National Language Support Functions

The National Language Support (NLS) functions provide support for applications that deal with multiple locales at once, especially for applications supporting OLE Automation. Locale information is passed to allow the application to interpret both the member names and the argument data in the proper locale context. On 32-bit Windows systems, an NLS API is part the system software; the information in this appendix applies only to 16-bit Windows systems.

Implemented by	Used by	Header filename	Import library name
OLE2NLS.DLL	Applications that support multiple national languages.	OLENLS.H	OLE2NLS.LIB

For OLE Automation, applications need to get locale information and to compare and transform strings into the proper format for each locale.

A *locale* is simply user preference information that denotes the user's language and sublanguage, represented as a list of values. National language support incorporates several disparate definitions of a locale into one coherent model. It is designed to be general enough at a low level to support multiple distinct high-level functions, such as the ANSI C locale functions.

A *code page* is the mapping between character glyphs (shapes) and the 1- or 2-byte numeric values that are used to represent them. Microsoft Windows version 3.1 uses one of several code pages, depending on the localized version of Windows installed. For example, the Russian version uses code page 1251 (Cyrillic), while the English/US and Western European versions use code page 1252 (Multilingual). For historical reasons, the Windows code page in effect is referred to as the ANSI code page.

Because only one code page is in effect at a time, it is impossible for a computer running English/US Windows version 3.1 to correctly display or print data from the Cyrillic code page; the fonts do not contain the Cyrillic characters. However, it can still manipulate the characters internally, and they will display correctly again if moved back to a machine running Russian Windows.

All NLS functions use the locale ID to identify which code page a piece of text is assumed to lie in; for example, when returning locale information (like month names) for Russian, the returned string can be meaningfully displayed in the Cyrillic code page only, because other code pages don't contain the appropriate characters. Similarly, when attempting to change the case of a string with the Russian locale, the case-mapping rules assume the characters are in the Cyrillic code page.

These functions can be divided into two categories:

- String transformation − NLS functions support uppercasing, lowercasing, generating sort keys (all locale-dependent), and getting string type information.
- Locale manipulation − NLS functions return information about installed locales for use in string transformations.

Overview of NLS Functions

The following table lists the NLS functions.

Function	Purpose
CompareStringA	Compares two strings of the same locale.
LCMapStringA	Transforms the case or sort order of a string.
GetLocaleInfoA	Retrieves locale information from the user's system.
GetStringTypeA	Retrieves locale type information about each character in a string.
GetSystemDefaultLangID	Retrieves the default LANGID from a user's system. (1)
GetSystemDefaultLCID	Retrieves the default LCID from a user's system.
GetUserDefaultLangID	Retrieves the default LANGID from a user's system.
GetUserDefaultLCID	Retrieves the default LCID from a user's system. (1)

[1] Because Microsoft Windows is a single-user system, **GetUserDefaultLangID** and **GetUserDefaultLCID** return the same information as **GetSystemDefaultLangID** and **GetSystemDefaultLCID**.

Localized Member Names

An application may expose a set of objects whose members have names that differ across localized versions of the product. This poses a problem for programming languages that want to access such objects because it means that late binding will be sensitive to the locale of the application. The **IDispatch** and VTBL interfaces allow software developers a range of solutions which vary in cost of implementation and quality of national language support. All methods of the **IDispatch** interface that are potentially sensitive to language are passed a locale ID (LCID).

Following are some of the possible approaches a class implementation may take:

- Accept any LCID and use the same member names in all locales. This is acceptable if the interface will typically be accessed only by advanced users. For example, the member names for OLE interfaces will never be localized.
- Simply return an error (DISP_E_UNKNOWNLCID) if the caller's LCID doesn't match the localized version of the class. This would prevent users from being able to write late-bound code which runs on machines with different localized implementations of the class.
- Recognize the particular version's localized names, as well as one language which is recognized in all versions. For example, a French version might accept French and English names, where English is the language supported in all versions. This would constrain users, who want to write code which runs in all countries, to use English.
- Accept all LCIDs supported by all versions of the product. This means that the implementation of **GetIDsOfNames** would need to interpret the passed array of names based on the given LCID. This is the preferred solution because users would be able to write code in their national language and run the code on any localized version of the application.

At the very least, the application must check the LCID before interpreting member names. Also note that the meaning of parameters passed to a member function may depend on the caller's national language. For example, a spreadsheet application might interpret the arguments to a **SetFormula** method differently, depending on the LCID.

Locale ID (LCID)

The **IDispatch** interface uses the 32-bit Windows definition of a locale ID (LCID) to identify locales. An LCID is a DWORD value which contains the language ID (LANGID) in the lower word and a reserved value in the upper word. The bits are as follows:

{ewc msdncd, EWGraphic, group10448 0 /a "SDK_01.bmp"}

This LCID has the components necessary to uniquely identify one of the installed system-defined locales.

```
/*
 * LCID creation/extraction macros:
 *
 *         MAKELCID - construct locale ID from language ID and
 *                           country code.
 */
#define MAKELCID(l)     ((DWORD)(((WORD)(l)) | (((DWORD)((WORD)(0))) << 16)))
```

There are two predefined LCID values: LOCALE_SYSTEM_DEFAULT is the system default locale, and LOCALE_USER_DEFAULT is the current user's locale. However, when querying the NLS APIs for several pieces of information, it is more efficient to query once for the current locale with **GetSystemDefaultLCID** or **GetUserDefaultLCID**, rather than using these constants.

Language ID (LANGID)

A LANGID is a 16-bit value which is the combination of a primary and sublanguage ID. The bits are as follows:

{ewc msdncd, EWGraphic, group10448 1 /a "SDK_02.bmp"}

Macros are provided for constructing a LANGID and extracting the fields:

```
/*
 * Language ID creation/extraction macros:
 *
 *    MAKELANGID - construct language ID from primary language ID and
 *                 sublanguage ID.
 *    PRIMARYLANGID - extract primary language ID from a language ID.
 *    SUBLANGID - extract sublanguage ID from a language ID.
 *    LANGIDFROMLCID - get the language ID from a locale ID.
 */
#define MAKELANGID(p, s)                  ((((USHORT)(s)) << 10) | (USHORT)
(p))
#define PRIMARYLANGID(lgid)         ((USHORT)(lgid) & 0x3ff)
#define SUBLANGID(lgid)             ((USHORT)(lgid) >> 10)
#define LANGIDFROMLCID(lcid)        ((WORD)(lcid))
```

The following three combinations of primary and sublanguage IDs have special meanings:

PRIMARYLANGID	SUBLANGID	Meaning
LANG_NEUTRAL	SUBLANG_NEUTRAL	Language neutral
LANG_NEUTRAL	SUBLANG_SYS_DEFAULT	System default language
LANG_NEUTRAL	SUBLANG_DEFAULT	User default language

For primary language IDs, the range 0x200 to 0x3ff is user definable. The range 0x000 to 0x1ff is reserved for system use. The following table lists the primary language IDs supported by OLE Automation:

Language	PRIMARYLANGID
Neutral	0x00
Chinese	0x04
Czech	0x05
Danish	0x06
Dutch	0x13
English	0x09
Finnish	0x0b
French	0x0c
German	0x07
Greek	0x08
Hungarian	0x0e
Icelandic	0x0F
Italian	0x10
Japanese	0x11

Korean	0x12
Norwegian	0x14
Polish	0x15
Portuguese	0x16
Russian	0x19
Serbo Croatian	0x1a
Slovak	0x1b
Spanish	0x0a
Swedish	0x1d
Turkish	0x1F

For sublanguage IDs, the range 0x20 to 0x3f is user definable. The range 0x00 to 0x1f is reserved for system use. The following table lists the sublanguage IDs supported by OLE Automation:

Sublanguage	**SUBLANGID**
Neutral	0x00
Default	0x01
System Default	0x02
Chinese (Simplified)	0x02
Chinese (Traditional)	0x01
Dutch	0x01
Dutch (Belgian)	0x02
English (US)	0x01
English (UK)	0x02
English (Australian)	0x03
English (Canadian)	0x04
English (Irish)	0x06
English (New Zealand)	0x05
French	0x01
French (Belgian)	0x02
French (Canadian)	0x03
French (Swiss)	0x04
German	0x01
German (Swiss)	0x02
German (Austrian)	0x03
Greek	0x01
Icelandic	0x01
Italian	0x01
Italian (Swiss)	0x02
Japanese	0x01
Korean	0x01
Norwegian (Bokmal)	0x01
Norwegian (Nynorsk)	0x02
Portuguese	0x02
Portuguese (Brazilian)	0x01
Serbo Croatian (Latin)	0x01

Spanish (Castilian)1	0x01
Spanish (Mexican)	0x02
Spanish (Modern)1	0x03
Turkish	0x01

[1] The only difference between Spanish (Castilian) and Spanish (Modern) is the sort ordering. All of the LCType values are the same.

Locale Constants (LCTYPE)

An LCTYPE is a constant which specifies a particular piece of locale information.

```
typedef  DWORD  LCTYPE;
```

The list of supported LCTYPES follows. All values are null-terminated, variable length strings. Numeric values are expressed as strings of decimal digits unless otherwise noted. The values in the brackets indicate the maximum number of characters allowed for the string (including the null-termination). If no maximum is indicated, the string may be of variable length.

Constant name	Description
LOCALE_ILANGUAGE	A language ID represented in hexadecimal digits; see previous sections. [5]
LOCALE_SLANGUAGE	The full localized name of the language.
LOCALE_SENGLANGUAGE	The full English name of the language from the ISO Standard 639. This will always be restricted to characters that can be mapped into the ASCII 127-character subset.
LOCALE_SABBREVLANGNAME	The abbreviated name of the language, created by taking the two-letter language abbreviation, as found in ISO Standard 639, and adding a third letter as appropriate to indicate the sublanguage.
LOCALE_SNATIVELANGNAME	The native name of the language.
LOCALE_ICOUNTRY	The country code, based on international phone codes, also referred to as IBM country codes. [6]
LOCALE_SCOUNTRY	The full localized name of the country.
LOCALE_SENGCOUNTRY	The full English name of the country. This will always be restricted to characters that can be mapped into the ASCII 127-character subset.
LOCALE_SABBREVCTRYNAME	The abbreviated name of the country as found in ISO Standard 3166.
LOCALE_SNATIVECTRYNAME	The native name of the country.
LOCALE_IDEFAULTLANGUAGE	Language ID for the principal language spoken in this locale. This is provided so that partially specified locales can be completed with default

	values. [5]
LOCALE_IDEFAULTCOUNTRY	Country code for the principal country in this locale. This is provided so that partially specified locales can be completed with default values. [6]
LOCALE_IDEFAULTANSICODEPAGE	The ANSI code page associated with this locale. Format: 4 Unicode decimal digits plus a Unicode null-terminator. [10] [6]
LOCALE_IDEFAULTCODEPAGE	The OEM code page associated with the country. [6]
LOCALE_SLIST	Characters used to separate list items; for example, comma is used in many locales.
LOCALE_IMEASURE	This value is "0" for the metric system (S.I.) and "1" for the U.S. system of measurements. [2]
LOCALE_SDECIMAL	Characters used for the decimal separator.
LOCALE_STHOUSAND	Characters used as the separator between groups of digits left of the decimal.
LOCALE_SGROUPING	Sizes for each group of digits to the left of the decimal. An explicit size is required for each group; sizes are separated by semicolons. If the last value is 0, the preceding value is repeated. To group thousands, specify "3;0."
LOCALE_IDIGITS	The number of fractional digits. [3]
LOCALE_ILZERO	Whether to use leading zeros in decimal fields. [2] A setting of 0 means use no leading zeros; 1 means use leading zeros.
LOCALE_SNATIVEDIGITS	The ten characters that are the native equivalent of the ASCII '0-9'.
LOCALE_INEGNUMBER	Negative number mode. [2] "0" (1.1) "1" -1.1 "2" -1.1 "3" 1.1 "4" 1.1
LOCALE_SCURRENCY	The string used as the local monetary symbol.

LOCALE_SINTLSYMBOL	Three characters of the International monetary symbol specified in ISO 4217 *Codes for the Representation of Currencies and Funds,* followed by the character separating this string from the amount.
LOCALE_SMONDECIMALSEP	Characters used for the monetary decimal separators.
LOCALE_SMONTHOUSANDSEP	Characters used as monetary separator between groups of digits left of the decimal.
LOCALE_SMONGROUPING	Sizes for each group of monetary digits to the left of the decimal. An explicit size is needed for each group; sizes are separated by semicolons. If the last value is 0, the preceding value is repeated. To group thousands, specify "3;0".
LOCALE_ICURRDIGITS	Number of fractional digits for the local monetary format. [3]
LOCALE_IINTLCURRDIGITS	Number of fractional digits for the international monetary format. [3]
LOCALE_ICURRENCY	Positive currency mode. [2] "0" Prefix, no separation "1" Suffix, no separation "2" Prefix, 1-character separation "3" Suffix, 1-character separation
LOCALE_INEGCURR	Negative currency mode. [2] "0" ($1.1) "1" -$1.1 "2" $-1.1 "3" $1.1- "4" $(1.1$) "5" -1.1$ "6" 1.1-$ "7" 1.1$- "8" -1.1 $ (space before $) "9" -$ 1.1 (space after $) "10" 1.1 $- (space before $)
LOCALE_ICALENDARTYPE	The type of calendar currently in use. [2] "1" Gregorian (as in U.S.) "2" Gregorian (always English strings)

	"3" Era: Year of the Emperor (Japan) "4" Era: Year of the Republic of China "5" Tangun Era (Korea)
LOCALE_IOPTIONALCALENDAR	The additional calendar types available for this LCID; can be a null-separated list of all valid optional calendars. [2] "0" None available "1" Gregorian (as in U.S.) "2" Gregorian (always English strings) "3" Era: Year of the Emperor (Japan) "4" Era: Year of the Republic of China "5" Tangun Era (Korea)
LOCALE_SDATE	Characters used for the date separator.
LOCALE_STIME	Characters used for the time separator.
LOCALE_STIMEFORMAT	Time formatting string. [80]
LOCALE_SSHORTDATE	Short Date_Time formatting strings for this locale.
LOCALE_SLONGDATE	Long Date_Time formatting strings for this locale.
LOCALE_IDATE	Short Date format ordering specifier. [2] "0" Month - Day - Year "1" Day - Month - Year "2" Year - Month - Day
LOCALE_ILDATE	Long Date format ordering specifier. [2] "0" Month - Day - Year "1" Day - Month - Year "2" Year - Month - Day
LOCALE_ITIME	Time format specifier. [2] "0" AM/PM 12-hour format "1" 24-hour format
LOCALE_ITIMEMARKPOSN	Whether the time marker string (AM|PM) precedes or follows the time string. (The registry value is named ITimePrefix for previous Far East version compatibility.) "0" Suffix (9:15 AM) "1" Prefix (AM 9:15)
LOCALE_ICENTURY	Whether to use full 4-digit

	century. [2] "0" Two digit "1" Full century
LOCALE_ITLZERO	Whether to use leading zeros in time fields. [2] "0" No leading zeros "1" Leading zeros for hours
LOCALE_IDAYLZERO	Whether to use leading zeros in day fields. [2] "0" No leading zeros "1" Leading zeros
LOCALE_IMONLZERO	Whether to use leading zeros in month fields. [2] "0" No leading zeros "1" Leading zeros
LOCALE_S1159	String for the AM designator.
LOCALE_S2359	String for the PM designator.
LOCALE_IFIRSTWEEKOFYEAR	Specifies which week of the year is considered first. [2] "0" Week containing 1/1 is the first week of the year. "1" First full week following 1/1is the first week of the year. "2" First week with at least 4 days is the first week of the year.
LOCALE_IFIRSTDAYOFWEEK	Specifies the day considered first in the week. [2] "0" SDAYNAME1 "1" SDAYNAME2 "2" SDAYNAME3 "3" SDAYNAME4 "4" SDAYNAME5 "5" SDAYNAME6 "6" DAYNAME7
LOCALE_SDAYNAME1	Long name for Monday.
LOCALE_SDAYNAME2	Long name for Tuesday.
LOCALE_SDAYNAME2	Long name for Tuesday.
LOCALE_SDAYNAME3	Long name for Wednesday.
LOCALE_SDAYNAME4	Long name for Thursday.
LOCALE_SDAYNAME5	Long name for Friday.
LOCALE_SDAYNAME6	Long name for Saturday.
LOCALE_SDAYNAME7	Long name for Sunday.
LOCALE_SABBREVDAYNAME1	Abbreviated name for Monday.
LOCALE_SABBREVDAYNAME2	Abbreviated name for Tuesday.
LOCALE_SABBREVDAYNAME3	Abbreviated name for Wednesday.

LOCALE_SABBREVDAYNAME4	Abbreviated name for Thursday.
LOCALE_SABBREVDAYNAME5	Abbreviated name for Friday.
LOCALE_SABBREVDAYNAME6	Abbreviated name for Saturday.
LOCALE_SABBREVDAYNAME7	Abbreviated name for Sunday.
LOCALE_SMONTHNAME1	Long name for January.
LOCALE_SMONTHNAME2	Long name for February.
LOCALE_SMONTHNAME3	Long name for March.
LOCALE_SMONTHNAME4	Long name for April.
LOCALE_SMONTHNAME5	Long name for May.
LOCALE_SMONTHNAME6	Long name for June.
LOCALE_SMONTHNAME7	Long name for July.
LOCALE_SMONTHNAME8	Long name for August.
LOCALE_SMONTHNAME9	Long name for September.
LOCALE_SMONTHNAME10	Long name for October.
LOCALE_SMONTHNAME11	Long name for November.
LOCALE_SMONTHNAME12	Long name for December.
LOCALE_SMONTHNAME13	Native name for 13th month, if it exists.
LOCALE_SABBREVMONTHNAME1	Abbreviated name for January.
LOCALE_SABBREVMONTHNAME2	Abbreviated name for February.
LOCALE_SABBREVMONTHNAME3	Abbreviated name for March.
LOCALE_SABBREVMONTHNAME4	Abbreviated name for April.
LOCALE_SABBREVMONTHNAME5	Abbreviated name for May.
LOCALE_SABBREVMONTHNAME6	Abbreviated name for June.
LOCALE_SABBREVMONTHNAME7	Abbreviated name for July.
LOCALE_SABBREVMONTHNAME8	Abbreviated name for August.
LOCALE_SABBREVMONTHNAME9	Abbreviated name for September.
LOCALE_SABBREVMONTHNAME10	Abbreviated name for October.
LOCALE_SABBREVMONTHNAME11	Abbreviated name for November.
LOCALE_SABBREVMONTHNAME12	Abbreviated name for December.
LOCALE_SABBREVMONTHNAME13	Native abbreviated name for 13th month, if it exists.
LOCALE_SPOSITIVESIGN	String value for the positive sign.
LOCALE_SNEGATIVESIGN	String value for the negative sign.
LOCALE_IPOSSIGNPOSN	Formatting index for positive values. [2] "0" Parentheses surround the amount and the monetary symbol.

	"1" The sign string precedes the amount and the monetary symbol. "2" The sign string precedes the amount and the monetary symbol. "3" The sign string precedes the amount and the monetary symbol. "4" The sign string precedes the amount and the monetary symbol.
LOCALE_INEGSIGNPOSN	Formatting index for negative values. [2] "0" Parentheses surround the amount and the monetary symbol. "1" The sign string precedes the amount and the monetary symbol. "2" The sign string precedes the amount and the monetary symbol. "3" The sign string precedes the amount and the monetary symbol. "4" The sign string precedes the amount and the monetary symbol.
LOCALE_IPOSSYMPRECEDES	"1" if the monetary symbol precedes; "0" if it succeeds a positive amount. [2]
LOCALE_IPOSSEPBYSPACE	"1" if the monetary symbol is separated by a space from a positive amount; "0" otherwise. [2]
LOCALE_INEGSYMPRECEDES	"1" if the monetary symbol precedes; "0" if it succeeds a negative amount. [2]
LOCALE_INEGSEPBYSPACE	"1" if the monetary symbol is separated by a space from a negative amount; "0" otherwise. [2]

The following table shows the equivalence between LCTYPE values and the information stored in the [intl] section of WIN.INI. These values will be retrieved from WIN.INI if information for the current system locale is queried. Values for LCTYPEs not in the following table do not depend on information stored in WIN.INI.

WIN.INI settings	**LCTYPE**
sLanguage (1)	LOCALE_SABBREVLANGNAME
iCountry	LOCALE_ICOUNTRY

sCountry	LOCALE_SCOUNTRY
sList	LOCALE_SLIST
iMeasure	LOCALE_IMEASURE
sDecimal	LOCALE_SDECIMAL
sThousand	LOCALE_STHOUSAND
iDigits	LOCALE_IDIGITS
iLZero	LOCALE_ILZERO
sCurrency	LOCALE_SCURRENCY
iCurrDigits	LOCALE_ICURRDIGITS
iCurrency	LOCALE_ICURRENCY
iNegCurr	LOCALE_INEGCURR
sDate	LOCALE_SDATE
sTime	LOCALE_STIME
sShortDate	LOCALE_SSHORTDATE
sLongDate	LOCALE_SLONGDATE
iDate	LOCALE_IDATE
iTime	LOCALE_ITIME
iTLZero	LOCALE_ITLZERO
s1159	LOCALE_S1159
s2359	LOCALE_S2359

[1] Unlike in WIN.INI, values returned by LOCALE_SABBREVLANGNAME are always in uppercase.

CompareStringA

int CompareStringA(*LCID*, *dwCmpFlags*, *lpString1*, *cchCount1*, *lpString2*, *cchCount2*)
LCID *LCID*
DWORD *dwCmpFlags*
LPCSTR *lpString1*
int *cchCount1*
LPCSTR *lpString2*
int *cchCount2*

Compares two character strings of the same locale according to the supplied LCID.

Parameters

LCID
Locale context for the comparison. The strings are assumed to be represented in the default ANSI code page for this locale.

dwCmpFlags
Flags that indicate the character traits to use or ignore when comparing the two strings. Several flags can be combined (in the case of this function, there are no illegal combinations of flags), or none can be used at all. Compare flags include the following.

Value	Meaning
NORM_IGNORECASE	Ignore case; default is OFF.
NORM_IGNOREKANATYPE	Ignore Japanese hiragana/katakana character differences; default is OFF.
NORM_IGNORENONSPACE	Ignore nonspacing marks (accents, diacritics and vowel marks); default is OFF.
NORM_IGNORESYMBOLS	Ignore symbols; default is OFF.
NORM_IGNOREWIDTH	Ignore character width, default is OFF.

lpString1 and *lpString2*
The two strings to be compared.

cchCount1 and *cchCount2*
The character counts of the two strings. The count does *not* include the null-terminator (if any). If either *cchCount1* or *cchCount2* is -1, the corresponding string is assumed to be null-terminated and the length will be calculated automatically.

Return Value

Value	Meaning
0	Failure.
1	lpString1 is less than lpString2.
2	lpString1 is equal to lpString2.
3	lpString1 is greater than lpString2.

Comments

When used without any flags, this function uses the same sorting algorithm as **lstrcmp** in the given locale. When used with NORM_IGNORECASE, the same algorithm as **lstrcmpi** is used.

For DBCS locales, the flag NORM_IGNORECASE has an effect on all the wide (two-byte) characters as well as the narrow (one-byte) characters. This includes the wide Greek and Cyrillic characters.

In Chinese Simplified, the sorting order used to compare the strings is based on the following sequence: symbols, digit numbers, English letters and Chinese Simplified characters. The characters within each group sort in character code order.

In Chinese Traditional, the sorting order used to compare the strings is based on the "number of strokes" in the characters. Symbols, digit numbers, and English characters are considered to have zero strokes. The sort sequence is symbols, digit numbers, English letters and Chinese Traditional characters. The characters within each stroke-number group sort in character code order.

In Japanese, the sorting order used to compare the strings is based on the Japanese 50-on sorting sequence. The Kanji ideographic characters sort in character code order.

In Japanese, the flag NORM_IGNORENONSPACE has an effect on the daku-on, handaku-on, chou-on, you-on, and soku-on modifiers, and on the repeat kana/kanji characters.

In Korean, the sort order is based on the sequence: symbols, digit numbers, Jaso and Hangeul, Hanja, and English. Within the Jaso-Hangeul group, each Jaso character is followed by the Hangeuls that start with that Jaso. Hanja characters are sorted in Hangeul pronounciation order. Where multiple Hanja have the same Hangeul pronounciation they are sorted in character code order.

The NORM_IGNORENONSPACE flag only has an effect for the locales in which accented characters are sorted in a second pass from main characters (that is, all characters in the string are first compared without regard to accents and, if the strings are equal, a second pass over the strings to compare accents is performed). In this case, this flag causes the second pass to not be performed. Some locales sort accented characters in the first pass, in which case this flag will have no effect.

Note that if the return value is 2, the two strings are "equal" in the collation sense, though not necessarily identical (case might be ignored, and so on).

If the two strings are of different lengths, they are compared up to the length of the shortest one. If they are equal to that point, the return value will indicate that the longer string is greater.

To maintain the C run-time convention of comparing strings, the value 2 can be subtracted from a nonzero return value. The meaning of < 0, == 0, and > 0 is then consistent with the C run-time conventions.

LCMapStringA

int LCMapStringA(*LCID*, *dwMapFlags*, *lpSrcStr*, *cchSrc*, *lpDestStr*, *cchDest*)
LCID *LCID*
DWORD *dwMapFlags*
LPCSTR *lpSrcStr*
int *cchSrc*
LPSTR *lpDestStr*
int *cchDest*

Parameters

LCID
Locale context for the mapping. The strings are assumed to be represented in the default ANSI code page for this locale.

dwMapFlags
Flags that indicate what type of transformation is to occur during mapping. Several flags can be combined on a single transformation (though some combinations are illegal). Mapping options include the following.

Name	Meaning
LCMAP_LOWERCASE	Lowercase.
LCMAP_UPPERCASE	Uppercase.
LCMAP_SORTKEY	Character sort key.
LCMAP_HALFWIDTH	Narrow characters (where applicable).
LCMAP_FULLWIDTH	Wide characters (where applicable).
LCMAP_HIRAGANA	Hiragana.
LCMAP_KATAKANA	Katakana.
NORM_IGNORECASE	Ignore case; default is OFF.
NORM_IGNORENONSPACE	Ignore nonspacing; default is OFF.
NORM_IGNOREWIDTH	Ignore character width; default is OFF.
NORM_IGNOREKANATYPE	Ignore Japanese hiragana/katakana character differences; default is OFF.
NORM_IGNORESYMBOLS	Ignore symbols; default is OFF.

The latter five options (NORM_IGNORECASE, NORM_IGNORENONSPACE, NORM_IGNOREWIDTH, NORM_IGNOREKANATYPE,and NORM_IGNORESYMBOLS) are normalization options that can only be used in combination with the LCMAP_SORTKEY conversion option.

Conversion options can be combined only when they are taken from the following three groups, and then only when there is no more than one option from each group:

- Casing options (LCMAP_LOWERCASE, LCMAP_UPPERCASE)
- Width options (LCMAP_HALFWIDTH, LCMAP_FULLWIDTH)
- Kana options (LCMAP_HIRAGANA, LCMAP_KATAKANA)

lpSrcStr
Pointer to the supplied string to be mapped.

cchSrc
Character count of the input string buffer. If -1, *lpSrcStr* is assumed to be null-terminated and the length will be calculated automatically.

lpDestStr
Pointer to the memory buffer to store the resulting mapped string.

cchDest
Character count of the memory buffer pointed to by *lpDestStr*. If *cchDest* is 0, then the return value of this function is the number of characters required to hold the mapped string. The *lpDestStr* pointer is not referenced in this case.

Return Value

Value	**Meaning**
0	Failure
The number of characters written to *lpDestSt*	Success

Comments

LCMapStringA maps one character string to another, performing the specified locale-dependent translation.

The flag LCMAP_UPPER produces the same result as **AnsiUpper** in the given locale; the flag LCMAP_LOWER produces the same result as **AnsiLower**. In particular, like these functions, this function always maps a single character to a single character.

The mapped string will be null-terminated if the source string is null-terminated.

When used with LCMAP_UPPER and LCMAP_LOWER, the *lpSrcStr* and *lpDestStr* may be the same to produce an in-place mapping. When LCMAP_SORTKEY is used, the *lpSrcStr* and *lpDestStr* pointers may *not* be the same; an error will result in this case.

The LCMAP_SORTKEY transformation transforms two strings such that when compared with the standard C library function **strcmp** (by strict numerical valuation of their characters), the same order would result as if the original strings were compared with **CompareStringA**. When LCMAP_SORTKEY is specified, the output string will be a string (without NULLs, except for the terminator), but the "character" values will not be meaningful display values. This is similar behavior to the ANSI C function **strxfrm**.

GetLocaleInfoA

int GetLocaleInfoA(*LCID*, *LCType*, *lpLCData*, *cchData*)
  LCID *LCID*
  LCTYPE *LCType*
  LPSTR *lpLCData*
  int *cchData*

Parameters

LCID
  ID for a locale. The returned string will be represented in the default ANSI code page for this locale.

LCType
  Flag that indicates the type of information to be returned by the call. See the listing of constant values in this chapter. LOCALE_NOUSEROVERRIDE **|** LCTYPE indicates that the desired information will always be retrieved from the locale database, even if the LCID is the current one, and the user has changed some of the values with the control panel. If this flag is not specified, values in WIN.INI take precedence over the database settings when getting values for the current system default locale.

lpLCData
  Pointer to the memory where **GetLocaleInfoA** will return the requested data. This pointer is not referenced if *cchData* is 0.

cchData
  Character count of the supplied *lpLCData* memory buffer. If *cchData* is 0, the return value is the number of characters required to hold the string, including the terminating NULL character. *lpLCData* is not referenced in this case.

Return Value

Value	**Meaning**
0	Failure
The number of characters copied, including the terminating NULL character	Success

Comments

GetLocaleInfoA returns one of the various pieces of information about a locale by querying the stored locale database or WIN.INI. The call also indicates how much memory is necessary to contain the desired information.

The information returned is always a null-terminated string. No integers are returned by this function ; numeric values are returned as text (see format descriptions under LCTYPE).

GetStringTypeA

BOOL GetStringTypeA(*LCID*, *dwInfoType*, *lpSrcStr*, *cchSrc, lpCharType*)
LCID *LCID*
DWORD *dwInfoType*
LPCSTR *lpSrcStr*
int *cchSrc*
LPWORD *lpCharType*

Parameters

LCID
Locale context for the mapping. The string is assumed to be represented in the default ANSI code page for this locale.

dwInfoType
Type of character information to retrieve. The various types are divided into different levels (see Comments for a list of the information included in each type). The options are mutually exclusive. The following types are supported:

- CT_CTYPE1
- CT_CTYPE2
- CT_CTYPE3

lpSrcStr
String for which character types are requested. If *cchSrc* is -1, *lpSrcStr* is assumed to be null-terminated.

cchSrc
Character count of *lpSrcStr*. If *cchSrc* is -1, *lpSrcStr* is assumed to be null-terminated. Note that this must also be the character count of *lpCharType*.

lpCharType
Array of the same length as *lpSrcStr* (*cchSrc*). On output, the array contains one word corresponding to each character in *lpSrcStr*.

Return Value

Value	**Meaning**
0	Failure
1	Success

Comments

The *lpSrcStr* and *lpCharType* pointers may *not* be the same; in this case the error ERROR_INVALID_PARAMETER results.

The character type bits are divided up into several levels. One level's information can be retrieved by a single call.

This function supports three character types:

- Ctype 1
- Ctype 2
- Ctype 3

Ctype 1 types support ANSI C and POSIX character typing functions. A bitwise OR of these values is returned when *dwInfoType* is set to CT_CTYPE1. For DBCS locales, the Ctype 1 attributes apply to both the narrow characters and the wide characters. The Japanese hiragana and katakana characters, and the kanji ideograph characters all have the C1_ALPHA attribute.

The following table lists the Ctype 1 character types.

Name	Value	Meaning
C1_UPPER	0x0001	Uppercase1
C1_LOWER	0x0002	Lowercase1
C1_DIGIT	0x0004	Decimal digits
C1_SPACE	0x0008	Space characters
C1_PUNCT	0x0010	Punctuation
C1_CNTRL	0x0020	Control characters
C1_BLANK	0x0040	Blank characters
C1_XDIGIT	0x0080	Hexadecimal digits
C1_ALPHA	0x0100	Any letter

[1] The Windows version 3.1 functions **IsCharUpper** and **IsCharLower** do not always produce correct results for characters in the range 0x80-0x9f, so they may produce different results than this function for characters in that range. (For example, the German Windows version 3.1 language driver incorrectly reports 0x9a, lowercase s hacek, as uppercase).

Ctype 2 types support the proper layout of text. For DBCS locales, Ctype 2 applies to both narrow and wide characters. The directional attributes are assigned so that the BiDi layout algorithm standardized by Unicode produces the correct results. See *The Unicode Standard: Worldwide Character Encoding* from Addison-Wesley for more information on the use of these attributes.

	Name	Value	Meaning
Strong	C2_LEFTTORIGHT	0x1	Left to right
	C2_RIGHTTOLEFT	0x2	Right to left
Weak	C2_EUROPENUMBER	0x3	European number, European digit
	C2_EUROPESEPARATOR	0x4	European numeric separator
	C2_EUROPETERMINATOR	0x5	European numeric terminator
	C2_ARABICNUMBER	0x6	Arabic number
	C2_COMMONSEPARATOR	0x7	Common numeric separator
Neutral	C2_BLOCKSEPARATOR	0x8	Block separator
	C2_SEGMENTSEPARATOR	0x9	Segment separator
	C2_WHITESPACE	0xA	White space
	C2_OTHERNEUTRAL	0xB	Other neutrals
Not applicable	C2_NOTAPPLICABLE	0x0	No implicit direction (for example, control codes)

Ctype 3 types are general text-processing information. A bitwise OR of these values is returned when *dwInfoType* is set to CT_CTYPE3. For DBCS locales, the Ctype 3 attributes apply to both the narrow characters and the wide characters. The Japanese hiragana and katakana characters and the kanji ideograph characters all have the C3_ALPHA attribute.

Name	**Value**	**Meaning**
C3_NONSPACING	0x1	Nonspacing mark
C3_DIACRITIC	0x2	Diacritic nonspacing mark
C3_VOWELMARK	0x4	Vowel nonspacing mark
C3_SYMBOL	0x8	Symbol
C3_KATAKANA	0x10	Katakana character
C3_HIRAGANA	0x20	Hiragana character
C3_HALFWIDTH	0x40	Narrow character
C3_FULLWIDTH	0x80	Wide character
C3_IDEOGRAPH	0x100	Ideograph
C3_ALPHA	0x8000	Any letter
C3_NOTAPPLICABLE	0x0	Not applicable

GetSystemDefaultLangID

Returns the system default language ID.

LANGID GetSystemDefaultLangID(void);

Return Value

Value	Meaning
0	Failure
The system default LANGID	Success

Comments

See **GetSystemDefaultLCID** for information on how this value is determined.

GetSystemDefaultLCID

Returns the system default LCID.

LCID GetSystemDefaultLCID(void)

Return Value

Value	Meaning
0	Failure
System default locale ID	Success

Comments

The return value is determined by examining the values of *sLanguage* and *iCountry* in WIN.INI, and comparing the values to those in the stored locale database. If no matching values are found, or the required values cannot be read from WIN.INI, or if the stored locale database cannot be loaded, the value 0 is returned.

GetUserDefaultLangID

Returns the user default LANGID.

LANGID GetUserDefaultLangID(void)

Return Value

Value	Meaning
0	Failure
The user default LANGID	Success

Comments

Because Windows version 3.1 is a single-user system, the value returned from this function is always the same as that returned from **GetSystemDefaultLangID**.

GetUserDefaultLCID

Returns the user default LCID.

LCID GetUserDefaultLCID(void)

Return Value

Value	**Meaning**
0	Failure
The user default locale ID	Success

Comments

Because Windows version 3.1 is a single-user system, the value returned by this function is always the same as that returned from **GetSystemDefaultLCID**.

Files You Need

To use OLE Automation, you need the following files. The filenames shown in **bold** are required by your application at run time. Note that you should not ship the files with your 32-bit Windows application; these files are provided as part of the32-bit Windows operating system.

32-bit filenames	16-bit filenames	Purpose
None	OLE2.REG	Registers OLE and OLE Automation. OLE is a system component on 32-bit systems, and therefore no .REG file is required.
None	**OLE2NLS.DLL** OLE2NLS.LIB OLENLS.H	Provides functions for applications that support multiple national languages. Note that on 32-bit systems, NLS features are provided by the Win32 NLS API.
OLEPRX32.DLL	**OLE2PROX.DLL**	Coordinates object access across processes.
MKTYPLIB.EXE	MKTYPLIB.EXE	Builds type libraries from interface descriptions.
OLEAUT32.DLL OLEAUT32.LIB OLEAUTO.H	**TYPELIB.DLL** DISPATCH.H	Accesses type libraries.
32-bit filenames	**16-bit filenames**	**Purpose**
	OLE2DISP.DLL OLE2DISP.LIB DISPATCH.H	Provides functions for creating OLE Automation objects and retrieving active objects at run time. Accesses OLE Automation objects by invoking methods and properties.
OLE32.DLL OLE32.LIB OLE2.H	**OLE2.DLL** OLE2.LIB OLE2.H	Provides OLE functions that may be used by OLE objects or containers.
	COMPOBJ.DLL COMPOBJ.LIB OLE2.H COMPOBJ.H	Supports component object creation and access.
	STORAGE.DLL STORAGE.LIB OLE2.H STORAGE.H	Supports access to subfiles, such as type libraries, within compound documents.

Information for Visual Basic Programmers

Visual Basic provides full support for OLE Automation. The following table lists how Visual Basic statements translate into OLE APIs.

Visual Basic Statement	**OLE APIs**
CreateObject ("*ProgID*")	**CLSIDFromProgID**() **CoCreateInstance**() **QueryInterface**() to get **IDispatch** interface
GetObject ("*filename*", "*ProgID*")	**CLSIDFromProgID**() **CoCreateInstance**() **QueryInterface**() for **IPersistFile** interface **Load**() on **IPersistFile** interface **QueryInterface**() to get **IDispatch** interface
GetObject ("*filename*")	**CreateBindCtx**() creates the bind context for the subsequent functions. **MkParseDisplayName**() returns a moniker handle for **BindMoniker**. **BindMoniker**() returns a pointer to the IDispatch interface. **Release**() on moniker handle. **Release**() on context.
GetObject (, "*ProgID*")	**CLSIDFromProgID**() **GetActiveObject**() on class ID. **QueryInterface**() to get **IDispatch** interface.
Dim *x* **As New** *interface*	Find CLSID for *interface*. **CoCreateInstance**() **QueryInterface**()

How OLE Automation Compares Strings

This appendix describes how the OLE Automation implementations compare strings. Understanding these comparisons is useful when creating applications that support national languages that use accents and digraphs. The information in this appendix applies to the following:

- **CreateStdDispatch** (OLEAUT32.DLL on 32-bit Windows systems, OLE2DISP.DLL on 16-bit Windows systems)
- **DispGetIDsOfNames** (OLEAUT32.DLL on 32-bit Windows systems, OLE2DISP.DLL on 16-bit Windows systems)
- **ITypeLib::FindName** (OLEAUT32.DLL on 32-bit Windows systems, TYPELIB.DLL on 16-bit Windows systems)
- **ITypeInfo::GetIDsOfNames** (OLEAUT32.DLL on 32-bit Windows systems, TYPELIB.DLL on 16-bit Windows systems)
- MkTypLib (MKTYPLIB.EXE)

When comparing strings, the listed OLE Automation components use the following rules:

- Comparisons are sensitive to locale based on the string's LCID. A string must have an LCID that is supported by the application or type library. Locales and LCIDs are described in the section "Supporting Multiple National Languages," in Chapter 2.
- Accent characters are ignored. For example, the string "à" compares the same as "a."
- Case is ignored. For example, the string "A" compares the same as "a."
- Comparisons are sensitive to digraphs. For example, the string "Æ" is not the same as "AE."
- For Japanese, Korean, and Chinese locales, **ITypeLib::FindName** and **ITypeLib::GetIDsOfNames** ignore width and kanatype.

Handling GUIDs

Globally unique identifiers (GUIDs) appear in many places in a typical OLE Automation application. GUID errors can cause persistent bugs. To help you avoid GUID problems, this appendix lists all the places GUIDs appear in a typical OLE Automation application, describes common characteristics of GUID bugs, and offers some GUID management techniques.

GUIDs are the same as UUIDs (Universally Unique Identifiers). A class identifier (CLSID) is a UUID/GUID that refers to a class.

The System Registry

The system registry is a central repository containing information about objects. GUIDs are used to index that information. You can view the registration information on your system by running REGEDIT with the **/V** option:

```
regedit /v
```

Typically, a name is connected with a GUID (for example, "Hello.Application" maps to a GUID) and then the GUID is connected to all the other relevant aspects (for example, the GUID maps to "HELLO.EXE").

You create GUIDs with the tool GUIDGEN.EXE. Running GUIDGEN.EXE produces a huge hex number which uniquely identifies your object, whether it be a class, an interface, a library, or some other kind of object.

Where GUIDs Live

GUIDs appear in the following places:

- .REG files – When you create an application, you usually create one or more .REG files. The .REG files contain the GUIDs for the classes that your application exposes. These GUIDs are added to the registry when you run REGEDIT.EXE to register your classes, or when you register type information with **LoadTypeLib**.
- The system registry – Contains the GUIDs for your classes in multiple places. This is where OLE and applications get information about your classes.
- .ODL files – When you describe objects in an Object Description Language (.ODL) file, you provide a GUID for each object. Compiling the .ODL file with MKTYPLIB.EXE places the GUIDs in a type library, which usually exists as a .TLB file. If you change a GUID in an .ODL file, make sure that you run MkTypLib again.
- .TLB files – Type libraries describe your classes, and this information includes the GUIDs for the classes. You can browse .TLB files using the TIBROWSE sample application supplied with OLE.
- .H files – Most application developers will declare CLSIDs for their classes in a header file using the DEFINE_GUID macro.

Troubleshooting

The following are common problems with GUIDs.

Problem

"**GetObject** can't seem to create an instance of my application."

Solution

Visual Basic uses the OLE calls listed in Appendix C to find the .EXE file that creates your application instance.

{ewl msdncd, EWGraphic, group10452 0 /a "SDK.BMP"} Visual Basic proceeds as follows

1. Looks up the GUID for your object. For the Hello application, Visual Basic maps the ProgID Hello.Application into a GUID.
2. Finds the object's server. This is HELLO.EXE for the Hello application.
3. Launches the application.

If an error occurs, check the following:

- Did you remember to run the .REG file?
- Are the entries in the registry correct? Check to see if all the GUIDs match.
- Can the application be launched? The executable (.EXE) file for the application, listed in the LocalServer entry, should either be on the path or it should be fully specified, for example, `c:\ole2\sample\hello\hello.bin`.

Problem

"When I use **GetObject**, the application launches and then the **GetObject** call fails."

Solution

Normally, a class factory is registered using **CoRegisterClassObject** when your application is started. Some applications register their class factories only when launched with the **/Automation** switch. If you inherited code or copied a sample, you should find out whether it checks for this switch. The **/Automation** option could appear in the .REG file, the registry, or in your development environment.

Problem

"**GetTypeInfoOfGuid**() is failing to get the type information from my type library."

Solution

When you call **GetTypeInfoOfGuid**, you provide a GUID. If this GUID doesn't match the GUID in your .TLB file, no type information will be returned. The GUID in your code is likely to be declared in a header file. You can check the GUID in your .TLB file by using BROWSE, which is provided with OLE, or with the Visual Basic Object Browser.

GUID Management

The problem with managing GUIDs is that they are pervasive, and their length prohibits simple comparisons.

The single most important technique in managing GUIDs is to keep a central list of all the GUIDs you use. Use the MAKE_GUID macro or the GUIDGEN tool with the **-n** option to generate the required number of GUIDs, and place the resulting strings in the first column of a spreadsheet. Each time you use a new GUID, enter a description of its purpose in the second column of the spreadsheet.

Keeping a central list has several advantages:

- Listing all the GUIDs in one location may prevent you from accidentally reusing a GUID. This often happens when you clone an application in order to create another one.
- You can use the spreadsheet to compare GUIDs. You can check that a GUID is correct by copying it from the place where it is being used (for example, a .REG file), pasting it into the spreadsheet, and then comparing the two cells with the "="operator.
- You'll have a record of your GUID use in case of future problems, and you'll have a single source of information to find the GUID for your object.

Overview of OLE Automation

OLE Automation is a technology that lets software packages expose their unique features to scripting tools and other applications. OLE Automation uses the OLE Component Object Model (COM), but may be implemented independently from other OLE features, such as *in-place activation*. Using OLE Automation, you can:

- Create applications and programming tools that expose objects.
- Create and manipulate objects exposed in one application from another application.
- Create tools that access and manipulate objects. These tools can include embedded macro languages, external programming tools, object browsers, and compilers.

The objects an application or programming tool exposes are called *OLE Automation objects*. Applications and programming tools that access those objects are called *OLE Automation controllers*. OLE Automation objects and controllers interact as follows:

{ewc msdncd, EWGraphic, group10453 0 /a "SDK_01.bmp"}

Applications and other software packages define and expose objects, which can be acted upon by OLE Automation controllers. Type information describes the exposed objects, and can be used by OLE Automation controllers at either compile time or run time.

Why Expose Objects?

Exposing objects provides a way to manipulate an application's tools programmatically. This allows your customers to use a programming tool to automate repetitive tasks that you might not anticipate.

For example, Microsoft Excel exposes a variety of objects that you can use to build applications. One such object is the Workbook, which contains a group of related worksheets, charts, and macros – the Excel equivalent of a three-ring binder. Using OLE Automation, you could write an application that accesses Excel Workbook objects, possibly to print them, as in the following diagram:

{ewc msdncd, EWGraphic, group10453 1 /a "SDK_02.bmp"}

With OLE Automation, solution providers can use your general-purpose objects to build applications that target a specific task. For example, a general-purpose drawing tool could expose objects that draw boxes, lines, and arrows, or insert text, and so forth. Another programmer could build a flowcharting tool by accessing the exposed objects and adding a user interface and other application-specific features.

Exposing objects to OLE Automation or supporting OLE Automation within a macro language offers several benefits:

- Exposed objects from many applications are available in a single programming environment. Software developers can choose from these objects to create solutions that span applications.
- Exposed objects are accessible from any macro language or programming tool that implements OLE Automation. Systems integrators are not limited to the programming language in which the objects were developed; instead, they can choose the programming tool or macro language that best suits their own needs and capabilities.
- Object names can remain consistent across versions of an application.
- Object names can automatically conform to the user's national language.

What Is an OLE Automation Object?

An *OLE Automation object* is an instance of a class that exposes properties, methods, and events to OLE Automation controllers. An *OLE Automation server* is an application or library that is capable of creating one or more OLE Automation objects. For example, Microsoft Excel exposes many objects, which you can use to create new applications and programming tools. Within Excel, objects are organized hierarchically, with an object named Application at the top of the hierarchy. The following figure shows some of Excel's objects.

{ewc msdncd, EWGraphic, group10453 2 /a "SDK_03.bmp"}

Each OLE Automation object has its own unique member functions. Exposing the member functions makes the object programmable by OLE Automation controllers. You can expose two types of members for an object:

- *Methods* are actions that an object can perform. For example, Excel's Worksheet object provides a Calculate method, which recalculates the values in the worksheet.
- *Properties* are functions that access information about the state of an object. The Worksheet object's Visible property determines whether the worksheet is visible on the display.

For example, you could expose the following objects in a document-based application by implementing these methods and properties:

OLE Automation object	**Methods**	**Properties**
Application	Help Quit Save Repeat Undo	ActiveDocument Application Caption DefaultFilePath Documents Height Name Parent Path Printers StatusBar Top Value Visible Width
Document	Activate Close NewWindow Print PrintPreview RevertToSaved Save SaveAs	Application Author Comments FullName Keywords Name Parent Path ReadOnly Saved Subject Title Value

Often, an application needs several instances of an object. For example, an OLE application based on Excel may have multiple workbooks. To provide an easy way to access and program the workbooks, Excel exposes an object named Workbooks, which refers to all the current Workbook objects. Workbooks is a *collection object*; other collection objects in Excel are shaded in the preceding figure. Collection objects let you iterate over the objects they manage. If you create an application with a multiple-document interface (MDI), you might expose a collection object named Documents with the following methods and properties:

Collection object	Methods	Properties
Documents	Add Close Item Open	Application Count Parent

What Is an OLE Automation Controller?

An *OLE Automation controller* is an application or programming tool that manipulates one or more OLE Automation objects. The objects may exist in the same application or in another application. OLE Automation controllers can use existing objects, create new instances of objects, get and set properties, and invoke methods that the object supports.

Microsoft Visual Basic™ is an OLE Automation controller. Using Visual Basic and similar programming tools, you can create packaged scripts to access OLE Automation objects. You can also create controllers by:

- Writing code within an application that accesses another application's exposed objects through OLE Automation.
- Revising an existing programming tool, such as an embedded macro language, to add support for OLE Automation.
- Developing a new application, such as a compiler or type information browser, that supports OLE Automation.

How Do Controllers and Objects Interact?

OLE Automation controllers can access OLE Automation objects in two different ways:

- By using the **IDispatch** interface.
- By directly calling one of the member functions in the object's virtual function table (VTBL).

An *interface* is a group of related functions that provide a service. All OLE Automation objects must implement the **IUnknown** interface, which manages all the other interfaces that the object supports. The **IDispatch** interface, which derives from **IUnknown**, consists of functions that allow access to the methods and properties of OLE Automation objects. A *custom interface* is a Component Object Model interface that is not defined as part of OLE; in short, any user-defined interface is a custom interface.

The *virtual function table* (VTBL) lists the addresses of all the properties and methods that are members of the object, including the member functions of the interfaces that it supports. The first three members of the VTBL are the members of the **IUnknown** interface; and subsequent entries are the members of the other supported interfaces. The following figure shows the VTBL for an object that supports the **IUnknown** and **IDispatch** interfaces:

{ewc msdncd, EWGraphic, group10453 3 /a "SDK_04.bmp"}

If the object does not support **IDispatch**, the entries for the members of the object's custom interfaces immediately follow the members of **IUnknown**. For example, the following figure shows the VTBL for an object that supports a custom interface named IMyInterface:

{ewc msdncd, EWGraphic, group10453 4 /a "SDK_05.bmp"}

When you expose an object for OLE Automation, you must decide whether to implement an **IDispatch** interface, a VTBL interface, or both. Microsoft strongly recommends that your objects provide a *dual interface*, which supports both access methods. In a dual interface, the first three entries in the VTBL are the members of **IUnknown**, the next four entries are the members of **IDispatch**, and the subsequent entries are the addresses of the members of the dual interface. The following figure shows the VTBL for an object that supports a dual interface named IMyInterface:

{ewc msdncd, EWGraphic, group10453 5 /a "SDK_06.bmp"}

In addition to providing access to objects, OLE Automation provides information about exposed objects. Using **IDispatch** or a type library, an OLE Automation controller or programming tool can determine the interfaces a object supports and the names of its members. Type libraries are especially useful for this purpose because they can be accessed at compile time. For more information, see What Is a Type Library?, and Type Libraries.

Accessing an Object Through the IDispatch Interface

OLE Automation controllers can use the **IDispatch** interface to access objects that implement this interface. The controller must first create the object, then query the object's **IUnknown** interface for a pointer to its **IDispatch** interface.

Although programmers know objects, methods, and properties by name, **IDispatch** keeps track of them internally by a number, the *Dispatch ID* (DISPID). Before an OLE Automation controller can access a property or method, it must have the DISPID that maps to the name of the member.

With the DISPID, it can call **IDispatch::Invoke** to access the property or invoke the method, packaging the parameters for the property or method into one of the **IDispatch::Invoke** parameters.

The object's implementation of **IDispatch::Invoke** must then unpackage the parameters, call the property or method, and be prepared to handle any errors that occur. When the property or method returns, the object passes its return value back to the controller through an **IDispatch::Invoke** parameter.

DISPIDs are available at run time and, in some circumstances, at compile time. At run time, controllers get DISPIDs by calling the **IDispatch::GetIDsOfNames** function. This is called *late binding* because the controller binds to the property or method at run time.

The DISPID of each property or method is fixed, and is part of the object's type description. If the object is described in a type library, an OLE Automation controller can read the DISPIDs from the type library at compile time, and thus avoid calling **IDispatch::GetIDsOfNames**. This is called *ID binding*. Because it requires only one call to IDispatch (that is, the call to **Invoke**) rather than the two calls required by late binding, it is generally about twice as fast. Late binding controllers can improve performance by caching DISPIDs after retrieving them, so that **IDispatch::GetIDsOfNames** is called only once for each property or method.

Accessing an Object Through the VTBL

OLE Automation allows an OLE Automation controller to call a method or property accessor function directly, either within or across processes. This approach, called *VTBL binding*, does not use the **IDispatch** interface. The controller gets type information from the type library at compile time, then calls the methods and functions directly. VTBL binding is faster than both ID binding and late binding because access is direct; no calls are made through **IDispatch**.

In-Process and Local Server Objects

OLE Automation objects may exist in the same process as their controller or in a different process. *In-process* objects are implemented in a DLL and run in the process space of the controller. Because they're contained in a DLL, they can't be run stand-alone. *Local server* objects are implemented in an .EXE file and run in a separate process space. Access to in-process objects is much faster than access to local server objects, because OLE Automation does not need to make remote procedure calls across the process boundary.

The BrowseH sample is an in-process OLE Automation object. The Hello and Lines samples are both local server OLE Automation objects.

The access mechanism (**IDispatch** or VTBL) and the location of an object (in-process or local server) determine the fixed overhead required for access. Note that the most important factor in performance, however, is the quantity and nature of the work performed by the methods and procedures being invoked. If the method is time-consuming or requires remote procedure calls, the overhead of one additional call to **IDispatch** may be negligible.

What Is a Type Library?

A *type library* is a file or part of a file that describes the type of one or more OLE Automation objects. Type libraries do not store objects; type libraries store type information. By accessing a type library, applications and browsers can determine the characteristics of an object, such as the interfaces the object supports and the names and addresses of each interface's members. You can also invoke a member through a type library; see Chapter 8, "Type Description Interfaces," for details.

When you expose OLE Automation objects, you should create a type library to make your objects easily accessible to other developers. The simplest way is to describe your objects in an Object Description Language (ODL) file, and then compile the file with the MkTypLib tool, as described in Chapter 7, "MkTypLib and Object Description Language."

Exposing OLE Automation Objects

Exposing objects makes them available for programmatic use by other applications and programming tools. To expose objects, you need to implement the objects using OLE Automation; create a type library that describes the exposed objects; and register the objects with the system.

This chapter discusses how you should design an application that exposes objects, then uses the Hello and Lines samples from the OLE 2 Software Development Kit (SDK) to demonstrate how to implement the design. Throughout the chapter, the filename for the example precedes the example in parentheses.

Design Considerations

When you expose objects to OLE Automation, you need to decide which interfaces to implement and how to organize your objects. In addition, you should create a type library. This section provides information to guide you in designing an OLE Automation application.

Dual Interfaces

Although OLE Automation allows you to implement an **IDispatch** interface, a VTBL interface, or a **dual** interface (which encompasses both), Microsoft strongly recommends that you implement dual interfaces for all exposed OLE Automation objects. Dual interfaces have significant advantages over **IDispatch**-only or VTBL-only interfaces:

- Binding can take place at compile time through the VTBL interface, or at run time through **IDispatch**.
- OLE Automation controllers that can use the VTBL interface may benefit from improved performance.
- Existing OLE Automation controllers that use the **IDispatch** interface will continue to work.
- The VTBL interface is easier to call from C++.
- Dual interfaces are required for compatibility with Visual Basic object support features.

Converting Existing Objects to Dual Interfaces

If you have already implemented exposed objects that support only the **IDispatch** interface, you should convert them to support **dual** interfaces.

{ewl msdncd, EWGraphic, group10454 0 /a "SDK.BMP"} To convert an IDispatch interface to a dual interface

1. Edit the Object Description Language (.ODL) file to declare a **dual** interface instead of an **IDispatch** interface.
2. Rearrange the parameter lists so that the methods and properties of your exposed objects return an HRESULT and pass their return values in a parameter.
3. If your object implements an exception handler, revise your code to use the OLE Automation error handling interface. This interface provides for detailed, contextual error information through both **IDispatch** and VTBL interfaces.

See "Setting Up the VTBL Interface," "Returning an Error," and "Creating a Type Library" later in this chapter for details.

Type Libraries

In addition to implementing dual interfaces, you should create a type library for each set of objects you expose. Because VTBL references are bound at compile time, exposed objects that support VTBL binding must be described in a type library.

Type libraries provide these important benefits:

- Type checking can be performed at compile time, helping developers of OLE Automation controllers to write fast, correct code to access your objects.
- You can use the **DispInvoke** function to implement **IDispatch** automatically. Using **DispInvoke** ensures that your implementation will be correct.
- Visual Basic applications can create objects with specific interface types, rather than the generic **Object** type, and take advantage of early binding.
- OLE Automation controllers that don't support VTBLs can read and cache DISPIDs at compile time, thereby improving run time performance.
- Type browsers can scan the library, allowing others to see the characteristics of your objects.
- The **RegisterTypeLib** function can be used to register your exposed objects.
- Local server access is improved, because OLE Automation uses information from the type library to package the parameters passed to an object in another process.

The Application Object

Document-based, user-interactive applications that expose OLE Automation objects should have one top-level object called the *Application object*. This object is initialized as the active object when the application starts.

The Application object identifies the application and provides a way for OLE Automation controllers to bind to and navigate the application's exposed objects. All other exposed objects are subordinate to the Application object; it is the root-level object in the object hierarchy.

The names of the Application object's members are part of the global name space, so OLE Automation controllers need not qualify them. For example, if MyApplication is the name of the Application object, a Visual Basic program can refer to a method of MyApplication as MyApplication.MyMethod or simply MyMethod. However, you should be careful not to overload the Application object;too many members can cause ambiguity and decrease performance. A large, complicated application with many members should be organized hierarchically, with a few generalized objects at the top, branching out into smaller, more specialized objects. Microsoft Excel is a good example of this kind of application.

Shutdown Behavior

OLE Automation objects must shut down in the following way:

- If the object's application is visible, the object should shut down only in response to an explicit user command (for example, the Exit command from the File menu) or from the equivalent command from an OLE Automation controller.
- If the object's application is not visible, the object should shut down only when the last external reference to it disappears.

Steps to Exposing Objects

To expose OLE Automation objects, you must write code to initialize the objects, implement the objects, and release OLE when your application terminates.

{ewl msdncd, EWGraphic, group10454 1 /a "SDK.BMP"} To initialize your exposed objects

1. Initialize OLE.
2. Register the class factories of your exposed objects.
3. Register the active object.

{ewl msdncd, EWGraphic, group10454 2 /a "SDK.BMP"} To implement your exposed objects

1. Implement the **IUnknown**, **IDispatch**, and VTBL interfaces for the objects.
2. Implement the properties and methods of the objects.

{ewl msdncd, EWGraphic, group10454 3 /a "SDK.BMP"} To release OLE when your application terminates

1. Revoke the registration of the class factories and revoke the active object.
2. Unintialize OLE.

{ewl msdncd, EWGraphic, group10454 4 /a "SDK.BMP"} To make your objects available for use by others

1. Create an object description (.ODL) file that describes the properties and methods of the exposed objects, and use MkTypLib to compile the .ODL file into a type library.
2. Create a registration (.REG) file for your application.

Initializing OLE to Expose Objects

To initialize OLE and your exposed objects, you use the following functions:

- **OleInitialize** – Initializes OLE.
- **CoRegisterClassObject** – Registers the object's class factory with OLE so that other applications may use it to create new objects.
- **RegisterActiveObject** – Registers the active object, allowing other applications to connect to an existing object.

Implementing Exposed Objects

The following figure shows the interfaces you should implement in order to expose OLE Automation objects:

{ewc msdncd, EWGraphic, group10454 5 /a "SDK_01.bmp"}

The member functions are listed under each interface name.

The *class factory* object implements the **IClassFactory** and **IUnknown** interfaces. (All objects must implement **IUnknown**, which allows OLE Automation controllers to determine what interfaces the object supports.) A class factory can create instances of a class.

The object implements two interfaces: **IUnknown** and IMyInterface. IMyInterface is a **dual** interface, which supports both late binding through **IDispatch** and early binding through the VTBL. The dual interface thus provides two ways to invoke the object's methods and properties. **IDispatch** includes the member functions **GetIDsOfNames**, **GetTypeInfo**, **GetTypeInfoCount**, and **Invoke**.

Member1 and Member2 are the members of IMyInterface. These members are available as direct entry points through the object's VTBL. They can also be accessed through **IDispatch::Invoke**.

You must also decide how to handle errors that occur in your exposed objects. If your object supports a dual interface and needs to return detailed, contextual error information, you also need to implement the OLE Automation error interface, **IErrorInfo**.

In addition to writing code to implement objects, you need to create a type library and a registration file. You describe the types of the exposed objects in the object description language (ODL), and use the MkTypLib tool to compile the ODL file into a type library (.TLB) and a header file (.H). The registration file provides information that the operating system and OLE need to locate objects.

Releasing OLE to Expose Objects

To release OLE and your exposed objects, you use the following functions:

- **RevokeActiveObject**–Ends an object's status as the active object.
- **CoRevokeClassObject**–Informs OLE that a class factory is no longer available for use by other applications.
- **OLEUninitialize**–Releases OLE.

"Hello World" Example

The Hello example in the OLE 2 SDK is a simple OLE Automation application with one object. It has these characteristics:

- Supports VTBL binding
- Permits multiple instances of its exposed object to exist at the same time
- Implements **IErrorInfo** for exception handling

The sections that follow demonstrate how the Hello sample exposes a simple class. The code is abridged to illustrate the essential parts; see the source code in the OLE 2 SDK for a complete listing.

Initializing OLE for the "Hello World" Example

When the Hello application starts, it initializes OLE and the creates the object to be exposed through OLE Automation. For example (MAIN.CPP):

```
BOOL InitInstance (HINSTANCE hinst)
{
     HRESULT hr;
     char ach[STR_LEN];

     // Intialize OLE
     hr = OleInitialize(NULL);
     if (FAILED(hr))
           return FALSE;

     // Create an instance of the Hello Application object. The object is
     // created with refcount 0.
     LoadString(hinst, IDS_HelloMessage, ach, sizeof(ach));
     hr = CHello::Create(hinst, ach, &g_phello);
     if (FAILED(hr))
           return FALSE;
     return TRUE;
}
```

This function calls **OleInitialize** to initialize OLE. It loads the string ach with the initial hello message, obtained from the string table through the constant IDS_HelloMessage. Then it calls CHello::Create to create a single, global instance of the application object, passing it the initial hello message and receiving a value for g_phello, a pointer to the instance. If the function is successful, it returns True.

Registering the Active Object

After Hello creates an instance of the object, it exposes and registers the class factory (if necessary) and registers the active object (MAIN.CPP):

```
BOOL ProcessCmdLine(LPSTR pCmdLine,
                             LPDWORD pdwRegisterCF,
                             LPDWORD pdwRegisterActiveObject,
                             int nCmdShow)
{
     LPCLASSFACTORY pcf = NULL;
     HRESULT hr;

     *pdwRegisterCF = 0;
     *pdwRegisterActiveObject = 0;

     // Expose class factory for application object if command line
     // contains the /Automation switch.
     if (_fstrstr(pCmdLine, "-Automation") != NULL
           || _fstrstr(pCmdLine, "/Automation") != NULL)
     {
           pcf = new CHelloCF;
           if (pcf)
                 goto error;
           pcf->AddRef();
           hr = CoRegisterClassObject(CLSID_Hello, pcf,
                                                    CLSCTX_LOCAL_SERVER,
                                                    REGCLS_SINGLEUSE,
                                                    pdwRegisterCF);
           if (hr != NOERROR)
                 goto error;
           pcf->Release();
     }
     else g_phello->ShowWindow(nCmdShow); //Show if started stand-alone.

RegisterActiveObject(g_phello, CLSID_Hello, ACTIVEOBJECT_WEAK,
                             pdwRegisterActiveObject);
     return TRUE;

error:
     if (!pcf)
           pcf->Release();
     return FALSE;
}
```

The sample first checks the command line for the **/Automation** switch. This switch indicates that the application should be started for programmatic access, so that OLE Automation controllers can create additional instances of the application's class. In this case, the class factory must be created and registered. If the switch is present, Hello creates a new CHelloCF object and calls its AddRef method, thereby creating the class factory.

Next, the sample calls **CoRegisterClassObject** to register the class factory. It passes the object's class ID (CLSID_Hello), a pointer to the CHelloCF object (pcf), and two constants (CLSCTX_LOCAL_SERVER and REGCLS_SINGLEUSE) that govern the class factory's use. CLSCTX_LOCAL_SERVER indicates that the executable code for the object runs in a separate process from the controller. REGCLS_SINGLEUSE allows only one OLE Automation controller to use each instance of the class factory. The value returned through pdwRegisterCF must later be used to revoke the class factory.

Finally, the sample registers the active object. Registering an active object lets OLE Automation controllers retrieve an object that is already running, rather than create a new instance of the object. The Application object must be registered.

Implementing IUnknown

Every OLE object must implement the **IUnknown** interface, which allows controllers to query the object to find out what interfaces it supports. **IUnknown** has three member functions: **QueryInterface**, **AddRef**, and **Release**. The sample implements them for the CHello object as follows (HELLO.CPP):

```
STDMETHODIMP
CHello::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
     *ppv = NULL;
     if (iid == IID_IUnknown || iid == IID_IDispatch || iid == IID_IHello
           *ppv = this;
     else if (iid == IID_ISupportErrorInfo)
           *ppv = &m_SupportErrorInfo;
     else return ResultFromScode(E_NOINTERFACE);

     AddRef();
     return NOERROR;
}

STDMETHODIMP_(ULONG)
CHello::AddRef(void)
{
     return ++m_cRef;
}

STDMETHODIMP_(ULONG)
CHello::Release(void)
{
if (--m_cRef == 0)
     {
           delete this;
           return 0;
     }
     return m_cRef;
}
```

Implementing IClassFactory

A class factory is a class that is capable of creating instances of another class. Hello implements a single class factory, called CHelloCF, as follows (HELLOCF.CPP):

```
CHelloCF::CHelloCF(void)
{
     m_cRef = 0;
}

STDMETHODIMP
CHelloCF::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
     *ppv = NULL;
     if (iid == IID_IUnknown || iid == IID_IClassFactory)
          *ppv = this;
     else
          return ResultFromScode(E_NOINTERFACE);
     AddRef();
     return NOERROR;
}

STDMETHODIMP_(ULONG)
CHelloCF::AddRef(void)
{
     return ++m_cRef;
}

STDMETHODIMP_(ULONG)
CHelloCF::Release(void)
{
     if (--m_cRef == 0)
     {
          delete this;
          return 0;
     }
     return m_cRef;
}

STDMETHODIMP
CHelloCF::CreateInstance(IUnknown FAR* punkOuter,
                                   REFIID riid,
                                   void FAR* FAR* ppv)
{
     HRESULT hr;

     *ppv = NULL;

     // This implementation doesn't allow aggregation.
     if (punkOuter)
          return ResultFromScode(CLASS_E_NOAGGREGATION);

     hr = g_phello->QueryInterface(riid, ppv);
     if (FAILED(hr))
     {
            g_phello->Quit();
            return hr;
      }
      return NOERROR;
}
STDMETHODIMP
CHelloCF::LockServer(BOOL fLock)
{
      CoLockObjectExternal(g_phello, fLock, TRUE);
      return NOERROR;
}
```

The class factory supports six member functions. **QueryInterface**, **AddRef**, and **Release** are the required **IUnknown** members, and **CreateInstance** and **LockServer** are the required **IClassFactory** members. The function CHelloCF::CHelloCF is a C++ constructor function. By default, the constructor function initializes the object's VTBLs; CHelloCF::CHelloCF also initializes the reference count for the class.

Implementing IDispatch

The **IDispatch** interface provides access to and information about an object. The interface requires the member functions **GetTypeInfoCount**, **GetTypeInfo**, **GetIdsOfNames**, and **Invoke**. The Hello sample implements **IDispatch** as follows (HELLO.CPP):

```
STDMETHODIMP
CHello::GetTypeInfoCount(UINT FAR* pctinfo)
{
     *pctinfo = 1;
     return NOERROR;
}

STDMETHODIMP
CHello::GetTypeInfo(
           UINT itinfo,
           LCID lcid,
           ITypeInfo FAR* FAR* pptinfo)
{
     *pptinfo = NULL;

     if(itinfo != 0)
           return ResultFromScode(DISP_E_BADINDEX);

     m_ptinfo->AddRef();
     *pptinfo = m_ptinfo;

     return NOERROR;
}

STDMETHODIMP
CHello::GetIDsOfNames(
           REFIID riid,
           OLECHAR FAR* FAR* rgszNames,
           UINT cNames,
           LCID lcid,
           DISPID FAR* rgdispid)
{
     return DispGetIDsOfNames(m_ptinfo, rgszNames, cNames, rgdispid);
}

STDMETHODIMP
CHello::Invoke(
           DISPID dispidMember,
           REFIID riid,
           LCID lcid,
           WORD wFlags,
           DISPPARAMS FAR* pdispparams,
           VARIANT FAR* pvarResult,
           EXCEPINFO FAR* pexcepinfo,
           UINT FAR* puArgErr)
{
     HRESULT hr;

     m_bRaiseException = FALSE;
     hr =  DispInvoke(
           this, m_ptinfo,
           dispidMember, wFlags, pdispparams,
           pvarResult, pexcepinfo, puArgErr);
     if (m_bRaiseException)
     {
           if (NULL != pexcepinfo)
                 _fmemcpy(pexcepinfo, &m_excepinfo, sizeof(EXCEPINFO));
           return ResultFromScode(DISP_E_EXCEPTION);
     }
     else return hr;
}
```

OLE Automation includes two functions, **DispGetIdsOfNames** and **DispInvoke**, which provide standard implementations for **IDispatch::GetIdsOfNames** and **IDispatch::Invoke**. The Hello sample uses these two functions to simplify the code.

Implementing VTBL Binding

In addition to the **IDispatch** interface, the Hello sample supports VTBL binding. Objects that support a VTBL interface return an HRESULT instead of a value when a member is invoked, and pass their return value as the last parameter. They may also accept a locale identifier (LCID) parameter, which allows them to parse strings correctly for the local language. The following example shows how the Visible property is implemented (HELLO.CPP):

```
STDMETHODIMP
CHello::put_Visible(BOOL bVisible)
{
     ShowWindow(bVisible ? SW_SHOW : SW_HIDE);
     return NOERROR;
}

STDMETHODIMP
CHello::get_Visible(BOOL FAR* pbool)
{
     *pbool =  m_bVisible;
     return NOERROR;
}
```

Additional information must be specified in ODL to create a dual interface, as shown in "Creating the Type Information."

Handling Errors

Hello includes an exception handler, which passes exceptions through **IDispatch::Invoke** and supports rich error information through VTBLs (HELLO.CPP):

```
STDMETHODIMP
CHello::RaiseException(int nID)
{
     extern SCODE g_scodes[];
     char szError[STR_LEN];
     ICreateErrorInfo *pcerrinfo;
     IErrorInfo *perrinfo;
     HRESULT hr;

     _fmemset(&m_excepinfo, 0, sizeof(EXCEPINFO));

     m_excepinfo.wCode = nID;
     if (LoadString(m_hinst, nID, szError, sizeof(szError)))
           m_excepinfo.bstrDescription = SysAllocString(szError);
     m_excepinfo.bstrSource = SysAllocString(m_bstrName);

     m_bRaiseException = TRUE;

     // Set ErrInfo object so that VTBL binding container
     // applications can get rich error information.
     hr = CreateErrorInfo(&pcerrinfo);
     if (SUCCEEDED(hr))
     {
           pcerrinfo->SetGUID(IID_IHello);
           if (m_excepinfo.bstrSource)
                 pcerrinfo->SetSource(m_excepinfo.bstrSource);
           if (m_excepinfo.bstrDescription)
                 pcerrinfo->SetDescription(m_excepinfo.bstrDescription);
           hr = pcerrinfo->QueryInterface(IID_IErrorInfo,
           (LPVOID FAR*) &perrinfo);
           if (SUCCEEDED(hr))
           {
                 SetErrorInfo(0, perrinfo);
                 perrinfo->Release();
           }
           pcerrinfo->Release();
     }
     return ResultFromScode(g_scodes[nID-1001]);
}
```

Hello's member functions call this routine when an exception occurs. RaiseException fills in an EXCEPINFO structure that is available through **IDispatch::Invoke**. It also sets the system's error object, so that container applications that call through VTBLs can retrieve rich error information. Note that because **IDispatch::Invoke** is effectively such a container, it retrieves the rich error information and returns it through the EXCEPINFO parameter.

Hello also implements the **ISupportErrorInfo** interface, which allows OLE Automation controllers to query whether an error object will be available (HELLO.CPP):

```
CSupportErrorInfo::CSupportErrorInfo(IUnknown FAR* punkObject,
                                                     REFIID riid)
```

```
{
     m_punkObject = punkObject;
     m_iid = riid;
}

STDMETHODIMP
CSupportErrorInfo::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
     return m_punkObject->QueryInterface(iid, ppv);
}

STDMETHODIMP_(ULONG)
CSupportErrorInfo::AddRef(void)
{
     return m_punkObject->AddRef();
}

STDMETHODIMP_(ULONG)
CSupportErrorInfo::Release(void)
{
     return m_punkObject->Release();
}

STDMETHODIMP
CSupportErrorInfo::InterfaceSupportsErrorInfo(REFIID riid)
{
     return (riid == m_iid) ? NOERROR : ResultFromScode(S_FALSE);
}
```

Releasing Objects and OLE

When the Hello application ends, it revokes the class factory and the active object, and uninitializes OLE. For example (MAIN.CPP):

```
void Uninitialize(DWORD dwRegisterCF, DWORD dwRegisterActiveObject)
{
     if (dwRegisterCF != 0)
           CoRevokeClassObject(dwRegisterCF);
     if (dwRegisterActiveObject != 0)
           RevokeActiveObject(dwRegisterActiveObject, NULL);
     OleUninitialize();
}
```

Creating the Type Information

Type information for the Hello sample is described in ODL. MkTypLib compiles the ODL file to create a type library (HELLOTL.TLB) and a header file (HELLOTL.H). The following example shows the description for the Hello type library, interface, and Application object (HELLO.ODL):

```
[
     uuid(F37C8060-4AD5-101B-B826-00DD01103DE1),      // LIBID_Hello
     helpstring("Hello 2.0 Type Library"),
     lcid(0x0409),
     version(2.0)
]
library Hello
{
    importlib("stdole.tlb");
    [
      uuid(F37C8062-4AD5-101B-B826-00DD01103DE1),   // IID_IHello
      helpstring("Application object for the Hello application."),
      oleautomation,
      dual
    ]
    interface IHello : IDispatch
    {
           [propget, helpstring("Returns the application of the object.")]
           HRESULT Application([out, retval] IHello** retval);

     [propget,
           helpstring("Returns the full name of the application.")]
           HRESULT FullName([out, retval] BSTR* retval);

           [propget, id(0),
           helpstring("Returns the name of the application.")]
           HRESULT Name([out, retval] BSTR* retval);

        [propget, helpstring("Returns the parent of the object.")]
        HRESULT Parent([out, retval] IHello** retval);

        [propput]
        HRESULT Visible([in] boolean VisibleFlag);
        [propget,
           helpstring
           ("Sets or returns whether the main window is visible.")]
        HRESULT Visible([out, retval] boolean* retval);

        [helpstring("Exits the application.")]
        HRESULT Quit();

        [propput,
           helpstring("Sets or returns the hello message to be used.")]
        HRESULT HelloMessage([in] BSTR Message);
        [propget]
        HRESULT HelloMessage([out, retval] BSTR *retval);

        [helpstring("Say Hello using HelloMessage.")]
        HRESULT SayHello();
    }


    [
       uuid(F37C8061-4AD5-101B-B826-00DD01103DE1),      // CLSID_Hello
       helpstring("Hello Class"),
       appobject
    ]
    coclass Hello
    {
        [default]          interface IHello;
                           interface IDispatch;
    }
}
```

The items enclosed by square brackets are *attributes*, which provide further information about the objects in the file. The **oleautomation** and **dual** attributes, for example, indicate that the IHello interface supports both **IDispatch** and VTBL binding. The **appobject** attribute indicates that the Hello is the Application object.

Creating a Registration File for the "Hello World" Example

The system registration database lists all the OLE objects in the system. OLE uses this database to locate objects and to determine their capabilities. The registration file registers the application, the type library, and the exposed classes of the sample (HELLO.REG):

```
REGEDIT
; Registration information for OLE Automation Hello 2.0 Application

; Version independent registration. Points to Version 2.0
HKEY_CLASSES_ROOT\Hello.Application = Hello 2.0 Application
HKEY_CLASSES_ROOT\Hello.Application\Clsid = {F37C8061-4AD5-101B-B826-
00DD01103DE1}

; Version 2.0 registration
HKEY_CLASSES_ROOT\Hello.Application.2 = Hello 2.0 Application
HKEY_CLASSES_ROOT\Hello.Application.2\Clsid = {F37C8061-4AD5-101B-B826-
00DD01103DE1}
HKEY_CLASSES_ROOT\CLSID\{F37C8061-4AD5-101B-B826-00DD01103DE1} = Hello 2.0
Application
HKEY_CLASSES_ROOT\CLSID\{F37C8061-4AD5-101B-B826-00DD01103DE1}\ProgID =
Hello.Application.2
HKEY_CLASSES_ROOT\CLSID\{F37C8061-4AD5-101B-B826-
00DD01103DE1}\VersionIndependentProgID = Hello.Application
HKEY_CLASSES_ROOT\CLSID\{F37C8061-4AD5-101B-B826-00DD01103DE1}\LocalServer =
hello.exe /Automation

; Type library registration information
HKEY_CLASSES_ROOT\TypeLib\{F37C8060-4AD5-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\TypeLib\{F37C8060-4AD5-101B-B826-00DD01103DE1}\2.0 = Hello
2.0 Type Library
HKEY_CLASSES_ROOT\TypeLib\{F37C8060-4AD5-101B-B826-00DD01103DE1}\2.0\HELPDIR
=
;US english
HKEY_CLASSES_ROOT\TypeLib\{F37C8060-4AD5-101B-B826-
00DD01103DE1}\2.0\409\win16 = hello.tlb

; Interface registration. All interfaces that support vtable binding must be
; registered as follows. RegisterTypeLib & LoadTypeLib will do this
automatically.

; IID_IHello = {F37C8062-4AD5-101B-B826-00DD01103DE1}
; LIBID_Hello = {F37C8060-4AD5-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\Interface\{F37C8062-4AD5-101B-B826-00DD01103DE1} = IHello
HKEY_CLASSES_ROOT\Interface\{F37C8060-4AD5-101B-B826-00DD01103DE1}\TypeLib =
{F37C8060-4AD5-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\Interface\{F37C8060-4AD5-101B-B826-
00DD01103DE1}\ProxyStubClsid = {00020424-0000-0000-C000-000000000046}
```

To merge your object's registration information with the system registry, your object should expose the **DLLRegisterServer** API, as described in the *OLE Controls Reference*. **DLLRegisterServer** should call **RegisterTypeLib** to register the type library and the interfaces supported by your application.

Limitations of the "Hello World" Example

The "Hello World" example was simplified for demonstration purposes. It has the following limitations:

- Has only one object.
- Uses only scalar argument types. OLE Automation also supports methods and properties that accept arguments of complex types, including arrays, references to objects, and formatted data, but not structures.
- Supports one national language. **DispInvoke** does not accommodate multiple localized member names, and therefore is useful only for single-language applications.

The remainder of this chapter explains in more detail the OLE Automation features that appear in Hello, as well as covering these more advanced features. Filenames in the rest of the chapter refer to the Lines sample.

Creating Class IDs

Each object you expose for creation must have a unique class identifier (CLSID). CLSIDs are universally unique identifiers (UUIDs, also called globally unique identifiers, or GUIDs) that identify class objects to OLE. The CLSID is included in your application and must be registered with the operating system when your application is installed.

Run the GUIDGEN.EXE utility included in the \OLE2\BIN directory to generate UUIDs. By default, GUIDGEN puts a DEFINE_GUID macro on the clipboard which you can paste into your source.

In the following example, the macro defines a class ID for Lines (TLB.H):

```
DEFINE_GUID(CLSID_Lines,0x3C591B21,0x1F13,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x
10,0x3D,0xE1);
```

When **MkTypLib** creates the optional header file (TLB.H), it inserts DEFINE_GUID macros for each library, interface, and class in your application.

Initializing OLE

OLE2.DLL provides functions to initialize OLE and to initialize exposed objects for creation using OLE Automation.

{ewl msdncd, EWGraphic, group10454 6 /a "SDK.BMP"} To initialize OLE

1. Initialize the OLE DLLs by calling **OleInitialize**.
2. Create an instance of each class factory you expose. You need a class factory for the Application object and any other top-level objects.
3. Call **CoRegisterClassObject** for each class your application exposes.

The following routine initializes OLE, then creates an instance of the Lines Application object (Lines sample, MAIN.CPP):

```
BOOL InitInstance (HINSTANCE hinst)
{
     HRESULT hr;

     // Intialize OLE
     hr = OleInitialize(NULL);
     if (FAILED(hr))
           return FALSE;

     // Create an instance of the Lines Application object. The object is
     // created with refcount 0.
     hr = CApplication::Create(hinst, &g_pApplication);
     if (FAILED(hr))
           return FALSE;
     return TRUE;
}
```

Initializing the Active Object

The following function creates and registers the application's class factory, then registers the Lines Application object as the active object (MAIN.CPP):

```
BOOL ProcessCmdLine(LPSTR lpCmdLine, LPDWORD pdwRegisterCF,
                             LPDWORD pdwRegisterActiveObject, int nCmdShow)
{
     LPCLASSFACTORY pcf = NULL;
     HRESULT hr;

     *pdwRegisterCF = 0;
     *pdwRegisterActiveObject = 0;

     // Expose class factory for application object if command line
     // contains the /Automation switch.
     if (_fstrstr(lpCmdLine, "-Automation") != NULL
           || _fstrstr(lpCmdLine, "/Automation") != NULL)
     {
           pcf = new CApplicationCF;
           if (pcf)
                 goto error;
           pcf->AddRef();
           hr = CoRegisterClassObject(CLSID_Lines, pcf,
                                               CLSCTX_LOCAL_SERVER,
REGCLS_SINGLEUSE,
                                               pdwRegisterCF);
           if (hr != NOERROR)
                 goto error;
           pcf->Release();
     }
     else                    // Show window if started stand-alone
     g_pApplication->ShowWindow(nCmdShow );

// Register Lines application object in the Running Object Table (ROT) //
Use weak registration so that the ROT releases its reference when
// all external references are released.
     RegisterActiveObject(g_pApplication, CLSID_Lines, ACTIVEOBJECT_WEAK,
                 pdwRegisterActiveObject);
     return TRUE;

error:
     if (!pcf)
           pcf->Release();
     return FALSE;
}
```

OLE Automation provides several functions to identify and retrieve the running instance of an object or application:

- **RegisterActiveObject** – Sets the active object for an application. (Use when application starts.)
- **RevokeActiveObject** – Revokes the active object. (Use when application ends.)
- **GetActiveObject** – Retrieves a pointer to the object that is active. (In Visual Basic, this is implemented by the **GetObject** function.)

Applications can have more than one active object at a time. To be initialized as an active object, an object must:

- Have a class factory (that is, the object provides an interface for creating instances of itself).
- Identify its class factory by a programmatic ID (ProgID) in the system registry.
- Be registered by a call to **RegisterActiveObject** when the object is created or becomes active.

The Application object must be registered as an active object.

The example exposes the class factory for the Lines application, CApplicationCF, if the command line contains the **/Automation** switch. OLE applies this switch if it appears on the command line or in the application's registration entry. The switch indicates that the application was started for programmatic access, and therefore OLE needs to register the class factory and create an instance of the Application object. OLE also supports the **/Embedding** switch, which indicates that an application was started by a container application.

You should register the Application object's class factory only if the application is launched with the **/Automation** switch. When **/Automation** is not specified, the application was started for some reason other than programmatic access through OLE Automation. If you register the class factory under these circumstances, and the user later requests a new instance of the Application object, OLE Automation will return the existing instance instead of creating a new instance.

The sample calls **CoRegisterClassObject** to register the class factory as the active object. The CLSCTX_LOCAL_SERVER flag means that the code that creates and manages Application objects will run in a separate process space on this machine.

Because the Application object's class factory is exposed, the call specifies the REGCLS_SINGLEUSE flag. When a multiple-document interface (MDI) application starts, it typically registers the class factory for its Document object, specifying REGCLS_MULTIPLEUSE. This attribute allows the existing application instance to be used later, when instances of the document objects need to be created. Each new Application object, however, requires a new instance of the application to be launched, and therefore should specify REGCLS_SINGLEUSE. If the application registered its class factory using REGCLS_MULTIPLEUSE, the next **CreateObject** call that tried to create the application would get an existing copy.

The following chart shows how applications should expose their Application and Document objects.

Command line	MDI application	SDI application
/Embedding	Expose class factories for document classes, but not for the application. Call **RegisterActiveObject** for the Application object.	Expose class factories for document class, but not for the application. Call **RegisterActiveObject** for the Application object.
/Automation	Expose class factories for document classes. Expose class factory for the application using **RegisterClassObject**. Call **RegisterActiveObject** for the Application object.	Do not expose class factory for document class. Expose class factory for the Application object using **RegisterClassObject**. Call **RegisterActiveObject** for the Application object.
No OLE	Expose class factories	Call

switches	for document classes, but not for the application. Call **RegisterActiveObject** for the Application object.	**RegisterActiveObject** for the Application object.

The call to **RegisterActiveObject** enters the Application object in OLE's running object table, so that OLE Automation controllers can retrieve the running object instead of creating a new instance. Thus, Visual Basic applications can use the **GetObject** statement to access an existing object.

The example specifies weak registration (ACTIVEOBJECT_WEAK), which means that OLE will release the object when all external connections to it have disappeared. OLE Automation objects should always be given weak registration. See **RegisterActiveObject** in Chapter 5 for more information.

The *OLE 2 Programmer's Reference, Volume 1* provides more information on the **OleInitialize** and **CoRegisterClassObject** functions. *Inside OLE 2* provides more information on verifying your application's entries in the registration database.

Creating the IUnknown Interface

IUnknown defines three member functions you must implement for each object you expose. The prototypes for these functions reside in OLE2.H:

- **QueryInterface** – Identifies which OLE interfaces the object supports.
- **AddRef** – Increments a member variable that tracks the number of references to the object.
- **Release** – Decrements the member variable that tracks the instances of the object. If an object has zero references, **Release** frees the object.

These functions provide the fundmental interface through which OLE can access your objects. The *OLE 2 Programmer's Reference, Volume 1*, describes in detail how to implement the functions.

The **IUnknown** interface for the Line object looks like this (Lines sample, LINE.CPP):

```
STDMETHODIMP
CLine::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
      *ppv = NULL;

      if (iid == IID_IUnknown || iid == IID_IDispatch || iid == IID_ILine)
            *ppv = this;
      else if (iid == IID_ISupportErrorInfo)
            *ppv = &m_SupportErrorInfo;
      else return ResultFromScode(E_NOINTERFACE);

      AddRef();
      return NOERROR;
}

STDMETHODIMP_(ULONG)
CLine::AddRef(void)
{
      return ++m_cRef;
}

STDMETHODIMP_(ULONG)
CLine::Release(void)
{
      if(--m_cRef == 0)
      {
            delete this;
            return 0;
      }
      return m_cRef;
}
```

The Line object implements the IID_ILine dual interface for VTBL binding. It also implements the IID_ISupportErrorInfo interface, so that it can return rich, contextual error information through VTBLs.

Exposing Objects for Creation with IClassFactory

Before OLE can create an object, it needs access to the object's class factory. The class factory implements the **IClassFactory** interface. For detailed information about this interface, see *Inside OLE* and the *OLE 2 Programmer's Reference, Volume1*; this chapter only describes what you must do to expose objects for OLE Automation.

You need to implement a class factory for objects that may be created explicitly through the OLE function **CoCreateInstance** or the Visual Basic **New** operator. For example, an application exposes an Application object for creation, but may have many other programmable objects that can be created or destroyed by referencing a member of the Application object. In this case, only the Application object needs a class factory.

For each class factory, you need to implement the following two member functions, which provide services for OLE API functions. The prototypes for the member functions reside in OLE2.H:

- **CreateInstance** – Creates an instance of the object's class.
- **LockServer** – Prevents the object's server from shutting down, even if the last instance of the object is released. **LockServer** can improve the performance of applications that create and release objects frequently.

In general, the **CreateInstance** method should create a new instance of the object's class. For the Application object, however, the **CreateInstance** method should return the existing instance of the Application object, which is registered in the running object table.

The Lines sample implements a class factory for its Application object, as follows (APPCF.CPP):

```
STDMETHODIMP
CApplicationCF::CreateInstance(IUnknown FAR* punkOuter,
                                                REFIID riid,
                                                void FAR* FAR* ppv)
{
     HRESULT hr;

     *ppv = NULL;

     // This implementation doesn't allow aggregation.
     if (punkOuter)
           return ResultFromScode(CLASS_E_NOAGGREGATION);

     // This is REGCLS_SINGLEUSE class factory, so CreateInstance will be
     // called at most once. An application object has a REGCLS_SINGLEUSE
     // class factory. The global application object has already been
     // created when CreateInstance is called. A REGCLS_MULTIPLEUSE class
     // factory's CreateInstance would be called multiple times and would
     // create a new object each time. An MDI application would have a
     // REGCLS_MULTIPLEUSE class factory for its document objects.

     hr = g_pApplication->QueryInterface(riid, ppv);
     if (FAILED(hr))
     {
           g_pApplication->Quit();
           return hr;
     }
     return NOERROR;
}
```

```
STDMETHODIMP
CApplicationCF::LockServer(BOOL fLock)
{
     CoLockObjectExternal(g_pApplication, fLock, TRUE);
     return NOERROR;
}
```

The object's class factory must also implement an **IUnknown** interface. For example (APPCF.CPP):

```
STDMETHODIMP
CApplicationCF::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
     *ppv = NULL;

     if (iid == IID_IUnknown || iid == IID_IClassFactory)
           *ppv = this;
     else
           return ResultFromScode(E_NOINTERFACE);
     AddRef();
     return NOERROR;
}

STDMETHODIMP_(ULONG)
CApplicationCF::AddRef(void)
{
     return ++m_cRef;
}

STDMETHODIMP_(ULONG)
CApplicationCF::Release(void)
{
     if(--m_cRef == 0)
     {
           delete this;
           return 0;
     }
     return m_cRef;
}
```

Creating the IDispatch Interface

IDispatch provides a late-bound mechanism to access and retrieve information about an object's methods and properties. In addition to the member functions inherited from **IUnknown**, you need to implement the following member functions within the class definition of each object you expose through OLE Automation:

- **GetTypeInfoCount** – Returns the number of type descriptions for the object. For objects that support **IDispatch**, the type information count is always 1.
- **GetTypeInfo** – Retrieves a description of the object's programmable interface.
- **GetIDsOfNames** – Maps the name of a method or property to a Dispatch ID, which can later be used to invoke the method or property.
- **Invoke** –Calls one of the object's methods, or gets or sets one of its properties.

You may implement **IDispatch** by any of the following means:

- Calling the **CreateStdDispatch** function. This approach is the simplest, but it does not provide for rich error handling or multiple national languages.
- Delegating to the **DispInvoke** and **DispGetIDsOfNames** functions, or to **ITypeInfo::Invoke** and **ITypeInfo::GetIDsOfNames**. This is the recommended approach, because it supports multiple locales and allows you to return exceptions.
- Implementing the member functions without delegating to the dispatch functions. This approach is seldom necessary. Because **Invoke** is a complex interface with many subtle semantics that are difficult to emulate, Microsoft strongly recommends that your code delegate to **ITypeInfo::Invoke** to implement this mechanism.

The following sections explain how to use CreateStdDispatch and DispInvoke to implement IDispatch.

Implementing IDispatch by Calling CreateStdDispatch

The simplest way to implement the **IDispatch** interface is to call **CreateStdDispatch**. This approach works for OLE Automation objects that return only the standard dispatch exception codes, support a single national language, and do not support dual interfaces.

CreateStdDispatch returns a pointer to the created **IDispatch** interface. It takes three pointers as input: a pointer to the object's **IUnknown** interface; a pointer to the object to expose; and a pointer to the type information for the object. The following example implements IDispatch for an object named CCalc by calling **CreateStdDispatch** on the loaded type information:

```
CCalc FAR*
CCalc::Create()
{
     HRESULT hresult;
     CCalc FAR* pcalc;
     ITypeLib FAR* ptlib;
     ITypeInfo FAR* ptinfo;
     IUnknown FAR* punkStdDisp;

     ptlib = NULL;
     ptinfo = NULL;

     // Some error handling code omitted...
     if ((pcalc = new FAR CCalc()) == NULL)
           return NULL;
     pcalc->AddRef();

     // Load the type library from the information in the registry.
     if ((hresult = LoadRegTypeLib(LIBID_DspCalc2, 1, 0, 0x0409, &ptlib))
           !=    NOERROR){
           goto LError0;
     }
     if ((hresult = ptlib->GetTypeInfoOfGuid(IID_ICalculator, &ptinfo))
           != NOERROR){
           goto LError0;
     }

     // Create an aggregate with an instance of the default
     // implementation of IDispatch that is initialized with our
     // TypeInfo.
     //
     hresult = CreateStdDispatch(
                 pcalc,                                     // controlling
unknown
                 &(pcalc->m_arith),                  // VTBL pointer to
dispatch on
                 ptinfo,
                 &punkStdDisp);
```

Implementing IDispatch by Delegating

Another way to implement **IDispatch** is to use the dispatch functions **DispInvoke** and **DispGetIDsOfNames**. These functions give you the option of supporting multiple national languages and creating application-specific exceptions that are passed back to OLE Automation controllers.

The Lines sample implements **IDispatch::GetIDsOfNames** and **IDispatch::Invoke** using these functions (LINES.CPP):

```
STDMETHODIMP
CLines::GetIDsOfNames(
           REFIID riid,
           char FAR* FAR* rgszNames,
           UINT cNames,
           LCID lcid,
           DISPID FAR* rgdispid)
{
     return DispGetIDsOfNames(m_ptinfo, rgszNames, cNames, rgdispid);
}

STDMETHODIMP
CLines::Invoke(
           DISPID dispidMember,
           REFIID riid,
           LCID lcid,
           WORD wFlags,
           DISPPARAMS FAR* pdispparams,
           VARIANT FAR* pvarResult,
           EXCEPINFO FAR* pexcepinfo,
           UINT FAR* puArgErr)
{

     HRESULT hr;

     m_bRaiseException = FALSE;
     hr =  DispInvoke(
           this, m_ptinfo,
           dispidMember, wFlags, pdispparams,
           pvarResult, pexcepinfo, puArgErr);
     if (m_bRaiseException)
     {
           if (NULL != pexcepinfo)
                 _fmemcpy(pexcepinfo, &m_excepinfo, sizeof(EXCEPINFO));
           return ResultFromScode(DISP_E_EXCEPTION);
     }
     else return hr;
}
```

In the preceding example, CLines::Invoke handles exceptions by initializing the variable m_bRaiseException to False before calling **DispInvoke**. If an exception occurs, the invoked member places error information in the variable m_excepinfo, and sets m_bRaiseException to True. Then CLines::Invoke copies m_excepinfo and returns DISP_E_EXCEPTION.

Setting Up the VTBL Interface

The Lines sample supports VTBL binding as well as the **IDispatch** interface. By supporting this **dual** interface, the sample allows OLE Automation controllers both the flexibility of the **IDispatch** interface and the speed of VTBLs. Controllers that know the names of the members can compile directly against the function pointers in the VTBL; controllers that don't have this information can use **IDispatch** at run time.

To have a **dual** interface, an interface must:

- Declare all its members to return an HRESULT and pass their actual return values as the last parameter.
- Have only OLE Automation-compatible parameters and return types, as described in Chapter 7, "MkTypLib and Object Description Language."
- Specify the **dual** attribute on the interface description in the ODL file.
- Initialize the VTBLs with the appropriate member function pointers.

The IPoint, IPoints, ILine, ILines, IPane, and IApplication interfaces in the Lines sample are all **dual** interfaces. The IPoint interface defines functions that get and put the values of the X and Y properties, as follows (POINT.CPP):

```
STDMETHODIMP
CPoint::get_x(int FAR* pnX)
{
     *pnX = m_nX;
     return NOERROR;
}

STDMETHODIMP
CPoint::put_x(int nX)
{
     m_nX = nX;
     return NOERROR;
}

STDMETHODIMP
CPoint::get_y(int FAR* pnY)
{
     *pnY = m_nY;
     return NOERROR;
}

STDMETHODIMP
CPoint::put_y(int nY)
{
     m_nY = nY;
     return NOERROR;
}
```

The get_x and get_y accessor functions pass their return values in the last parameter, pnX and pnY, and return an HRESULT as the function value.

In the ODL file, the interface is described as follows (LINES.ODL):

```
[
           uuid(3C591B25-1F13-101B-B826-00DD01103DE1),    // IID_IPoint
```

```
        helpstring("Point object."),
        oleautomation,
        dual
    ]
    interface IPoint : IDispatch
    {
        [propget, helpstring("Returns and sets x coordinate.")]
        HRESULT x([out, retval] int* retval);
        [propput, helpstring("Returns and sets x coordinate.")]
        HRESULT x([in] int Value);

        [propget, helpstring("Returns and sets y coordinate.")]
        HRESULT y([out, retval] int* retval);
        [propput, helpstring("Returns and sets y coordinate.")]
        HRESULT y([in] int Value);
    }
```

The **oleautomation** and **dual** attributes indicate that the interface supports both **IDispatch** and VTBL binding. All the member functions are declared with HRESULT return values. The get accessor functions, indicated by the **propget** attribute, return their value in the last parameter. This parameter has the **out** and **retval** attributes.

For more information on dual interfaces, see the reference page for the **dual** attribute in Chapter 7.

Creating the Programmable Interface

An object's programmable interface comprises the properties and methods it defines. Organizing the objects, properties, and methods that an application exposes is like creating an object-oriented framework for an application. Chapter 4, "Standards and Guidelines," discusses some of the concepts behind naming and organizing the programmable elements your application may expose.

Creating Methods

A *method* is an action that an object can perform, such as drawing a line or clearing the display. Methods can take any number of arguments, including optional arguments, and they may be passed either by value or by reference. A method may or may not return a value.

In the Lines sample, the Application object exposes the following method (APP.CPP):

```
STDMETHODIMP
CApplication::CreatePoint(IPoint FAR* FAR* ppPoint)
{
     CPoint FAR* ppoint = NULL;
     HRESULT hr;

     // Create new item and QI for IDispatch.
     hr = CPoint::Create(&ppoint);
     if (FAILED(hr))
           {hr = RaiseException(IDS_OutOfMemory); goto error;}

     hr = ppoint->QueryInterface(IID_IDispatch, (void FAR* FAR*)ppPoint);
     if (FAILED(hr))
           {hr = RaiseException(IDS_Unexpected); goto error;}
     return NOERROR;

error:
     if (ppoint)
           delete ppoint;
     return hr;
}
```

The CreatePoint method creates a new point and returns a pointer to it in the parameter pPoint.

Creating Properties

A *property* is a member function that sets or returns information about the state of the object, such as color or visibility. Most properties have a pair of accessor functions – a function to get the property value and a function to set the property value. Properties that are read-only or write-only, however, have only one accessor function.

In the Lines sample, the CLine object exposes the Color property. This property is implemented by the following accessor functions (LINE.CPP):

```
STDMETHODIMP
CLine::get_Color(long FAR* plColorref)
{
     *plColorref =  m_colorref;
     return NOERROR;
}

STDMETHODIMP
CLine::put_Color(long lColorref)
{
     m_colorref = (COLORREF)lColorref;
     return NOERROR;
}
```

The accessor functions for a single property have the same dispatch ID (DISPID). The purpose of each function is indicated by attributes set for the function. These attributes are set in the ODL description of the function (as in LINES.ODL) and are passed in the *wFlags* parameter to Invoke to set the context for the call. The attributes and flags are shown in the following table:

Purpose of function	ODL attribute	wFlags
Return a value	**propget**	DISPATCH_PROPERTYGET
Set a value	**propput**	DISPATCH_PROPERTYPUT
Set a reference	**propputref**	DISPATCH_PROPERTYPUTREF

The **propget** attribute designates the accessor function that gets the value of a property. When OLE Automation controller needs to get the value of the property, it passes the DISPATCH_PROPERTYGET flag to **Invoke**.

The **propput** attribute designates the accessor function that sets the value of a property. When an OLE Automation controller needs to set a property by value, it passes the DISPATCH_PROPERTYPUT flag to **Invoke**. In Visual Basic, **Let** statements set properties by value.

The **propputref** attribute indicates that the property should be set by reference, rather than by value. OLE Automation controllers that need to set a reference to a property pass DISPATCH_PROPERTYPUTREF in these cases. Visual Basic treats the **Set** statement as a by-reference property assignment.

The Value Property

The Value property defines the default behavior of an object when no property or method is specified. It is typically used for the property that users associate most closely with the object. For example, a cell in a spreadsheet might have many properties (Font, Width, Height, and so on), but its Value property is special, yielding the value of the cell. To refer to this property, a user need not specify the property name Cell(1,1).Value, but can simply use Cell(1,1).

The Value property is identified by the dispatch ID DISPID_VALUE. In an ODL file, the Value property for an object has the attribute **id(0)**.

In the Lines sample, ILines.Item, IPoints.Item, and IApplication.Name are the Value properties of the ILines, IPoints, and IApplication objects, respectively. The ILines.Item object is described as follows:

```
interface ILines : IDispatch
     {
.
.//Some descriptions omitted
.
     [propget, id(0), helpstring(
"Given an integer index, returns one of the lines in the collection")]
     HRESULT Item([in] long Index,[out, retval] ILine** retval);
.
.
.
}
```

Using this property, a user can refer to the fourth line in the collection as ILines(4).Item or simply as ILines(4).

See Chapter 4, "Standards and Guidelines," for more on recommended objects, properties, and methods.

Returning Objects

To return an object from a property or method, the application should return a pointer to the object's implementation of the **IDispatch** interface. The data type of the return value should be VT_DISPATCH or VT_UNKNOWN (if the object doesn't support **IDispatch**). The Object Description Language (ODL) for the object should specify the name of the interface, rather than **IDispatch***, as follows:

```
ICustom * MyMember(...) {...};
```

The example declares a member that returns a pointer to a custom interface called ICustom.

Passing Formatted Data

Often, an application needs to accept formatted data as an argument to a method or property. Examples include a bitmap, formatted text, or a spreadsheet range. When handling formatted data, the application should pass an object that implements the OLE **IDataObject** interface. For detailed information about the interface, see the *OLE 2 Programmer's Reference, Volume 1*.

Using this interface, applications can retrieve data of any Clipboard format. Because an **IDataObject** instance can provide data of more than one format, a caller can provide data in several formats, and let the called object choose which format is most appropriate (like the Clipboard).

If the data object implements **IDispatch**, it should be passed using the VT_DISPATCH flag. If the data object does not support **IDispatch**, it should be passed with the VT_UNKNOWN flag.

Restricting Access

OLE Automation provides several ways of restricting access to objects. The simplest approach is not to document the properties and methods you don't want customers to see. Alternatively, you can prevent a property or method from appearing in type library browsers by specifying the **hidden** attribute in ODL.

The **restricted** attribute goes one step further, preventing user calls from binding to the property or method as well as hiding it from type browsers. For example, the following restricts access to the _NewEnum property of the ILines object:

```
[propget, restricted, id(DISPID_NEWENUM)]  // Must be propget.
     HRESULT _NewEnum( [out, retval] IUnknown** retval);
```

Restricted properties and methods can be invoked by OLE Automation controllers, but are not visible to the customer or end user who may be using a language such as Visual Basic. In addition, they can't be bound to by user calls.

Creating Collection Objects

A collection object contains a group of exposed objects of the same type and can iterate over them. Collection objects do not need an **IClassFactory** implementation, because they are accessed from elements that have their own class factories.

For example, the Lines sample has a collection object named CLines that iterates over a group of Line objects. The following routine creates and initializes the CLines collection object (LINES.CPP):

```
STDMETHODIMP
CLines::Create(ULONG lMaxSize, long lLBound, CPane FAR* pPane,
                       CLines FAR* FAR* ppLines)
{
     HRESULT hr;
     CLines FAR* pLines = NULL;
     SAFEARRAYBOUND sabound[1];

     *ppLines = NULL;

     // Create new collection.
     pLines = new CLines();
     if (pLines == NULL)
           goto error;

     pLines->m_cMax = lMaxSize;
     pLines->m_cElements = 0;
     pLines->m_lLBound = lLBound;
     pLines->m_pPane = pPane;

     // Load type information for the Lines collection from type library.
     hr = LoadTypeInfo(&pLines->m_ptinfo, IID_ILines);
     if (FAILED(hr))
           goto error;

     // Create a safe array of variants used to implement the collection.
     sabound[0].cElements = lMaxSize;
     sabound[0].lLbound = lLBound;
     pLines->m_psa = SafeArrayCreate(VT_VARIANT, 1, sabound);
     if (pLines->m_psa == NULL)
     {
           hr = ResultFromScode(E_OUTOFMEMORY);
           goto error;
     }

     *ppLines = pLines;
     return NOERROR;

error:
     if (pLines == NULL)
           return ResultFromScode(E_OUTOFMEMORY);
     if (pLines->m_ptinfo)
           pLines->m_ptinfo->Release();
     if (pLines->m_psa)
           SafeArrayDestroy(pLines->m_psa);
```

```
    pLines->m_psa = NULL;
    pLines->m_ptinfo = NULL;

    delete pLines;
    return hr;
}
```

The parameters to CLines::Create specify the maximum number of lines that the collection can contain, the lower bound of the indexes of the collection, and a pointer to a pane, which contains the lines and points in the sample.

OLE Automation defines the **IEnumVARIANT** interface to provide a standard way for OLE Automation controllers to iterate over collections. In addition, every collection object must expose a read-only property named _NewEnum to let OLE Automation controllers know that the object supports iteration. The _NewEnum property returns an enumerator object that supports **IEnumVariant**.

Implementing IEnumVARIANT

The **IEnumVARIANT** interface provides a way to iterate through the items contained by a collection object. This interface is supported by an enumerator object that is returned by the _NewEnum property of the collection object, as in the following figure:

{ewc msdncd, EWGraphic, group10454 7 /a "SDK_02.bmp"}

The **IEnumVARIANT** interface defines these member functions:

- **Next** – Retrieves one or more elements in a collection, starting with the current element.
- **Skip** – Skips over one or more elements in a collection.
- **Reset** – Resets the current element to the first element in the collection.
- **Clone** – Copies the current state of the enumeration so you can return to the current element after using **Skip** or **Reset**.

In the Lines example, CEnumVariant implements the **Next, Skip, Reset,** and **Clone** member functions (ENUMVAR.CPP):

```
STDMETHODIMP
CEnumVariant::Next(ULONG cElements, VARIANT FAR* pvar,
                             ULONG FAR* pcElementFetched)
{
      HRESULT hr;
      ULONG l;
      long l1;
      ULONG l2;

      if (pcElementFetched != NULL)
            *pcElementFetched = 0;

      for (l=0; l<cElements; l++)
            VariantInit(&pvar[l]);

      // Retrieve the next cElements elements.
      for (l1=m_lCurrent, l2=0; l1<(long)(m_lLBound+m_cElements) &&
            l2<cElements; l1++, l2++)
      {
            hr = SafeArrayGetElement(m_psa, &l1, &pvar[l2]);
            if (FAILED(hr))
                  goto error;
      }
      // Set count of elements retrieved
      if (pcElementFetched != NULL)
            *pcElementFetched = l2;
      m_lCurrent = l1;

      return  (l2 < cElements) ? ResultFromScode(S_FALSE) : NOERROR;
error:
      for (l=0; l<cElements; l++)
            VariantClear(&pvar[l]);
      return hr;
}

STDMETHODIMP
CEnumVariant::Skip(ULONG cElements)
```

```
{
    m_lCurrent += cElements;
    if (m_lCurrent > (long)(m_lLBound+m_cElements))
    {
        m_lCurrent =  m_lLBound+m_cElements;
        return ResultFromScode(S_FALSE);
    }
    else return NOERROR;
}

STDMETHODIMP
CEnumVariant::Reset()
{
    m_lCurrent = m_lLBound;
    return NOERROR;
}

STDMETHODIMP
CEnumVariant::Clone(IEnumVARIANT FAR* FAR* ppenum)
{
    CEnumVariant FAR* penum = NULL;
    HRESULT hr;

    *ppenum = NULL;

    hr = CEnumVariant::Create(m_psa, m_cElements, &penum);
    if (FAILED(hr))
        goto error;
    penum->AddRef();
    penum->m_lCurrent = m_lCurrent;

    *ppenum = penum;
    return NOERROR;

error:
    if (penum)
        penum->Release();
    return hr;
}
```

Implementing the _NewEnum Property

The _NewEnum property identifies an object as supporting iteration through the **IEnumVARIANT** interface. The _NewEnum property has the following requirements:

- Must be named _NewEnum and must not be localized.
- Must return a pointer to the enumerator object's **IUnknown** interface.
- Must have DISPID = DISPID_NEWENUM (-4)

The Lines example contains two collections, Lines and Points, and implements a _NewEnum property for each.   Both are restricted properties, available to OLE Automation controllers, but invisible to users of scripting or macro languages supported by OLE Automation controllers. The property returns an enumerator (**IEnumVariant**) for the items in the collection.

The following code implements the _NewEnum property for the Lines collection (LINES.CPP):

```
STDMETHODIMP
CLines::get__NewEnum(IUnknown FAR* FAR* ppunkEnum)
{
     CEnumVariant FAR* penum = NULL;;
     HRESULT hr;

     *ppunkEnum = NULL;

     // Create a new enumerator for items currently in the collection and
     // QueryInterface for IUnknown.
     hr = CEnumVariant::Create(m_psa, m_cElements, &penum);
     if (FAILED(hr))
           {hr = RaiseException(IDS_OutOfMemory); goto error;}
     hr = penum->QueryInterface(IID_IUnknown, (VOID FAR* FAR*)ppunkEnum);
     if (FAILED(hr))
           {hr = RaiseException(IDS_Unexpected); goto error;}
     return NOERROR;

error:
     if (penum)
           delete penum;
     return hr;
}
```

Handling Events

In addition to supporting properties and methods, OLE Automation objects can be a source of events. In OLE Automation, an *event* is a method that is called by an OLE Automation object, rather than implemented by the object. For example, an object might include an event method named Button that retrieves clicks of the mouse button. Instead of being implemented by the object, the Button method returns an object that is a source of events.

In OLE Automation, you use the **source** attribute to identify a member that is a source of events. See the reference page for the **source** attribute in Chapter 7 for more information.

Details of the OLE Automation event interfaces are provided in the *CDK Programmer's Guide*.

Returning an Error

OLE Automation objects typically return rich contextual error information, including an error number, a description of the error, and the path of a Help file that supplies further documentation. Objects that do not need to return detailed error information can simply return an HRESULT that indicates the nature of the error.

Passing Exceptions through IDispatch

When an error occurs, objects invoked through **IDispatch** can return DISP_E_EXCEPTION and pass the details in the *pexcepinfo* parameter (an EXCEPINFO structure) to **IDispatch::Invoke**. The EXCEPINFO structure is defined in Chapter 5, "Dispatch Interfaces."

The Lines sample defines an exception handler that fills the EXCEPINFO structure and signals **IDispatch** to return DISP_E_EXCEPTION (APP.CPP):

```
STDMETHODIMP
CApplication::RaiseException(int nID)
{
     extern SCODE g_scodes[];
     char szError[STR_LEN];
     ICreateErrorInfo *pcerrinfo;
     IErrorInfo *perrinfo;
     HRESULT hr;

     _fmemset(&m_excepinfo, 0, sizeof(EXCEPINFO));

     m_excepinfo.wCode = nID;
     if (LoadString(m_hinst, nID, szError, sizeof(szError)))
           m_excepinfo.bstrDescription = SysAllocString(szError);
     m_excepinfo.bstrSource = SysAllocString(m_bstrName);

     m_bRaiseException = TRUE;

     // Set ErrInfo object so that vtable binding containers can get
     // rich error information.
     hr = CreateErrorInfo(&pcerrinfo);
     if (SUCCEEDED(hr))
     {
           pcerrinfo->SetGUID(IID_IApplication);
           if (m_excepinfo.bstrSource)
                 pcerrinfo->SetSource(m_excepinfo.bstrSource);
           if (m_excepinfo.bstrDescription)
                 pcerrinfo->SetDescription(m_excepinfo.bstrDescription);
           hr = pcerrinfo->QueryInterface(IID_IErrorInfo,
                                    (LPVOID FAR*) &perrinfo);
           if (SUCCEEDED(hr))
           {
                 SetErrorInfo(0, perrinfo);
                 perrinfo->Release();
           }
           pcerrinfo->Release();
     }

     return ResultFromScode(g_scodes[nID-1001]);
}
```

To raise an exception, properties and methods of the CApplication object call RaiseException, passing an integer error ID This routine fills the global EXCEPINFO structure, m_b_excepinfo, with information about the error. It also sets the global flag m_b_RaiseException, which signals **IDispatch::Invoke** to return DISP_E_EXCEPTION.

Passing Exceptions through VTBLs

The Lines example also provides rich error information for members invoked through VTBLs. Because VTBL-bound calls bypass the IDispatch interface, they cannot return exceptions through IDispatch. Instead, they must use OLE Automation's error handling interfaces. The RaiseException function shown in the example calls **CreateErrorInfo** to create an error object, then fills the object's data fields with information about the error. When all the information has been successfully recorded, it calls **SetErrorInfo** to associate the error object with the current thread of execution.

OLE Automation objects similar to the CApplication object, which use the error interfaces, must implement the **ISupportErrorInfo** interface as well. This interface identifies the object as supporting the error interfaces, and ensures that error information can be propagated correctly up the call chain. The following example shows how Lines implements this interface (ERRINFO.CPP):

```
STDMETHODIMP
CSupportErrorInfo::CSupportErrorInfo(IUnknown FAR* punkObject,
            REFIID riid)
{
     m_punkObject = punkObject;
     m_iid = riid;
}

CSupportErrorInfo::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
     return m_punkObject->QueryInterface(iid, ppv);
}

STDMETHODIMP_(ULONG)
CSupportErrorInfo::AddRef(void)
{
     return m_punkObject->AddRef();
}

STDMETHODIMP_(ULONG)
CSupportErrorInfo::Release(void)
{
     return m_punkObject->Release();
}

STDMETHODIMP
CSupportErrorInfo::InterfaceSupportsErrorInfo(REFIID riid)
{
     return (riid == m_iid) ? NOERROR : ResultFromScode(S_FALSE);
}
```

ISupportErrorInfo has the **QueryInterface**, **AddRef**, and **Release** methods inherited from **IUnknown**, along with the **InterfaceSupportsErrorInfo** method. OLE Automation controllers call **InterfaceSupportsErrorInfo** to check whether the OLE Automation object supports the **IErrorInfo** interface, so that they can access the error object.

Releasing OLE

{ewl msdncd, EWGraphic, group10454 8 /a "SDK.BMP"} To release OLE on exit

1. Revoke the active object by calling **RevokeActiveObject**.
2. Revoke the classes of your exposed objects by calling **CoRevokeClassObject**.
3. Uninitialize OLE by calling **OleUninitialize**.

The following code revokes an active Lines object, revokes the Lines class, then uninitializes OLE (MAIN.CPP):

```
void Uninitialize(DWORD dwRegisterCF, DWORD dwRegisterActiveObject)
{
     if (dwRegisterCF != 0)
           CoRevokeClassObject(dwRegisterCF);
     if (dwRegisterActiveObject != 0)
           RevokeActiveObject(dwRegisterActiveObject, NULL);
     OleUninitialize();
}
```

Creating a Type Library

Type information is the OLE Automation standard for describing the objects, properties, and methods that your OLE Automation server exposes. Browsers and compilers use the type information to display and access the exposed objects.

Most OLE Automation servers create type libraries. Type libraries contain type information, Help filenames and contexts, and function-specific documentation strings, and enable access at both compile time and run time. Type libraries are described in Object Description Language (ODL) and are compiled by MkTypLib, as the following figure illustrates:

{ewc msdncd, EWGraphic, group10454 9 /a "SDK_03.bmp"}

OLE Automation also supports the creation of alternative tools that compile and access type information. Chapter 9, "Type Building Interfaces," describes interfaces for creating these tools.

A type library stores complete type information for all of an application's exposed objects. It may be included as a resource in a DLL or remain as a stand-alone file (.TLB).

{ewl msdncd, EWGraphic, group10454 10 /a "SDK.BMP"} To create a type library

1. Write an object description script (.ODL file) for the objects you expose.
2. Build the type library (.TLB) and class description header file (.H) from the script using MkTypLib.

Writing an Object Description Script

An object description script is essentially an annotated header file, written in Object Description Language (ODL). The following example shows a portion of LINES.ODL, the object description script for the Lines sample.

```
[
    uuid(3C591B20-1F13-101B-B826-00DD01103DE1),          // LIBID_Lines
    helpstring("Lines 1.0 Type Library"),
    lcid(0x0409),
    version(1.0)
]

library Lines
{
    importlib("stdole.tlb");
    #define DISPID_NEWENUM -4
.
.
.
```

The preceding entry describes the type library (LINES.TLB) created by the sample. The items in square brackets are attributes, which provide additional information about the library. In the example, the attributes give the library's universally unique identifier (UUID), a Help string, a locale identifier, and a version number.

The **importlib** directive is similar to the C or C++ #include directive. It allows access to the type descriptions in STDOLE.TLB from the Lines library. However, it doesn't copy those types into LINES.TLB. Therefore, to use LINES.TLB, both LINES.TLB and STDOLE.TLB must be available.

By default, ODL files are preprocessed with the C preprocessor, so the #include and #define directives may be used.

The ODL script continues with information on the objects in the type library:

```
    [
        uuid(3C591B25-1F13-101B-B826-00DD01103DE1),          //
IID_IPoint
        helpstring("Point object."),
        oleautomation,
        dual
    ]
    interface IPoint : IDispatch
    {
        [propget, helpstring("Returns and sets x coordinate.")]
        HRESULT x( [out, retval] int* retval);
        [propput, helpstring("Returns and sets x coordinate.")]
        HRESULT x([in] int Value);

        [propget, helpstring("Returns and sets y coordinate.")]
        HRESULT y( [out, retval] int* retval);
        [propput, helpstring("Returns and sets y coordinate.")]
        HRESULT y([in] int Value);
    }
    // .
    // . Additional definitions omitted
    // .
```

```
}
```

This entry describes the IPoint interface. The interface has the attributes **oleautomation** and **dual**, indicating that the types of all its properties and methods are compatible with OLE Automation, and that it supports binding through both **IDispatch** and VTBLs. IPoint has two pairs of property accessor functions, which set and return the X and Y properties.

The Value parameter of both the X and Y properties has the **in** attribute. These parameters supply a value and are read-only. Conversely, the retval parameter of each property has the **out** and **retval** attributes, indicating that it returns the value of the property.

Because IPoint supports VTBL binding and rich error information, its properties return HRESULTs and pass their function return values through **retval** parameters. For more information, see Chapter 7, "MkTypLib and Object Description Language."

```
[
            uuid(3C591B21-1F13-101B-B826-00DD01103DE1),            //
CLSID_Lines
            helpstring("Lines Class"),
            appobject
      ]
      coclass Lines
      {
            [default] interface IApplication;
                  interface IDispatch;
      }
}
```

The file concludes with the description of the Lines Application object, as specified by the **appobject** attribute. The **default** attribute applies to the IApplication interface, indicating that this interface will be returned by default.

Building the Type Library

The MkTypLib tool builds type libraries. MkTypLib is described in detail in Chapter 7, "MkTypLib and Object Description Language."

{ewl msdncd, EWGraphic, group10454 11 /a "SDK.BMP"} To create a type library from an object description script

From Windows, run the MkTypLib tool on the script. For example:

```
MKTYPLIB /TLB output.tlb /H output.h inscript.odl
```

The example creates a type library named OUTPUT.TLB and a header file named OUTPUT.H, based on the object description script INSCRIPT.ODL.

Note that MkTypLib requires Windows. If you run it from NMAKE, you must use the WX server support provided with OLE. For example, start WXSRVR in Windows and use the following command line in your NMAKE script:

```
MKTYPLIB /o odlscrip.err /TLB output.tlb /h output.h inscript.odl
```

Alternatively, you can add MkTypLib to the Tools menu in the Microsoft Visual C++ programming environment. If you do this, be sure to check the "Ask for Arguments" check box, so the programming environment will prompt you for the options and input file before running the tool.

After creating the type library (.TLB), you can include it in the resource step of building your application or leave it as a stand-alone file. In either case, be sure to specify the filename and path of the library in the application's registration file (.REG), so that OLE Automation can find the type library when necessary. See the following section for information on registering the type library.

{ewl msdncd, EWGraphic, group10454 12 /a "SDK.BMP"} To build an application that uses a type library

1. Include the header file output by MkTypLib in your project.
2. Compile the project.
3. Optionally, use the Resource Compiler to bind the type library with your compiled project. You can bind the type library with DLLs or with .EXE files. For example, to bind a type library named OUTPUT.TLB with a DLL, use the following statement in the .RC file for the DLL:

   ```
   1 typelib output.tlb
   ```

   A DLL that contains a typelib resource usually has the .OLB (object library) extension.

The following figure illustrates the process.

{ewc msdncd, EWGraphic, group10454 13 /a "SDK_04.bmp"}

Creating a Registration File

Before an application can use OLE and OLE Automation, the OLE objects must be registered with the user's system registration database. Registration makes the following possible:

- OLE Automation controllers can create instances of the OLE Automation objects through **CoCreateInstance.**
- OLE Automation tools can find the type libraries that are installed on the user's computer.
- OLE can find remoting code for the interfaces.

OLE provides sample registration files to perform this task for the OLE objects and the sample applications.

{ewl msdncd, EWGraphic, group10454 14 /a "SDK.BMP"} To create a registration file

1. Copy LINES.REG.
2. Rename and edit the file, adding entries for your application.

The registration file provides information about the application, Application object, classes of objects, type libraries, and interfaces. Entries for objects and interfaces start with the constant HKEY_CLASSES_ROOT, which represents the root key of the entire registration database. Entries for type libraries start with HKEY_TYPELIB_ROOT. After the constant, each entry supplies specific information about an object, type library, or interface.

The following sections give a brief overview of the syntax used in registering OLE Automation objects. For detailed information, refer to the *OLE 2 Programmer's Reference, Volume 1*.

Note that you can use the **DLLRegisterServer** function to register all the objects implemented by a DLL. This function registers the class IDs for each object, the ProgIDs for each application, and the type library. For details, refer to the description of **DLLRegisterServer** in the *CDK Programmer's Guide*.

Registering the Application

Registration maps the programmatic ID (ProgID) of the application to a unique class ID (CLSID), so that you can create instances of the application by name, rather than by CLSID. For example, registering Microsoft Excel associates a CLSID with the ProgID "Excel.Application." In Visual Basic, you use the ProgId to create an instance of the application, as follows:

```
SET xl = CreateObject("Excel.Application")
```

By passing the ProgID to **CLSIDFromProgID**, you can get the corresponding CLSID for use in **CoCreateInstance**. Only applications that will be used in this way need to be registered.

The registration file uses the following syntax for the application:

\ *AppName.ObjectName*[.*VersionNumber*] **=** *human_readable_string*
\ *AppName.ObjectName*\**CLSID = {***UUID***}**

AppName
The name of the application.

ObjectName
The name of the object to be registered, in this case, Application.

VersionNumber
The optional version number of the object.

human_readable_string
A string that describes the application, as you want it to appear to users. The recommended maximum length is 40 characters.

UUID
The universally unique ID for the application class (CLSID). To generate a universally unique ID for your class, run the utility GUIDGEN.EXE provided in the \OLE2\BIN directory.

Example

```
REGEDIT
; Registration information for OLE Automation Hello 2.0 Application

; Version independent registration. Points to Version 2.0
HKEY_CLASSES_ROOT\Hello.Application = OLE Automation Hello Application
HKEY_CLASSES_ROOT\Hello.Application\Clsid = {F37C8061-4AD5-101B-B826-
00DD01103DE1}

; Version 2.0 registration
HKEY_CLASSES_ROOT\Hello.Application.2 = OLE Automation Hello 2.0 Application
HKEY_CLASSES_ROOT\Hello.Application.2\Clsid = {F37C8061-4AD5-101B-B826-
00DD01103DE1}
```

Registering Classes

Objects that can be created with **CoCreateInstance** must also be registered with the system. For these objects, registration maps a CLSID to the OLE Automation server file (.DLL or .EXE) in which the object resides. The CLSID also maps an OLE Automation object back to its application and ProgID. The following figure shows how registration connects ProgIDs, CLSIDs, and OLE Automation servers:

{ewc msdncd, EWGraphic, group10454 15 /a "SDK_05.bmp"}

The registration file uses the following syntax for each class of each object your application exposes:

\CLSID\**{***UUID***}** = *human_readable_string*
  \CLSID\**{***UUID***}\ProgID** = *AppName.ObjectName.VersionNumber*
  \CLSID\**{***UUID***}\VersionIndependentProgID** = *AppName.ObjectName*
  \CLSID\**{***UUID***}\LocalServer**[**32**] = *filepath*[/**Automation**]
  \CLSID\**{***UUID***}\InProcServer**[**32**] = *filepath*[/**Automation**]

human_readable_string
  A string that describes the object, as you want it to appear to users. The recommended maximum length is 40 characters.

AppName
  The name of the application, as specified previously in the application registration string.

ObjectName
  The name of the object to be registered.

VersionNumber
  The version number of the object.

UUID
  The universally unique ID for the application class (CLSID). To generate a universally unique ID for your class, run the utility GUIDGEN.EXE provided in the \OLE2\BIN directory.

filepath
  The full path and name of the file that contains the object. The optional **/Automation** switch tells the application it was launched for OLE Automation purposes. The switch should be specified for the Application object's class. For more information on **/Automation**, see the section "Initializing the Active Object" earlier in this chapter.

The programmatic IDs (ProgID and VersionIndependentProgID) are used by other programmers to gain access to the objects you expose. These IDs should follow consistent naming guidelines across all your applications. They may contain up to 39 characters, must not contain any punctuation (except for the period), and must not start with a digit.

Version-independent names consist of an *AppName*.*ObjectName*, without a version number; for example, Word.Document or Excel.Chart.

Version-dependent names consist of an *AppName*.*ObjectName*.*VersionNumber*, such as Excel.Application.5.

LocalServer indicates that the OLE Automation server is an EXE file and runs in a separate process from the OLE Automation controller. **InProcServer** indicates that the server is a DLL and runs in the same process as the OLE Automation controller. The optional **32** specifies a server intended for use on 32-bit Windows systems.

The *filepath* you register should give the full path and name, so that applications need not rely on the MS-DOS PATH variable to find the object.

Example

```
HKEY_CLASSES_ROOT\CLSID\{F37C8061-4AD5-101B-B826-00DD01103DE1} = Hello 2.0
Application
HKEY_CLASSES_ROOT\CLSID\{F37C8061-4AD5-101B-B826-00DD01103DE1}\ProgID =
Hello.Application.2
HKEY_CLASSES_ROOT\CLSID\{F37C8061-4AD5-101B-B826-
00DD01103DE1}\VersionIndependentProgID = Hello.Application
HKEY_CLASSES_ROOT\CLSID\{F37C8061-4AD5-101B-B826-00DD01103DE1}\LocalServer =
hello.exe /Automation
```

Registering Type Library Information

Tools and applications that expose type information must register the information so that it is available to type browsers and programming tools. You can generate the correct registration entries for a type library by calling the **RegisterTypeLib** function on the type library. You can then use REGEDIT, supplied with the Windows SDK, to write the registration entries to a text file from your system registration database.

The following information is registered for a type library:

\TypeLib\{*libUUID***}**
  \TypeLib\{*libUUID***}***major.minor* **=** *human_readable_string*
  \TypeLib\{*libUUID***}***major.minor***\HELPDIR =** [*helpfile_path*]
  \TypeLib\{*libUUID***}***major.minor***\Flags =** *typelib_flags*
  \TypeLib\{*libUUID***}***major.minor**lcid**platform* **=** *localized_typelib_filename*

libUUID
  The universally unique ID of the type library.

major.minor
  The two-part version number of the type library. If only the minor version number increases, all the features of the previous type library are supported in a compatible way. If the major version number changes, code that compiled against the type library must be recompiled. The version number of the type library may differ from the version number of the application.

human_readable_string
  A string that describes the type library, as you want it to appear to users. The recommended maximum length is 40 characters.

helpfile_path
  The directory where the Help file for the types in the type library is located. Note that if the application supports type libraries for multiple languages, the libraries may refer to different filenames in the Help file directory.

typelib_flags
  The hexadecimal representation of the type library flags for this type library. These are the values of the LIBFLAGS enumeration, and are the same flags specified in the *uLibFlags* parameter to **ICreateTypeLib::SetLibFlags**. They can't have leading zeros or the 0x prefix.

lcid
  The hexadecimal string representation of the locale ID (LCID). It is one to four hexadecimal digits with no 0x prefix and no leading zeros. The LCID may have a neutral sublanguage.

platform
  The target operating system platform: 16-bit Windows, 32-bit Windows, or Mac.

localized_typelib_filename
  The full name of the localized type library.

Using the LCID specifier, an application can explicitly register the filenames of type libraries for different languages. This allows the application to find the desired language without having to open all type libraries with a given name. To find the type library for Australian English (309), the application would first look for it and, if that fails, look for an entry for Standard English (9). If there is no entry for Standard English, the application looks for the LANG_SYSTEM_DEFAULT (0). For more information on locale support, refer to your operating system documentation for the NLS interface; for 16-bit systems, see Appendix A, "National Language Support Functions."

Example

```
; Type library registration information
HKEY_CLASSES_ROOT\TypeLib\{F37C8060-4AD5-101B-B826-00DD01103DE1}
```

```
HKEY_CLASSES_ROOT\TypeLib\{F37C8060-4AD5-101B-B826-00DD01103DE1}\2.0 = OLE
Automation Hello 2.0 Type Library
HKEY_CLASSES_ROOT\TypeLib\{F37C8060-4AD5-101B-B826-00DD01103DE1}\2.0\HELPDIR
=
;US english
HKEY_CLASSES_ROOT\TypeLib\{F37C8060-4AD5-101B-B826-
00DD01103DE1}\2.0\409\win16 = hello.tlb
```

Registering Interfaces

Applications that add interfaces need to register the interfaces, so that OLE can find the appropriate remoting code for interprocess communication. By default, OLE Automation registers dispinterfaces that appear in the ODL file. It also registers remote OLE Automation-compatible interfaces that are not registered elsewhere under the ProxyStubClsid for PSAutomation.

The information registered for an interface is as follows:

Interface\{*IID***}** = *InterfaceName*
  Interface\**{***IID***}**\Typelib = *LIBID*
  Interface\**{***IID***}**\ProxyStubClsid[**32**] = *CLSID*

IID
  The universally unique ID of the interface.

InterfaceName
  The name of the interface.

LIBID
  The UUID associated with the type library in which the interface is described.

CLSID
  The UUID associated with the proxy/stub implementation of the interface, used internally by OLE for interprocess communication. OLE Automation objects use the proxy/stub implementation of **IDispatch**.

Example

The following lines from HELLO.REG register the interface for VTBL binding. The ProxyStubClsid in the example refers to the proxy/stub implementation of **IDispatch**.

```
HKEY_CLASSES_ROOT\Interface\{F37C8062-4AD5-101B-B826-00DD01103DE1} = IHello
HKEY_CLASSES_ROOT\Interface\{F37C8060-4AD5-101B-B826-00DD01103DE1}\TypeLib =
{F37C8060-4AD5-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\Interface\{F37C8060-4AD5-101B-B826-
00DD01103DE1}\ProxyStubClsid = {00020424-0000-0000-C000-000000000046}
```

Registration File Example

Lines uses the following entries to register its Application object (Lines.Application) and its type library with the system (LINES.REG):

```
REGEDIT
; Registration information for the Lines Automation Object application

; Version independent registration
HKEY_CLASSES_ROOT\Lines.Application = Lines
HKEY_CLASSES_ROOT\Lines.Application\Clsid = {3C591B21-1F13-101B-B826-
00DD01103DE1}

; Version 1.0 registration
HKEY_CLASSES_ROOT\Lines.Application.1 = Lines 1.0
HKEY_CLASSES_ROOT\Lines.Application.1\Clsid = {3C591B21-1F13-101B-B826-
00DD01103DE1}
HKEY_CLASSES_ROOT\CLSID\{3C591B21-1F13-101B-B826-00DD01103DE1} = Lines 1.0
HKEY_CLASSES_ROOT\CLSID\{3C591B21-1F13-101B-B826-00DD01103DE1}\ProgID =
Lines.Application.1
HKEY_CLASSES_ROOT\CLSID\{3C591B21-1F13-101B-B826-
00DD01103DE1}\VersionIndependentProgID = Lines.Application
HKEY_CLASSES_ROOT\CLSID\{3C591B21-1F13-101B-B826-00DD01103DE1}\LocalServer =
lines.exe /Automation

; Type library registration information
HKEY_CLASSES_ROOT\TypeLib\{3C591B20-1F13-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\TypeLib\{3C591B20-1F13-101B-B826-00DD01103DE1}\1.0 = Lines
1.0 Type Library
HKEY_CLASSES_ROOT\TypeLib\{3C591B20-1F13-101B-B826-00DD01103DE1}\1.0\HELPDIR
=
;US english
HKEY_CLASSES_ROOT\TypeLib\{3C591B20-1F13-101B-B826-
00DD01103DE1}\1.0\409\win16 = lines.tlb

; Interface registration. All interfaces that support vtable binding must be
; registered as follows. RegisterTypeLib & LoadTypeLib will do this
automatically.

; LIBID_Lines = {3C591B20-1F13-101B-B826-00DD01103DE1}

; IID_IPoint = {3C591B25-1F13-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\Interface\{3C591B25-1F13-101B-B826-00DD01103DE1} = IPoint
HKEY_CLASSES_ROOT\Interface\{3C591B25-1F13-101B-B826-00DD01103DE1}\TypeLib =
{3C591B20-1F13-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\Interface\{3C591B25-1F13-101B-B826-
00DD01103DE1}\ProxyStubClsid = {00020424-0000-0000-C000-000000000046}

; IID_ILine = {3C591B24-1F13-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\Interface\{3C591B24-1F13-101B-B826-00DD01103DE1} = ILine
HKEY_CLASSES_ROOT\Interface\{3C591B24-1F13-101B-B826-00DD01103DE1}\TypeLib =
{3C591B20-1F13-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\Interface\{3C591B24-1F13-101B-B826-
00DD01103DE1}\ProxyStubClsid = {00020424-0000-0000-C000-000000000046}
```

```
; IID_ILines = {3C591B26-1F13-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\Interface\{3C591B26-1F13-101B-B826-00DD01103DE1} = ILines
HKEY_CLASSES_ROOT\Interface\{3C591B26-1F13-101B-B826-00DD01103DE1}\TypeLib =
{3C591B20-1F13-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\Interface\{3C591B26-1F13-101B-B826-
00DD01103DE1}\ProxyStubClsid = {00020424-0000-0000-C000-000000000046}

; IID_IPoints = {3C591B27-1F13-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\Interface\{3C591B27-1F13-101B-B826-00DD01103DE1} = IPoints
HKEY_CLASSES_ROOT\Interface\{3C591B27-1F13-101B-B826-00DD01103DE1}\TypeLib =
{3C591B20-1F13-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\Interface\{3C591B27-1F13-101B-B826-
00DD01103DE1}\ProxyStubClsid = {00020424-0000-0000-C000-000000000046}

; IID_IPane = {3C591B23-1F13-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\Interface\{3C591B23-1F13-101B-B826-00DD01103DE1} = IPane
HKEY_CLASSES_ROOT\Interface\{3C591B23-1F13-101B-B826-00DD01103DE1}\TypeLib =
{3C591B20-1F13-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\Interface\{3C591B23-1F13-101B-B826-
00DD01103DE1}\ProxyStubClsid = {00020424-0000-0000-C000-000000000046}

; IID_IApplication = {3C591B22-1F13-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\Interface\{3C591B22-1F13-101B-B826-00DD01103DE1} =
IApplication
HKEY_CLASSES_ROOT\Interface\{3C591B22-1F13-101B-B826-00DD01103DE1}\TypeLib =
{3C591B20-1F13-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\Interface\{3C591B22-1F13-101B-B826-
00DD01103DE1}\ProxyStubClsid = {00020424-0000-0000-C000-000000000046}
```

Supporting Multiple National Languages

Applications sometimes need to expose objects with names that differ across localized versions of the product. The names pose a problem for programming languages that need to access these objects, because late binding will be sensitive to the locale of the application. The **IDispatch** interface provides a range of solutions that vary in cost of implementation and quality of language support. All methods of the **IDispatch** interface that are potentially sensitive to language are passed a locale ID (LCID), which identifies the local language context.

The following are some of the approaches a class implementation may take:

- Accept any LCID and use the same member names in all locales. This is acceptable if the exposed interface will typically be accessed only by very advanced users. For example, the member names for OLE 2 interfaces will never be localized.
- Accept all LCIDs supported by all versions of the product. In this case, the implementation of **GetIDsOfNames** would need to interpret the passed array of names based on the given LCID. This is the most acceptable solution because it allows users to write code in their natural language and run the code on any localized version of the application.
- Simply return an error (DISP_E_UNKNOWNLCID) from **GetIDsOfNames** if the caller's LCID doesn't match the localized version of the class. This would prevent your customers from being able to write late-bound code that runs on machines with different localized implementations of the class.
- Recognize the particular version's localized names, as well as one language that is recognized in all versions. For example, a French version might accept French and English names, where English is the language supported in all versions. Users who want to write code that runs in all countries would have to use English.

However, to provide general language support, the application should check the LCID before interpreting member names. Because **Invoke** is passed an LCID, methods can properly interpret parameters whose meaning varies by locale. The following sections provide examples and guidelines of how to create multilingual applications.

Implementing IDispatch for Multilingual Applications

When creating applications that will support multiple languages, you need to create separate type libraries for each supported language and write versions of the IDispatch member functions that include dependencies for each language. In the following example, the Hello example code is modified to define locale IDs for both US English and German.

Write a separate type library for each supported language.

The type libraries use the same DISPIDs and GUIDs but localize names and Help strings based on the language. The following registration file example includes entries for US English and German.

```
; Type library registration information
HKEY_CLASSES_ROOT\TypeLib\{F37C8060-4AD5-101B-B826-00DD01103DE1}
HKEY_CLASSES_ROOT\TypeLib\{F37C8060-4AD5-101B-B826-00DD01103DE1}\2.0 =
Hello 2.0 Type Library
HKEY_CLASSES_ROOT\TypeLib\{F37C8060-4AD5-101B-B826-00DD01103DE1}\2.0\HELPDIR
=
;US english
HKEY_CLASSES_ROOT\TypeLib\{F37C8060-4AD5-101B-B826-
00DD01103DE1}\2.0\409\win16 = helloeng.tlb
;German
HKEY_CLASSES_ROOT\TypeLib\{F37C8060-4AD5-101B-B826-
00DD01103DE1}\2.0\407\win16 = helloger.tlb
```

Define the locale IDs for the supported languages.

Refer to the next section to obtain the language IDs for your supported languages.

```
// Locale IDs for the languages that are supported
#define LCID_ENGLISH MAKELCID(MAKELANGID(0x09, 0x01))
#define LCID_GERMAN  MAKELCID(MAKELANGID(0x07, 0x01))
```

Using the Hello code example, you define member variables that will be used to hold English and German type information.

```
class FAR CHello : public IHello
{
public:
     :

private:
     LPTYPEINFO m_ptinfoEnglish;   // English type information of Hello
                                                     application interface.
     LPTYPEINFO m_ptinfoGerman;    // German type information of Hello
                                                           application
interface.
     :
};
```

Load type information for each supported language

The following example uses the LoadTypeInfo function to load locale-specific type library information when the Hello object is created.

```
LoadTypeInfo(&phello->m_ptinfoEnglish, IID_IHello, LCID_ENGLISH);
LoadTypeInfo(&phello->m_ptinfoGerman, IID_IHello, LCID_GERMAN);
```

```
/* LoadTypeInfo - Gets type information of an object's interface from
 * the type library.
 *
 * Parameters:
 *  ppunkStdDispatch         Returns type information.
 *  clsid                    Interface id of object in type library.
 *  lcid                           Locale ID of typeinfo to be loaded.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT LoadTypeInfo(ITypeInfo FAR* FAR* pptinfo, REFCLSID clsid,
LCID lcid)
{
     HRESULT hr;
     LPTYPELIB ptlib = NULL;
     LPTYPEINFO ptinfo = NULL;

     *pptinfo = NULL;

     // Load Type Library.
     hr = LoadRegTypeLib(LIBID_Hello, 2, 0, lcid, &ptlib);
     if (FAILED(hr))
           return hr;

     // Get type information for interface of the object.
     hr = ptlib->GetTypeInfoOfGuid(clsid, &ptinfo);
     if (FAILED(hr))
     {
           ptlib->Release();
           return hr;
     }

     ptlib->Release();
     *pptinfo = ptinfo;
     return NOERROR;
}
```

Implement the IDispatch member functions

The following code implements language-sensitive versions of **GetTypeInfoCount**, **GetIDsOfNames**, and **Invoke**. Note that Invoke doesn't check the locale ID, but merely passes it to **DispInvoke**.

```
STDMETHODIMP
CHello::GetTypeInfoCount(UINT FAR* pctinfo)
{
     *pctinfo = 1;
     return NOERROR;
}

STDMETHODIMP
CHello::GetTypeInfo(
           UINT itinfo,
           LCID lcid,
           ITypeInfo FAR* FAR* pptinfo)
{
     LPTYPEINFO ptinfo;
     *pptinfo = NULL;

     if(itinfo != 0)
           return ResultFromScode(DISP_E_BADINDEX);

     if(lcid == LOCALE_SYSTEM_DEFAULT || lcid == 0)
           lcid = GettSystemDefaultLCID();

     if(lcid == LOCALE_USER_DEFAULT)
           lcid = GetUserDefaultLCID();

     switch(lcid)
     {
           case LCID_GERMAN:
                 ptinfo = m_ptinfoGerman;
                 break;

           case LCID_ENGLISH:
                 ptinfo = m_ptinfoEnglish;
                 break;

           default:
                 return ResultFromScode(DISP_E_UNKNOWNLCID);
     }

     ptinfo->AddRef();
     *pptinfo = ptinfo;
     return NOERROR;
}

STDMETHODIMP
CHello::GetIDsOfNames(
           REFIID riid,
           OLECHAR FAR* FAR* rgszNames,
           UINT cNames,
           LCID lcid,
           DISPID FAR* rgdispid)
{
     LPTYPEINFO ptinfo;

     if(lcid == LOCALE_SYSTEM_DEFAULT || lcid == 0)
           lcid = GetSystemDeraultLCID();

     if(lcid == LOCALE_USER_DEFAULT)
           lcid = GetUserDefaultLCID();

     switch(lcid)
     {
           case LCID_GERMAN:
                 ptinfo = m_ptinfoGerman;
                 break;
```

```
          case LCID_ENGLISH:
                ptinfo = m_ptinfoEnglish;
                break;

          default:
                return ResultFromScode(DISP_E_UNKNOWNLCID);
     }
     return DispGetIDsOfNames(ptinfo, rgszNames, cNames, rgdispid);
}

STDMETHODIMP
CHello::Invoke(
          DISPID dispidMember,
          REFIID riid,
          LCID lcid,
          WORD wFlags,
          DISPPARAMS FAR* pdispparams,
          VARIANT FAR* pvarResult,
          EXCEPINFO FAR* pexcepinfo,
          UINT FAR* puArgErr)
{
     HRESULT hr;
     m_bRaiseException = FALSE;
     hr =  DispInvoke(
           this, m_ptinfoEnglish,
           dispidMember, wFlags, pdispparams,
           pvarResult, pexcepinfo, puArgErr);
     if (m_bRaiseException)
     {
           if (NULL != pexcepinfo)
                 _fmemcpy(pexcepinfo, &m_excepinfo, sizeof(EXCEPINFO));
           return ResultFromScode(DISP_E_EXCEPTION);
     }
     else return hr;
}
```

Interpreting Arguments and Strings Based on LCID

Some methods or properties need to interpret arguments based on the LCID. Such methods or properties can require that an LCID be passed as an argument.

The following code example implements a property that takes an LCID. In this example, **get_CheckingBalance** returns a currency string that contains the amount of money in the checking account. The currency string should be correctly formated depending on the locale that is passed in. **ConvertCurrency** is a private function that converts the checking balance to the currency of the country described by **llcid**. The string form of converted currency is placed in **m_szBalance**. **GetCurrencyFormat** is a 32-bit Windows function that that formats a currency string for the given locale.

The following represents the .ODLfile:

```
    [
    uuid(83219430-CB36-11cd-B774-00DD01103DE1),
    helpstring("Bank Account object."),
    oleautomation,
    dual
    ]
    interface IBankAccount : IDispatch
    {
          [propget, helpstring("Returns account balance formatted for the
                country described by localeID.")]
          HRESULT CheckingBalance([in, lcid] long localeID, [out, retval]
          BSTR* retval);
          :
    }
```

The following represents the .H file:

```
class FAR CBankAccount : public IBankAccount
{
    public:
    // IUnknown methods
          :

    // IDispatch methods
          :

    // IBankAccount methods
    STDMETHOD(get_CheckingBalance)(long llcid, BSTR FAR* pbstr);
          :
}
```

The following represents the .CPP file:

```
STDMETHODIMP
CBankAccount::get_CheckingBalance(long llcid, BSTR FAR* pbstr)
{
    TCHAR ach[100];
    ConvertCurrency(llcid);
          GetCurrencyFormat(llcid, 0, m_szBalance, NULL, ach,
          sizeof(ach));
          *pbstr = SysAllocString(ach); // Return currency string formated
```

```
                                                            // according to locale
id.
            return NOERROR;
}
```

The LCID is commonly used to parse strings that contain locale-dependent information. For example, a function that takes a string such as "6/11/59" needs the LCID to determine whether the month is June (6) or November (11). The LCID should not be used for output strings, including error strings; these should always be displayed in the current system language.

Locale, Language, and Sublanguage IDs

The following macro is defined for creating LCIDs (WINNT.H for 32-bit systems; OLENLS.H for 16-bit systems):

```
/*
 * LCID creation/extraction macros:
 *
 *   MAKELCID - construct locale ID from language ID and
 *                          country code.
 */
#define MAKELCID(l)     ((DWORD)(((WORD)(l)) | (((DWORD)((WORD)(0))) << 16)))
```

There are two predefined LCID values: LOCALE_SYSTEM_DEFAULT is the system default locale, and LOCALE_USER_DEFAULT is the current user's locale.

Another macro constructs a language ID:

```
/*
 * Language ID creation/extraction macros:
 *
 *         MAKELANGID - construct language ID from primary language ID and
 *                          sublanguage ID
*/
#define MAKELANGID(p, s)                  ((((USHORT)(s)) << 10) | (USHORT)
(p))
```

The following three combinations of primary language ID and sublanguage ID have special meanings:

PRIMARYLANGID	**SUBLANGID**	**Result**
LANG_NEUTRAL	SUBLANG_NEUTRAL	Language neutral
LANG_NEUTRAL	SUBLANG_SYS_DEFAULT	System default language
LANG_NEUTRAL	SUBLANG_DEFAULT	User default language

For primary language IDs, the range 0x200 to 0x3ff is user definable. The range 0x000 to 0x1ff is reserved for system use. For sublanguage IDs, the range 0x20 to 0x3f is user definable. The range 0x00 to 0x1f is reserved for system use.

Language tables

The following table lists the primary language IDs supported by OLE Automation. For more information on national language support for 16-bit Windows systems, refer to Appendix A, "National Language Support Functions." On 32-bit Windows systems, see your operating system documentation.

Language	PRIMARYLANGID
Neutral	0x00
Chinese	0x04
Czech	0x05
Danish	0x06
Dutch	0x13
English	0x09
Finnish	0x0b
French	0x0c
German	0x07
Greek	0x08
Hungarian	0x0e
Icelandic	0x0F
Italian	0x10
Japanese	0x11
Korean	0x12
Norwegian	0x14
Polish	0x15
Portuguese	0x16
Russian	0x19
Slovak	0x1b
Spanish	0x0a
Swedish	0x1d
Turkish	0x1F

The following table lists the sublanguage IDs supported by OLE Automation. For more information on national language support for 16-bit systems, refer to Appendix A, "National Language Support Functions." On 32-bit Windows systems, see your operating system documentation.

Sublanguage	SUBLANGID
Neutral	0x00
Default	0x01
System Default	0x02
Chinese (Simplified)	0x02
Chinese (Traditional)	0x01
Czech	0x01
Danish	0x01
Dutch	0x01
Dutch (Belgian)	0x02
English (US)	0x01
English (UK)	0x02

English (Australian)	0x03
English (Canadian)	0x04
English (Irish)	0x06
English (New Zealand)	0x05
Finnish	0x01
French	0x01
French (Belgian)	0x02
French (Canadian)	0x03
French (Swiss)	0x04
German	0x01
German (Swiss)	0x02
German (Austrian)	0x03
Greek	0x01
Hungarian	0x01
Icelandic	0x01
Italian	0x01
Italian (Swiss)	0x02
Japanese	0x01
Korean	0x01
Norwegian (Bokmal)	0x01
Norwegian (Nynorsk)	0x02
Polish	0x01
Portuguese	0x02
Portuguese (Brazilian)	0x01
Russian	0x01
Slovak	0x01
Spanish (Castilian)[1]	0x01
Spanish (Mexican)	0x02
Spanish (Modern)[1]	0x03
Swedish	0x01
Turkish	0x01

[1] The only difference between Spanish (Castilian) and Spanish (Modern) is the sort ordering. The LCType values are all the same.

Accessing OLE Automation Objects

To access exposed objects, you can create OLE Automation controllers using Microsoft Visual Basic™, Microsoft Visual C++™, Microsoft Excel, Microsoft Project, and other applications and programming languages that support OLE. This chapter discusses several strategies for accessing exposed objects:

- Creating scripts with Microsoft Visual Basic
- Creating your own controllers that manipulate objects
- Creating type information browsers

Regardless of the strategy you choose, your OLE Automation controller needs to follow these steps:

{ewl msdncd, EWGraphic, group10455 0 /a "SDK.BMP"} To initialize and create the object

1. Initialize OLE.
2. Create an instance of the exposed object.

{ewl msdncd, EWGraphic, group10455 1 /a "SDK.BMP"} To manipulate methods and properties

1. Get information about the object's methods and properties.
2. Invoke the methods and properties.

{ewl msdncd, EWGraphic, group10455 2 /a "SDK.BMP"} To release OLE when your application or programming tool terminates

1. Revoke the active object.
2. Unitialize OLE.

Packaging Scripts Using Visual Basic

Microsoft Visual Basic provides a complete programming environment for creating Windows applications.

Visual Basic can manipulate the exposed OLE objects of other applications. Internally, it fully supports OLE Automation **dual** interfaces. For the syntax and semantics of the OLE Automation features, see the Visual Basic Help file, VB.HLP. Appendix C lists how the Visual Basic statements translate into OLE application programming interfaces (APIs).

Note   You don't need Visual Basic in order to use OLE Automation. It is presented here as an example of a programming tool that supports OLE Automation and is convenient for packaging OLE Automation scripts. Optionally, you can use a different OLE Automation controller for your testing.

You can call exposed objects directly from programs written with Visual Basic. This figure shows how this was done for the sample program HELLO.EXE:

{ewc msdncd, EWGraphic, group10455 3 /a "SDK_01.bmp"}

{ewl msdncd, EWGraphic, group10455 4 /a "SDK.BMP"}   To access an exposed object

1. Start Visual Basic. Initialization and release of OLE is handled automatically by Visual Basic.
2. Use the Tools/References menu to select the type library of the object.
3. Add code to declare a variable of the interface type. For example:

```
Dim HelloObj As IHello
```

4. Add code in event procedures to create an instance of the object and to manipulate the object using its properties and methods. For example:

```
Sub Form_Load ( )
   Set HelloObj = New Hello.Hello
End Sub
Sub SetVisible_Click ( )
   HelloObj.Visible = True
End Sub
```

5. From the Run menu, choose Start and trigger the event by clicking the form.

The following figure shows the interfaces you use when accessing exposed objects through Visual Basic.

{ewc msdncd, EWGraphic, group10455 5 /a "SDK_02.bmp"}

Accessing a Remote Object

With Visual Basic, accessing the remote object requires only that the program declare an object variable and assign the return of a **New** statement to the variable. The following is the syntax for the statements:

Dim *ObjectVar* **As** *InterfaceName*
  Set *ObjectVar* **= New** *CoClassName*

The **Dim** statement declares a variable of an interface type. The **New** keyword creates an instance of an object. Used together, the two declare and create an instance of an OLE Automation object. For example:

```
Dim MyLines As ILines
Set MyLines = New Lines.Lines
```

The **Dim** statement declares the object variable MyLines of the interface type ILines. The **Set** statement assigns a new object of the *component object class* (coclass) Lines to the variable MyLines. When you use the **Dim** statement to set a variable to an interface type, susbsequent uses of the variable execute faster than with the generic **Object** syntax.

The **New** keyword applies only to creating coclasses of interface or dispinterface types. To create other types of objects, you must declare the variable with the **Dim** statement, and then use the **CreateObject** function as follows:

Set *ObjectVar* **= CreateObject(***ProgID***)**

CreateObject creates an OLE Automation object, based on the specified *ProgID*. The *ProgID* has the form:

"*AppName.ObjectName*"

The *AppName* is the name of the application, and the *ObjectName* identifies the type of object to create. See "Registering the Application" in Chapter 2 for more information on ProgIDs.

The **GetObject** function can be used to re-establish the reference to the most recently used object that corresponds to the *Filename* and *AppName.ObjectName* specification.

Set *ObjectVar* **= GetObject("***Filename***",** *ProgID***)**

For example, if the OLE Automation controller needs an existing instance of an object instead of a new instance, it could use **GetObject**.

The Hello sample application included in the OLE 2 Software Development Kit (SDK) displays a Hello message in response to a mouse click. You can add a simple form that accesses the Hello application's exposed object from another process.

{ewl msdncd, EWGraphic, group10455 6 /a "SDK.BMP"} To add a form

1. Start Visual Basic.
2. Choose Open Project from the File menu.
3. In the dialog box, select VB.MAK from the \OLE2\SAMPLE\HELLO\VB directory.
4. In the Forms box, choose View Form to see the form, or choose View Code to see the Visual Basic code.

```
'Module-level declarations
Dim HelloObj As IHello

Sub Form_load ( )
   Set HelloObj = New Hello.Hello
End Sub
```

```
  Sub Invoke_SayHello_Method_Click ( )
     HelloObj.SayHello
  End Sub

  Sub Get_HelloMsg_Property_Click ()
     Debug.Print HelloObj.HelloMessage
  End Sub

  Sub Set_HelloMsg_Property_Click ( )
     HelloObj.HelloMessage = "Hello Universe"
  End Sub

  Sub SetVisible_Click ( )
     HelloObj.Visible = True
  End Sub
```

The Form_Load subroutine creates the Hello Application object, and the other subroutines manipulate Hello's Application object through the Visible and HelloMessage properties and the SayHello method.

To program an object in Visual Basic, you need its class name and the names and parameters of its properties and methods. For the Hello sample, you need to know the exact names of the SayHello method and the Visible and HelloMsg properties, and the types of their arguments. This information is published as documentation for many objects, such as those exposed by Microsoft Excel. You can also obtain the information by viewing the object's type library with an object browser like the one included in Visual Basic. A sample browser, BROWSE, is provided in the OLE 2 SDK.

Creating an Invisible Object

In the preceding section, Visual Basic was used to access and program a form-based interface for Hello. The Hello Application object was started as invisible and was later displayed when its Visible property was set to True. Some objects are not visible, and some objects are never displayed to the user. For example, a word processing application may expose its spelling checker engine as an object. This object supports a method called CheckWord that takes a string as an argument. If the string is spelled correctly, the method returns True; otherwise, the method returns False. If the string is spelled incorrectly, you could pass it to another (hypothetical) method called SuggestWord that returns a suggestion for its correct spelling. The code might look something like this:

```
Sub CheckSpelling ()
     Dim ObjVar As New SpellChecker
     Dim MyWord, Result

     MyWord = "potatoe"

     ' Check the spelling.
     Result = ObjVar.CheckWord MyWord

     ' If False, get suggestion.
     If Not Result Then
           MyWord = ObjVar.SuggestWord MyWord
     End If
End Sub
```

In this example, the spelling checker is never displayed to the user. Its capabilities are exposed through the properties and methods of the spelling checker object.

As shown in the example, you create and reference invisible objects the same way as any other type of object.

Activating an Object from a File

Many OLE Automation applications allow the user to save objects in files. For example, a spreadsheet application that supports Worksheet objects allows the user to save the worksheet in a file. The same application may also support a chart object that the user can save in a file. To activate an object that has been saved in a file, you use the **GetObject** function.

To activate an object from a file, first declare an object variable, then call the **GetObject** function using the following syntax:

GetObject **(*filename*[, *ProgID*])**

The *filename* argument is a string containing the full path and name of the file you want to activate. For example, an application named SPDSHEET.EXE creates an object that was saved in a file called REVENUE.SPD. The following code invokes SPDSHEET.EXE, loads the file REVENUE.SPD, and assigns REVENUE.SPD to an object variable:

```
Dim Ss As Spreadsheet
Set Ss = GetObject("C:\ACCOUNTS\REVENUE.SPD")
```

If the *filename* argument is omitted, **GetObject** returns the currently active object of the specified *ProgID*. For example:

```
Set Ss = GetObject (,"SpdSheet.Application")
```

If there is no active object of the class SpdSheet.Application, an error occurs.

In addition to activating an entire file, some applications let you activate part of a file. To activate part of a file, add an exclamation point (!) or a backslash (\) to the end of the filename, followed by a string that identifies the part of the file you want to activate. Refer to the object's documentation for information on how to create this string.

For example, if SPDSHEET.EXE is a spreadsheet application that uses R1C1 syntax, the following code could be used to activate a range of cells within REVENUE.SPD:

```
Set Ss = GetObject("C:\ACCOUNTS\REVENUE.SPD!R1C1:R10C20")
```

These examples invoke an application and activate an object. Notice that in these examples the application name (SPDSHEET.EXE) is never specified. When you use **GetObject** to activate an object, the registry files determine the application to invoke and the object to activate based on the filename or ProgID you provide. If you do not include a ProgID, OLE Automation activates the default object of the specified file.

Some OLE Automation servers, however, support more than one class of object. Suppose the spreadsheet file, REVENUE.SPD, supports three different classes of objects: an Application object, a Worksheet object, and a Toolbar object, all of which are part of the same file. To specify which object to activate, you must supply an argument for the optional *ProgID* parameter. For example:

```
Set Ss = GetObject("C:\REVENUE.SPD", "SPDSHEET.TOOLBAR")
```

This statement activates the Spdsheet.Toolbar object in the file REVENUE.SPD.

Accessing Linked and Embedded Objects

Some applications that supply objects support linking and embedding as well as OLE Automation. Using the OLE custom control (MSOLE2.VBX) from the OLE toolkit, you can create and display linked and embedded objects in a Visual Basic application. If the objects also support OLE Automation, you can access their properties and methods using the Object property. The Object property returns the object in the OLE control. This property refers to an OLE object in the same way an object variable created using the **New**, **CreateObject**, or **GetObject** functions refers to the object.

For example, an OLE control named Ole1 contains an object that supports OLE Automation. This object has an Insert method, a Select method, and a Bold property. In this case, you could write the following code to manipulate the OLE control's object:

```
' Insert text in the object.
Ole1.Object.Insert "Hello, world."
' Select the text.
Ole1.Object.Select
' Format the text as bold.
Ole1.Object.Bold = True
```

Manipulating Objects

Once you have created a variable that references an OLE object, the object can be manipulated in the same way as any other Visual Basic object. To get and set an object's properties or to perform an object's methods, you use the *object.property* or *object.method* syntax. You can include multiple objects, properties, and methods on the same line of code using the following syntax:

```
ObjVar.Cell(1,1).FontBold = True
```

Accessing an Object's Properties

To assign a value to a property of an object, put the object variable and property name on the left side of an assignment and the desired property setting on the right side. For example:

```
Dim ObjVar As IMyInterface
Dim RowPos, ColPos
Set ObjVar = New MyObject

ObjVar.Text = "Hello, world"
ObjVar.Cell(RowPos, ColPos) = "This property accepts two arguments."
' Sets the font for ObjVar.Selection.
ObjVar.Selection.Font = 12
```

You can also retrieve property values from an object:

```
Dim X As Object

X = ObjVar.Text
X = ObjVar.Range(12, 32)
```

Invoking Methods

In addition to getting and setting properties, you can manipulate an object using the methods it supports. Some methods may return a value, as in the following example:

```
X = ObjVar.Calculate(1,2,3)
```

Methods that do not return a value behave like subroutines. For example:

```
' This method requires two arguments.
ObjVar.Move XPos, YPos
```

If you assign such a method to a variable, an error occurs.

Creating Applications and Tools That Access Objects

OLE Automation provides interfaces for accessing exposed objects from an application or programming tool written in C or C++. The following sections show C++ code that uses the same type of access method as the Visual Basic code described earlier in this chapter. Although the process is more complicated than with Visual Basic, the approach is similar. Note, however, that this section shows the minimum code necessary to access and manipulate a remote object.

You can use the **IDispatch** interface to access OLE Automation objects, or you can access objects directly through the VTBL. Because VTBL references can be bound at compile time, VTBL access is generally faster than access through **IDispatch**. Whenever possible, you should use the **ITypeInfo** interface to get information about an object, including the VTBL addresses of the object's members. Then use this information to access OLE Automation objects through the VTBL.

To create compilers and other programming tools that use information from type libraries, you use the **ITypeComp** interface. This interface binds to exposed objects at compile time. For details on **ITypeComp**, see Chapter 8, "Type Description Interfaces."

Accessing Members through VTBLs

For objects that have dual interfaces, the first seven members of the VTBL are the members of **IUnknown** and **IDispatch**, and the subsequent members are standard OLE Component Object Model (COM) entries for the interface's member functions. You can call these entries directly from C++.

{ewl msdncd, EWGraphic, group10455 7 /a "SDK.BMP"} To access a method or property through the VTBL

1. Initialize OLE.
2. Create an instance of the exposed object.
3. Manipulate the properties and methods of the object.
4. Uninitialize OLE.

The code excerpt that follows shows how to access a property of the Hello object. Error handling has been omitted for brevity.

```
HRESULT hr;
CLSID clsid;                             //CLSID of Hello object
LPUNKNOWN punk = NULL;       //IUnknown of Hello object
IHello* phello = NULL;       //IHello interface of Hello object

//Initialize OLE.
hr = OleInitialize(NULL);

// Retrieve CLSID from the progID for Hello.
hr = CLSIDFromProgID("Hello.Application", &clsid);

// Create an instance of the Hello object and ask for its
// IDispatch interface.
hr = CoCreateInstance(clsid, NULL, CLSCTX_SERVER,
                                   IID_IUnknown, (void FAR* FAR*)&punk);

hr = punk->QueryInterface(IID_IHello, (void FAR* FAR*)&pHello);

punk->Release();       //Release when no longer needed.

hr = pHello->put_Visible (TRUE);

//Additional code to work with other methods and properties.
// .
// .
// .

OleUninitialize();
```

The example initializes OLE, then calls the **CLSIDFromProgID** function to obtain the CLSID for the Hello application. With the CLSID, the example can call **CoCreateInstance** to create an instance of the Hello Application object. **CoCreateInstance** returns a pointer to the object's IUnknown interface (`punk`), and this, in turn, is used to call **QueryInterface** to get `pHello`, a pointer to the IID_IHello dual interface. The `punk` is no longer needed, so the example releases it. The example then sets the value of the Visible property to True.

If the function returns an error HRESULT, you can get detailed, contextual information through the **IErrorInfo** interface.

Accessing Members through IDispatch

To bind to exposed objects at run time, you use **IDispatch**.

{ewl msdncd, EWGraphic, group10455 8 /a "SDK.BMP"} To create an OLE Automation controller using IDispatch

1. Initialize OLE.
2. Create an instance of the object you want to access. The object's OLE Automation server creates the object.
3. Obtain a reference to the object's **IDispatch** interface (if it has implemented one).
4. Manipulate the object through the methods and properties exposed in its **IDispatch** interface.
5. Terminate the object by invoking the appropriate method in its **IDispatch** interface or by releasing all references to the object.
6. Uninitialize OLE.

The following table shows minimum set of functions necessary to manipulate a remote object.

Function	Purpose	Interface
OleInitialize	Initializes OLE	OLE API function
CoCreateInstance	Creates an instance of the class represented by the specified class ID, and returns a pointer to the object's **IUnknown** interface.	Component object API function
QueryInterface	Checks whether **IDispatch** has been implemented for the object. If so, returns a pointer to the IDispatch implementation.	IUnknown
GetIDsOfNames	Returns DISPIDs for properties and methods and their parameters.	IDispatch
Invoke	Invokes a method, or sets or gets a property of the remote object.	IDispatch
Release	Decrements the reference count for an **IUnknown** or **IDispatch** object.	IUnknown
OleUninitialize	Uninitializes OLE.	OLE API function

The code that follows is extracted from is a generalized Windows-based OLE Automation controller for the Hello sample. The controller relies on helper functions provided in INVHELP.CPP, available in \OLE2\SAMPLE\BROWSE. Error checking is omitted to save space, but would normally be used where an HRESULT is returned.

The two functions that follow initialize OLE, then create an instance of an object and get a pointer to the object's **IDispatch** interface (INVHELP.CPP):

```
BOOL InitOle(void)
{
     if(OleInitialize(NULL) != 0)
           return FALSE;

     return TRUE;
}

HRESULT CreateObject(LPSTR pszProgID, IDispatch FAR* FAR* ppdisp)
{
     CLSID clsid;                              // CLSID of OLE Automation
object
     HRESULT hr;
     LPUNKNOWN punk = NULL;         // IUnknown of OLE Automation object
     LPDISPATCH pdisp = NULL;             // IDispatch of OLE Automation
object

     *ppdisp = NULL;

     // Retrieve CLSID from the progID that the user specified.
     hr = CLSIDFromProgID(pszProgID, &clsid);
     if (FAILED(hr))
           goto error;

     // Create an instance of the OLE Automation object and ask for the
     // IDispatch interface.
     hr = CoCreateInstance(clsid, NULL, CLSCTX_SERVER,
                                        IID_IUnknown, (void FAR*
FAR*)&punk);
     if (FAILED(hr))
           goto error;

     hr = punk->QueryInterface(IID_IDispatch, (void FAR* FAR*)&pdisp);
     if (FAILED(hr))
           goto error;

     *ppdisp = pdisp;
     punk->Release();
     return NOERROR;

error:
     if (punk) punk->Release();
     if (pdisp) pdisp->Release();
     return hr;
}
```

The CreateObject function is passed a ProgID and returns a pointer to the **IDispatch** implementation of the specified object. CreateObject calls the OLE API **CLSIDFromProgID** to get the CLSID that corresponds to the requested object, then passes the CLSID to **CoCreateInstance** to create an instance of the object and get a pointer to the object's **IUnknown** interface. (The **CLSIDFromProgID** function is described in the *OLE 2 Programmer's Reference, Volume 1*). With this pointer, CreateObject calls **IUnknown::QueryInterface**, specifying IID_IDispatch, to get a pointer to the object's **IDispatch** interface.

```
HRESULT FAR
Invoke(LPDISPATCH pdisp,
     WORD wFlags,
     LPVARIANT pvRet,
     EXCEPINFO FAR* pexcepinfo,
     UINT FAR* pnArgErr,
     LPSTR pszName,
     char *pszFmt,
     ...)
{
     va_list argList;
     va_start(argList, pszFmt);
     DISPID dispid;
     HRESULT hr;
     VARIANTARG* pvarg = NULL;

     if (pdisp == NULL)
           return ResultFromScode(E_INVALIDARG);

     // Get DISPID of property/method
     hr = pdisp->GetIDsOfNames(IID_NULL, &pszName, 1,
           LOCALE_SYSTEM_DEFAULT, &dispid);
     if(FAILED(hr))
           return hr;

     DISPPARAMS dispparams;
     _fmemset(&dispparams, 0, sizeof dispparams);

     // Determine number of arguments.
     if (pszFmt != NULL)
           CountArgsInFormat(pszFmt, &dispparams.cArgs);

     // Property puts have a named argument that represents the value
     // being assigned to the property.
     DISPID dispidNamed = DISPID_PROPERTYPUT;
     if (wFlags & DISPATCH_PROPERTYPUT)
     {
           if (dispparams.cArgs == 0)
                 return ResultFromScode(E_INVALIDARG);
           dispparams.cNamedArgs = 1;
           dispparams.rgdispidNamedArgs = &dispidNamed;
     }

     if (dispparams.cArgs != 0)
     {
           // Allocate memory for all VARIANTARG parameters.
           pvarg = new VARIANTARG[dispparams.cArgs];
           if(pvarg == NULL)
                 return ResultFromScode(E_OUTOFMEMORY);
           dispparams.rgvarg = pvarg;
           _fmemset(pvarg, 0, sizeof(VARIANTARG) * dispparams.cArgs);

           // Get ready to walk vararg list.
           LPSTR psz = pszFmt;
           pvarg += dispparams.cArgs - 1;   // Params go in opposite order.

           while (psz = GetNextVarType(psz, &pvarg->vt))
           {
```

```
                 if (pvarg < dispparams.rgvarg)
                 {
                       hr = ResultFromScode(E_INVALIDARG);
                       goto cleanup;
                 }
                 switch (pvarg->vt)
                 {
                 case VT_I2:
                       V_I2(pvarg) = va_arg(argList, short);
                       break;
                 case VT_I4:
                       V_I4(pvarg) = va_arg(argList, long);
                       break;
                 // Additional cases omitted to save space...
                 default:
                       {
                             hr = ResultFromScode(E_INVALIDARG);
                             goto cleanup;
                       }
                       break;
                 }
                 --pvarg; // Get ready to fill next argument.
           } //while
     } //if

     // Initialize return variant, in case caller forgot. Caller can pass
     // NULL if no return value is expected.
     if (pvRet)
           VariantInit(pvRet);
     // Make the call.
     hr = pdisp->Invoke(dispid, IID_NULL, LOCALE_SYSTEM_DEFAULT, wFlags,
           &dispparams, pvRet, pexcepinfo, pnArgErr);

cleanup:
     // Clean up any arguments that need it.
     if (dispparams.cArgs != 0)
     {
           VARIANTARG FAR* pvarg = dispparams.rgvarg;
           UINT cArgs = dispparams.cArgs;
           while (cArgs--)
           {
                 switch (pvarg->vt)
                 {
                 case VT_BSTR:
                       VariantClear(pvarg);
                       break;
                 }
                 ++pvarg;
           }
     }
     delete dispparams.rgvarg;
     va_end(argList);
     return hr;
}
```

The Invoke function in the example is a general-purpose function that calls **IDispatch::Invoke** to invoke a property or method of an OLE Automation object. As arguments, it accepts the object's IDispatch implementation, the name of the member to invoke, flags that control the invocation, and a variable list of the member's arguments.

Using the object's IDispatch implementation and the name of the member, it calls **GetIDsOfNames** to get the DISPID of the requested member. The member's DISPID must be used later, in the call to **IDispatch::Invoke**.

The invocation flags specify whether a method, property put, or property get function is being invoked. The helper function simply passes these flags directly to **IDispatch::Invoke**.

The helper function next fills in the DISPPARAMS structure with the parameters of the member. DISPPARAMS structures have the following form:

```
typedef struct FARSTRUCT tagDISPPARAMS{
     VARIANTARG FAR* rgvarg;                   // Array of arguments
     DISPID FAR* rgdispidNamedArgs;      // Dispids of named arguments
     UINT cArgs;                                     // Number of arguments
     UINT cNamedArgs;                    // Number of named arguments
} DISPPARAMS;
```

The *rgvarg* field is a pointer to an array of VARIANTARG structures. Each element of the array specifies an argument, whose position in the array corresponds to its position in the parameter list of the method definition. The *cArgs* field specifies the total number of arguments, and the *cNamedArgs* field specifies the number of named arguments. For methods and property get functions,   all arguments may be accessed as positional, or they may be accessed as named arguments. Property put functions have a named argument that is the new value for the property. The DISPID of this argument is DISPID_PROPERTYPUT.

To build the *rgvarg* array, the Invoke helper function retrieves the parameter values and types from its own argument list, and constructs a VARIANTARG structure for each one. (See INVHELP.CPP for a description of the format string that specifies the types of the parameters.) Parameters are put in the array in reverse order, so that the last parameter is in *rgvarg*[0] and so forth. Although VARIANTARG has the following five fields, only the first and fifth are used:

```
typedef struct FARSTRUCT tagVARIANT VARIANTARG;

struct FARSTRUCT tagVARIANT{
     VARTYPE vt;
     unsigned short wReserved1;
     unsigned short wReserved2;
     unsigned short wReserved3;
     union {
           short       iVal;       /* VT_I2          */
.
.    // The rest of this union specifies numerous other types.
.

     };
} VARIANTARG;
```

The first field contains the argument's type, and the fifth contains its value. To pass a long integer, for example, the *vt* and *iVal* fields of the VARIANTARG structure would be filled with VT_I4 (long integer) and the actual value of the long integer.

In addition, for property put functions, the first element of the *rgdispidNamedArgs* array must contain DISPID_PROPERTYPUT.

After filling the DISPPARAMS structure, the Invoke helper function initializes *pvRet*, a VARIANT in which **IDispatch::Invoke** returns a value from the method or property. The following is the actual call to **IDispatch::Invoke**:

```
hr = pdisp->Invoke(dispid, IID_NULL, LOCALE_SYSTEM_DEFAULT, wFlags,
            &dispparams, pvRet, pexcepinfo, pnArgErr);
```

The variable `pdisp` is a pointer to the object's IDispatch interface. The `dispid` is the DISPID of the method or property being invoked. The value IID_NULL must be specified for all **IDispatch::Invoke** calls, and LOCALE_SYSTEM_DEFAULT is a constant denoting the default LCID for this system. The `wFlags`, `dispparams`, and `pvRet` arguments have already been discussed. In the final two arguments, `pexcepinfo` and `pnArgErr`, **IDispatch::Invoke** can return error information. If the invoked member has defined an exception handler, it returns exception information in `pexcepinfo`. If certain errors occur in the argument vector, `pnArgErr` points to the errant argument. The function return value `hr` is an HRESULT that indicates success or various types of failure.

For more information, including how to pass optional arguments, see "IDispatch::Invoke" in Chapter 5, "Dispatch Interfaces."

Creating Type Information Browsers

Type information browsers allow you and your customers to scan type libraries to determine what types of objects are available. The following figure shows the interfaces you use when you create compilers or browsers that access type libraries.

{ewc msdncd, EWGraphic, group10455 9 /a "SDK_03.bmp"}

The Browse sample in \OLE2\SAMPLE\BROWSE shows how a browser might access a type library. The Browse sample is a Windows-based type browser that presents a dialog box from which you can select the type information items to display. This function prompts you for the name of the type library, opens the library, and gathers and displays information.

Standards and Guidelines

This chapter describes the standard OLE Automation objects and discusses naming guidelines for creating objects that are unique to applications, especially user-interactive applications that support a multiple-document interface (MDI). However, if your OLE Automation object is not user interactive or supports only a single-document interface (SDI), you should adapt the standards and guidelines as appropriate.

- *Standard objects* comprise a standard set of objects defined by OLE. They should be used as appropriate to your application. The objects described in this chapter are oriented toward document-based, user-interactive applications. Other applications (such as noninteractive database servers) may have different requirements.
- *Naming guidelines* are recommendations meant to improve consistency across applications.

This chapter covers each of these points in turn. Examples are shown in a hypothetical syntax derived from Visual Basic.

Note These standards and guidelines are subject to change.

Standard Objects

The following table lists the OLE Automation standard objects. Although none of these objects is required, user-interactive applications with subordinate objects should include an Application object.

Object name	Description
Application	Top-level object; provides a standard way for OLE Automation controllers to retrieve and navigate an application's subordinate objects.
Document	Provides a way to open, print, change, and save an application document.
Documents	Provides a way to iterate over and select open documents in multiple-document interface (MDI) applications.
Font	Describes fonts that are used to display or print text.

The following figure shows how the standard objects fit into the organization of the objects an application provides.

{ewc msdncd, EWGraphic, group10456 0 /a "SDK_01.bmp"}

The following sections describe the standard properties and methods for all objects, all collection objects, and each of the standard objects. These sections list only the standard methods and properties for each object, as well as the standard arguments for those properties and methods.

Note You may define additional application-specific properties and methods for each object. You may also provide additional optional arguments for any of the listed properties or methods; however, the optional arguments should follow the standard arguments in a positional argument list.

All Objects

All objects, including the Application object and collection objects, must provide the following properties:

Property name	Return type	Description
Application	VT_DISPATCH	Returns the Application object; read only.
Parent	VT_DISPATCH	Returns the creator of the object; read only.

Note The Application and Parent properties of the Application object return the Application object.

All Collection Objects

A collection provides a set of objects over which iteration can be performed. All collection objects must provide the following properties:

Property name	Return type	Description
Count	VT_I4	Returns the number of items in the collection; read only. Required.
_NewEnum	VT_DISPATCH	A special property that returns an enumerator object that implements **IEnumVARIANT**. Required.

Collection Methods

Methods for collections are described in the following table. The Item method is required; other methods are optional.

Method name	Return type	Description
Add	VT_DISPATCH or VT_EMPTY	Adds an item to a collection. Returns VT_DISPATCH if object is created (object can't exist outside the collection) or VT_EMPTY if no object is created (object can exist outside the collection).
Item	Varies with type of collection	Returns the indicated item in the collection. Required. The Item method may take one or more arguments to indicate the element within the collection to return. This method is the default member (DISPID_VALUE) for the collection object.
Remove	VT_EMPTY	Removes an item from a collection. Uses indexing arguments in the same way as the Item method.

All collection objects must provide at least one form of indexing through the Item method. The dispatch ID of the Item method is DISPID_VALUE. Because it is the default member, it can be used in the following convenient form:

```
ThirdDef = MyWords(3).Definition   ' Equivalent to
                                                  '

MyWords.Item(3).Definition
```

The Item method takes one or more arguments to indicate the index. Indexes may be numbers, strings, or other types. For example:

```
DogDef = MyWords("dog").Definition
```

Important Within the application's type library, the _NewEnum property has a special dispatch ID: DISPID_NEWENUM. The name "_NewEnum" should not be localized.

The Add method may take one or more arguments. For example, if MyWord is an object with the properties Letters and Definition:

```
Dim MyWord As New Word
Dim MyDictionary as Words
MyWord = "dog"
MyWord.Letters = "Dog"
MyWord.Definition = "My best friend."
MyDictionary.Add MyWord
MyDictionary.Remove("Dog")
```

For more information on creating collection objects, see Chapter 2, "Exposing OLE Automation Objects."

Kinds of Collections

The standard for collections allows you to describe two kinds of collections, depending on whether it makes sense for the collected objects to exist outside the collection.

In some cases, it isn't logical for an object to exist independently of its collection. For example, an application's Documents collection contains all currently open Document objects. Opening a document means adding it to the collection, and closing the document means removing it from the collection. All open documents are, therefore, part of the collection; the application can't have open documents that aren't part of the collection. The relationship between the collection and the members of the collection can be shown in the following ways:

- Documents.Add creates an object (an open document) and adds it to the collection. Since an object is created, a reference to it is returned:

  ```
  Set MyDoc = Documents.Add
  ```

- Document.Close removes an object from the collection:

  ```
  Set SomeDoc = Documents(3)
  SomeDoc.Close
  ```

In other cases, it is logical for the objects to exist outside the collection. For example, a Mail application might have Name objects, and many collections of these Name objects. Each Name object would have a user's email name, full name, and possibly other information. The email name and full name would likely be properties named EmailName and FullName.

Additionally, the application might have the following collections of Name objects:

- A collection for the "to" list on each piece of mail
- A collection of the names of all of the people to whom a user has sent mail

The collections of Name objects could be indexed by using either EmailName or FullName.

For these collections, the Add method doesn't create an object because the object already exists. Therefore, the Add method should take an object as an argument and should not return a value. Assuming the existence of two collections (AddressBook and ToList), a user might execute the following code to add a Name object to the ToList collection:

```
Dim Message as Object
Dim AddressBook as Object
Dim NameRef as Object
.
.
.

Set NameRef = AddressBook.Names("Fred Funk")
Message.ToList.Add   NameRef
```

The Name object already exists and is contained in the AddressBook collection. The first line of code gets a reference to the Name object for "Fred Funk" and makes NameRef point to it. The second line of code adds a reference to the object to the ToList collection. No new object is created, so no reference is returned from the Add method. Unlike the relationship between Documents and Document, there is no way for the collected object (the Name) to know how to remove itself from the collections in which it is contained. Thus, to remove an item from a collection, the Remove method is used:

```
Message.ToList.Remove("Fred Funk")
```

This line of code removes the Name object that has the FullName "Fred Funk." The "Fred Funk" object may exist in other collections. These other collections will be unaffected.

Application Object Properties

If a type library is used, the Application object should be the object that has the **appobj** attribute. Because some OLE Automation controllers use the type information to allow unqualified access to the Application object's members, it is important to avoid overloading the Application object with too many members.

The Application object should have the properties listed in the following table. The Application, FullName, Name, Parent, and Visible properties are required; other properties are optional.

Property name	Return type	Description
ActiveDocument	VT_DISPATCH, VT_EMPTY	Returns the active document object or VT_EMPTY if none; read only.
Application	VT_DISPATCH	Returns the Application object; read only. Required.
Caption	VT_BSTR	Sets or returns the title of the application window; read/write. Setting the Caption to VT_EMPTY returns control to the application.
DefaultFilePath	VT_BSTR	Sets or returns the default path specification used by the application for opening files; read/write.
Documents	VT_DISPATCH	Returns a collection object for the open documents; read only.
FullName	VT_BSTR	Returns the file specification for the application, including path; read only. For example, C:\DRAWDIR\SCRIBBLE. Required.
Height	VT_R4	Sets or returns the distance between the top and bottom edge of the main application window; read/write.
Interactive	VT_BOOL	Sets or returns True if the application accepts actions from the user, and False otherwise; read/write.
Left	VT_R4	Sets or returns the distance between the left edge of the physical screen and the main application window; read/write.
Name	VT_BSTR	Returns the name of the application, such as "Microsoft Excel"; read only. The Name property is the default member (DISPID_VALUE) for the Application object. Required.
Parent	VT_DISPATCH	Returns the Application object;

		read only. Required.
Path	VT_BSTR	Returns the path specification for the application's executable file; read only. For example, C:\DRAWDIR if the executable is C:\DRAWDIR\SCRIBBLE.EXE.
StatusBar	VT_BSTR	Sets or returns the text displayed in the status bar; read/write.
Top	VT_R4	Sets or returns the distance between the top edge of the physical screen and main application window; read/write.
Visible	VT_BOOL	Sets or returns whether the application is visible to the user; read/write. The default is False when the application is started with **/Automation** command-line switch. Required.
Width	VT_R4	Sets or returns the distance between the left and right edges of the main application window; read/write.

The Application object should have the following methods. The Quit method is required; other methods are optional.

Method name	**Return type**	**Description**
Help	VT_EMPTY	Displays online Help. May take three optional arguments: *helpfile* (VT_BSTR), *helpcontextID* (VT_I4), and *helpstring* (VT_BSTR). The *helpfile* argument specifies the Help file to display; if omitted, the main Help file for the application is displayed. The *helpcontextID* and *helpstring* arguments specify a Help context to display, and only one of them may be supplied; if both are omitted, the default Help topic is displayed.
Quit	VT_EMPTY	Exits the application and closes all open documents. Required.
Repeat	VT_EMPTY	Repeats the previous action in the user interface.
Undo	VT_EMPTY	Reverses the previous action in the user interface.

Document Object Properties

If your application is document-based, it should provide a Document object named "Document." Use a different name only if "Document" is inappropriate, for example, if your application uses highly technical or otherwise specialized terminology within its user interface.

The Document object should have the properties listed in the table that follows. The Application, FullName, Name, Parent, Path, and Saved properties are required; other properties are optional.

Property name	Return type	Description
Application	VT_DISPATCH	Returns the Application object; read only. Required.
Author	VT_BSTR	Sets or returns the summary information about the document's author; read/write.
Comments	VT_BSTR	Sets or returns summary information comments for the document; read/write.
FullName	VT_BSTR	Returns the file specification of the document, including path; read only. Required.
Keywords	VT_BSTR	Sets or returns summary information keywords associated with the document; read/write.
Name	VT_BSTR	Returns the filename of the document, not including the file's path specification; read only.
Parent	VT_DISPATCH	Returns the parent of the Document object; read only. Required.
Path	VT_BSTR	Returns the path specification for the document, not including the filename or filename extension; read only. Required.
ReadOnly	VT_BOOL	Returns True if the file is read only and False otherwise; read only.
Saved	VT_BOOL	Returns True if the document has never been saved, but has not changed since it was created. Returns True if it has been saved and has not changed since last saved. Returns False if it has never been saved and has changed since it was created; or if it was saved, but has changed since last saved. Read only; required.
Subject	VT_BSTR	Sets or returns the summary information about the subject of the document; read/write.
Title	VT_BSTR	Sets or returns the summary information about the title of the document; read/write.

The Document object should have the following methods. The Activate, Close, Print, Save, and SaveAs methods are required; other methods are optional.

Method	Return type	Description

name		
Activate	VT_EMPTY	Activates the first window associated with the document. Required.
Close	VT_EMPTY	Closes all windows associated with the document and removes the document from the Documents collection. Required. Takes two optional arguments, *saveChanges* (VT_BOOL) and *fileName* (VT_BSTR). The *fileName* argument specifies the name of the file in which to save the document.
NewWindow	VT_EMPTY	Creates a new window for the document.
Print	VT_EMPTY	Prints the document. Required. Takes three optional arguments: *from* (VT_I2), *to* (VT_I2), and *copies* (VT_I2). The *from* and *to* arguments specify the page range to print. The *copies* argument specifies the number of copies to print.
PrintOut	VT_EMPTY	Same as Print method, but provides an easier way to use the method in Visual Basic version 3.0, because Print is a Visual Basic keyword.
PrintPreview	VT_EMPTY	Previews the pages and page breaks of the document. Equivalent to choosing Print Preview from the File menu.
RevertToSaved	VT_EMPTY	Reverts to the last saved copy of the document, discarding any changes.
Save	VT_EMPTY	Saves changes to the file specified in the document's FullName property. Required.
SaveAs	VT_EMPTY	Saves changes to a file. Required. Takes one optional argument, *filename* (VT_BSTR). The *filename* argument may optionally include a path specification.

Documents Collection Object Properties

If your application supports a multiple-document interface (MDI), you should provide a Documents collection object. Use the name "Documents" for this collection unless the name is inappropriate for your application.

The Documents collection object should have all of the following properties.

Property name	Return type	Description
Application	VT_DISPATCH	Returns the Application object; read only. Required.
Count	VT_I4	Returns the number of items in the collection; read only. Required.
_NewEnum	VT_DISPATCH	A special property that returns an enumerator object that implements **IEnumVARIANT**. Required.
Parent	VT_DISPATCH	Returns the parent of the Documents collection object; read only. Required.

The Documents collection object should have all of the following methods.

Method name	Return type	Description
Add	VT_DISPATCH	Creates a new document and adds it to the collection. Returns the document that was created. Required.
Close	VT_EMPTY	Closes all documents in the collection. Required.
Item	VT_DISPATCH or VT_EMPTY	Returns a Document object from the collection or returns VT_EMPTY if the document does not exist. Takes an optional argument, *index*, which may be a string (VT_BSTR) indicating the document name, a number (VT_I4) indicating the ordered position within the collection, or either (VT_VARIANT). If *index* is omitted, returns the Document collection. The Item method is the default member (DISPID_VALUE). Required.
Open	VT_DISPATCH or VT_EMPTY	Opens an existing document and adds it to the collection. Returns the document that was opened or VT_EMPTY if the object could not be opened. Takes one required argument, *filename*, and one optional argument, *password*. Both arguments have type VT_BSTR. Required.

Font Object Properties

The Font object may be appropriate for some applications. It should have the following properties. The Application, Bold, Italic, Parent, and Size properties are required; other properties are optional.

Property name	Return type	Description
Application	VT_DISPATCH	Returns the Application object; read only. Required.
Bold	VT_BOOL	Sets or returns True if the font is boldface, and False otherwise; read/write. Required.
Color	VT_I4	Sets or returns the RGB color of the font; read/write.
Italic	VT_BOOL	Sets or returns True if the font is italic, and False otherwise; read/write. Required.
Name	VT_BSTR	Returns the name of the font; read only.
OutlineFont	VT_BOOL	Sets or returns True if the font is scaleable, and False otherwise. For example, bitmapped fonts are not scaleable, whereas TrueType fonts are scaleable; read/write.
Parent	VT_DISPATCH	Returns the parent of the Font object; read only. Required.
Shadow	VT_BOOL	Sets or returns True if the font appears with a shadow, and False otherwise; read/write.
Size	VT_R4	Sets or returns the size of the font in points; read/write. Required.
Strikethrough	VT_BOOL	Sets or returns True if the font appears with a line running through it, and False otherwise; read/write.
Subscript	VT_BOOL	Sets or returns True if the font is subscripted, and False otherwise; read/write.
Superscript	VT_BOOL	Sets or returns True if the font is superscripted and False otherwise; read/write.

Storage Object Naming Conventions

Choose names for exposed objects, properties, and methods that can be easily understood by the users of your application. The guidelines in this section apply to all the items you expose:

- Objects (implemented as classes in your application)
- Properties and methods (implemented as members of a class)
- Named arguments (implemented as named parameters in a member function)
- Constants and enumerations (implemented as settings for properties and methods)

Use entire words or syllables

It is easier for users to remember complete words than to remember whether you abbreviated Window as Wind, Wn, or Wnd.

When you need to abbreviate because an identifier would be too long, try to use complete initial syllables. For example, use AltExpEval instead of AlternateExpressionEvaluation.

Use	Don't use
Application	App
Window	Wnd

Use mixed case

All identifiers should use mixed case, rather than underscores, to separate words.

Use	Don't use
ShortcutMenus	Shortcut_Menus, Shortcutmenus, SHORTCUTMENUS, SHORTCUT_MENUS
BasedOn	basedOn

Use the same word you use in the interface

Use consistent terminology; don't use names like HWND that are based on Hungarian notation. Try to use the same word your users would use to describe a concept.

Use	Don't use
Name	Lbl

Use the correct plural for the class name

Collection classes should use the correct plural for the class name. For example, if you have a class named Axis, you should store the collection of Axis objects in an Axes class. Similarly, a collection of Vertex objects is stored in a Vertices class. In cases where English uses the same word for the plural, append the word "Collection."

Use	Don't use
Axes	Axiss
SeriesCollection	CollectionSeries
Windows	ColWindow

Using plurals rather than inventing new names for collections reduces the number of items a user must remember. It also simplifies the selection of names for collections.

Note, however, that for some collections this may not be appropriate, especially where a set of objects exists independently of the collection. For example, a Mail program might have a Name object that exists in several collections, such as ToList, CCList, and GroupList. In this case, you might specify the individual name collections as ToNames, CCNames, and GroupNames.

Dispatch Interfaces

The dispatch interfaces provide a way to expose and access objects within an application. OLE Automation defines the following dispatch interfaces and functions:

- **IDispatch** interface – Exposes objects, methods, and properties to OLE Automation programming tools and other applications.
- Dispatch functions – Simplify the implementation of an **IDispatch** interface. You can use these functions to automatically generate an **IDispatch** interface.
- **IEnumVARIANT** interface – Provides a way for OLE Automation controllers to iterate over collection objects.

Overview of Dispatch Interfaces

The following table describes the member functions of each of the dispatch interfaces.

Interface	Member name	Purpose
IDispatch	**Invoke**	Provides access to properties and methods exposed by the object.
	GetIDsOfNames	Maps a single member name and an optional set of argument names to a corresponding set of integer DISPIDs, which may then be used on subsequent calls to **Invoke**.
	GetTypeInfo	Retrieves the type information for an object.
	GetTypeInfoCount	Retrieves the number of type information interfaces that an object provides (either 0 or 1).
IEnumVARIANT	**Clone**	Creates a copy of the current enumeration state.
	Next	Gets the next item or items in the enumeration sequence and returns them through an array.
	Reset	Resets the enumeration sequence to the beginning.
	Skip	Skips over the next item or items in the enumeration sequence.

Overview of Dispatch Functions

The dispatch functions are summarized in the following table. For 32-bit systems, these functions are provided in OLEAUT32.DLL; the header file is OLEAUTO.H, and the import library is OLEAUT32.LIB. For 16-bit systems, these functions are provided in OLE2DISP.DLL; the header file is DISPATCH.H, and the import library is OLE2DISP.LIB.

Category	Function name	Purpose
Dispatch interface creation	**CreateDispTypeInfo**	Creates simplified type information for an object.
	CreateStdDispatch	Creates a standard **IDispatch** implementation for an object.
	DispGetIDsOfNames	Converts a set of names to DISPIDs.
	DispGetParam	Retrieves and coerces elements from a DISPPARAMS structure.
	DispInvoke	Calls a member function of an **IDispatch** interface.
Active object initialization	**GetActiveObject**	Retrieves an instance of an object that is initialized with OLE.
	RegisterActiveObject	Initializes a running object with OLE. (Use when application starts.)
	RevokeActiveObject	Revokes a running application's initialization with OLE. (Use when application ends.)

IDispatch Interface

Implemented by	Used by	Header filename
Applications that expose programmable objects.	Applications that access programmable objects.	OLEAUTO.H (32-bit systems) DISPATCH.H (16-bit systems)

OLE Automation objects may implement the **IDispatch** interface for access by OLE Automation controllers, such as Visual Basic. The object's properties and methods can be accessed using **IDispatch::GetIDsOfNames** and **IDispatch::Invoke**.

The following examples show how to access an OLE Automation object through the **IDispatch** interface. Note that the code is abbreviated and omits error handling.

```
// Declarations of variables used
     DEFINE_GUID(CLSID_Hello,            //...portions omitted for
brevity...

     HRESULT hresult;
     IUnknown * punk;
     IDispatch * pdisp;
     OLECHAR FAR* szMember = "SayHello";
     DISPID dispid;
     DISPPARAMS dispparamsNoArgs = {NULL, NULL, 0, 0};
     EXCEPINFO excepinfo;
     UINT nArgErr;
```

In the following lines, **OleInitialize** loads the OLE DLLs and **CoCreateInstance** initializes the OLE Automation object's class factory. For more information on these two functions, see the *OLE 2 Programmer's Reference, Volume 1*.

```
// Initialize OLE DLLs
hresult = OleInitialize(NULL);

// OLE function CoCreateInstance starts application using GUID
hresult = CoCreateInstance(CLSID_Hello, &punk);
```

QueryInterface checks whether the object supports **IDispatch**. (As with any call to **QueryInterface**, the returned pointer must be released when it is no longer needed.)

```
// QueryInterface to see if it supports IDispatch.
hresult = punk->QueryInterface(IID_IDispatch, &pdisp);
```

GetIDsOfNames retrieves the dispatch identifier (DISPID) for the indicated method or property, in this case, `szMember`.

```
// Retrieve the dispatch identifier for the SayHello method.
// Use defaults where possible.
hresult = pdisp->GetIDsOfNames(
                             IID_NULL,
                             &szMember,
                             1,
                             LOCALE_SYSTEM_DEFAULT,
                             &dispid);
```

In the following call to **Invoke**, the dispatch identifier (`dispid`) indicates the property or method to invoke. The **SayHello** method takes no parameters, so the fifth argument (`&dispparamsNoArgs`*)*, contains NULL and 0, as initialized at declaration. To invoke a property or method that requires parameters, you would need to supply the parameters in the DISPPARAMS structure.

```
// Invoke the method. Use defaults where possible.
hresult = pdisp->Invoke(
                             dispid,
                             IID_NULL,
                             LOCALE_SYSTEM_DEFAULT,
                             DISPATCH_METHOD,
                             &dispparamsNoArgs,
                             NULL,
                             NULL,
                             NULL);
```

Data Types, Structures, and Enumerations

The **IDispatch** interface uses these data types and structures:

Name	Purpose
BSTR	A length-prefixed string.
CALLCONV	Identifies the calling convention used by a member function.
CURRENCY	Provides a precise data type of monetary data.
DISPID	Identifies a method, property, or argument to **Invoke**.
DISPPARAMS	Contains arguments passed to a method or property.
EXCEPINFO	Describes an error that occurred during **Invoke**.
INTERFACEDATA	Describes the members of an interface.
LCID	Provides locale information for international string comparisons and localized member names.
METHODDATA	Describes a method or property.
PARAMDATA	Describes a parameter to a method.
VARIANT	Describes a VARIANTARG that can't have the VT_BYREF bit set. Because VT_BYREF is not set, data of type VARIANT can't be passed within DISPPARAMS.
VARIANTARG	Describes arguments that may be passed within DISPPARAMS.
VARTYPE	Identifies the available variant types.

BSTR

A length-prefixed string used by OLE Automation data manipulation functions.

```
typedef OLECHAR *BSTR;
```

For details on this data type, see Chapter 6, "Data Manipulation Functions."

CALLCONV

Identifies the calling convention used by a member function described in METHODDATA.

```
typedef enum tagCALLCONV {
      CC_CDECL = 1
    , CC_MSCPASCAL
    , CC_PASCAL = CC_MSCPASCAL
    , CC_MACPASCAL
    , CC_STDCALL
    , CC_RESERVED
    , CC_SYSCALL
    , CC_MPWCDECL
    , CC_MPWPASCAL
    , CC_MAX                 /* end of enum marker */
} CALLCONV;
```

On 16-bit Windows systems, functions implemented with the CC_CDECL calling convention can't have return type **float** or **double**. This includes functions returning DATE, which is a floating-point type.

CURRENCY

A currency number is stored as an 8-byte, two's complement integer, scaled by 10,000 to give a fixed-point number with 15 digits to the left of the decimal point and 4 digits to the right. This representation provides a range of ±922337203685477.5807. The currency data type is useful for calculations involving money, or for any fixed-point calculation where accuracy is particularly important.

```
typedef CY CURRENCY;
```

The data type is defined as a structure for working with currency more conveniently:

```
typedef struct tagCY
     {
     unsigned long Lo;
     long Hi;
     }
CY;
```

DISPID

Used by **IDispatch::Invoke** to identify methods, properties, and arguments.

```
typedef LONG DISPID;
```

The following DISPIDs have special meaning:

DISPID	Description
DISPID_VALUE	The default member for the object. This property or method is invoked when an OLE Automation controller specifies the object name without a property or method.
DISPID_NEWENUM	The _NewEnum property. This special, restricted property is required for collection objects. It returns an enumerator object that supports **IEnumVariant** and should have the **restricted** attribute specified in ODL.
DISPID_EVALUATE	The Evaluate method. This method is implicitly invoked when the OLE Automation controller encloses the arguments in square brackets. For example, the following two lines are equivalent: x.[A1:C1].value = 10 x.Evaluate("A1:C1").value = 10 The Evaluate method has the dispatch ID DISPID_EVALUATE.
DISPID_PROPERTYPUT	The parameter that receives the value of an assignment in a property "put."
DISPID_CONSTRUCTOR	The C++ constructor function for the object.
DISPID_DESTRUCTOR	The C++ destructor function for the object.
DISPID_UNKNOWN	Value returned by **IDispatch::GetIDsOfNames** to indicate that a member or parameter name was not found.

DISPPARAMS

Used by **IDispatch::Invoke** to contain the arguments passed to a method or property. For more information, see the reference page for **IDispatch::Invoke** later in this chapter.

```
typedef struct FARSTRUCT tagDISPPARAMS{
VARIANTARG FAR* rgvarg;                   // Array of arguments
     DISPID FAR* rgdispidNamedArgs;       // Dispatch IDs of named arguments
     unsigned int cArgs;                        // Number of arguments
     unsigned int cNamedArgs;                   // Number of named arguments
} DISPPARAMS;
```

EXCEPINFO

Describes an exception that occurred during **IDispatch::Invoke**. See the reference page for "**IDispatch::Invoke**" for more information on exceptions.

```
typedef struct FARSTRUCT tagEXCEPINFO {
     unsigned short wCode;         // An error code describing the error.
     unsigned short wReserved;
     BSTR bstrSource;                    // Source of the exception.
     BSTR bstrDescription;         // Textual description of the error.
     BSTR bstrHelpFile;                  // Help file path.
     unsigned long dwHelpContext;  // Help context ID.
     void FAR* pvReserved;
     /* Pointer to function that fills in Help and description info */
     HRESULT (STDAPICALLTYPE FAR* pfnDeferredFillIn)
                 (struct tagEXCEPINFO FAR*);
     SCODE scode;                              // An SCODE describing the
error.
} EXCEPINFO, FAR* LPEXCEPINFO;
```

The following table describes the fields of the EXCEPINFO structure:

Name	Type	Description
wCode	**unsigned short**	An error code identifying the error. Error codes should be greater than 1000. Either this field or the *scode* field must be filled in; the other must be set to 0.
wReserved	**unsigned short**	Reserved; should be set to 0.
bstrSource	BSTR	A textual, human-readable name of the source of the exception. Typically this will be an application name. This field should be filled in by the implementor of **IDispatch.**
bstrDescription	BSTR	A textual, human-readable description of the error intended for the customer. If no description is available, use NULL.
bstrHelpFile	BSTR	The fully qualified drive, path, and filename of a Help file with more information about the error. If no Help is available, use NULL.
dwHelpContext	**unsigned long**	The Help context of the topic within the Help file. This field should be filled in if and only if the *bstrHelpFile* field is not NULL.
pvReserved	**void FAR***	Must be set to NULL.
pfnDeferredFillIn	HRESULT (STDAPICALLTYPE FAR* pfnDeferredFillIn) (struct tagEXCEPINFO FAR*)	Pointer to a function that takes an EXCEPINFO structure as an argument and returns an HRESULT value. If deferred fill-in is not desired, this field should be

		set to NULL.
scode	SCODE	An SCODE describing the error. Either this field or *wCode* (but not both) must be filled in; the other must be set to 0.

Use the *pfnDeferredFillIn* field to allow an object to defer filling in the *bstrDescription*, *bstrHelpFile*, and *dwHelpContext* fields until they are needed. This field might be used, for example, if loading the string for the error is a time-consuming operation. To use deferred fill-in, the object puts a function pointer in this slot and does not fill any of the other fields except *wCode* (which is required in any case). To get additional information, the caller passes the EXCEPINFO structure back to the pexcepinfo callback function, which fills in the additional information. When the OLE Automation object and the OLE Automation controller are in different processes, the OLE Automation object calls *pfnDeferredFillIn* before returning to the controller.

INTERFACEDATA

Describes the OLE Automation object's properties and methods.

```
typedef struct FARSTRUCT tagINTERFACEDATA {
     METHODDATA FAR* pmethdata;    // Pointer to an array of METHODDATAs
     unsigned int cMembers;        // Count of members
} INTERFACEDATA;
```

LCID and National Language Support

Identifies a locale for national language support. Locale information is used for international string comparisons and localized member names. For information on LCIDs, see "Supporting Multiple National Languages" in Chapter 2.

```
typedef unsigned long LCID;
```

METHODDATA

Used to describe a method or property.

```
typedef struct FARSTRUCT tagMETHODDATA {
     OLECHAR FAR* szName;          // Member name
     PARAMDATA FAR* ppdata;  // Pointer to array of PARAMDATAs
     DISPID dispid;                // Member ID
     unsigned int iMeth;           // Method index
     CALLCONV cc;                        // Calling convention
     unsigned int cArgs;           // Count of arguments
     unsigned short wFlags;  // Whether this is a method or
                                   // a property get, put, or putref
     VARTYPE vtReturn;       // Return type
} METHODDATA;
```

The following table describes the fields of the METHODDATA structure:

Name	Type	Description
szName	OLECHAR FAR*	The method name.
ppdata	PARAMDATA FAR*	The parameters for the method. The first parameter is *ppdata*[0], and so on.
dispid	DISPID	The ID of the method as used in **IDispatch**.
iMeth	unsigned int	The index of the method in the interface's VTBL. The indexes start with 0.
cc	CALLCONV	The calling convention. The CDECL and Pascal calling conventions are supported by the dispatch interface creation functions, such as **CreateStdDispatch**.
cArgs	unsigned int	The number of arguments for the method.
wFlags	unsigned short	Flags that indicate whether the method is used for getting or setting a property. The flags are the same as in **IDispatch::Invoke**. DISPATCH_METHOD indicates this is not used for a property. DISPATCH_PROPERTYGET indicates the method is used to get a property value. DISPATCH_PROPERTYPUT indicates the method is used to set the value of a property. DISPATCH_PROPERTYPUTREF indicates the method is used to make the property refer to a passed-in object.
vtReturn	VARTYPE	Return type for the method.

PARAMDATA

Used to describe a parameter accepted by a method or property.

```
typedef struct FARSTRUCT tagPARAMDATA {
     OLECHAR FAR* szName;           // Parameter name
     VARTYPE vt;                    // Parameter type
} PARAMDATA;
```

The following table describes the fields of the PARAMDATA structure:

Name	Type	Description
szName	OLECHAR FAR *	The parameter name. Names should follow standard conventions for programming language access; that is, no embedded spaces or control characters, and 32 or fewer characters. This name should be localized, because each type description describes names for a particular locale.
vt	VARTYPE	The VARTYPE that will be used by the receiver. If more than one parameter type is accepted, VT_VARIANT should be specified.

VARIANT and VARIANTARG

Use VARIANTARG to describe arguments passed within DISPPARAMS. Use VARIANT to specify variant data that can't be passed by reference; the VARIANT type can't have the VT_BYREF bit set. Note that VARIANTs can be passed by value even if VARIANTARGs cannot.

```
typedef struct FARSTRUCT tagVARIANT VARIANT;
typedef struct FARSTRUCT tagVARIANT VARIANTARG;

typedef struct tagVARIANT  {
     VARTYPE vt;
     unsigned short wReserved1;
     unsigned short wReserved2;
     unsigned short wReserved3;
     union {
           unsigned char     bVal;             /* VT_UI1
     */
           short             iVal;             /* VT_I2
           */
           long              lVal;             /* VT_I4
           */
           float             fltVal;                 /* VT_R4
                 */
           double                  dblVal;                 /* VT_R8
                       */
           VARIANT_BOOL            bool;             /* VT_BOOL
           */
           SCODE             scode;                  /* VT_ERROR
           */
           CY                      cyVal;                  /* VT_CY
                       */
           DATE              date;             /* VT_DATE
     */
           BSTR              bstrVal;                /* VT_BSTR
           */
           Iunknown                FAR* punkVal;     /* VT_UNKNOWN
           */
           Idispatch         FAR* pdispVal;    /* VT_DISPATCH
     */
           SAFEARRAY         FAR* parray;            /* VT_ARRAY|*
           */
           unsigned char     FAR *pbVal;       /* VT_BYREF|VT_UI1
     */
           short             FAR* piVal;       /* VT_BYREF|VT_I2       */
           long              FAR* plVal;       /* VT_BYREF|VT_I4       */
           float             FAR* pfltVal;     /* VT_BYREF|VT_R4       */
           double                  FAR* pdblVal;     /* VT_BYREF|VT_R8
     */
           VARIANT_BOOL            FAR* pbool;       /* VT_BYREF|VT_BOOL
           */
           SCODE             FAR* pscode;            /* VT_BYREF|VT_ERROR
           */
           CY                      FAR* pcyVal;            /* VT_BYREF|VT_CY
           */
```

```
          DATE              FAR* pdate;        /* VT_BYREF|VT_DATE
     */
          BSTR              FAR* pbstrVal;     /* VT_BYREF|VT_BSTR
     */
          IUnknown FAR*     FAR* ppunkVal;     /* VT_BYREF|VT_UNKNOWN  */
          IDispatch FAR*    FAR* ppdispVal;    /* VT_BYREF|VT_DISPATCH */
          SAFEARRAY FAR*    FAR* parray;            /* VT_ARRAY|*
          */
          VARIANT                FAR* pvarVal;      /* VT_BYREF|VT_VARIANT
     */
          void              FAR* byref;        /* Generic ByRef
     */
     };
};
```

To simplify extracting values from VARIANTARGs, OLE Automation provides a set of functions for manipulating this type. Use of these functions is strongly recommended to ensure that applications apply consistent coercion rules. The functions are described in Chapter 6, "Data Manipulation Functions."

The *vt* value governs the interpretation of the union as follows:

Value	**Description**
VT_EMPTY	No value was specified. If an argument is left blank, you should **not** return VT_EMPTY for the argument. Instead, you should return the VT_ERROR value: DISP_E_MEMBERNOTFOUND.
VT_EMPTY | VT_BYREF	Illegal.
VT_UI1	An unsigned 1-byte character is stored in *bVal*.
VT_UI1 | VT_BYREF	A reference to an unsigned 1-byte character was passed; a pointer to the value is in *pbVal*.
VT_I2	A 2-byte integer value is stored in *iVal*.
VT_I2 | VT_BYREF	A reference to a 2-byte integer was passed; a pointer to the value is in *piVal*.
VT_I4	A 4-byte integer value is stored in *lVa*l.
VT_I4 | VT_BYREF	A reference to a 4-byte integer was passed; a pointer to the value is in *plVa*l.
VT_R4	An IEEE 4-byte real value is stored in *fltVal*.
VT_R4 | VT_BYREF	A reference to an IEEE 4-byte real was passed; a pointer to the value is in *pfltVal*.
VT_R8	An 8-byte IEEE real value is stored in *dblVal*.
VT_R8 | VT_BYREF	A reference to an 8-byte IEEE real was passed; a pointer to its value is in *pdblVal*.
VT_CY	A currency value was specified. A currency number is stored as an 8-byte, two's complement integer, scaled by 10,000 to give a fixed-point number with 15 digits to the left of the decimal point and 4 digits to the right. The value is in *cyVal*.
VT_CY | VT_BYREF	A reference to a currency value was passed; a pointer to the value is in *pcyVal*.

VT_BSTR	A string was passed; it is stored in *bstrVa*l. This pointer must be obtained and freed via the BSTR functions, which are described in Chapter 6, "Data Manipulation Functions."
VT_BSTR | VT_BYREF	A reference to a string was passed. A BSTR* which points to a BSTR is in *pbstrVal*. The referenced pointer must be obtained or freed via the BSTR functions.
VT_NULL	A propagating NULL value was specified. This should not be confused with the NULL pointer. The NULL value is used for tri-state logic as with SQL.
VT_NULL | VT_BYREF	Illegal.
VT_ERROR	An SCODE was specified. The type of the error is specified in *scode*. Generally, operations on error values should raise an exception or propagate the error to the return value, as appropriate.
VT_ERROR | VT_BYREF	A reference to an SCODE was passed. A pointer to the value is in *pscode*.
VT_BOOL	A Boolean (True/False) value was specified. A value of 0xFFFF (all bits one) indicates True; a value of 0 (all bits zero) indicates False. No other values are legal.
VT_BOOL | VT_BYREF	A reference to a Boolean value. A pointer to the Boolean value is in *pbool*.
VT_DATE	A value denoting a date and time was specified. Dates are represented as double-precision numbers, where midnight, January 1, 1900 is 2.0, January 2, 1900 is 3.0, and so on. The value is passed in *date*. This is the same numbering system used by most spreadsheet programs, although some incorrectly believe that February 29, 1900 existed, and thus set January 1, 1900 to 1.0. The date can be converted to and from an MS-DOS representation using **VariantTimeToDosDateTime**, discussed in Chapter 6, "Data Manipulation Functions."
VT_DATE | VT_BYREF	A reference to a date was passed. A pointer to the value is in *pdate*.
VT_DISPATCH	A pointer to an object was specified. The pointer is in *pdispVal*. This object is only known to implement **IDispatch**; the object can be queried as to whether it supports any other desired interface by calling **QueryInterface** on the object. Objects that do not implement **IDispatch** should be passed using VT_UNKNOWN.
VT_DISPATCH | VT_BYREF	A pointer to a pointer to an object was specified. The pointer to the object is stored in the location referred to by *ppdispVal*.
VT_VARIANT	Illegal. VARIANTARGs must be passed by reference.

VT_VARIANT | VT_BYREF	A pointer to another VARIANTARG is passed in *pvarVal*. This referenced VARIANTARG will never have the VT_BYREF bit set in *vt*, so only one level of indirection can ever be present. This value can be used to support languages that allow functions to change the types of variables passed by reference.
VT_UNKNOWN	A pointer to an object that implements the **IUnknown** interface is passed in *punkVal*.
VT_UNKNOWN | VT_BYREF	A pointer to a pointer to the **IUnknown** interface is passed in *ppunkVal*. The pointer to the interface is stored in the location referred to by *ppunkVal*.
VT_ARRAY | <anything>	An array of data type <anything> was passed. (VT_EMPTY and VT_NULL are illegal types to combine with VT_ARRAY). The pointer in *pByrefVal* points to an array descriptor, which describes the dimensions, size, and in-memory location of the array. The array descriptor is never accessed directly, but instead is read and modified using the functions described in Chapter 6, "Data Manipulation Functions."

VARTYPE

VARTYPE is an enumeration type used in VARIANT, TYPEDESC, OLE property sets, and safe arrays. The enumeration constants listed below are valid in the *vt* field of a VARIANT structure.

```
typedef unsigned short VARTYPE;
enum VARENUM{
     VT_EMPTY          = 0,              // Not specified
     VT_NULL           = 1,              // Null
     VT_I2       = 2,              // 2-byte signed int
     VT_I4       = 3,              // 4-byte signed int
     VT_R4       = 4,              // 4-byte real
     VT_R8       = 5,              // 8-byte real
     VT_CY       = 6,              // Currency
     VT_DATE           = 7,              // Date
     VT_BSTR           = 8,              // Binary string
     VT_DISPATCH = 9,              // IDispatch FAR*
     VT_ERROR          = 10,             // SCODE
     VT_BOOL           = 11,             // Boolean; True=-1, False=0
     VT_VARIANT  = 12,             // VARIANT FAR*
     VT_UNKNOWN  = 13,             // IUnknown FAR*
     VT_UI1            = 17,             // Unsigned char
.
.    //Other constants that are not valid in VARIANTs
.
};
     VT_RESERVED = (int) 0x8000
     // By reference: a pointer to the data is passed
     VT_BYREF    = (int) 0x4000
     VT_ARRAY    = (int) 0x2000    // A safe array of the data is passed
```

IDispatch::GetIDsOfNames

HRESULT IDispatch::GetIDsOfNames(*riid, rgszNames, cNames, lcid, rgdispid***)**
REFIID *riid*
OLECHAR FAR* FAR* *rgszNames*
unsigned int *cNames*
LCID *lcid*
DISPID FAR* *rgdispid*

Maps a single member and an optional set of argument names to a corresponding set of integer DISPIDs, which may be used on subsequent calls to **IDispatch::Invoke**. The dispatch function **DispGetIDsOfNames** provides a standard implementation of **GetIDsOfNames**.

Parameters

riid
Reserved for future use. Must be NULL.

rgszNames
Passed-in array of names to be mapped.

cNames
Count of the names to be mapped.

lcid
The locale context in which to interpret the names.

rgdispid
Caller-allocated array, each element of which contains an ID corresponding to one of the names passed in the *rgszNames* array. The first element represents the member name; the subsequent elements represent each of the member's parameters.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
DISP_E_UNKNOWNNAME	One or more of the names were not known. The returned array of DISPIDs contains DISPID_UNKNOWN for each entry that corresponds to an unknown name.
DISP_E_UNKNOWNLCID	The LCID was not recognized.

Comments

An **IDispatch** implementation may choose to associate any positive integer ID value with a given name. Zero is reserved for the default, or Value property; -1 is reserved to indicate an unknown name; and other negative values are defined for other purposes. For example, if **GetIDsOfNames** is called and the implementation does not recognize one or more of the names, it will return DISP_E_UNKNOWNNAME and the *rgdispid* array will contain DISPID_UNKNOWN for the entries that correspond to the unknown names.

The member and parameter DISPIDs must remain constant for the lifetime of the object. This allows a client to obtain the DISPIDs once and cache them for later use.

When **GetIDsOfNames** is called with more than one name, the first name (*rgszNames*[0]) corresponds to the member name, and subsequent names correspond to the names of the member's parameters.

The same name may map to different DISPIDs, depending on context. For example, a name may have a DISPID when it is used as a member name with a particular interface, a different DISPID as a member of a different interface, and different mapping for each time it appears as a parameter.

The **IDispatch** interface binds to names at run time. To bind at compile time instead, an **IDispatch** client can map names to DISPIDs using the type information interfaces described in Chapter 8, "Type Description Interfaces." This allows a client to bind to members at compile time and avoid calling **GetIDsOfNames** at run time.

The implementation of **GetIDsOfNames** must be case insensitive. Clients that need case-sensitive name mapping should use the type information interfaces to map names to DISPIDs, rather than calling **GetIDsOfNames**.

Examples

The following code from the Lines sample file LINE.CPP implements the **GetIDsOfNames** member function for the CLine class. The OLE Automation object uses the standard implementation, **DispGetIDsOfNames**.

```
STDMETHODIMP
CLine::GetIDsOfNames(
           REFIID riid,
           OLECHAR FAR* FAR* rgszNames,
           UINT cNames,
           LCID lcid,
           DISPID FAR* rgdispid)
{
           return DispGetIDsOfNames(m_ptinfo, rgszNames, cNames, rgdispid);
}
```

The following code might appear in an OLE Automation controller that calls **GetIDsOfNames** to get the DISPID of the CLine Color property.

```
HRESULT hresult;
IDispatch FAR* pdisp = (IDispatch FAR*)NULL;
DISPID dispid;
OLECHAR FAR* szMember = "color";

// Omitted code that sets pdisp...
hresult = pdisp->GetIDsOfNames(
     IID_NULL,
     &szMember,
     1, LOCALE_SYSTEM_DEFAULT,
     &dispid);
```

See Also

CreateStdDispatch, **DispGetIDsOfNames**, **ITypeInfo::GetIDsOfNames**

IDispatch::GetTypeInfo

HRESULT IDispatch::GetTypeInfo(*itinfo, lcid, pptinfo***)**
  unsigned int *itinfo*
  LCID *lcid*
  ITypeInfo FAR* FAR* *pptinfo*

Retrieves a type information object, which can be used to get the type information for an interface.

Parameters

itinfo
  The type information to return. Pass 0 to retrieve type information for the **IDispatch** implementation.

lcid
  The locale ID for the type information. An object may be able to return different type information for different languages. This is important for classes that support localized member names. For classes that don't support localized member names, this parameter can be ignored.

pptinfo
  Receives a pointer to the type information object requested.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success; the TypeInfo element exists.
DISP_E_BADINDEX	Failure; *itinfo* argument was not 0.
TYPE_E_ELEMENTNOTFOUND	Failure; *itinfo* argument was not 0.

Example

The following code from the sample file LINES.CPP loads information from the type library and implements the member function **GetTypeInfo**:

```
// These lines from CLines::Create load type information for the
// Lines collection from the type library.
     hr = LoadTypeInfo(&pLines->m_ptinfo, IID_ILines);
     if (FAILED(hr))
           goto error;

// Additional code omitted...

// This function implements GetTypeInfo for the CLines collection.
STDMETHODIMP
CLines::GetTypeInfo(
           UINT itinfo,
           LCID lcid,
           ITypeInfo FAR* FAR* pptinfo)
{
     *pptinfo = NULL;

     if(itinfo != 0)
           return ResultFromScode(DISP_E_BADINDEX);

     m_ptinfo->AddRef();
     *pptinfo = m_ptinfo;

     return NOERROR;
}
```

See Also

CreateStdDispatch, **CreateDispTypeInfo**

IDispatch::GetTypeInfoCount

HRESULT IDispatch::GetTypeInfoCount(*pctinfo***)**
   unsigned int FAR* *pctinfo*

Retrieves the number of type information interfaces that an object provides
(either 0 or 1).

Parameters

pctinfo
   Points to location that receives the number of type information interfaces that the object provides. If the object provides type information, this number is 1; otherwise the number is 0.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success
E_NOTIMPL	Failure

Comments

The function may return zero, which indicates that the object does not provide any type information. In this case, the object may still be programmable through **IDispatch**, but does not provide type information for browsers, compilers, or other programming tools that access type information. This may be useful for hiding an object from browsers or for preventing early binding on an object.

Example

This code from the Lines sample file LINES.CPP implements the **GetTypeInfoCount** member function for the CLines class. (OLE Automation object.)

```
STDMETHODIMP
CLines::GetTypeInfoCount(UINT FAR* pctinfo)
{
     *pctinfo = 1;
     return NOERROR;
}
```

See Also

CreateStdDispatch

IDispatch::Invoke

HRESULT IDispatch::Invoke(*dispidMember*, *riid*, *lcid*, *wFlags*, *pdispparams*, *pvarResult*, *pexcepinfo*, *puArgErr***)**
DISPID *dispidMember*
REFIID *riid*
LCID *lcid*
unsigned short *wFlags*
DISPPARAMS FAR* *pdispparams*
VARIANT FAR* *pvarResult*
EXCEPINFO FAR* *pexcepinfo*
unsigned int FAR* *puArgErr*

Provides access to properties and methods exposed by an object. The dispatch function **DispInvoke** provides a standard implementation of **IDispatch::Invoke**.

Parameters

dispidMember
Identifies the member. Use **GetIDsOfNames** or the object's documentation to obtain the dispatch identifier.

riid
Reserved for future use. Must be IID_NULL.

lcid
The locale context in which to interpret arguments. The *lcid* is used by the **GetIDsOfNames** function, and is also passed to **Invoke** to allow the object to interpret its arguments in a locale-specific way. Applications that don't support multiple national languages can ignore this parameter. See "Supporting Multiple National Languages," in Chapter 2, for more information.

wFlags
Flags describing the context of the **Invoke** call, as follows:

Value	Description
DISPATCH_METHOD	The member is being invoked as a method. If a property has the same name, both this and the DISPATCH_PROPERTYGET flag may be set.
DISPATCH_PROPERTYGET	The member is being retrieved as a property or data member.
DISPATCH_PROPERTYPUT	The member is being changed as a property or data member.
DISPATCH_PROPERTYPUTREF	The member is being changed via a reference assignment, rather than a value assignment. This flag is valid only when the property accepts a reference to an object.

pdispparams
Pointer to a structure containing an array of arguments, array of argument dispatch IDs for named arguments, and counts for number of elements in the arrays. See the Comments section for a description of the DISPPARAMS structure.

pvarResult
Pointer to where the result is to be stored, or NULL if the caller expects no result. This argument is ignored if DISPATCH_PROPERTYPUT or DISPATCH_PROPERTYPUTREF is specified.

pexcepinfo
Pointer to a structure containing exception information. This structure should be filled in if DISP_E_EXCEPTION is returned.

puArgErr
The index within *rgvarg* of the first argument that has an error. Arguments are stored in *pdispparams->rgvarg* in reverse order, so the first argument is the one with the highest index in the array. This parameter is returned only when the resulting SCODE is DISP_E_TYPEMISMATCH or DISP_E_PARAMNOTFOUND; see "Returning Errors" for details.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
DISP_E_BADPARAMCOUNT	The number of elements provided DISPPARAMS is different from the number of arguments accepted by the method or property.
DISP_E_BADVARTYPE	One of the arguments in *rgvarg* is not a valid variant type.
DISP_E_EXCEPTION	The application needs to raise an exception. In this case the structure passed in *pexcepinfo* should be filled in.
DISP_E_MEMBERNOTFOUND	The requested member does not exist, or the call to **Invoke** tried to set the value of a read-only property.
DISP_E_NONAMEDARGS	This implementation of **IDispatch** does not support named arguments.
DISP_E_OVERFLOW	One of the arguments in *rgvarg* could not be coerced to the specified type.
DISP_E_PARAMNOTFOUND	One of the parameter dispatch IDs does not correspond to a parameter on the method. In this case *puArgErr* should be set to the first argument that contains the error.
DISP_E_TYPEMISMATCH	One or more of the arguments could not be coerced. The index within *rgvarg* of the first parameter with the incorrect type is returned in the *puArgErr* parameter.
DISP_E_UNKNOWNINTERFACE	The interface ID passed in *riid* is not IID_NULL.
DISP_E_UNKNOWNLCID	The member being invoked interprets string arguments according to the locale ID (LCID), and the LCID is not recognized. If the LCID is not needed to interpret arguments, this error should not be returned.
DISP_E_PARAMNOTOPTIONAL	A required parameter was omitted.

To return other errors, you can define your own errors using the MAKE_SCODE macro.

Comments

Generally, you should not implement **Invoke** directly; instead, you should use the dispatch interface creation functions **CreateStdDispatch** and **DispInvoke**. For details, refer to "**CreateStdDispatch**" and "**DispInvoke**" in this chapter, and "Creating the IDispatch Interface" in Chapter 2.

However, if you need to perform some application-specific processing before calling a member, your code should perform the necessary actions and then call **ITypeInfo::Invoke** to invoke the member. **ITypeInfo::Invoke** acts exactly like **IDispatch::Invoke**. In fact, the standard implementations of **IDispatch::Invoke** that are created by **CreateStdDispatch** and **DispInvoke** defer to **ITypeInfo::Invoke**.

In an OLE Automation controller, use **IDispatch::Invoke** to get and set the values of properties or to call a method of an OLE Automation object. The *dispidMember* argument identifies the member to invoke. The dispatch IDs that identify members are defined by the implementor of the object, and can be determined by using the object's documentation, the **IDispatch::GetIDsOfNames** function, or the **ITypeInfo** interface.

Except as noted, the information that follows addresses developers of OLE Automation controllers. Nevertheless, if you're writing code that exposes OLE Automation objects, you should be familiar with all this information, because it describes the behavior that users of your exposed objects will expect.

Calling a Method with No Arguments

The simplest use of **Invoke** is to call a method that has no arguments. You need only pass the dispatch ID of the method, a locale ID, the DISPATCH_METHOD flag, and an empty DISPPARAMS stucture. For example:

```
HRESULT hresult;
IUnknown FAR* punk;
IDispatch FAR* pdisp = (IDispatch FAR*)NULL;
OLECHAR FAR* szMember = "simple";
DISPID dispid;
DISPPARAMS dispparamsNoArgs = {NULL, NULL, 0, 0};

hresult = CoCreateInstance(CLSID_CMyObject, NULL, CLSCTX_SERVER,
                                               IID_Unknown, (void FAR*
FAR*)&punk);

hresult = punk->QueryInterface(IID_IDispatch,
                        (void FAR* FAR*)&pdisp);

hresult = pdisp->GetIDsOfNames(IID_NULL, &szMember, 1,
                                               LOCALE_SYSTEM_DEFAULT,
&dispid) ;

hresult = pdisp->Invoke(
           dispid,
           IID_NULL,
           LOCALE_SYSTEM_DEFAULT,
           DISPATCH_METHOD,
           &dispparamsNoArgs, NULL, NULL, NULL);
```

The example invokes a method named Simple on an object of the class CMyObject. First, it calls **CoCreateInstance**, which instantiates the object and returns a pointer to the object's **IUnknown** interface (`punk`). Next, it calls **QueryInterface**, receiving a pointer to the object's **IDispatch** interface (pdisp), and then uses `pdisp` to call the object's **GetIDsOfNames** function, passing the string 'Simple' in szMember to get the dispatch ID for the Simple method. With the dispatch ID for Simple in dispid, it calls **Invoke** to invoke the method, specifying DISPATCH_METHOD for the `wFlags` parameter, and using the system default locale.

To further simplify the code, the example declares a DISPPARAMS structure named dispparamsNoArgs appropriate to an **Invoke** call with no arguments.

Because the Simple method takes no arguments and returns no result, the puArgErr and pvarResult parameters are NULL. In addition, the example passes NULL for `pexcepinfo`, indicating that it is not prepared to handle exceptions and will handle only HRESULT errors.

Most methods, however, take one or more arguments. To invoke these methods, you must fill in the DISPPARAMS structure, as described in "Passing Parameters."

OLE Automation defines special dispatch IDs for invoking an object's Value (default), _NewEnum, and Evaluate members. See DISPID.

Getting and Setting Properties

Properties are accessed the same way as methods, except that DISPATCH_PROPERTYGET or DISPATCH_PROPERTYPUT is specified instead of DISPATCH_METHOD. Note that some languages can't distinguish between retrieving a property and calling a method; both the DISPATCH_PROPERTYGET and DISPATCH_METHOD flags should be set in this case.

The following example gets the value of a property named On.   Assume that the object has been created and its interfaces queried as in the previous example:

```
VARIANT FAR *pvarResult;
//...code omitted for brevity...
szMember = "On";
hresult = pdisp->GetIDsOfNames(IID_NULL, &szMember, 1,
                                              LOCALE_SYSTEM_DEFAULT,
&dispid) ;

hresult = pdisp->Invoke(
           dispid,
           IID_NULL,
           LOCALE_SYSTEM_DEFAULT,
           DISPATCH_PROPERTYGET,
           &dispparamsNoArgs, pvarResult, NULL, NULL);
```

As in the previous example, the code calls **GetIDsOfNames** for the dispatch ID of the On property, then passes the ID to **Invoke**. **Invoke** returns the property's value in pvarResult. In general, the return value does not have VT_BYREF set. However, implementors may set this bit and return a pointer to the return value, if the lifetime of the return value is the same as that of the object.

To change the property's value, the call looks like this:

```
VARIANT FAR *pvarResult;
DISPPARAMS dispparams;

//...Code omitted for brevity...

szMember = "On";
dispparams.rgvarg[0].vt = VT_BOOL;
dispparams.rgvarg[0].bool = FALSE;
dispparams.rgdispidNamedArgs = DISPID_PROPERTYPUT;
dispparams.cArgs = 1;
disparams.cNamedArgs = 1;
```

```
hresult = pdisp->GetIDsOfNames(IID_NULL, &szMember, 1,
                                                LOCALE_SYSTEM_DEFAULT,
&dispid) ;

hresult = pdisp->Invoke(
           dispid,
           IID_NULL,
           LOCALE_SYSTEM_DEFAULT,
           DISPATCH_PROPERTYPUT,
           &dispparams, NULL, NULL, NULL);
```

The new value for the property (the Boolean value False) is passed as an argument when the On property's put function is invoked. Note that the dispatch ID for the argument is DISPID_PROPERTYPUT. This special dispatch ID is defined by OLE Automation to designate the parameter that contains the new value for a property's put function. The remaining details of the DISPPARAMS structure are described in the next section, "Passing Parameters."

The DISPATCH_PROPERTYPUT flag in the example above indicates that a property is being set by value. In Visual Basic, the following statement assigns the Value (default) property of YourObj to the property Prop:

```
MyObj.Prop = YourObj
```

This statement should be flagged as a DISPATCH_PROPERTYPUT. Similarly, statements like the following assign the Value property of one object to the Value property of another object:

```
Worksheet.Cell(1,1) = Worksheet.Cell(6,6)
MyDoc.Text1 = YourDoc.Text1
```

These statements result in a PROPERTY_PUT operation on Worksheet.Cell(1,1) and MyDoc.Text1.

Use the DISPATCH_PROPERTYPUTREF flag to indicate a property or data member that should be set by reference. For example, the following Visual Basic statement assigns the pointer to YourObj to the property Prop, and should be flagged as DISPATCH_PROPERTYPUTREF:

```
Set MyObj.Prop = YourObj
```

The Set statement causes a reference assignment, rather than a value assignment.

Note that the right side parameter is always passed by name, and should not be accessed positionally.

Passing Parameters

Arguments to the method or property being invoked are passed in the DISPPARAMS structure. This structure consists of a pointer to an array of arguments represented as variants, a pointer to an array of dispatch IDs for named arguments, and the number of arguments in each array.

```
typedef struct FARSTRUCT tagDISPPARAMS{
     VARIANTARG FAR* rgvarg;            // Array of arguments
     DISPID FAR* rgdispidNamedArgs;     // Dispatch IDs of named arguments
     unsigned int cArgs;                      // Number of arguments
     unsigned int cNamedArgs;           // Number of named arguments
} DISPPARAMS;
```

The arguments are passed in the array *rgvarg*[ ], with the number of arguments passed in *cArgs*. Place the arguments in the array from last to first, so *rgvarg*[0] has the last argument and *rgvarg*[*cArgs* -1] has the first argument. The method or property may change the values of elements within the array *rgvarg* only if they have the VT_BYREF flag set; otherwise they should be considered read only.

A dispatch invocation can have named arguments as well as positional arguments. If *cNamedArgs* is 0, all the elements of *rgvarg*[ ] represent positional arguments. If *cNamedArgs* is nonzero, each element of *rgdispidNamedArgs*[ ] contains the DISPID of a named argument, and the value of the argument is in the matching element of *rgvarg*[ ]. The dispatch IDs of the named arguments are always contiguous in *rgdispidNamedArgs*, and their values are in the first *cNamedArgs* elements of *rgvarg*. Named arguments can't be accessed positionally and vice versa.

The DISPID of an argument is its zero-based position in the argument list. For example, the following method takes three arguments:

```
BOOL _export CDECL
CCredit::CheckCredit(BSTR bstrCustomerID,// DISPID = 0
                            BSTR bstrLenderID,          // DISPID = 1
                            CURRENCY cLoanAmt)          // DISPID = 2
{
... // Code omitted.
}
```

If you include the DISPID with each named argument, you may pass the named arguments to **Invoke** in any order. For example, if a method is to be invoked with two positional arguments, followed by three named arguments (A, B, and C), using the following hypothetical syntax, then *cArgs* would be 5, and *cNamedArgs* would be 3:

object.method("arg1", "arg2", A := "argA", B := "argB", C: = "argC")

The first positional argument would be in *rgvarg*[4]. The second positional argument would be in *rgvarg*[3]. The ordering of named arguments should be immaterial to the **IDispatch** implementation, but these are also generally passed in reverse order. The argument named A would be in *rgvarg*[2], with the DISPID of A in *rgdispidNamedArgs*[2]. The argument named B would be in *rgvarg*[1], with the corresponding DISPID in *rgdispidNamedArgs*[1]. The argument named C would be in *rgvarg*[0], with the DISPID corresponding to C in *rgdispidNamedArgs*[0]. The following diagram illustrates the arrays and their contents.

{ewc msdncd, EWGraphic, group10457 0 /a "SDK_01.bmp"}

You can also use **Invoke** on members with optional arguments, but all the optional arguments must be of type VARIANT. As with required arguments, the contents of the argument vector depend on whether the arguments are positional or named. The invoked member must ensure that the arguments are valid; **Invoke** merely passes the DISPPARAMS structure it receives.

Omitting named arguments is straightforward. You simply pass the arguments in *rgvarg* and their DISPIDs in *rgdispidNamedArgs*. To omit the argument named B, in the preceding example, you would set *rgvarg*[0] to the value of C, with its DISPID in *rgdispidNamedArgs*[0]; and *rgvarg*[1] to the value of A, with its DISPID in *rgdispidNamedArgs*[1]. The subsequent positional arguments would occupy elements 2 and 3 of the arrays. In this case, *cArgs* is 4, and *cNamedArgs* is 2.

If the arguments are positional (unnamed), set *cArgs* to the total number of possible arguments, *cNamedArgs* to zero, and pass VT_ERROR as the type of the omitted arguments, and DISP_E_PARAMNOTFOUND as the value. For example, the following code invokes ShowMe (,1):

```
VARIANT FAR *pvarResult;
EXCEPINFO FAR *pExcepinfo;
unsigned int FAR *puArgErr;
DISPPARAMS dispparams;

//...Code omitted for brevity...

szMember = "ShowMe";
```

```
hresult = pdisp->GetIDsOfNames(IID_NULL, &szMember, 1,
                                                LOCALE_SYSTEM_DEFAULT,
&dispid) ;
dispparams.rgvarg[0].vt = VT_I2;
dispparams.rgvarg[0].ival = 1;
dispparams.rgvarg[1].vt = VT_ERROR;
dispparams.rgvarg[1].scode = DISP_E_PARAMNOTFOUND;
dispparams.cArgs = 2;
disparams.cNamedArgs = 0;

hresult = pdisp->Invoke(
           dispid,
           IID_NULL,
           LOCALE_SYSTEM_DEFAULT,
           DISPATCH_METHOD,
           &dispparams, pvarResult, pExcepinfo, puArgErr);
```

The example takes two positional arguments, but omits the first. Therefore, *rgvarg*[0] contains 1, the value of the last argument in the argument list, and *rgvarg*[1] contains VT_ERROR and the error scode, indicating the omitted first argument.

The calling code is responsible for releasing all strings and objects referred to by *rgvarg*[ ] or placed in **pvarResult*. As with other parameters that are passed by value, if the invoked member must maintain access to a string after returning, the string should be copied. Similarly, if the member needs access to a passed object pointer after returning, it must call **AddRef** on the object. A common example occurs when an object property is changed to refer to a new object, using the DISPATCH_PROPERTYPUTREF flag.

For those implementing **IDispatch::Invoke**, OLE Automation provides the **DispGetParam** function to retrieve parameters from the argument vector and coerce them to the proper type. See the reference page for **DispGetParam** later in this chapter, for details.

Indexed Properties

When you invoke indexed properties of any dimension, you need to pass the indexes as additional arguments. To set an indexed property, place the new value in the first element of the *rgvarg*[ ] vector, and the indexes in the subsequent elements. To get an indexed property, pass the indexes in the first *n* elements of *rgvarg* and the number of indexes in *cArg*. **Invoke** returns the value of the property in *pvarResult*.

OLE Automation stores array data in column-major order, which is the same ordering scheme used by Visual Basic and FORTRAN, but different from C, C++, and Pascal. If you're programming in C, C++, or Pascal, you need to pass the indexes in the reverse order. The following example shows how to fill the DISPPARAMS structure in C++:

```
dispparams.rgvarg[0].vt = VT_I2;
dispparams.rgvarg[0].iVal = 99;
dispparams.rgvarg[1].vt = VT_I2;
dispparams.rgvarg[1].iVal = 2;
dispparams.rgvarg[2].vt = VT_I2;
dispparams.rgvarg[2].iVal = 1;
dispparams.rgdispidNamedArgs = DISPID_PROPERTYPUT;
dispparams.cArgs = 3;
disparams.cNamedArgs = 1;
```

The example changes the value of Prop[1,2] to 99. The new property value is passed in *rgvarg*[0]. The rightmost index is passed in *rgvarg*[1], and the next index in *rgvarg*[2]. The *cArgs* field specifies the number of elements of *rgvarg*[ ] that contain data, and *cNamedArgs* is 1, indicating the new value for the property.

Property collections are an extension of this feature.

Raising Exceptions During Invoke

When you implement **IDispatch::Invoke**, you can choose to communicate errors either through the normal return value or by raising an exception. An exception is a special situation that is normally dealt with by jumping to the nearest enclosing exception handler.

To raise an exception, **IDispatch::Invoke** returns DISP_E_EXCEPTION and fills the structure passed through *pexcepinfo* with information about the cause of the exception or error. You can use the information to understand the cause of the exception and deal with it as necessary.

The exception information structure includes an error code number which identifies the kind of exception, a string which describes the error in a human-readable way, and a Help file and Help context number which can be passed to Windows Help for details about the error. At a minimum, the error code number must be filled in with a valid number.

If you consider **IDispatch** as another way to call C++ style methods in an interface, EXCEPINFO models the throwing of an exception or **longjmp()** call by such a method.

Returning Errors

Invoke returns DISP_E_MEMBERNOTFOUND if one of the following conditions occurs:

- A member or parameter with the specified DISPID and matching *cArgs* can't be found*,* and the parameter is not optional.
- The member is a void function and the caller didn't set *pvarResult* to NULL.
- The member is a read-only property and the caller set *wFlags* to DISPATCH_PROPERTYPUT or DISPATCH_PROPERTYPUTREF.

If **Invoke** finds the member, but uncovers errors in the argument list, it returns one of several other errors. DISP_E_BAD_PARAMCOUNT means that the DISPPARAMS structure contains an incorrect number of parameters for the property or method. DISP_E_NONAMEDARGS means that **Invoke** received named arguments, but they are not supported by this member.

DISP_E_PARAMNOTFOUND means that the correct number of parameters was passed, but the dispatch ID for one or more parameters was incorrect. If **Invoke** can't convert one of the arguments to the desired type, it returns DISP_E_TYPEMISMATCH. In these two cases, if it can identify which argument is incorrect, **Invoke** sets **puArgErr* to the index within *rgvarg* of the argument with the error. For example, if an OLE Automation method expects a reference to a double-precision number as an argument, but receives a reference to an integer, the argument should be coerced. However, if the method receives a date, **IDispatch::Invoke** returns DISP_E_TYPEMISMATCH and sets **puArgErr* to the index of the integer in the argument array.

OLE Automation provides functions to perform standard conversions of VARIANT, and these should be used for consistent operation. Only when these functions fail is DISP_E_TYPEMISMATCH returned. For more information on converting arguments, see Chapter 6, "Data Manipulation Functions."

Example

This code from the Lines sample file LINES.CPP implements the **Invoke** member function for the CLines class:

```
STDMETHODIMP
CLines::Invoke(
           DISPID dispidMember,
           REFIID riid,
           LCID lcid,
           WORD wFlags,
           DISPPARAMS FAR* pdispparams,
           VARIANT FAR* pvarResult,
           EXCEPINFO FAR* pexcepinfo,
           UINT FAR* puArgErr)
{
     HRESULT hr;

     m_bRaiseException = FALSE;
     hr =  DispInvoke(
           this, m_ptinfo,
           dispidMember, wFlags, pdispparams,
           pvarResult, pexcepinfo, puArgErr);
     if (m_bRaiseException)
     {
           if (NULL != pexcepinfo)
           _fmemcpy(pexcepinfo, &m_excepinfo, sizeof(EXCEPINFO));
           return ResultFromScode(DISP_E_EXCEPTION);
     }
     else return hr;
}
```

This code calls the CLine **Invoke** member function to get the value of the Color property:

```
HRESULT hr;
EXCEPINFO excepinfo;
UINT nArgErr;
VARIANT vRet;
DISPPARAMS FAR * pdisp;
OLECHAR FAR* szMember;
DISPPARAMS dispparamsNoArgs = {NULL, NULL, 0, 0};

// Initialization code omitted for brevity.
szMember = "Color";
hr = pdisp->GetIDsOfNames(IID_NULL, &szMember, 1, LOCALE_SYSTEM_DEFAULT,
     &dispid);

// Get Color property.
hr = pdisp->Invoke(dispid, IID_NULL, LOCALE_SYSTEM_DEFAULT,
 DISPATCH_PROPERTYGET, &dispparams, &vRet, &excepinfo, &nArgErr);
```

See Also

CreateStdDispatch, **DispInvoke**, **DispGetParam**, **ITypeInfo::Invoke**

Dispatch Interface Creation Functions

Implemented by	Used by	Header filename	Import library name
OLEAUT32.DLL (32-bit systems) OLE2DISP.DLL (16-bit systems)	Applications that expose programmable objects.	OLEAUTO.H DISPATCH.H	OLEAUT32.LIB OLE2DISP.LIB

These functions simplify the creation of IDispatch interfaces.

CreateDispTypeInfo

HRESULT CreateDispTypeInfo (*pInterfacedata, lcid, pptinfo***)**
INTERFACEDATA *pInterfacedata*
LCID *lcid*
ITypeInfo FAR* FAR* *pptinfo*

Creates simplified type information for use in an implementation of **IDispatch**.

Parameters

pInterfacedata
The interface description that this type information describes.

lcid
The locale ID (LCID) for the names used in the type information.

pptinfo
On return, pointer to a pointer to a type information implementation for use in **DispGetIDsOfNames** and **DispInvoke**.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	The interface is supported.
E_INVALIDARG	Either the interface description or the locale ID is invalid.
E_OUTOFMEMORY	Insufficient memory to complete operation.

Comments

Type information may be constructed at run time using **CreateDispTypeInfo** and an INTERFACEDATA structure that describes the object being exposed.

The type information returned by this function is primarily designed to automate the implementation of **IDispatch**. **CreateDispTypeInfo** doesn't return all the type information described in Chapter 9, "Type Building Interfaces." The argument *pInterfaceData* is not a complete description of an interface. It does not include Help information, comments, optional parameters, and other type information that is useful in different contexts.

The recommended method for providing type information about an object is to describe the object using the Object Description Language (ODL) and to compile the object description into a type library using MkTypLib.

To use type information from a type library, use the **LoadTypeLib** and **GetTypeInfoOfGuid** functions instead of **CreateDispTypeInfo**. For more information, see Chapter 8, "Type Description Interfaces."

Example

The code that follows creates type information from the INTERFACEDATA shown to expose the CCalc object.

```
static METHODDATA NEARDATA rgmdataCCalc[] =
{
      PROPERTY(VALUE,   IMETH_ACCUM,    IDMEMBER_ACCUM,    VT_I4)
    , PROPERTY(ACCUM,   IMETH_ACCUM,    IDMEMBER_ACCUM,    VT_I4)
    , PROPERTY(OPND,    IMETH_OPERAND,  IDMEMBER_OPERAND,  VT_I4)
    , PROPERTY(OP,      IMETH_OPERATOR, IDMEMBER_OPERATOR, VT_I2)
    ,  METHOD0(EVAL,    IMETH_EVAL,     IDMEMBER_EVAL,     VT_BOOL)
    ,  METHOD0(CLEAR,   IMETH_CLEAR,    IDMEMBER_CLEAR,    VT_EMPTY)
    ,  METHOD0(DISPLAY, IMETH_DISPLAY,  IDMEMBER_DISPLAY,  VT_EMPTY)
    ,  METHOD0(QUIT,    IMETH_QUIT,     IDMEMBER_QUIT,     VT_EMPTY)
    ,  METHOD1(BUTTON,  IMETH_BUTTON,   IDMEMBER_BUTTON,   VT_BOOL)
};

INTERFACEDATA NEARDATA g_idataCCalc =
{
     rgmdataCCalc, DIM(rgmdataCCalc)
};

// Use Dispatch Interface creation functions to implement IDispatch
CCalc FAR*
CCalc::Create()
{
     HRESULT hresult;
     CCalc FAR* pcalc;
     CArith FAR* parith;
     ITypeInfo FAR* ptinfo;
     IUnknown FAR* punkStdDisp;
extern INTERFACEDATA NEARDATA g_idataCCalc;

     if((pcalc = new FAR CCalc()) == NULL)
           return NULL;
     pcalc->AddRef();

     parith = &(pcalc->m_arith);
     // Build a TypeInfo for the functionality on this object that
     // is being exposed for external programmability.
     hresult = CreateDispTypeInfo(
           &g_idataCCalc, LOCALE_SYSTEM_DEFAULT, &ptinfo);
     if(hresult != NOERROR)
           goto LError0;

     // Create an aggregate with an instance of the default
     // implementation of IDispatch that is initialized with our
     // TypeInfo.
     hresult = CreateStdDispatch(
           pcalc,                        // controlling unknown
           parith,                 // instance to dispatch on
           ptinfo,                 // typeinfo describing the instance
           &punkStdDisp);

     ptinfo->Release();

     if(hresult != NOERROR)
           goto LError0;

     pcalc->m_punkStdDisp = punkStdDisp;

     return pcalc;

LError0:;
     pcalc->Release();
     return NULL;
}
```

CreateStdDispatch

HRESULT CreateStdDispatch (*punkOuter, pvThis, ptinfo, ppunkStdDisp*)
IUnknown FAR* *punkOuter*
void FAR* *pvThis*
ITypeInfo FAR* *ptinfo*
IUnknown FAR* FAR* *ppunkStdDisp*

Creates a standard implementation of the **IDispatch** interface through a single function call. This simplifies exposing objects through OLE Automation.

Parameters

punkOuter
Pointer to the object's **IUnknown** implementation.

pvThis
Pointer to the object to expose.

ptinfo
Pointer to the type information that describes the exposed object.

ppunkStdDisp
Pointer to the location where the implementation of the **IDispatch** interface for this object is returned. This pointer is NULL if the function fails.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
E_INVALIDARG	One of the first three arguments is invalid.
E_OUTOFMEMORY	There was insufficient memory to complete the operation.

Comments

You can use **CreateStdDispatch** when you create an object instead of implementing the **IDispatch** member functions for the object. The implementation that **CreateStdDispatch** creates has these limitations:

- Supports only one national language
- Supports only dispatch-defined exception codes returned from **Invoke**

LoadTypeLib, **GetTypeInfoOfGuid**, and **CreateStdDispatch** comprise the minimum set of functions you need to cal l to expose an object using a type library. For more information on **LoadTypeLib** and **GetTypeInfoOfGuid**, see Chapter 8, "Type Description Interfaces."

CreateDispTypeInfo and **CreateStdDispatch** comprise the minimum set of dispatch components you need to call to expose an object using type information provided by the INTERFACEDATA structure.

Example

The following code implements the **IDispatch** interface for the CCalc class using **CreateStdDispatch**.

```
CCalc FAR*
CCalc::Create()
{
      HRESULT hresult;
      CCalc FAR* pcalc;
```

```
     CArith FAR* parith;
     ITypeInfo FAR* ptinfo;
     IUnknown FAR* punkStdDisp;
extern INTERFACEDATA NEARDATA g_idataCCalc;

     if((pcalc = new FAR CCalc()) == NULL)
           return NULL;
     pcalc->AddRef();

     parith = &(pcalc->m_arith);

     // Build a TypeInfo for the functionality on this object that
     // is being exposing for external programmability.
     hresult = CreateDispTypeInfo(
           &g_idataCCalc, LOCALE_SYSTEM_DEFAULT, &ptinfo);
     if(hresult != NOERROR)
           goto LError0;

     // Create an aggregate with an instance of the default
     // implementation of IDispatch that is initialized with our
     // TypeInfo.
     hresult = CreateStdDispatch(
           pcalc,                        // controlling unknown
           parith,                 // instance to dispatch on
           ptinfo,                 // typeinfo describing the instance
     &punkStdDisp);

     ptinfo->Release();

     if(hresult != NOERROR)
           goto LError0;

     pcalc->m_punkStdDisp = punkStdDisp;

     return pcalc;

LError0:;
     pcalc->Release();
     return NULL;
}
```

DispGetIDsOfNames

HRESULT DispGetIDsOfNames (*ptinfo, rgszNames, cNames, rgdispid***)**
  ITypeInfo * *ptinfo*
  OLECHAR FAR* FAR* *rgszNames*
  unsigned int *cNames*
  DISPID FAR* *rgdispid*

Uses type information to convert a set of names to DISPIDs. This is the recommended implementation of **IDispatch::GetIDsOfNames**.

Parameters

ptinfo
Pointer to the type information for an interface. Note that this type information is specific to one interface and language code, so it is not necessary to pass an IID or LCID to this function.

rgszNames
An array of name strings, which can be the same array passed to **DispInvoke** in the DISPPARAMS structure. If *cNames* is greater than one, the first name is interpreted as a method name and subsequent names are interpreted as parameters to that method.

cNames
The number of elements in *rgszNames*.

rgdispid
Pointer to an array of DISPIDs to be filled in by this function. The first ID corresponds to the method name; subsequent IDs are interpreted as parameters to the method.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	The interface is supported.
E_INVALIDARG	One of the arguments is invalid.
DISP_E_UNKNOWNNAME	One or more of the given names were not known. The returned array of DISPIDs will contain DISPID_UNKNOWN for each entry that corresponds to an unknown name.
Other returns	Any of the **ITypeInfo::Invoke** errors may also be returned.

Example

This code from the Lines sample file POINTS.CPP implements the member function **GetIDsOfNames** for the CPoints class using **DispGetIDsOfNames**.

```
STDMETHODIMP
CPoints::GetIDsOfNames(
            REFIID riid,
            char FAR* FAR* rgszNames,
            UINT cNames,
            LCID lcid,
            DISPID FAR* rgdispid)
{
     return DispGetIDsOfNames(m_ptinfo, rgszNames, cNames, rgdispid);
}
```

See Also

CreateStdDispatch, **IDispatch::GetIDsOfNames**

DispGetParam

HRESULT DispGetParam(*dispparams, iPosition, vt, pvarResult, puArgErr***)**
  DISPPARAMS FAR* *dispparams*
  unsigned int *iPosition*
  VARTYPE *vt*
  VARIANT FAR* *pvarResult*
  unsigned int FAR* *puArgErr*

Retrieves a parameter from the DISPPARAMS structure, checking both named parameters and positional parameters, and coerces it to the specified type.

Parameters

dispparams
  Pointer to the parameters passed to **IDispatch::Invoke**.

iPosition
  The position of the parameter in the parameter list. **DispGetParam** starts at the end of the array, so if *iPosition* is 0, the last parameter in the array is returned.

vt
  The type to which the argument should be coerced.

pvarResult
  Pointer to the variant into which to pass the parameter.

puArgErr
  On return, pointer to the index of the argument that caused a DISP_E_TYPEMISMATCH error. This pointer should be returned to **Invoke** to indicate the position of the argument in DISPPARAMS that caused the error.

Return Value

The SCODE obtained from the HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
DISP_E_BADVARTYPE	The variant type *vt* is not supported.
DISP_E_OVERFLOW	The retrieved parameter could not be coerced to the specified type.
DISP_E_PARAMNOTFOUND	The parameter indicated by *iPosition* could not be found.
DISP_E_TYPEMISMATCH	The argument could not be coerced to the specified type.
E_INVALIDARG	One of the arguments was invalid.
E_OUTOFMEMORY	Insufficient memory to complete operation.

Comments

The output parameter *pvarResult* must be a valid VARIANT; any existing contents will be released in the standard way. The contents of the VARIANT should be freed with **VariantFree**.

If **DispGetParam** is used to get the right side of a property put operation, the second parameter should be DISPID_PROPERTYPUT. For example:

```
DispGetParam(&dispparams, DISPID_PROPERTYPUT, VT_BOOL, &varResult)
```

Note also that named parameters can't be accessed positionally, and vice versa.

Example

The following example uses **DispGetParam** to set X and Y properties:

```
STDMETHODIMP
CPoint::Invoke(
     DISPID dispidMember,
     REFIID riid,
     LCID lcid,
     unsigned short wFlags,
     DISPPARAMS FAR* pdispparams,
     VARIANT FAR* pvarResult,
     EXCEPINFO FAR* pexcepinfo,
     unsigned int FAR* puArgErr)
{
     unsigned int uArgErr;
     HRESULT hresult;
     VARIANTARG varg0;
     VARIANT varResultDummy;

     UNUSED(lcid);
     UNUSED(pexcepinfo);

     // Make sure the wFlags are legal
     if(wFlags & ~(DISPATCH_METHOD | DISPATCH_PROPERTYGET |
           DISPATCH_PROPERTYPUT | DISPATCH_PROPERTYPUTREF))
           return ResultFromScode(E_INVALIDARG);

     // This object only exposes a "default" interface.
     if(!IsEqualIID(riid, IID_NULL))
           return ResultFromScode(DISP_E_UNKNOWNINTERFACE);

     // It simplifies the following code if the caller
     // ignores the return value.
     if(puArgErr == NULL)
           puArgErr = &uArgErr;
     if(pvarResult == NULL)
           pvarResult = &varResultDummy;

     VariantInit(&varg0);

     // Assume the return type is void, unless we find otherwise.
     VariantInit(pvarResult);

     switch(dispidMember){
     case IDMEMBER_CPOINT_GETX:
           V_VT(pvarResult) = VT_I2;
           V_I2(pvarResult) = GetX();
           break;

     case IDMEMBER_CPOINT_SETX:
           hresult = DispGetParam(pdispparams, 0, VT_I2, &varg0, puArgErr);
           if(hresult != NOERROR)
                 return hresult;
           SetX(V_I2(&varg0));
           break;

     case IDMEMBER_CPOINT_GETY:
           V_VT(pvarResult) = VT_I2;
           V_I2(pvarResult) = GetY();
           break;

     case IDMEMBER_CPOINT_SETY:
           hresult = DispGetParam(pdispparams, 0, VT_I2, &varg0, puArgErr);
           if(hresult != NOERROR)
           return hresult;
           SetY(V_I2(&varg0));
           break;

     default:
           return ResultFromScode(DISP_E_MEMBERNOTFOUND);
     }
     return NOERROR;
}
```

See Also

CreateStdDispatch, IDispatch::Invoke

DispInvoke

HRESULT DispInvoke(_*this, ptinfo, dispidMember, wFlags, pparams, pvarResult, pexcepinfo, puArgErr*)
void FAR* *_this*
ITypeInfo FAR* *ptinfo*
DISPID *dispidMember*
unsigned short *wFlags*
DISPPARAMS FAR* *pparams*
VARIANT FAR* *pvarResult*
EXCEPINFO *pexcepinfo*
unsigned int FAR* *puArgErr*

Automatically calls member functions on an interface, given type information for the interface. You can describe an interface with type information and implement **IDispatch::Invoke** for the interface using this single call.

Parameters

_this
Pointer to an implementation of the **IDispatch** interface described by *ptinfo*.

ptinfo
Pointer to the type information describing the interface.

dispidMember
Identifies the member. Use **GetIDsOfNames** or the object's documentation to obtain the dispatch ID.

wFlags
Flags describing the context of the **Invoke** call, as follows:

Value	Description
DISPATCH_METHOD	The member is being invoked as a method. If a property has the same name, both this and the DISPATCH_PROPERTYGET flag may be set.
DISPATCH_PROPERTYGET	The member is being retrieved as a property or data member.
DISPATCH_PROPERTYPUT	The member is being changed as a property or data member.
DISPATCH_PROPERTYPUTREF	The member is being changed via a reference assignment, rather than a value assignment. This flag is valid only when the property accepts a reference to an object.

pparams
Pointer to a structure containing an array of arguments, an array of argument dispatch IDs for named arguments, and counts for number of elements in the arrays.

pvarResult
Pointer to where the result is to be stored, or NULL if the caller expects no result. This argument is ignored if DISPATCH_PROPERTYPUT or DISPATCH_PROPERTYPUTREF is specified.

pexcepinfo
Pointer to a structure containing exception information. This structure should be filled in if DISP_E_EXCEPTION is returned.

puArgErr

The index within *rgvarg* of the first argument that has an error. Arguments are stored in *pdispparams->rgvarg* in reverse order, so the first argument is the one with the highest index in the array. This parameter is returned only when the resulting SCODE is DISP_E_TYPEMISMATCH or DISP_E_PARAMNOTFOUND.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
DISP_E_BADPARAMCOUNT	The number of elements provided in DISPPARAMS is different from the number of arguments accepted by the method or property.
DISP_E_BADVARTYPE	One of the arguments in DISPPARAMS is not a valid variant type.
DISP_E_EXCEPTION	The application needs to raise an exception. In this case, the structure passed in *pexcepinfo* should be filled in.
DISP_E_MEMBERNOTFOUND	The requested member does not exist.
DISP_E_NONAMEDARGS	This implementation of **IDispatch** does not support named arguments.
DISP_E_OVERFLOW	One of the arguments in DISPPARAMS could not be coerced to the specified type.
DISP_E_PARAMNOTFOUND	One of the parameter IDs does not correspond to a parameter on the method. In this case *puArgErr* is set to the first argument that contains the error.
DISP_E_PARAMNOTOPTIONAL	A required parameter was omitted.
DISP_E_TYPEMISMATCH	One or more of the arguments could not be coerced. The index within *rgvarg* of the first parameter with the incorrect type is returned in *puArgErr*.
E_INVALIDARG	One of the arguments is invalid.
E_OUTOFMEMORY	Insufficient memory to complete operation.
Other returns	Any of the **ITypeInfo::Invoke** errors may also be returned.

Comments

The parameter *_this* is a pointer to an implementation of the interface being deferred to. **DispInvoke** builds a stack frame, coerces parameters using standard coercion rules, pushes them on the stack, and calls the correct member function in the virtual function table (VTBL).

Example

The following code from the Lines sample file LINES.CPP implements **IDispatch::Invoke** using **DispInvoke**. This function uses `m_bRaiseException` to signal that an error occurred during the **DispInvoke** call.

```
STDMETHODIMP
CLines::Invoke(
           DISPID dispidMember,
           REFIID riid,
           LCID lcid,
           WORD wFlags,
           DISPPARAMS FAR* pdispparams,
           VARIANT FAR* pvarResult,
           EXCEPINFO FAR* pexcepinfo,
           UINT FAR* puArgErr)
{
      HRESULT hr;

      if (NULL == pexcepinfo)
            return ResultFromScode(E_INVALIDARG);

      m_bRaiseException = FALSE;
      hr =  DispInvoke(
            this, m_ptinfo,
            dispidMember, wFlags, pdispparams,
            pvarResult, pexcepinfo, puArgErr);
      if (m_bRaiseException)
      {
            _fmemcpy(pexcepinfo, &m_excepinfo, sizeof(EXCEPINFO));
            return ResultFromScode(DISP_E_EXCEPTION);
      }
      else return hr;
}
```

See Also

CreateStdDispatch, **IDispatch::Invoke**

Active Object Registration Functions

Implemented by	Used by	Header filename	Import library name
OLEAUT32.DLL (32-bit systems) OLE2DISP.DLL (16-bit systems)	Applications that expose or access programmable objects.	OLEAUTO.H DISPATCH.H	OLEAUT32.LIB OLE2DISP.LIB

These functions let you identify a running instance of an object. Because they use the OLE object table (**GetRunningObjectTable**), they also require either OLE32.DLL (for 32-bit systems) or OLE2.DLL (for 16-bit systems).

When an application is started with the **/Automation** switch, it should initialize its Application object as the active object by calling **RegisterActiveObject** after it initializes OLE.

Applications may also initialize other top-level objects as the active object. For example, an application that exposes a Document object may want to let OLE Automation controllers retrieve and modify the currently active document.

For more information on initializing the active object, see Chapter 2, "Exposing OLE Automation Objects."

GetActiveObject

HRESULT GetActiveObject(*rclsid*, *pvreserved*, *ppunk***)**
    REFCLSID *rclsid*
    void FAR* *pvreserved*
    IUnknown FAR* FAR* *ppunk*

Retrieves a pointer to a running application initialized with OLE.

Parameters

rclsid
    Pointer to the class ID of the active object from the OLE registration database.

pvreserved
    Reserved for future use. Must be NULL.

ppunk
    On return, a pointer to the requested active object.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success
Other returns	Failure

RegisterActiveObject

HRESULT RegisterActiveObject (*punk*, *rclsid*, *pvreserved*, *pdwRegister***)**
IUnknown FAR* *punk*
REFCLSID *rclsid*
DWORD *dwFlags*
unsigned long FAR* *pdwRegister*

Registers an object as the active object for its class.

Parameters

punk
Pointer to the **IUnknown** interface of the active object.

rclsid
Pointer to the class ID of the active object.

dwFlags
Flags controlling registration of the object. Possible values are ACTIVEOBJECT_STRONG and ACTIVEOBJECT_WEAK.

pdwRegister
On return, pointer to a handle. You must pass this handle to **RevokeActiveObject** to end the object's status as active.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success
Other returns	Failure

Comments

The **RegisterActiveObject** function registers the object to which *punk* points as the active object for the class denoted by *rclsid.* Registration causes the object to be listed in OLE's running object table, a globally accessible lookup table that keeps track of the objects that are currently running on your computer. (See the *OLE 2 Programmer's Reference*, *Volume 1* for more information on the running object table.) The *dwFlags* parameter specifies the strength or weakness of the registration, which affects the way the object is shut down.

In general, OLE Automation objects should behave in the following manner:

- If the object is visible, it should shut down only in response to an explicit user command (such as the Exit command from the File menu), or to the equivalent command from an OLE Automation controller (invoking the Quit or Exit method on the Application object).
- If the object is not visible, it should shut down only when the last external connection to it disappears.

Strong registration performs an **AddRef** on the object, thereby incrementing the reference count of the object (and its associated stub) in the running object table. A strongly registered object must be explicitly revoked from the table with **RevokeActiveObject**. Strong registration (ACTIVEOBJECT_STRONG) is the default.

Weak registration keeps a pointer to the object in the running object table, but does not increment the reference count. Consequently, when the last external connection to a weakly registered object disappears, OLE releases the object's stub and the object itself is no longer available.

To ensure the desired behavior, you must consider not only OLE's default actions, but also the following:

- Even though your code creates an invisible object, the object may become visible at some later time. Once the object is visible, it should remain visible and active until it receives an explicit command to shut down, which may occur after references from your code disappear.
- Other OLE Automation controllers may be using the object. If so, your code should not force the object to shut down.

To avoid possible conflicts, OLE Automation objects should always register with ACTIVEOBJECT_WEAK, and call **CoLockObjectExternal** when necessary to guarantee that the object remains active. **CoLockObjectExternal** adds a strong lock, thereby preventing the object's reference count from reaching zero. For detailed information on this function, refer to the *OLE 2 Programmer's Reference*, *Volume 1*.

Most commonly, objects need to call **CoLockObjectExternal** when they become visible, so that they remain active until the user requests shutdown.

{ewl msdncd, EWGraphic, group10457 1 /a "SDK.BMP"} To shut down correctly, your code should follow these steps

1. When the object becomes visible, make the following call to add a lock on behalf of the user:

   ```
   CoLockObjectExternal(punk, TRUE, TRUE)
   ```

   The lock should remain in effect until the user explicitly requests shutdown, such as with a Quit or Exit command.
2. When the user requests shutdown, call **CoLockObjectExternal** again to free the lock, as follows:

   ```
   CoLockObjectExternal(punk, FALSE, TRUE)
   ```

3. Call **RevokeActiveObject** to make the object inactive.
4. To end all connections from remote processes, call **CoDisconnectObject** as follows:

   ```
   CoDisconnectObject(punk, 0)
   ```

   This function is described in the *OLE 2 Programmer's Reference*, *Volume 1*.

RevokeActiveObject

HRESULT RevokeActiveObject (*dwRegister*, *pvreserved*)
unsigned long *dwRegister*
void FAR* *pvreserved*

Ends an object's status as active.

Parameters

dwRegister
A handle previously returned by **RegisterActiveObject**.

pvreserved
Reserved for future use. Must be NULL.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success
Other returns	Failure

IEnumVARIANT Interface

Implemented by	Used by	Header filename
Applications that expose collections of objects.	Applications that access collections of objects.	OLEAUTO.H (32-bit systems) DISPATCH.H (16-bit systems)

The **IEnumVARIANT** interface provides a method for enumerating a collection of variants, including heterogeneous collections of objects and intrinsic types. Callers of this interface need not know the specific type, or types, of the elements in the collection.

The following is the definition that results from expanding the parameterized type **IEnumVARIANT**:

```
 interface IEnumVARIANT : IUnknown {
     virtual HRESULT Next(unsigned long celt,
                                    VARIANT FAR* rgvar,
                                    unsigned long FAR* pceltFetched) =
0;
     virtual HRESULT Skip(unsigned long celt) = 0;
     virtual HRESULT Reset() = 0;
     virtual HRESULT Clone(IEnumVARIANT FAR* FAR* ppenum) = 0;
     };
```

See the sample file ENUMVAR.CPP in the Lines sample to see how to implement a collection of objects using **IEnumVARIANT**.

IEnumVARIANT::Clone

HRESULT IEnumVARIANT::Clone(*ppenum***)**
   IEnumVARIANT FAR* FAR* *ppenum*

Creates a copy of the current enumeration state.

Parameter

ppenum
   On return, pointer to the location of the clone enumerator.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Comments

Using this function, you can record a particular point in the enumeration sequence, then return to it at a later time. The enumerator returned is of the same actual interface as the one that is being cloned.

There is no guarantee that exactly the same set of variants will be enumerated the second time as was enumerated the first. Although an exact duplicate is desirable, the outcome depends on the collection being enumerated. Some collections (for example, an enumeration of the files in a directory) will find it impractical to maintain this condition.

Example

The following code implements **IEnumVariant::Clone** for collections in the Lines sample (ENUMVAR.CPP):

```
STDMETHODIMP
CEnumVariant::Clone(IEnumVARIANT FAR* FAR* ppenum)
{
     CEnumVariant FAR* penum = NULL;
     HRESULT hr;

     *ppenum = NULL;

     hr = CEnumVariant::Create(m_psa, m_cElements, &penum);
     if (FAILED(hr))
           goto error;
     penum->AddRef();
     penum->m_lCurrent = m_lCurrent;

     *ppenum = penum;
     return NOERROR;

error:
     if (penum)
           penum->Release();
     return hr;
}
```

IEnumVARIANT::Next

HRESULT IEnumVARIANT::Next(*celt, rgvar, pceltFetched***)**
unsigned long *celt*
VARIANT FAR* *rgvar*
unsigned long FAR* *pceltFetched*

Attempts to get the next *celt* items in the enumeration sequence and return them through the array pointed to by *rgvar*.

Parameters

celt
The number of elements to be returned.

rgvar
An array of at least size *celt* in which the elements are to be returned.

pceltFetched
Pointer to the number of elements returned in *rgvar*, or NULL.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	The number of elements returned is *celt.*
S_FALSE	The number of elements returned is less than *celt*.

Comments

If fewer than the requested number of elements remain in the sequence, return only the remaining elements; the actual number of elements returned is passed through **pceltFetched* (unless it is NULL).

Example

The following code implements **IEnumVariant::Next** for collections in the Lines sample (ENUMVAR.CPP):

```
STDMETHODIMP
CEnumVariant::Next(ULONG cElements, VARIANT FAR* pvar, ULONG FAR*
pcElementFetched)
{
      HRESULT hr;
      ULONG l;
      long l1;
      ULONG l2;

      if (pcElementFetched != NULL)
            *pcElementFetched = 0;

      for (l=0; l<cElements; l++)
            VariantInit(&pvar[l]);

      // Retrieve the next cElements elements.
      for (l1=m_lCurrent, l2=0; l1<(long)(m_lLBound+m_cElements) &&
            l2<cElements; l1++, l2++)
      {
            hr = SafeArrayGetElement(m_psa, &l1, &pvar[l2]);
            if (FAILED(hr))
                  goto error;
      }
      // Set count of elements retrieved
      if (pcElementFetched != NULL)
            *pcElementFetched = l2;
      m_lCurrent = l1;

      return  (l2 < cElements) ? ResultFromScode(S_FALSE) : NOERROR;

 error:
      for (l=0; l<cElements; l++)
            VariantClear(&pvar[l]);
      return hr;
 }
```

IEnumVARIANT::Reset

HRESULT IEnumVARIANT::Reset()

Resets the enumeration sequence to the beginning.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success
S_FALSE	Failure

Comments

There is no guarantee that exactly the same set of variants will be enumerated the second time as was enumerated the first. Although an exact duplicate is desirable, the outcome depends on the collection being enumerated. Some collections (for example, an enumeration of the files in a directory) will find it impractical to maintain this condition.

Example

The following code implements **IEnumVariant::Reset** for collections in the Lines sample (ENUMVAR.CPP):

```
STDMETHODIMP
CEnumVariant::Reset()
{
     m_lCurrent = m_lLBound;
     return NOERROR;
}
```

IEnumVARIANT::Skip

HRESULT IEnumVARIANT::Skip(*celt***)**
  unsigned long *celt*

Attempts to skip over the next *celt* elements in the enumeration sequence.

Parameter

celt
  The number of elements to skip.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	The specified number of elements was skipped.
S_FALSE	The end of the sequence was reached before skipping the requested number of elements.

Example

The following code implements **IEnumVariant::Reset** for collections in the Lines sample (ENUMVAR.CPP):

```
STDMETHODIMP
CEnumVariant::Skip(ULONG cElements)
{
     m_lCurrent += cElements;
     if (m_lCurrent > (long)(m_lLBound+m_cElements))
     {
           m_lCurrent =  m_lLBound+m_cElements;
           return ResultFromScode(S_FALSE);
     }
     else return NOERROR;
}
```

Data Manipulation Functions

Data manipulation functions access and manipulate the array, string, and variant data types used by OLE Automation.

Overview of Data Manipulation Functions

The data manipulation functions are summarized in the following table.

Category	Function name	Purpose
Array manipulation	SafeArrayAccessData	Increments the lock count of an array and returns a pointer to array data.
	SafeArrayAllocData	Allocates memory for a safe array based on a descriptor created with SafeArrayAllocDescriptor.
	SafeArrayAllocDescriptor	Allocates memory for a safe array descriptor.
	SafeArrayCopy	Copies an existing array.
	SafeArrayCreate	Creates a new array descriptor.
	SafeArrayDestroy	Destroys an array descriptor.
	SafeArrayDestroyData	Frees memory used by the data elements in a safe array.
	SafeArrayDestroyDescriptor	Frees memory used by a safe array descriptor.
	SafeArrayGetDim	Returns the number of dimensions in an array.
	SafeArrayGetElement	Retrieves an element of an array.
	SafeArrayGetElemsize	Returns the size of an element.
	SafeArrayGetLBound	Retrieves the lower bound for a given dimension.
	SafeArrayGetUBound	Retrieves the upper bound for a given dimension.
	SafeArrayLock	Increments the lock count of an array.
	SafeArrayPtrOfIndex	Returns a pointer to an array element.
	SafeArrayPutElement	Assigns an element into an array.
	SafeArrayRedim	Resizes a safe array.
	SafeArrayUnAccessData	Frees a pointer to array data and decrements the lock count of the array.
	SafeArrayUnlock	Decrements the lock count of an array.
String manipulation	SysAllocString	Creates and initializes a string.
	SysAllocStringByteLen	Creates a zero-terminated string of a specified length (32-bit only).
	SysAllocStringLen	Creates a string of a specified length.

	SysFreeString	Frees a previously created string.
	SysReAllocString	Changes the size and value of a string.
	SysReAllocStringLen	Changes the size of an existing string.
	SysStringByteLen	Returns the length of a string in bytes (32-bit only).
	SysStringLen	Returns the length of a string.
Variant manipulation	**VariantChangeType**	Converts a variant to another type.
	VariantChangeTypeEx	Converts a variant to another type using a locale ID.
	VariantClear	Releases resources and sets a variant to VT_EMPTY.
	VariantCopy	Copies a variant.
	VariantCopyInd	Copies variants that may contain a pointer.
	VariantInit	Initializes a variant.
	Low-level conversion functions	Converts specific types of variants to other variant types. These functions are called by **VariantChangeType** and **VariantChangeTypeEx**.
Time conversion	**DosDateTimeToVariantTime**	Converts MS-DOS date and time representations to a variant time.
	VariantTimeToDosDateTime	Converts a variant time to MS-DOS date and time representations.

Array Manipulation Functions

Implemented by	Used by	Header filename	Import library name
OLEAUT32.DLL (32-bit systems) OLE2DISP.DLL (16-bit systems)	Applications that expose or access programmable objects.	OLEAUTO.H DISPATCH.H	OLEAUT32.LIB OLE2DISP.LIB

The arrays passed by IDispatch::Invoke within VARIANTARGs are called *safe arrays*. Safe arrays contain information about the number of dimensions and bounds within them. When an array is an argument or the return value of a function, the *parray* field of VARIANTARG points to an array descriptor. This array descriptor should not be accessed directly unless you are creating arrays containing elements with nonvariant data types. Instead, use the functions SafeArrayAccessData and SafeArrayUnaccessData to access the data.

The base type of the array is indicated by VT_ tag | VT_ARRAY. The data referenced by an array descriptor is stored in column-major order, which is the same ordering scheme used by Visual Basic and FORTRAN, but different from C and Pascal. *Column-major order* means that the leftmost dimension (as specified in a programming language syntax) changes first.

Following are the definitions of the safe array descriptor and the functions to use when accessing the data in the descriptor and the array itself.

Data Types and Structures

SAFEARRAY
SAFEARRAYBOUND

SAFEARRAY

```
typedef struct FARSTRUCT tagSAFEARRAY {
  unsigned short cDims;            // Count of dimensions in this array.
  unsigned short fFeatures;  // Flags used by the SafeArray
                                         // routines documented below:
#if defined(WIN32)
  unsigned long cbElements;  // Size of an element of the array;
                                         // Does not include size of
                                         // pointed-to data.
  unsigned long cLocks;      // Number of times the array has been
                                         // locked without corresponding
  unlock.
#else
  unsigned short cbElements;
  unsigned short cLocks;
  unsigned long handle;      // Unused but kept for compatibility
#endif
  void HUGEP* pvData;                    // Pointer to the data.
  SAFEARRAYBOUND rgsabound[1];           // One bound for each dimension.
} SAFEARRAY;
```

Note that the definition varies, depending on the target operating system platform. On 32-bit Windows systems, both the *cbElements* and *cLocks* parameters are unsigned long integers, and the *handle* parameter is omitted. On 16-bit Windows systems, *cbElements* and *cLocks* are unsigned short integers, and the *handle* parameter has been retained for compatibility with earlier software.

The array *rgsabound* is stored with the leftmost dimension in *rgsabound*[0] and the rightmost dimension in *rgsabound*[*cDims* - 1]. If an array were specified in C-like syntax as a[2][5], it would have two elements in the *rgsabound* vector. Element 0 has an *lLbound* of 0 and a *cElements* of 2. Element 1 has an *lLbound* of 0 and a *cElements* of 5.

The *fFeatures* flags describe attributes of an array that may affect how the array is released. This allows freeing the array without referencing its containing variant. The bits are accessed using the following constants:

```
#define FADF_AUTO      0x0001      // Array is allocated on the stack.
#define FADF_STATIC    0x0002      // Array is statically allocated.
#define FADF_EMBEDDED  0x0004      // Array is embedded in a structure.
#define FADF_FIXEDSIZE 0x0010      // Array may not be resized or
                                               // reallocated.
#define FADF_BSTR      0x0100      // An array of BSTRs.
#define FADF_UNKNOWN   0x0200      // An array of IUnknown*.
#define FADF_DISPATCH  0x0400      // An array of IDispatch*.
#define FADF_VARIANT   0x0800      // An array of VARIANTs.
#define FADF_RESERVED  0xF0E8      // Bits reserved for future use.
```

SAFEARRAYBOUND

```
typedef struct tagSAFEARRAYBOUND {
  unsigned long cElements;
  long lLbound;
} SAFEARRAYBOUND;
```

This structure represents the bounds of one dimension of the array. The lower bound of the dimension is represented by *lLbound*, and *cElements* represents the number of elements in the dimension.

SafeArrayAccessData

HRESULT SafeArrayAccessData (*psa, ppvdata*)
  SAFEARRAY FAR* *psa*
  void HUGEP* FAR* *ppvdata*

Increments the lock count of an array and retrieves a pointer to the array data.

Parameters

psa
  Pointer to an array descriptor created by **SafeArrayCreate**.

ppvdata
  On exit, pointer to a pointer to the array data. Note that arrays may be larger than 64K, so huge pointers must be used in Windows version 3.1 or later.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
E_INVALIDARG	The argument *psa* was not a valid safe array descriptor.
E_UNEXPECTED	The array could not be locked.

SafeArrayAllocData

HRESULT SafeArrayAllocData(*psa*)
SAFEARRAY FAR* *psa*

Allocates memory for a safe array based on a descriptor created with **SafeArrayAllocDescriptor**.

Parameters

psa
Pointer to an array descriptor created by **SafeArrayAllocDescriptor**.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_INVALIDARG	The argument *psa* was not a valid safe array descriptor.
E_UNEXPECTED	The array could not be locked.

Example

The following example creates a safe array using the **SafeArrayAllocDescriptor** and **SafeArrayAllocData** functions.

```
SAFEARRAY FAR* FAR*ppsa;
unsigned int ndim = 2;
HRESULT hresult = SafeArrayAllocDescriptor( ndim, ppsa );
if( FAILED( hresult ) )
  return ERR_OutOfMemory;
(*ppsa)->rgsabound[ 0 ].lLbound = 0;
(*ppsa)->rgsabound[ 0 ].cElements = 5;
(*ppsa)->rgsabound[ 1 ].lLbound = 1;
(*ppsa)->rgsabound[ 1 ].cElements = 4;
hresult = SafeArrayAllocData( *ppsa );
if( FAILED( hresult ) ) {
  SafeArrayDestroyDescriptor( *ppsa )
  return ERR_OutOfMemory;
}
```

See Also

SafeArrayAllocData, **SafeArrayDestroyData**, **SafeArrayDestroyDescriptor**

SafeArrayAllocDescriptor

HRESULT SafeArrayAllocDescriptor(*cDims, ppsaOut*)
unsigned int *cDims*
SAFEARRAY FAR* FAR* *ppsaOut*

Allocates memory for a safe array descriptor.

Parameters

cDims
The number of dimensions of the array.

ppsaOut
Pointer to a location in which to store the created array descriptor.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_INVALIDARG	The argument *psa* was not a valid safe array descriptor.
E_UNEXPECTED	The array could not be locked.

Comments

This function lets you create safe arrays containing elements with data types other than those provided by **SafeArrayCreate**. After creating an array descriptor using **SafeArrayAllocDescriptor**, set the element size in the array descriptor and call **SafeArrayAllocData** to allocate memory for the array elements.

Example

The following example creates a safe array using the **SafeArrayAllocDescriptor** and **SafeArrayAllocData** functions.

```
SAFEARRAY FAR* FAR*ppsa;
unsigned int ndim = 2;
HRESULT hresult = SafeArrayAllocDescriptor( ndim, ppsa );
if( FAILED( hresult ) )
  return ERR_OutOfMemory;
(*ppsa)->rgsabound[ 0 ].lLbound = 0;
(*ppsa)->rgsabound[ 0 ].cElements = 5;
(*ppsa)->rgsabound[ 1 ].lLbound = 1;
(*ppsa)->rgsabound[ 1 ].cElements = 4;
hresult = SafeArrayAllocData( *ppsa );
if( FAILED( hresult ) ) {
  SafeArrayDestroyDescriptor( *ppsa )
  return ERR_OutOfMemory;
}
```

See Also

SafeArrayAllocData, **SafeArrayDestroyData**, **SafeArrayDestroyDescriptor**

SafeArrayCopy

HRESULT SafeArrayCopy(*psa, ppsaOut***)**
  SAFEARRAY FAR* *psa*
  SAFEARRAY FAR* FAR* *ppsaOut*

Creates a copy of an existing safe array.

Parameters

psa
  Pointer to an array descriptor created by **SafeArrayCreate**.

ppsaOut
  Pointer to a location in which to return the new array descriptor.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
E_INVALIDARG	The argument *psa* was not a valid safe array descriptor.
E_OUTOFMEMORY	Insufficient memory to create the copy.

Comments

SafeArrayCopy calls the string or variant manipulation functions if the array to copy contains either of those data types. If the array being copied contains object references, the reference counts for those objects are incremented.

See Also

SysAllocStringLen, **VariantCopy**, **VariantCopyInd**.

SafeArrayCreate

SAFEARRAY * SafeArrayCreate(*vt, cDims, rgsabound***)**
    VARTYPE *vt*
    unsigned int *cDims*
    SAFEARRRAYBOUND FAR* *rgsabound*

Creates a new array descriptor, allocates and initializes the data for the array, and returns a pointer to the new array descriptor.

Parameters

vt

The base type of the array (that is, the VARTYPE of each element of the array). The VARTYPE is restricted to a subset of the variant types. Neither the VT_ARRAY nor the VT_BYREF flag can be set. VT_EMPTY and VT_NULL are not valid base types for the array. All other types are legal.

cDims

Number of dimensions in the array. This can't be changed after the array is created.

rgsabound

Pointer to a vector of bounds (one for each dimension) to allocate for the array.

Return Value

Points to the array descriptor, or NULL if the array could not be created.

Example

```
HRESULT PASCAL __export CPoly::EnumPoints(IEnumVARIANT FAR* FAR* ppenum)
{
  unsigned int i;
  HRESULT hresult;
  VARIANT var;
  SAFEARRAY FAR* psa;
  CEnumPoint FAR* penum;
  POINTLINK FAR* ppointlink;
  SAFEARRAYBOUND rgsabound[1];

  rgsabound[0].lLbound = 0;
  rgsabound[0].cElements = m_cPoints;

  psa = SafeArrayCreate(VT_VARIANT, 1, rgsabound);
  if(psa == NULL){
    hresult = ReportResult(0, E_OUTOFMEMORY, 0, 0);
    goto LError0;
  }
.
.    // Code omitted here.
.
LError0:;
  return hresult;
}
```

SafeArrayDestroy

HRESULT SafeArrayDestroy(*psa***)**
  SAFEARRAY FAR* *psa*

Destroys an existing array descriptor and all the data in the array. If objects are stored in the array, **Release** is called on each object in the array.

Parameters

psa
  Pointer to an array descriptor created by **SafeArrayCreate**.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
DISP_E_ARRAYISLOCKED	The array is currently locked.
E_INVALIDARG	The item pointed to by *psa* is not a safe array descriptor.

Example

```
STDMETHODIMP_(ULONG) CEnumPoint::Release()
{
  if(--m_refs == 0){
    if(m_psa != NULL)
    SafeArrayDestroy(m_psa);
    delete this;
    return 0;
  }
  return m_refs;
}
```

SafeArrayDestroyData

HRESULT SafeArrayDestroyData(*psa***)**
    SAFEARRAY FAR* *psa*

Destroys all the data in a safe array. If objects are stored in the array, **Release** is called on each object in the array.

Parameters

psa
    Pointer to an array descriptor.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
DISP_E_ARRAYISLOCKED	The array is currently locked.
E_INVALIDARG	The item pointed to by *psa* is not a safe array descriptor.

Comments

This function is typically used when freeing safe arrays that contain elements with data types other than variants.

See Also

SafeArrayAllocData, **SafeArrayAllocDescriptor**, **SafeArrayDestroyDescriptor**

SafeArrayDestroyDescriptor

HRESULT SafeArrayDestroyDescriptor(*psa***)**
SAFEARRAY FAR* *psa*

Destroys a descriptor of a safe array.

Parameters

psa
Pointer to a safe array descriptor.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
DISP_E_ARRAYISLOCKED	The array is currently locked.
E_INVALIDARG	The item pointed to by *psa* is not a safe array descriptor.

Comments

This function is typically used to destroy the descriptor of a safe array that contains elements with data types other than variants. Destroying the array descriptor does not destroy the elements in the array. Call **SafeArrayDestroyData** to free the elements before destroying the array descriptor.

See Also

SafeArrayAllocData, **SafeArrayAllocDescriptor**, **SafeArrayDestroyData**

SafeArrayGetDim

unsigned int SafeArrayGetDim(*psa***)**
SAFEARRAY FAR* *psa*

Returns the number of dimensions in the array.

Parameters

psa
Pointer to an array descriptor created by **SafeArrayCreate**.

Return Value

Returns the number of dimensions in the array.

Example

```
HRESULT
CEnumPoint::Create(SAFEARRAY FAR* psa, CEnumPoint FAR* FAR* ppenum)
{
  long lBound;
  HRESULT hresult;
  CEnumPoint FAR* penum;

  // Verify that the SafeArray is the proper shape.
  //
  if(SafeArrayGetDim(psa) != 1)
    return ReportResult(0, E_INVALIDARG, 0, 0);
.
.    // Code omitted here.
.
}
```

SafeArrayGetElement

HRESULT SafeArrayGetElement(*psa, rgIndices, pvData***)**
  SAFEARRAY FAR* *psa*
  long FAR* *rgIndices*
  void FAR* *pvData*

Retrieves a single element of the array.

Parameters

psa

Pointer to an array descriptor created by **SafeArrayCreate**.

rgIndices

Pointer to a vector of indexes for each dimension of the array. The rightmost (least significant) dimension is *rgIndices*[0]. The leftmost dimension is stored at *rgIndices*[*psa*->*cDims* - 1].

pvData

Pointer to the location to place the element of the array.

Comments

This function automatically calls **SafeArrayLock** and **SafeArrayUnlock** before and after retrieving the element. The caller must provide a storage area of the correct size to receive the data. If the data element is a string, object, or variant, the function copies the element in the correct way.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
DISP_E_BADINDEX	The specified index is invalid.
E_INVALIDARG	One of the arguments is invalid.
E_OUTOFMEMORY	Memory could not be allocated for the element.

Example

```
STDMETHODIMP CEnumPoint::Next(
  ULONG celt,
  VARIANT FAR rgvar[],
  ULONG FAR* pceltFetched)
{
  unsigned int i;
  long ix;
  HRESULT hresult;

  for(i = 0; i < celt; ++i)
    VariantInit(&rgvar[i]);

  for(i = 0; i < celt; ++i){
    if(m_iCurrent == m_celts){
    hresult = ReportResult(0, S_FALSE, 0, 0);
    goto LDone;
  }

    ix = m_iCurrent++;
    hresult = SafeArrayGetElement(m_psa, &ix, &rgvar[i]);
    if(FAILED(hresult))
    goto LError0;
  }
  hresult = NOERROR;

LDone:;
  *pceltFetched = i;
  return hresult;

LError0:;
  for(i = 0; i < celt; ++i)
    VariantClear(&rgvar[i]);
  return hresult;
}
```

SafeArrayGetElemsize

unsigned int SafeArrayGetElemsize(*psa***)**
SAFEARRAY FAR* *psa*

Returns the size, in bytes, of the elements of a safe array.

Parameters

psa
Pointer to an array descriptor created by **SafeArrayCreate**.

SafeArrayGetLBound

HRESULT SafeArrayGetLBound(*psa, nDim, plLbound***)**
  SAFEARRAY FAR* *psa*
  unsigned int *nDim*
  long FAR* *plLbound*

Returns the lower bound for any dimension of a safe array.

Parameters

psa
Pointer to an array descriptor created by **SafeArrayCreate**.

nDim
The array dimension for which to get the lower bound.

plLbound
Pointer to the location to return the lower bound.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
DISP_E_BADINDEX	The specified index is out of bounds.
E_INVALIDARG	One of the arguments is invalid.

Example

```
HRESULT
CEnumPoint::Create(SAFEARRAY FAR* psa, CEnumPoint FAR* FAR* ppenum)
{
  long lBound;
  HRESULT hresult;
  CEnumPoint FAR* penum;


  // Verify that the SafeArray is the proper shape.
  //
  hresult = SafeArrayGetLBound(psa, 1, &lBound);
  if(FAILED(hresult))
    return hresult;
.
.    // Code omitted here.
.

}
```

SafeArrayGetUBound

HRESULT SafeArrayGetUBound(*psa, nDim, plUbound***)**
  SAFEARRAY FAR* *psa*
  unsigned int *nDim*
  long FAR* *plUbound*

Returns the upper bound for any dimension of a safe array.

Parameters

psa
  Pointer to an array descriptor created by **SafeArrayCreate()**.

nDim
  The array dimension for which to get the upper bound.

plUbound
  Pointer to the location to return the upper bound.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
DISP_E_BADINDEX	The specified index is out of bounds.
E_INVALIDARG	One of the arguments is invalid.

Example

```
HRESULT
CEnumPoint::Create(SAFEARRAY FAR* psa, CEnumPoint FAR* FAR* ppenum)
{
  long lBound;
  HRESULT hresult;
  CEnumPoint FAR* penum;

  // Verify that the SafeArray is the proper shape.
  //
  hresult = SafeArrayGetUBound(psa, 1, &lBound);
  if(FAILED(hresult))
    goto LError0;
.
.    // Code omitted
.
LError0:;
  penum->Release();

  return hresult;
}
```

SafeArrayLock

HRESULT SafeArrayLock(*psa***)**
SAFEARRAY FAR* *psa*

Increments the lock count of an array and places a pointer to the array data in *pvData* of the array descriptor.

Parameters

psa
Pointer to an array descriptor created by **SafeArrayCreate**.

Comments

The pointer in the array descriptor is valid until **SafeArrayUnlock** is called. Calls to **SafeArrayLock** can be nested; an equal number of calls to **SafeArrayUnlock** are required.

An array can't be deleted while it is locked.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_INVALIDARG	The argument *psa* was not a valid safe array descriptor.
E_UNEXPECTED	The array could not be locked.

SafeArrayPtrOfIndex

HRESULT SafeArrayPtrOfIndex(*psa, rgIndices, ppvData***)**
SAFEARRAY FAR* *psa*
long FAR* *rgIndices*
void HUGEP* FAR* *ppvData*

Returns a pointer to an array element.

Parameters

psa
Pointer to an array descriptor created by **SafeArrayCreate**.

rgIndices
An array of index values that identify an element of the array. All indexes for the element must be specified.

ppvData
On return, pointer to the element identified by the values in *rgIndices*.

Comments

The array should be locked before **SafeArrayPtrOfIndex** is called. Failing to lock the array may cause unpredictable results.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
E_INVALIDARG	The argument *psa* was not a valid safe array descriptor.
DISP_E_BADINDEX	The specified index was invalid.

SafeArrayPutElement

HRESULT SafeArrayPutElement(*psa, rgIndices, pvData***)**
  SAFEARRAY FAR* *psa*
  long FAR* *rgIndices*
  void FAR* *pvData*

Assigns a single element into the array.

Parameters

psa
  Pointer to an array descriptor created by **SafeArrayCreate**.

rgIndices
  Pointer to a vector of indexes for each dimension of the array. The rightmost (least significant) dimension is *rgIndices*[0]. The leftmost dimension is stored at *rgIndices*[*psa*->*cDims* - 1].

pvData
  Pointer to the data to assign to the array. VT_DISPATCH, VT_UNKNOWN, and VT_BSTR variant types are pointers and don't require another level of indirection.

Comments

This function automatically calls **SafeArrayLock** and **SafeArrayUnlock** before and after assigning the element. If the data element is a string, object, or variant, the function copies it correctly. If the existing element is a string, object or variant, it is cleared correctly.

Note that you can have multiple locks on an array, so you can put elements into an array while the array is locked by other operations.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
DISP_E_BADINDEX	The specified index was invalid.
E_INVALIDARG	One of the arguments is invalid.
E_OUTOFMEMORY	Memory could not be allocated for the element.

Example

```
HRESULT PASCAL __export CPoly::EnumPoints(IEnumVARIANT FAR* FAR* ppenum)
{
     unsigned int i;
     HRESULT hresult;
     VARIANT var;
     SAFEARRAY FAR* psa;
     CEnumPoint FAR* penum;
     POINTLINK FAR* ppointlink;
     SAFEARRAYBOUND rgsabound[1];
     rgsabound[0].lLbound = 0;
     rgsabound[0].cElements = m_cPoints;

     psa = SafeArrayCreate(VT_VARIANT, 1, rgsabound);
.
.    // Code omitted here.
.
            V_VT(&var) = VT_DISPATCH;
            hresult = ppointlink->ppoint->QueryInterface(
            IID_IDispatch, (void FAR* FAR*)&V_DISPATCH(&var));
            if(hresult != NOERROR)
                  goto LError1;

            ix[0] = i;
            SafeArrayPutElement(psa, ix, &var);

            ppointlink = ppointlink->next;
     }

     hresult = CEnumPoint::Create(psa, &penum);
     if(hresult != NOERROR)
            goto LError1;
     *ppenum = penum;
     return NOERROR;

LError1:;
     SafeArrayDestroy(psa);

LError0:;
     return hresult;
}
```

SafeArrayRedim

HRESULT SafeArrayRedim(*psa, psaboundNew***)**
SAFEARRAY FAR* *psa*
SAFEARRAYBOUND FAR* *psaboundNew*

Changes the least significant (rightmost) bound of a safe array.

Parameters

psa
Pointer to an array descriptor.

psaboundNew
Pointer to a new safe array bound structure containing the new array bound. Only the least significant dimension of an array may be changed.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
DISP_E_ARRAYISLOCKED	The array is currently locked.
E_INVALIDARG	The item pointed to by *psa* is not a safe array descriptor.

Comments

If you reduce the bound of an array, **SafeArrayRedim** deallocates the array elements outside the new array boundary. If you increase the bound of an array, **SafeArrayRedim** allocates and initializes the new array elements. The data is preserved for elements that exist in both the old and the new array.

SafeArrayUnaccessData

HRESULT SafeArrayUnaccessData(*psa***)**
SAFEARRAY FAR* *psa*

Decrements the lock count of an array and invalidates the pointer retrieved by **SafeArrayAccessData**.

Parameters

psa
Pointer to an array descriptor created by **SafeArrayCreate**.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_INVALIDARG	The argument *psa* was not a valid safe array descriptor.
E_UNEXPECTED	The array could not be unlocked.

SafeArrayUnlock

HRESULT SafeArrayUnlock(*psa***)**
   SAFEARRAY FAR* *psa*

Decrements the lock count of an array so it can be freed or resized.

Parameters

psa
   Pointer to an array descriptor created by **SafeArrayCreate**.

Comments

This function is called after access to the data in an array is finished.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_INVALIDARG	The argument *psa* was not a valid safe array descriptor.
E_UNEXPECTED	The array could not be unlocked.

String Manipulation Functions

Implemented by	Used by	Header filename	Import library name
OLEAUT32.DLL (32-bit systems) OLE2DISP.DLL (16-bit systems)	Applications that expose or access programmable objects.	OLEAUTO.H DISPATCH.H	OLEAUT32.LIB OLE2DISP.LIB

To handle strings that are allocated by one component and freed by another, OLE Automation defines a special set of functions. These functions use the following data type:

```
typedef OLECHAR FAR* BSTR;
```

These strings are zero-terminated, and in most cases they can be treated just like OLECHAR * strings. However, a BSTR can be queried for its length rather than scanned, so it may contain embedded null characters. The length is stored as an integer at the memory location preceding the data in the string. Instead of reading this location directly, applications should use the string manipulation functions to access the length of a BSTR. In situations where you can be sure that a BSTR will not be translated from ANSI to Unicode or vice versa, you can use BSTRs to pass binary data. For example, if your code will run only on 16-bit systems and interact only with other 16-bit systems, you may use BSTRs. However, the preferred method of passing binary data is to use a SAFEARRAY of VT_UI1.

A NULL pointer is a valid value for a BSTR variable; by convention, it is always treated the same as a pointer to a BSTR that contains zero characters. Also, by convention, calls to functions that take a BSTR reference parameter must pass either a NULL pointer or a pointer to an allocated BSTR. If the implementation of a function that takes a BSTR reference parameter assigns a new BSTR to the parameter, it must free the previously referenced BSTR.

SysAllocString

BSTR SysAllocString(*sz*)
OLECHAR FAR* *sz*

Allocates a new string and copies the passed string into it. Returns NULL if insufficient memory exists or if NULL is passed in.

Parameter

sz
A zero-terminated string to copy.

Return Value

If successful, points to a BSTR containing the string. If insufficient memory exists or *sz* was NULL, returns NULL.

Strings created with **SysAllocString** should be freed with **SysFreeString**.

Example

```
inline void CStatBar::SetText(OLECHAR FAR* sz)
{
  SysFreeString(m_bstrMsg);
  m_bstrMsg = SysAllocString(sz);
}
```

SysAllocStringByteLen

BSTR SysAllocStringByteLen(*psz, len***)**
char FAR* *psz*
unsigned int *len*

Allocates a new string of *len* bytes, copies *len* bytes from the passed string into it, and then appends a null character. Valid for 32-bit systems only.

Parameter

psz
A zero-terminated string to copy, or NULL to keep the string uninitialized.

len
Number of bytes to copy from *psz*. A null character is placed afterwards, making a total of *len*+1 bytes allocated.

Return Value

Points to a copy of the string or NULL, if insufficient memory exists.

Comments

If *psz* is NULL, a string of the requested length is allocated, but not initialized. The string *psz* can contain embedded null characters and need not end with a null. The returned string should later be freed with **SysFreeString**.

SysAllocStringLen

BSTR SysAllocStringLen(*pch, cch***)**
  OLECHAR FAR* *pch*
  unsigned int *cch*

Allocates a new string, copies *cch* characters from the passed string into it, and then appends a null character.

Parameter

pch
  A pointer to *cch* characters to copy, or NULL to keep the string uninitialized.

cch
  Number of characters to copy from *pch**. A null character is placed afterwards, making a total of *cch*+1 characters allocated.

Return Value

Points to a copy of the string or NULL, if insufficient memory exists.

Comments

If *pch* is NULL, a string of the requested length is allocated, but not initialized. The string *pch* can contain embedded null characters and need not end with a null. The returned string should later be freed with **SysFreeString**.

SysFreeString

void SysFreeString(*bstr***)**
  BSTR *bstr*

Frees a string previously allocated by **SysAllocString**, **SysAllocStringByteLen**, **SysReAllocString**, **SysAllocStringLen**, or **SysReAllocStringLen**.

Parameter

bstr
  A BSTR allocated earlier or NULL. If NULL, the function simply returns.

Return Value

None.

Example

```
CStatBar::~CStatBar()
{
  SysFreeString(m_bstrMsg);
}
```

SysReAllocString

BOOL SysReAllocString(*pbstr, sz***)**
  BSTR FAR* *pbstr*
  OLECHAR FAR* *sz*

Allocates a new BSTR and copies the passed string into it, then frees the BSTR currently referenced by *pbstr* and resets *pbstr* to point to the new BSTR. Returns False if insufficient memory exists.

Parameter

pbstr
  Points to a variable containing a BSTR.

sz
  A zero-terminated string to copy.

Return Value

Returns False if insufficient memory exists.

SysReAllocStringLen

BOOL SysReAllocStringLen(*pbstr, pch, cch***)**
BSTR FAR* *pbstr*
OLECHAR FAR* *pch*
unsigned int *cch*

Creates a new BSTR containing a specified number of characters from an old BSTR and frees the old BSTR.

Parameter

pbstr
Pointer to a variable containing a BSTR.

pch
Pointer to *cch* characters to copy, or NULL to keep the string uninitialized.

cch
Number of characters to copy from *pch**. A null character is placed afterwards, making a total of *cch*+1 characters allocated.

Return Value

Returns True if the string is successfully reallocated, or False if insufficient memory exists.

Comments

Allocates a new string, copies *cch* characters from the passed string into it, and then appends a null character. Frees the BSTR currently referenced by *pbstr* and resets *pbstr* to point to the new BSTR. If *pch* is NULL, a string of length *cch* is allocated, but not initialized.

The string *pch* can contain embedded null characters and need not end with a null.

SysStringByteLen

unsigned int SysStringByteLen(*bstr***)**
BSTR *bstr*

Returns the length in bytes of a BSTR. Valid for 32-bit systems only.

Parameter

bstr
A BSTR previously allocated. It cannot be NULL.

Return Value

The number of bytes in *bstr*, not including a terminating null character.

Comments

The value returned may be different from **fstrlen**(*bstr*), if the BSTR was allocated with **Sys[Re]AllocStringLen** or **SysAllocStringByteLen**, and the passed-in characters included a null character in the first *len* characters. For a BSTR allocated with **Sys[Re]AllocStringLen** or **SysAllocStringByteLen**, this function always returns the number of bytes specified in the *len* parameter at allocation time.

Example

```
// Draw the status message
//
TextOut(
     hdc,
     rcMsg.left + (m_dxFont / 2),
     rcMsg.top + ((rcMsg.bottom - rcMsg.top - m_dyFont) / 2),
     m_bstrMsg, SysStringByteLen(m_bstrMsg));
```

SysStringLen

unsigned int SysStringLen(*bstr***)**
BSTR *bstr*

Returns the length of a BSTR.

Parameter

bstr
A BSTR previously allocated; can't be NULL.

Return Value

The number of characters in *bstr*, not including a terminating null character.

Comments

The value returned may be different from **_fstrlen**(*bstr*), if the BSTR was allocated with **Sys[Re]AllocStringLen** or **SysAllocStringByteLen**, and the passed-in characters included a null character in the first *cch* characters. For a BSTR allocated with **Sys[Re]AllocStringLen** or **SysAllocStringByteLen**, this function always returns the number of characters specified in the *cch* parameter at allocation time.

Example

```
// Draw the status message
//
TextOut(
     hdc,
     rcMsg.left + (m_dxFont / 2),
     rcMsg.top + ((rcMsg.bottom - rcMsg.top - m_dyFont) / 2),
     m_bstrMsg, SysStringLen(m_bstrMsg));
```

Variant Manipulation Functions

Implemented by	Used by	Header filename	Import library name
OLEAUT32.DLL (32-bit systems) OLE2DISP.DLL (16-bit systems)	Applications that expose or access programmable objects.	OLEAUTO.H DISPATCH.H	OLEAUT32.LIB OLE2DISP.LIB

These functions are provided to allow applications to manipulate VARIANTARG variables. Applications that implement **IDispatch** should test each VARIANTARG for all permitted types by attempting to coerce the variant to each type (using **VariantChangeType** or **VariantChangeTypeEx**). If objects are allowed, the application should always test for object types before other types. If an object type is expected, the application must test (via **IUnknown::QueryInterface**) whether the object is of the type desired.

Although applications can access and interpret the VARIANTARGs without these functions, using them ensures uniform conversion and coercion rules for all implementors of **IDispatch**. (For example, these functions automatically coerce numeric arguments to strings and vice versa, when necessary.)

Because variants can contain strings, references to scalars and objects, and arrays, certain data ownership rules must be obeyed. All the variant manipulation functions conform to the following rules:

1. All VARIANTARGs must be initialized by **VariantInit** before use.
2. For the types VT_UI1, VT_I2, VT_I4, VT_R4, VT_R8, VT_BOOL, VT_ERROR, VT_CY, VT_DATE, data is stored within the VARIANT structure. Any pointers to the data become invalid when the type of the variant is changed.
3. For VT_BYREF | any type, the memory pointed to by the variant is owned and freed by the caller of the function.
4. For VT_BSTR, there is only one owner for the string. All strings in variants must be allocated with the **SysAllocString** function. When releasing or changing the type of a variant with the VT_BSTR type, **SysFreeString** is called on the contained string.
5. For VT_ARRAY | any type, the rule is analogous to the rule for VT_BSTR. All arrays in variants must be allocated with **SafeArrayCreate**. When releasing or changing the type of a variant with the VT_ARRAY flag set, **SafeArrayDestroy** is called.
6. For VT_DISPATCH and VT_UNKNOWN, the pointed-to objects have reference counts that are incremented when they are placed in a variant. When releasing or changing the type of the variant, **Release** is called on the pointed-to object.

VariantChangeType

HRESULT VariantChangeType(*pvargDest, pvargSrc, wFlags, vtNew***)**
VARIANTARG FAR* *pvargDest*
VARIANTARG FAR* *pvargSrc*
unsigned short *wFlags*
VARTYPE *vtNew*

This function converts a variant from one type to another.

Parameters

pvargDest
Pointer to the VARIANTARG to receive the coerced type. If this is the same as *pvargSrc*, the variant will be converted in place.

pvargSrc
Pointer to the source VARIANTARG to be coerced.

wFlags
Flags that control the coercion. The only defined flag is VARIANT_NOVALUEPROP, which prevents the function from attempting to coerce an object to a fundamental type by getting the Value property. Applications should set this flag only if necessary, because it makes their behavior inconsistent with other applications.

vtNew
The type to coerce to. If the return code is S_OK, the *vt* field of the **pvargDest* will always be the same as this value.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
DISP_E_BADVARTYPE	The variant type *vtNew* is not a valid type of variant.
DISP_E_OVERFLOW	The data pointed to by *pvargSrc* does not fit in the destination type.
DISP_E_TYPEMISMATCH	The argument could not be coerced to the specified type.
E_INVALIDARG	One of the arguments is invalid.
E_OUTOFMEMORY	Memory could not be allocated for the conversion.

Comments

The **VariantChangeType** function handles coercions between the fundamental types (including numeric-to-string and string-to-numeric coercions). A variant that has VT_BYREF set is coerced to a value by fetching the referenced value. An object is coerced to a value by invoking the object's Value property (DISPID_VALUE).

Typically, the implementor of **IDispatch::Invoke** determines which member is being accessed, then calls **VariantChangeType** to get the value of one or more arguments. For example, if the **IDispatch** call specifies a SetTitle member that takes one string argument, the implementor would call **VariantChangeType** to attempt to coerce the argument to VT_BSTR. If **VariantChangeType** did not return an error, the argument could then be fetched directly from the *bstrVal* field of the VARIANTARG. If **VariantChangeType** returned DISP_E_TYPEMISMATCH, the implementor would set **puArgErr* to 0 (indicating the argument in error) and return DISP_E_TYPEMISMATCH from **IDispatch::Invoke**.

Note that you should not attempt to change the type of a VARIANTARG in the *rgvarg* array in place.

Arrays of one type can't be converted to arrays of another type with this function.

See Also

VariantChangeTypeEx

VariantChangeTypeEx

HRESULT VariantChangeTypeEx(*pvargDest, pvargSrc, lcid, wFlags, vtNew***)**
VARIANTARG FAR* *pvargDest*
VARIANTARG FAR* *pvargSrc*
LCID *lcid*
unsigned short *wFlags*
VARTYPE *vtNew*

This function converts a variant from one type to another using a locale ID.

Parameters

pvargDest
Pointer to the VARIANTARG to receive the coerced type. If this is the same as *pvargSrc*, the variant will be converted in place.

pvargSrc
Pointer to the source VARIANTARG to be coerced.

lcid
The locale ID for the variant to coerce. The locale ID is useful when the type of the source or destination VARIANTARG is VT_BSTR, VT_DISPATCH, or VT_DATE.

wFlags
Flags that control the coercion. The only defined flag is VARIANT_NOVALUEPROP, which prevents the function from attempting to coerce an object to a fundamental type by getting its Value property. Applications should set this flag only if necessary, because it makes their behavior inconsistent with other applications.

vtNew
The type to coerce to. If the return code is S_OK, the *vt* field of the **pvargDest* is guaranteed to be equal to this value.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
DISP_E_BADVARTYPE	The variant type *vtNew* is not a valid type of variant.
DISP_E_OVERFLOW	The data pointed to by *pvargSrc* does not fit in the destination type.
DISP_E_TYPEMISMATCH	The argument could not be coerced to the specified type.
E_INVALIDARG	One of the arguments is invalid.
E_OUTOFMEMORY	Memory could not be allocated for the conversion.

Comments

The **VariantChangeTypeEx** function handles coercions between the fundamental types (including numeric to string and string to numeric coercions). To change a type with the VT_BYREF flag set to one without VT_BYREF set, **VariantChangeTypeEx** fetches the referenced value. Objects are coerced to fundamental types by fetching the value of the Value property.

Typically, the implementor of **IDispatch::Invoke** determines which member is being accessed, then calls **VariantChangeType** to get the value of one or more arguments. For example, if the **IDispatch** call specifies a SetTitle member that takes one string argument, the implementor would call **VariantChangeTypeEx** to attempt to coerce the argument to VT_BSTR. If **VariantChangeTypeEx** did not return an error, the argument could then be fetched directly from the *bstrVal* field of the VARIANTARG. If **VariantChangeTypeEx** returned DISP_E_TYPEMISMATCH, the implementor would set **puArgErr* to 0 (indicating the argument in error) and return DISP_E_TYPEMISMATCH from **IDispatch::Invoke**.

Note that you should not attempt to change the type of a VARIANTARG in the *rgvarg* array in place.

Arrays of one type can't be converted to arrays of another type with this function.

See Also

VariantChangeType

VariantClear

HRESULT VariantClear(*pvarg***)**
VARIANTARG FAR* *pvarg*

Clears a variant.

Parameter

pvarg
Pointer to the VARIANTARG to clear.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
DISP_E_ARRAYISLOCKED	The variant contains an array that is locked.
DISP_E_BADVARTYPE	The variant type *pvarg* is not a valid type of variant.
E_INVALIDARG	One of the arguments is invalid.

Comments

This function should be used to clear variables of type VARIANTARG (or VARIANT) before the memory containing the VARIANTARG is freed (as when a local variable goes out of scope).

The function clears a VARIANTARG by setting the *vt* field to VT_EMPTY and the *wReserved* field to 0. The current contents of the VARIANTARG are released first. If the *vt* field is VT_BSTR, the string is freed; if the *vt* field is VT_DISPATCH, the object is released. If the *vt* field has the VT_ARRAY bit set, the array is freed.

In certain cases, you may prefer to clear a variant in your own code, without calling **VariantClear**. For example, it is permissible to change the type of a VT_I4 variant to some other type without calling this function. However, you must call **VariantClear** if you receive a VT_type that you are not prepared to handle. Using **VariantClear** in these cases ensures that your code will continue to work if OLE Automation adds new variant types in the future.

Example

```
for(i = 0; i < celt; ++i)
  VariantClear(&rgvar[i]);
```

VariantCopy

HRESULT VariantCopy(*pvargDest, pvargSrc***)**
  VARIANTARG FAR* *pvargDest*
  VARIANTARG FAR* *pvargSrc*

Frees the destination variant and makes a copy of the source variant.

Parameters

pvargDest
  Pointer to the VARIANTARG to receive the copy.

pvargSrc
  Pointer to the VARIANTARG to be copied.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
DISP_E_ARRAYISLOCKED	The variant contains an array that is locked.
DISP_E_BADVARTYPE	The source and destination have an invalid variant type (usually uninitialized).
E_OUTOFMEMORY	Memory could not be allocated for the copy.
E_INVALIDARG	The argument *pvargSrc* was VT_BYREF.

Comments

First, any memory owned by *pvargDest* is freed as in **VariantClear** (note that *pvargDest* must point to a valid initialized variant, and not simply an uninitialized memory location). Then, *pvargDest* receives an exact copy of the contents of *pvargSrc*. If *pvargSrc* is a VT_BSTR, a copy of the string is made. If *pvargSrc* is a VT_ARRAY, the entire array is copied. If *pvargSrc* is a VT_DISPATCH or VT_UNKNOWN, **AddRef** is called to increment the object's reference count.

VariantCopyInd

HRESULT VariantCopyInd(*pvarDest, pvargSrc***)**
VARIANT FAR* *pvarDest*
VARIANTARG FAR* *pvargSrc*

Frees the destination variant and makes a copy of the source VARIANTARG, performing the necessary indirection if the source is specified to be VT_BYREF.

Parameters

pvarDest
Pointer to the VARIANTARG to receive the copy.

pvargSrc
Pointer to the VARIANTARG to be copied.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
DISP_E_ARRAYISLOCKED	The variant contains an array that is locked.
DISP_E_BADVARTYPE	The source and destination have an invalid variant type (usually uninitialized).
E_OUTOFMEMORY	Memory could not be allocated for the copy.
E_INVALIDARG	The argument *pvargSrc* was VT_ARRAY.

Comments

This function is useful when you need to make a copy of a variant and guarantee that it is not VT_BYREF, such as when handling arguments in an implementation of **IDispatch::Invoke**.

For example, if the source is a (VT_BYREF | VT_I2), the destination will be a ByVal VT_I2. The same is true for all legal VT_BYREF combinations including VT_VARIANT.

If *pvargSrc* is (VT_BYREF | VT_VARIANT) and the contained variant is also VT_BYREF, the contained variant is also dereferenced.

This function frees any existing contents of *pvarDest*.

VariantInit

void VariantInit(*pvarg***)**
  VARIANTARG FAR* *pvarg*

Initializes a variant.

Parameter

pvarg
  Pointer to the VARIANTARG to initialize.

Comments

The **VariantInit** function initializes the VARIANTARG by setting the *vt* field to VT_EMPTY and the *wReserved* field to 0. Unlike **VariantClear**, this function does not interpret the current contents of the VARIANTARG. **VariantInit** should be used to initialize new local variables of type VARIANTARG (or VARIANT).

Example

```
for(i = 0; i < celt; ++i)
  VariantInit(&rgvar[i]);
```

Low-Level Variant Conversion Functions

The following low-level functions for converting variant data types are provided by OLEAUT32.DLL (for 32-bit systems) and OLE2DISP.DLL (for 16-bit systems). These functions are used by the higher-level variant manipulation functions, such as **VariantChangeType**, but may be called directly as well.

Convert to type	**From type**	**Function**
unsigned char	**unsigned char**	None
	short	**VarUI1FromI2**(*sIn, pbOut)*
	long	**VarUI1FromI4**(*lIn*, *pbOut*)
	float	**VarUI1FromR4**(*fltIn*, *pbOut*)
	double	**VarUI1FromR8**(*dblIn*, *pbOut*)
	CURRENCY	**VarUI1FromCy**(*cyIn*, *pbOut*)
	DATE	**VarUI1FromDate**(*dateIn*, *pbOut*)
	OLECHAR FAR*	**VarUI1FromStr**(*strIn*, *lcid*, *dwFlags, pbOut*)
	IDispatch FAR*	**VarUI1FromDisp**(*pdispIn*, *lcid, dwFlags, pbOut*)
	BOOL	**VarUI1FromBool**(*boolIn*, *pbOut*)
short	**unsigned char**	**VarI2FromUI1**(*bIn, psOut)*
	short	None
	long	**VarI2FromI4**(*lIn*, *psOut*)
	float	**VarI2FromR4**(*fltIn*, *psOut*)
	double	**VarI2FromR8**(*dblIn*, *psOut*)
	CURRENCY	**VarI2FromCy**(*cyIn*, *psOut*)
	DATE	**VarI2FromDate**(*dateIn*, *psOut*)
	OLECHAR FAR*	**VarI2FromStr**(*strIn*, *lcid*, *dwFlags, psOut*)
	IDispatch FAR*	**VarI2FromDisp**(*pdispIn*, *lcid, dwFlags, psOut*)
	BOOL	**VarI2FromBool**(*boolIn*, *psOut*)
long	**unsigned char**	**VarI4FromUI1**(*bIn*, *plOut*)
	short	**VarI4FromI2**(*sIn*, *plOut*)
	long	None
	float	**VarI4FromR4**(*fltIn*, *plOut*)
	double	**VarI4FromR8**(*dblIn*, *plOut*)
	CURRENCY	**VarI4FromCy**(*cyIn*, *plOut*)
	DATE	**VarI4FromDate**(*dateIn*, *plOut*)
	OLECHAR FAR*	**VarI4FromStr**(*strIn*, *lcid*, *dwFlags, plOut*)
	IDispatch FAR*	**VarI4FromDisp**(*pdispIn*, *lcid*, *dwFlags, plOut*)
	BOOL	**VarI4FromBool**(*boolIn*, *plOut*)
float	**unsigned char**	**VarR4FromUI1**(*bIn*, *prOut*)
	short	**VarR4FromI2**(*sIn*, *prOut*)

	long	**VarR4FromI4**(*lIn*, *prOut*)
	float	None
	double	**VarR4FromR8**(*dblIn*, *prOut*)
	CURRENCY	**VarR4FromCy**(*cyIn*, *prOut*)
	DATE	**VarR4FromDate**(*dateIn*, *prOut*)
	OLECHAR FAR*	**VarR4FromStr**(*strIn*, *lcid*, *dwFlags, prOut*)
	IDispatch FAR*	**VarR4FromDisp**(*pdispIn*, *lcid*, *dwFlags, prOut*)
	BOOL	**VarR4FromBool**(*boolIn*, *prOut*)
double	**unsigned char**	**VarR8FromUI1**(*bIn*, *pdblOut*)
	short	**VarR8FromI2**(*sIn*, *pdblOut*)
	long	**VarR8FromI4**(*lIn*, *pdblOut*)
	float	**VarR8FromR4**(*fltIn*, *pdblOut*)
	double	None
	CURRENCY	**VarR8FromCy**(*cyIn*, *pdblOut*)
	DATE	**VarR8FromDate**(*dateIn*, *pdblOut*)
	OLECHAR FAR*	**VarR8FromStr**(*strIn*, *lcid*, *dwFlags, pdblOut*)
	IDispatch FAR*	**VarR8FromDisp**(*pdispIn*, *lcid*, *dwFlags, pdblOut*)
	BOOL	**VarR8FromBool**(*boolIn*, *pdblOut*)
DATE	**unsigned char**	**VarDateFromUI1**(*bIn*, *pdateOut*)
	short	**VarDateFromI2**(*sIn*, *pdateOut*)
	long	**VarDateFromI4**(*lIn*, *pdateOut*)
	float	**VarDateFromR4**(*fltIn*, *pdateOut*)
	double	**VarDateFromR8**(*dblIn*, *pdateOut*)
	CURRENCY	**VarDateFromCy**(*cyIn*, *pdateOut*)
	DATE	None
	OLECHAR FAR*	**VarDateFromStr**(*strIn*, *lcid*, *dwFlags, pdateOut*)
	IDispatch FAR*	**VarDateFromDisp**(*pdispIn*, *lcid*, *dwFlags*, *pdateOut*)
	BOOL	**VarDateFromBool**(*boolIn*, *pdateOut*)
CURRENCY	**unsigned char**	**VarCyFromUI1**(*bIn*, *pcyOut*)
	short	**VarCyFromI2**(*sIn*, *pcyOut*)
	long	**VarCyFromI4**(*lIn*, *pcyOut*)
	float	**VarCyFromR4**(*fltIn*, *pcyOut*)
	double	**VarCyFromR8**(*dblIn*, *pcyOut*)
	CURRENCY	None

	DATE	**VarCyFromDate***(dateIn, pcyOut)*
	OLECHAR FAR*	**VarCyFromStr**(*strIn*, *lcid*, *dwFlags, pcyOut*)
	IDispatch FAR*	**VarCyFromDisp**(*pdispIn*, *lcid*, *dwFlags, pcyOut*)
	BOOL	**VarCyFromBool**(*boolIn*, *pcyOut*)
BSTR	**unsigned char**	**VarBstrFromUI1**(*bIn*, *lcid*, *dwFlags*, *pbstrOut*)
	short	**VarBstrFromI2**(*sIn*, *lcid*, *dwFlags*, *pbstrOut*)
	long	**VarBstrFromI4**(*lIn*, *lcid*, *dwFlags*, *pbstrOut*)
	float	**VarBstrFromR4**(*fltIn*, *lcid*, *dwFlags*, *pbstrOut*)
	double	**VarBstrFromR8**(*dblIn*, *lcid*, *dwFlags*, *pbstrOut*)
	CURRENCY	**VarBstrFromCy**(*cyIn*, *lcid*, *dwFlags*, *pbstrOut*)
	DATE	**VarBstrFromDate**(*dateIn*, *lcid*, *dwFlags*, *pbstrOut*)
	OLECHAR FAR*	None
	IDispatch FAR*	**VarBstrFromDisp**(*pdispIn*, *lcid*, *dwFlags*, *pbstrOut*)
	BOOL	**VarBstrFromBool**(*boolIn*, *lcid*, *dwFlags*, *pbstrOut*)
BOOL	**unsigned char**	**VarBoolFromUI1**(*bIn*, *pboolOut*)
	short	**VarBoolFromI2**(*sIn*, *pboolOut*)
	long	**VarBoolFromI4**(*lIn*, *pboolOut*)
	float	**VarBoolFromR4**(*fltIn*, *pboolOut*)
	double	**VarBoolFromR8**(*dblIn*, *pboolOut*)
	CURRENCY	**VarBoolFromCy**(*cyIn*, *pboolOut*)
	DATE	**VarBoolFromDate**(*dateIn*, *pboolOut*)
	OLECHAR FAR*	**VarBoolFromStr**(*strIn*, *lcid*, *dwFlags, pboolOut*)
	IDispatch FAR*	**VarBoolFromDisp**(*pdispIn*, *lcid*, *dwFlags, pboolOut*)
	BOOL	None

Parameters

bIn, sIn, lIn, fltIn, dblIn, cyIn, dateIn, strIn, pdispIn, boolIn

The value to coerce. These parameters have following data types:

Parameter	**Data type**
bIn	**unsigned char**
sIn	**short**
lIn	**long**
fltIn	**float**
dblIn	**double**
cyIn	CURRENCY
dateIn	DATE
strIn	OLECHAR FAR*
pdispIn	IDispatch FAR*
boolIn	BOOL

lcid

For conversions from string and VT_DISPATCH input, the locale ID to use for the conversion. For a list of locale IDs, see "Supporting Multiple National Languages" in Chapter 2.

dwFlags

One or more of the following flags:

Flag	**Description**
LOCALE_NOUSEROVERRIDE	Use the system default locale settings, rather than custom user locale settings.
VAR_TIMEVALUEONLY	Omit the date portion of a VT_DATE and return the time only. Applies to conversions to or from dates.
VAR_DATEVALUEONLY	Omit the time portion of a VT_DATE and return the time only. Applies to conversions to or from dates.

pbOut, psOut, plOut, pfltOut, pdblOut, pcyOut, pstrOut, pdispOut, pboolOut

A pointer to the coerced value. These parameters have following data types:

Parameter	**Data type**
pbOut	**unsigned char**
psOut	**short**
plOut	**long**
pfltOut	**float**
pdblOut	**double**
pcyOut	CURRENCY
pdateOut	DATE
pstrOut	OLECHAR FAR*
pdispOut	IDispatch FAR*
pboolOut	BOOL

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
DISP_E_BADVARTYPE	The input parameter is not a valid type of variant.
DISP_E_OVERFLOW	The data pointed to by the output parameter does not fit in the destination type.
DISP_E_TYPEMISMATCH	The argument could not be coerced to the specified type.
E_INVALIDARG	One of the arguments is invalid.
E_OUTOFMEMORY	Memory could not be allocated for the conversion.

Date and Time Conversion Functions

The following functions to convert between dates and times stored in MS-DOS format and the variant representation are provided by OLEAUT32.DLL (for 32-bit systems) and OLE2DISP.DLL (for 16-bit systems).

DosDateTimeToVariantTime

int DosDateTimeToVariantTime(*wDOSDate*, *wDOSTime*, *pvtime***)**
  unsigned short *wDOSDate*
  unsigned short *wDOSTime*
  double FAR* *pvtime*

Converts the MS-DOS representation of time to the date and time representation stored in variant.

Parameters

wDOSDate
  The MS-DOS date to convert.

wDOSTime
  The MS-DOS time to convert.

pvtime
  Pointer to the location to store the converted time.

Return Value

One of the following:

Result	Meaning
True	Success
False	Failure

Comments

MS-DOS records file dates and times as packed 16-bit values. An MS-DOS *date* has the following format:

Bits	Contents
0-4	Day of the month (1-31)
5-8	Month (1 = January, 2 = February, and so on)
9-15	Year offset from 1980 (add 1980 to get actual year)

An MS-DOS *time* has the following format:

Bits	Contents
0-4	Second divided by 2
5-10	Minute (0-59)
11-15	Hour (0-23 on a 24-hour clock)

VariantTimeToDosDateTime

int VariantTimeToDosDateTime(*vtime*, *pwDOSDate*, *pwDOSTime***)**
  double *vtime*
  unsigned short FAR* *pwDOSDate*
  unsigned short FAR* *pwDOSTime*

Converts the variant representation of a date and time to MS-DOS date and time values.

Parameters

vtime
  The variant time to convert.

pwDOSDate
  Pointer to location to store the converted MS-DOS date.

pwDOSTime
  Pointer to location to store the converted MS-DOS time.

Return Value

One of the following:

Result	**Meaning**
True	Success
False	Failure

Comments

A variant time is stored as an 8-byte real value (**double**), representing a date between January 1, 1753 and December 31, 2078, inclusive. The value 2.0 represents January 1, 1900; 3.0 represents January 2, 1900, and so on. Adding 1 to the value increments the date by a day. The fractional part of the value represents the time of day. Thus, 2.5 represents noon on January 1, 1900; 3.25 is 6:00 a.m. on January 2, 1900; and so on. Negative numbers represent the dates prior to December 30, 1899.

See the Comments section of "**DosDateTimeToVariantTime**" for a description of the MS-DOS date and time formats.

MkTypLib and Object Description Language

When you expose OLE Automation objects, you permit interoperability with other vendors' programs. For others to use your objects, they must have access to the objects' characteristics (properties and methods). To make this information public, you can:

- Publish object and type definitions (for example, as printed documentation).
- Code objects into a compiled .C or .CPP file, so they can be accessed using **IDispatch::GetTypeInfo** or implementations of the **ITypeInfo** and **ITypeLib** interfaces.
- Use the MkTypLib tool to create a type library that contains the objects, then make the type library available to others.

MkTypLib compiles scripts that are written in the Object Description Language (ODL). This chapter provides a reference to the MkTypLib tool and ODL.

Contents of a Type Library

Type libraries are OLE compound document files that include information about types and objects exposed by an OLE application. A type library may contain any of the following:

- Information about data types, such as aliases, enumerations, structures, or unions.
- Descriptions of one or more objects, such as a module, interface, **IDispatch** interface (dispinterface), or component object class (coclass). Each of these descriptions is commonly referred to as a *typeinfo.*
- References to type descriptions from other type libraries.

Including the type library with your product makes information about the objects it contains available to your customers' applications and programming tools. You can ship type libraries in any of the following forms:

- A resource in a dynamic-link library (DLL). This resource should have the type **typelib** and an integer ID. It must be declared in the resource (.RC) file as follows:

  ```
  1 typelib mylib1.tlb
  2 typelib mylib2.tlb
  ```

  There can be multiple type library resources in a DLL. Application developers use the resource compiler to add the .TLB file to their own DLL. A DLL with one or more type library resources typically has the extension .OLB (object library).
- A resource in an .EXE file. The file may contain multiple type libraries.
- A stand-alone binary file. The .TLB file output by MkTypLib is a binary file.

Object browsers, compilers, and similar tools access type libraries through the **ITypeLib**, **ITypeInfo**, and **ITypeComp** interfaces. Type library tools (such as MkTypLib) can be created using the **ICreateTypeLib** and **ICreateTypeInfo** interfaces.

MkTypLib: Type Library Creation Tool

MkTypLib processes scripts written in the Object Description Language (ODL), producing a type library (.TLB) and an optional C or C++ style header file.

MkTypLib uses the **ICreateTypeLib** and **ICreateTypeInfo** interfaces to create type libraries. Type libraries can, in turn, be accessed by tools such as type browsers and compilers that use the **ITypeLib** and **ITypeInfo** interfaces, as shown in the following figure.

{ewc msdncd, EWGraphic, group10459 0 /a "SDK_01.bmp"}

Invoking MkTypLib

To invoke MkTypLib, choose the Run command from either the Program Manager or the File Manager File menu, then enter the following command line in the dialog box:

MkTypLib [*options***]** *ODLfile*

MkTypLib creates a type library (.TLB) file based on the object description script in the file specified by *ODLfile*. It can optionally produce a header file (extension .H), which is a stripped version of the input file suitable for inclusion in C or C++ programs that want to access the types defined in the input file. In the header file, MkTypLib inserts DEFINE_GUID macros for each element (interface, dispinterface, and so forth) defined in the type library.

The *options* can be a series of options prefixed with a hyphen (-) or a slash (/) as follows:

Option	**Description**
/? or **/help**	Displays command line Help. The *ODLfile* does not need to be specified in this case.
/align:*alignment*	Sets the default alignment for types in the library. An *alignment* value of 1 indicates natural alignment; *n* indicates alignment on byte *n*.
/cpp_cmd *cpppath*	Specifies *cpppath* as the command to run the C preprocessor. By default, MkTypLib invokes CL.
/cpp_opt "*options"*	Specifies options for the C preprocessor. Default is "**/C /E /D__MKTYPLIB__**."
/D *define*[=*value*]	Defines the name *define* for the C preprocessor. The *value* is its optional value. No space is allowed between the equal sign (=) and the value.
/h *filename*	Specifies *filename* as the name for a stripped version of the input file. This file can be used as a C or C++ header file.
/I *includedir*	Specifies *includedir* as the directory where include files are located for the C preprocessor.
/nocpp	Suppresses invocation of the C preprocessor.
/nologo	Disables display of the copyright banner.
/o *outputfile*	Redirects output (for example, error messages) to specified *outputfile.*
/tlb *filename*	Specifies *filename* as the name of the output .TLB file. If not specified, this is the same as the name of the *ODLfile* with the extension .TLB.
/win16 /win32 /mac **/mips /alpha** **/ppc /ppc32**	Specifies type of output type library to produce. Default is the current operating system.
/w0	Disables warnings.

Although MkTypLib offers minimal error reporting, error messages include accurate line and column number information that can be used with text editors to locate the source of errors.

MkTypLib spawns the C preprocessor. The symbol **__MKTYPLIB__** is predefined for the preprocessor.

ODL File Syntax

The general syntax for an ODL file is as follows:

[*attributes***] library** *libname* **{***definitions***};**

The *attributes* associate characteristics with the library, such as its Help file and UUID. Attributes must be enclosed in square brackets.

The *definitions* consist of the descriptions of the imported libraries, data types, modules, interfaces, dispinterfaces, and coclasses that are part of the type library. Braces ({}) must surround the definitions.

The following table summarizes the elements that may appear in *definitions*. Each element is described in detail in the section, "ODL Reference."

Purpose	Library element	Description
Allows references to other type libraries.	importlib (*lib1*)	Specifies an external type library that contains definitions referenced in this type library.
Declares data types used by the objects in this type library.	**typedef [***attributes***]** *aliasname*	An alias declared using C syntax. Must have at least one *attribute* to be included in the type library.
	typedef [*attributes*] **enum**	An enumeration declared using C **typedef** and **enum** keywords.
	typedef [*attributes*] **struct**	A structure declared using C **typedef** and **struct** keywords.
	typedef [*attributes*] **union**	A union declared using C **typedef** and **union** keywords.
Describes functions that enable querying the DLL.	[*attributes*] **module**	Constants and general data functions whose actions are not restricted to any specified class of objects.
Describes interfaces.	[*attributes*] **dispinterface**	An interface describing the methods and properties for an object that must be accessed through **IDispatch::Invoke**.
	[*attributes*] **interface**	An interface describing the methods and properties for an object that may be accessed through **IDispatch::Invoke** or through VTBL entries.

Describes OLE classes.	[*attributes*] **coclass**	Specifies a top-level object, with all its interfaces and dispinterfaces.

Within the library description, modules, interfaces, dispinterfaces, and coclasses follow the same general syntax:

[*attributes***]** *elementname typename* **{**
    memberdescriptions
  };

The *attributes* set characteristics for the element. The *elementname* is a keyword that indicates the kind of item (**module**, **interface**, **dispinterface**, or **coclass**), and the *typename* defines the name of the item. The *memberdescriptions* define the members (constants, functions, properties, and methods) of each element.

Aliases, enumerations, unions, and structures have the following syntax:

typedef [*typeattributes***]** *typekind typename* **{**
    memberdescriptions
  };

For these types, the attributes follow the **typedef** keyword, and the *typekind* indicates the data type (**enum**, **union**, or **struct**). See the keyword descriptions for details.

Note The square brackets ([])and braces ({}) in these descriptions are part of the syntax, not descriptive symbols. The semicolon after the closing brace (**}**) that terminates the library definition (and all other type definitions) is optional.

ODL File Example

The following example shows the ODL file for the Lines sample, extracted from LINES.ODL:

```
[
     uuid(3C591B20-1F13-101B-B826-00DD01103DE1),           // LIBID_Lines
     helpstring("Lines 1.0 Type Library"),
     lcid(0x0409),
     version(1.0)
]
library Lines
{
     importlib("stdole.tlb");
     #define DISPID_NEWENUM -4

     [
           uuid(3C591B25-1F13-101B-B826-00DD01103DE1),  // IID_IPoint
           helpstring("Point object."),
           oleautomation,
           dual
     ]
     interface IPoint : IDispatch
     {
           [propget, helpstring("Returns and sets x coordinate.")]
           HRESULT x([out, retval] int* retval);
           [propput, helpstring("Returns and sets x coordinate.")]
           HRESULT x([in] int Value);

           [propget, helpstring("Returns and sets y coordinate.")]
           HRESULT y([out, retval] int* retval);
           [propput, helpstring("Returns and sets y coordinate.")]
           HRESULT y([in] int Value);
     }

// Additional interfaces omitted for brevity

[
           uuid(3C591B27-1F13-101B-B826-00DD01103DE1),           //
IID_IPoints
           helpstring("Points collection."),
           oleautomation,
           dual
     ]
     interface IPoints : IDispatch
     {
           [propget, helpstring("Returns number of points in collection.")]
           HRESULT Count([out, retval] long* retval);

           [propget, id(0),
           helpstring("Given an index, returns a point in the collection")]
           HRESULT Item([in] long Index, [out, retval] IPoint** retval);

           [propget, restricted, id(DISPID_NEWENUM)] // Must be propget.
           HRESULT _NewEnum([out, retval] IUnknown** retval);
     }
// Additional interface omitted for brevity

[
           uuid(3C591B22-1F13-101B-B826-00DD01103DE1),     //
IID_IApplication
           helpstring("Application object."),
           oleautomation,
           dual
     ]
     interface IApplication : IDispatch
     {
           [propget, helpstring("Returns the application of the object.")]
           HRESULT Application(    [out, retval] IApplication** retval);

           [propget,
                 helpstring("Returns the full name of the application.")]
           HRESULT FullName([out, retval] BSTR* retval);
           [propget, id(0),
           helpstring("Returns the name of the application.")]
           HRESULT Name([out, retval] BSTR* retval);

           [propget, helpstring("Returns the parent of the object.")]
           HRESULT Parent([out, retval] IApplication** retval);

           [propput]
           HRESULT Visible([in] boolean VisibleFlag);
           [propget, helpstring
           ("Sets or returns whether the main window is visible.")]
           HRESULT Visible(  [out, retval] boolean* retval);

           [helpstring("Exits the application.")]
           HRESULT Quit();

// Additional methods omitted for brevity

           [helpstring("Creates new Point object initialized to (0,0).")]
           HRESULT CreatePoint(    [out, retval] IPoint** retval);
     }

     [
           uuid(3C591B21-1F13-101B-B826-00DD01103DE1),     // CLSID_Lines
           helpstring("Lines Class"),
           appobject
     ]
     coclass Lines
     {
           [default] interface IApplication;
                 interface IDispatch;
     }
}
```

The example describes a library named Lines, which imports the standard OLE library STDOLE.TLB. The #define directive defines the constant DISPID_NEWENUM, which is needed for the _NewEnum property of the IPoints collection.

The example shows declarations for three interfaces in the library: IPoint, IPoints, and IApplication. Because all three are **dual** interfaces, their members may be invoked through **IDispatch** or directly through VTBLs. In addition, all their members return HRESULT values and pass their return values as **retval** parameters. Therefore, they can support the **IErrorInfo** interface, through which they can return detailed error information, no matter how they are invoked.

The IPoint interface has two properties, x and y, and two pairs of accessor functions to get and set the properties.

The IPoints interface is a collection of points. It supports three read-only properties, each of which has a single accessor function. The Count and Item properties return the number of points and the value of a single point, respectively. The _NewEnum property, required for collection objects, returns an enumerator object for the collection. This property has the **restricted** attribute, indicating that it should not be invoked from a macro language.

The IApplication interface describes the application object. It supports properties named Application, FullName, Name, Parent, Visible, and Pane; and methods named Quit, CreateLine, and CreatePoint.

Finally, the script defines a coclass named Lines. The **appobject** attribute makes the members of the coclass (IApplication and IDispatch) globally accessible in the type library. IApplication is defined as the **default** member, indicating that it is the programmability interface intended for use by macro languages.

Source File Contents

The following sections describe the proper format for comments, constants, identifiers, and other syntactic items within an ODL file.

Array Definitions

MkTypLib accepts both fixed-size arrays and arrays declared as SAFEARRAY.

Use C-style syntax for a fixed-size array:

type arrname[*size*];

To describe a SAFEARRAY, use the following syntax:

SAFEARRAY (*elementtype***)** ***arrayname***

A function returning a SAFEARRAY has the following syntax:

SAFEARRAY (*elementtype*) *myfunction*(*parameterlist*);

Comments

To include comments in an ODL file, use C-style syntax in either block form (/*...*/) or single-line form (\ \). MkTypLib ignores the comments, and does not preserve them in the header (.H) file.

Constants

A constant can be either numeric or a string, depending on the attribute.

Numerics

Numeric input is usually an integer (in either decimal or in hexadecimal using the standard **0x** format), but may also be a single character constant (for example, \0).

Strings

A string is delimited by double-quotation marks (**"**) and may not span multiple lines. The backslash character (****) acts as an escape character. The backslash character followed by any character (even another backslash) prevents the second character from being interpreted with any special meaning; the backslash is not included in the text.

For example, to include a double-quotation mark (**"**) in the text without causing it to be interpreted as the closing delimiter, precede it with a backslash (\"). Similarly, you can use a double backslash (\\) to put a backslash into the text. Some examples of valid strings are:

```
"commandName"
"This string contains a \"quote\"."
"Here's a pathname: c:\\bin\\binp"
```

A string can be up to 255 characters long.

Filenames

A filename is a string that represents either a full or partial path. OLE Automation expects to find files in directories referenced by the type library registration entries, so partial pathnames are typically used. See Chapter 2, "Exposing OLE Automation Objects," for more information about registration.

Forward Declarations

Forward declarations permit forward references to types. Forward references have the following form:

```
typedef struct mydata;
interface aninterface;
dispinterface fordispatch;
coclass pococlass;
```

The Globally Unique ID (GUID)

A UUID is a globally unique ID (GUID). This number is created by running the GUIDGEN.EXE command-line program. GUIDGEN will never produce the same number twice, no matter how many times it is run or how many different machines it runs on. Every entity (such as an interface) that needs to be uniquely identified has a GUID.

Identifiers

Identifiers can be up to 255 characters long, and must conform to C-style syntax. MkTypLib is case sensitive, but it generates type libraries that are case insensitive. Thus, it is possible to define a user-defined type whose name differs from that of a built-in type only by case. However, user-defined type names (and member names) that differ only in case refer to the same type or member. Except for property accessor functions, it is illegal for two members of a type to have the same name, regardless of case.

Intrinsic Data Types

The following data types are recognized by MkTypLib:

Type	Description
boolean	Data item that can have the values True or False. The size maps to VARIANT_BOOL.
char	8-bit signed data item.
double	64-bit IEEE floating-point number.
int	Signed integer, whose size is system-dependent.
float	32-bit IEEE floating-point number.
long	32-bit signed integer.
short	16-bit signed integer.
void	Allowed only as return type for a function, or in a function parameter list to indicate no arguments.
wchar_t	Unicode character accepted only for 32-bit type libraries.
BSTR	Length-prefixed string, as described in Chapter 5, "Dispatch Interfaces."
CURRENCY	8-byte fixed-point number.
DATE	64-bit floating-point fractional number of days since December 30, 1899.
HRESULT	Return type used for reporting error information in interfaces, as described in the *OLE 2 Programmer's Reference, Volume 1*.
Type	**Description**
LPWSTR	Unicode string accepted only for 32-bit type libraries.
LPSTR	Zero-terminated string.
SCODE	Built-in error type that corresponds to VT_ERROR. An SCODE does not contain the additional error information provided by HRESULT.
VARIANT	One of the variant data types as described in Chapter 5, "Dispatch Interfaces."
IDispatch *	Pointer to **IDispatch** interface.
IUnknown *	Pointer to **IUnknown** interface. (Any OLE interface can be represented by its **IUnknown** interface.)

The keyword **unsigned** may be specified before **int**, **char**, **short**, and **long**.

String Definitions

Strings can be declared using the LPSTR type, which indicates a zero-terminated string, and with the BSTR type, which indicates a length-prefixed string (as defined in Chapter 5, "Dispatch Interfaces." In 32-bit type libraries, Unicode strings can be defined with the LPWSTR type.

ODL Reference

The remainder of this chapter provides reference material on the attributes, statements, and directives that are part of the Object Description Language (ODL).

Attribute Descriptions

The following sections describe the ODL attributes and list the types of objects to which they apply, and the equivalent flags set in the object's typeinfo.

appobject

Description

Identifies the Application object.

Allowed on

Coclass

Comments

Indicates that the members of the class may be accessed without qualification when accessing this type library.

Flags

TYPEFLAG_FAPPOBJECT

bindable

Description

Indicates that the property supports data binding.

Allowed on

Property

Comments

Refers to the property as a whole, so it must be specified wherever the property is defined. Therefore, you need to specify the attribute on both the property get description and on the property set description.

Flags

FUNCFLAG_FBINDABLE, VARFLAG_FBINDABLE

control

Description

Indicates that the item represents a control from which a container site will derive additional typelibs or coclasses.

Allowed on

Typelib, coclass

Comments

This attribute allows you to mark type libraries that describe controls so that they will not be displayed in type browsers intended for nonvisual objects.

Flags

TYPEFLAG_FCONTROL, LIBFLAG_FCONTROL

default

Description

Indicates that the interface or dispinterface represents the default programmability interface, intended for use by macro languages.

Allowed on

Coclass member

Comments

A coclass may have at most two **default** members. One represents the source interface or dispinterface, and the other represents the sink interface or dispinterface. If the **default** attribute is not specified for any member of the coclass or cotype, the first source and sink members that do not have the **restricted** attribute are treated as the defaults.

Flags

IMPLTYPEFLAG_FDEFAULT

defaultbind

Description

Indicates the single, bindable property that best represents the object.

Allowed on

Property

Comments

Properties that have the **defaultbind** attribute must also have the **bindable** attribute. You can't specify **defaultbind** on more than one property in a dispinterface.

This attribute is used by containers that have a user model involving binding to an object rather than binding to a property of an object. An object can support data binding but not have this attribute.

Flags

FUNCFLAG_FDEFAULTBIND, VARFLAG_FDEFAULTBIND

displaybind

Description

Indicates a property that should be displayed to the user as bindable.

Allowed on

Property

Comments

Properties that have the **displaybind** attribute must also have the **bindable** attribute. An object can support data binding but not have this attribute.

Flags

FUNCFLAG_FDISPLAYBIND, VARFLAG_FDISPLAYBIND

dllname(*str*)

Description

Defines the name of the DLL that contains the entry points for a module.

Allowed on

Module (required)

Comments

The *str* argument gives the filename of the DLL.

dual

Description

Identifies an interface that exposes properties and methods through **IDispatch** and directly through the VTBL.

Allowed on

Interface

Comments

The interface must be compatible with OLE Automation and derive from **IDispatch**. Not allowed on dispinterfaces.

The **dual** attribute creates an interface that is both a **Dispatch** interface and a Component Object Model (COM) interface. The first seven entries of the VTBL for a dual interface are the seven members of **IDispatch**, and the remaining entries are OLE COM entries for direct access to members of the dual interface. All the parameters and return types specified for members of a dual interface must be OLE Automation-compatible types.

All members of a dual interface must pass an HRESULT as the function return value. Members that need to return other values should specify the last parameter as [**retval**, **out**] indicating an output parameter that returns the value of the function. In addition, members that need to support multiple locales should pass an **lcid** parameter.

A dual interface provides for both the speed of direct VTBL binding and the flexibility of **IDispatch** binding. For this reason, dual interfaces are recommended whenever possible.

Note If your application accesses object data by casting the THIS pointer within the interface call, you should check the VTBL pointers in the object against your own VTBL pointers to ensure that you are connected to the appropriate proxy.

Specifying **dual** on an interface implies that the interface is compatible with OLE Automation, and therefore causes both the TYPEFLAG_FDUAL and TYPEFLAG_FOLEAUTOMATION flags to be set.

Flags

TYPEFLAG_FDUAL, TYPEFLAG_FOLEAUTOMATION

entry(*entryid*)

Description

Identifies the entry point in the DLL.

Allowed on

Functions in a module (required)

Comments

If *entryid* is a string, this is a named entry point. If *entryid* is a number, the entry point is defined by an ordinal. This attribute provides a way to obtain the address of a function in a module.

helpcontext(*numctxt*)

Description

Sets the context within the Help file.

Allowed on

Library, interface, dispinterface, struct, enum, union, module, typedef, method, struct member, enum value, property, coclass, const

Comments

Retrieved via the **GetDocumentation** functions in the **ITypeLib** and **ITypeInfo** interfaces. The *numctxt* is a 32-bit Help context within the Help file.

helpfile(*filename*)

Description

Sets the name of the Help file.

Allowed on

Library

Comments

Retrieved via the **GetDocumentation** functions in the **ITypeLib** and **ITypeInfo** interfaces.

All types in a library share the same Help file.

helpstring(*string*)

Description

Sets the Help string.

Allowed on

Library, interface, dispinterface, struct, enum, union, module, typedef, method, struct member, enum value, property, coclass, const

Comments

Retrieved via the **GetDocumentation** functions in the **ITypeLib** and **ITypeInfo** interfaces.

hidden

Description

Indicates that the item exists but should not be displayed in a user-oriented browser.

Allowed on

Property, method, coclass, dispinterface, interface, library

Comments

This attribute allows you to remove members from your interface (by shielding them from further use) while maintaining compatibility with existing code.

When specified for a library, the attribute prevents the entire library from being displayed. It is intended for use by controls. Hosts need to create a new type library that wraps the control with extended properties.

Flags

VARFLAG_FHIDDEN, FUNCFLAG_FHIDDEN, TYPEFLAG_FHIDDEN

id(*num*)

Description

Identifies the DISPID of the member.

Allowed on

Method or property in an interface or dispinterface

Comments

The *num* is a 32-bit integral value in the following format:

Bits	Value
0-15	Offset. Any value is permissible.
16-21	The nesting level of this typeinfo in the inheritance hierarchy. For example: `interface mydisp : IDispatch` The nesting level of **IUnknown** is 0, **IDispatch** is 1, and mydisp is 2.
22-25	Reserved; must be zero
26-28	DISPID value.
29	True if this is the member ID for a FuncDesc; otherwise False.
30-31	Must be 01.

Negative IDs are reserved for use by OLE Automation.

in

Description

Specifies an input parameter.

Allowed on

Parameter

Comments

The parameter may be a pointer (such as **char** *) but the value it refers to is not returned.

lcid

Description

Indicates that the parameter is a locale ID.

Allowed on

Parameter in a member of an interface

Comments

At most one parameter may have this attribute. The parameter must have the **in** attribute and can't have the **out** attribute, and its type must be **long**. The **lcid** attribute is not allowed on dispinterfaces.

The **lcid** attribute allows members in the VTBL to receive an LCID at invocation. By convention, the **lcid** parameter is the last parameter that does not have the **retval** attribute. If the member specifies **propertyput** or **propertyputref**, the **lcid** parameter must precede the parameter that represents the right side of the property assignment.

ITypeInfo::Invoke passes the LCID of the typeinfo into the **lcid** parameter. Parameters with this attibute are not displayed in user-oriented browsers.

lcid(*numid*)

Description

Identifies the locale for a type library.

Allowed on

Library

Comments

The *numid* is a 32-bit locale ID as used in Win32 National Language Support. The locale ID is typically entered in hexadecimal.

licensed

Description

Indicates that the class is licensed.

Allowed on

Coclass

Flags

TYPEFLAG_FLICENSED

nonextensible

Description

Indicates that the **IDispatch** implementation includes only the properties and methods listed in the interface description.

Allowed on

Dispinterface, interface

Comments

The interface must have the **dual** attribute.

By default, OLE Automation assumes that interfaces may add members at run time; that is, it assumes they are extensible.

Flags

TYPEFLAG_FNONEXTENSIBLE

odl

Description

Identifies an interface as an ODL interface.

Allowed on

Interface (required)

Comments

This attribute must appear on all interfaces.

oleautomation

Description

Indicates that an interface is compatible with OLE Automation.

Allowed on

Interface

Comments

Not allowed on dispinterfaces.

The **oleautomation** attribute indicates that an interface is compatible with OLE Automation. The parameters and return types specified for its members must be OLE Automation-compatible, as listed in the following table.

Type	Description
boolean	Data item that can have the value True or False. The size corresponds to VARIANT_BOOL.
unsigned char	8-bit unsigned data item.
double	64-bit IEEE floating-point number.
float	32-bit IEEE floating-point number.
int	Signed integer, whose size is system-dependent.
long	32-bit signed integer.
short	16-bit signed integer.
BSTR	Length-prefixed string, as described in Chapter 5, "Dispatch Interfaces."
CURRENCY	8-byte fixed-point number.
DATE	64-bit floating-point fractional number of days since December 30, 1899.
SCODE	Built-in error type that corresponds to VT_ERROR.
typedef enum *myenum*	Signed integer, whose size is system-dependent.
interface IDispatch *	Pointer to **IDispatch** interface (VT_DISPATCH).
interface IUnknown *	Pointer to interface that does not derive from **IDispatch** (VT_UNKNOWN). (Any OLE interface can be represented by its **IUnknown** interface.)
dispinterface *Typename **	Pointer to **IDispatch**-derived interface (VT_DISPATCH).
coclass *Typename **	Pointer to a coclass name (VT_UNKNOWN).
[oleautomation] interface *Typename **	Pointer to an interface that derives from **IDispatch**.

A parameter is compatible with OLE Automation if its type is an OLE Automation-compatible type, a pointer to an OLE Automation-compatible type, or a SAFEARRAY of an OLE Automation-compatible type.

A return type is compatible with OLE Automation if its type is an HRESULT or is **void**. Methods in OLE Automation must return either HRESULT or **void**.

A member is compatible with OLE Automation if its return type and all of its parameters are OLE-Automation compatible.

An interface is compatible with OLE Automation if it derives from **IDispatch** or IUnknown, if it has the **oleautomation** attribute, and if all of its VTBL entries are OLE-Automation compatible. For 32-bit systems, the calling convention for all methods in the interface must be STDCALL. For 16-bit systems, all methods must have the CDECL calling convention. Every dispinterface is OLE Automation-compatible.

Flags

TYPEFLAG_FOLEAUTOMATION

optional

Description

Specifies an optional parameter.

Allowed on

Parameter

Comments

Valid only if the parameter is of type VARIANT or VARIANT*. All subsequent parameters of the function must also be **optional**.

out

Description

Specifies an output parameter.

Allowed on

Parameter

Comments

The parameter must be a pointer to memory that will receive a result.

propget

Description

Specifies a property accessor function.

Allowed on

Functions; methods in interfaces and dispinterfaces

Comments

The property must have the same name as the function. At most, one of **propget**, **propput**, and **propputref** can be specified for a function.

Flags

INVOKE_PROPERTYGET

propput

Description

Specifies a property-setting function.

Allowed on

Functions; methods in interfaces and dispinterfaces

Comments

The property must have the same name as the function. At most, one of **propget**, **propput** and **propputref** can be specified.

Flags

INVOKE_PROPERTYPUT

propputref

Description

Specifies a property-setting function that uses a reference instead of a value.

Allowed on

Functions; methods in interfaces and dispinterfaces

Comments

The property must have the same name as the function. At most, one of **propget**, **propput** and **propputref** can be specified.

Flags

INVOKE_PROPERTYPUTREF

public

Description

Includes an alias declared with the **typedef** keyword in the type library.

Allowed on

Aliases declared with **typedef**

Comments

By default, an alias that is declared with **typedef** and has no other attributes is treated as a **#define** and is not included in the type library. Using the **public** attribute ensures that the alias becomes part of the type library.

readonly

Description

Prohibits assignment to a variable.

Allowed on

Variable

Flags

VARFLAG_FREADONLY

requestedit

Description

Indicates that the property supports the OnRequestEdit notification.

Allowed on

Property

Comments

The property supports the OnRequestEdit notification, raised by a property before it is edited. An object can support data binding but not have this attribute.

Flags

FUNCFLAG_FREQUESTEDIT, VARFLAG_FREQUESTEDIT

restricted

Description

Prevents the item from being used by a macro programmer.

Allowed on

Type library, coclass member, member of a module or interface

Comments

Allowed on a member of a coclass, independent of whether the member is a dispinterface or interface, and independent of whether the member is a sink or source. A member of a coclass can't have both the **restricted** and **default** attributes.

Flags

IMPLTYPEFLAG_FRESTRICTED, FUNCFLAG_FRESTRICTED

retval

Description

Designates the parameter that receives the return value of the member.

Allowed on

Parameters of interface members that describe methods or get properties

Comments

This attribute may be used only on the last parameter of the member. The parameter must have the **out** attribute and must be a pointer type.

Parameters with this attibute are not displayed in user-oriented browsers.

Flags

IDLFLAG_FRETVAL

source

Description

Indicates that a member is a source of events.

Allowed on

Member of a coclass; property or method.

Comments

For a member of a coclass, this attribute means that the member is called rather than implemented.

On a property or method, indicates that the member returns an object or VARIANT that is a source of events. The object implements **IConnectionPointContainer**.

Flags

IMPLTYPEFLAG_FSOURCE, VARFLAG_SOURCE, FUNCFLAG_SOURCE

string

Description

Specifies a string.

Allowed on

Struct, member, parameter, property

Comments

Included only for compatibility with IDL; use LPSTR for a zero-terminated string.

uuid(*uuidval*)

Description

Specifies the UUID of the item.

Allowed on

Required for library, dispinterface, interface, and coclass; optional on struct, enum, union, module, and typedef

Comments

The *uuidval* is a 16-byte value formatted as hexadecimal digits in the following format: `12345678-1234-1234-1234-123456789ABC`. This value is returned in the TypeAttr structure retrieved by **TypeInfo::GetTypeAttr.**

vararg

Description

Indicates a variable number of arguments.

Allowed on

Function

Comments

Indicates that the last parameter is a safe array of VARIANT type, which contains all the remaining parameters.

version(*versionval*)

Description

Specifies a version number.

Allowed on

Library, struct, module, dispinterface, interface, coclass, enum, union.

Comments

The argument *versionval* is a real number in the format *n.m*, where *n* is a major version number and *m* is a minor version number.

ODL Statements and Directives

The following sections describe the statements and directives that make up the Object Description Language.

The coclass Statement

Describes the GUID and supported interfaces for a component object.

Syntax

```
[attributes]
   coclass classname {
      [attributes2] [interface | dispinterface] interfacename;
   . . .
   };
```

Syntax Elements

attributes
The **uuid** attribute is required on a coclass. This is the same **uuid** that is registered as a CLSID in the system registration database. The **helpstring**, **helpcontext, licensed**, **version**, **control**, **hidden**, and **appobject** attributes are accepted, but not required, before a **coclass** definition. See "Attribute Descriptions," earlier in this chapter, for more information on the attributes accepted before a coclass definition. The **appobject** attribute makes the functions and properties of the coclass globally available in the type library.

classname
Name by which the common object is known in the type library.

attributes2
Optional attributes for the interface or dispinterface. The **source, default,** and **restricted** attributes are accepted on an interface or dispinterface within a coclass.

interfacename
Either an interface declared with the **interface** keyword, or a dispinterface declared with the **dispinterface** keyword.

Comments

The Microsoft Component Object Model defines a class as an implementation that allows **QueryInterface** between a set of interfaces.

Examples

```
[ uuid(BFB73347-822A-1068-8849-00DD011087E8), version(1.0), helpstring("A
class"), helpcontext(2481), appobject]
coclass myapp {
     [source] interface IMydocfuncs;
     dispinterface DMydocfuncs;
};

[uuid 00000000-0000-0000-0000-123456789019]
coclass foo
{
     [restricted] interface bar;
     interface baz;
}
```

The dispinterface Statement

Defines a set of properties and methods on which you can call **IDispatch::Invoke**. A dispinterface may be defined by explicitly listing the set of supported methods and properties (Syntax 1) or by listing a single interface (Syntax 2).

Syntax 1

```
[attributes]
   dispinterface intfname {
      properties:
         proplist
      methods:
         methlist
   };
```

Syntax 2

```
[attributes]
   dispinterface intfname {
      interface interfacename
   };
```

Syntax Elements

attributes

The **helpstring**, **helpcontext**, **hidden**, **uuid**, and **version** attributes are accepted before **dispinterface**. See "Attribute Descriptions," earlier in this chapter, for more information on the attributes accepted before a **dispinterface** definition. Attributes (including the brackets) may be omitted. The **uuid** attribute is required.

intfname

The name by which the **dispinterface** is known in the type library. This name must be unique within the type library.

interfacename

(Syntax 2) The name of the interface to declare as an **IDispatch** interface.

proplist

(Syntax 1) An optional list of properties supported by the object, declared in the form of variables. This is the short form for declaring the property functions in the methods list. See the comments section for details.

methlist

(Syntax 1) A list comprising a function prototype for each method and property in the **dispinterface**. Any number of function definitions can appear in *methlist*. A function in *methlist* has the following form:

[*attributes***]** *returntype methname***(***params***);**

The following attributes are accepted on a method in a **dispinterface**: **helpstring**, **helpcontext**, **string** (for compatibility with IDL), **bindable**, **defaultbind**, **displaybind**, **propget**, **propput**, **propputref**, and **vararg.** If **vararg** is specified, the last parameter must be a safe array of VARIANT type.

The parameter list is a comma-delimited list, each element of which has the following form:

[*attributes***]** *type paramname*

The *type* can be any declared or built-in type, or a pointer to any type. Attributes on parameters are:

in, **out**, **optional**, **string**

If **optional** appears, it must only be specified on the rightmost parameters, and the types of those parameters must be VARIANT.

Comments

Method functions are specified exactly as described in "The module Statement," except that the **entry** attribute is not allowed. Note that STDOLE32.TLB (STDOLE.TLB on 16-bit systems) must be imported, because a **dispinterface** inherits from **IDispatch**.

You can declare properties in either the properties or methods lists. Declaring properties in the properties list does not indicate the type of access the property supports (that is, get, put, or putref). Specify the **readonly** attribute for properties that don't support put or putref. If you declare the property functions in the methods list, functions for one property all have the same ID.

Using Syntax 1, the **properties:** and **methods:** tags are required. The **id** attribute is also required on each member. For example:

```
properties:
     [id(0)] int Value;       // Default property.
methods:
     [id(1)] void Show();
```

Unlike **interface** members, **dispinterface** members cannot use the **retval** attribute to return a value in addition to an HRESULT error code. The **lcid** attribute is likewise invalid for dispinterfaces, because **IDispatch::Invoke** passes an LCID. However, it is possible to redeclare an interface that uses these attributes.

Using Syntax 2, interfaces that support **IDispatch** and are declared earlier in an ODL script can be redeclared as **IDispatch** interfaces as follows:

```
dispinterface helloPro {
     interface hello;
};
```

The preceding example declares all of the members of hello and all of the members that hello inherits as supporting **IDispatch**. In this case, if hello were declared earlier with **lcid** and **retval** members that returned HRESULTs, MkTypLib would remove each **lcid** parameter and HRESULT return type, and instead mark the return type as that of the **retval** parameter.

The properties and methods of a dispinterface are not part of the VTBL of the dispinterface. Consequently, **CreateStdDispatch** and **DispInvoke** can't be used to implement **IDispatch::Invoke**. The dispinterface is used when an application needs to expose existing non-VTBL functions through OLE Automation. These applications can implement **IDispatch::Invoke** by examining the *dispidMember* parameter and directly calling the corresponding function.

Example

```
[ uuid(BFB73347-822A-1068-8849-00DD011087E8), version(1.0),
helpstring("Useful help string."), helpcontext(2480)]
dispinterface MyDispatchObject {
     properties:
           [id(1)] int x;     //An integer property named x
           [id(2)] BSTR y;    //A string property named y
     methods:
           [id(3)] void show();     //No arguments, no result
           [id(11)] int computeit(int inarg, double *outarg);
};

[uuid 00000000-0000-0000-0000-123456789012]
```

```
dispinterface MyObject
{
     properties:
     methods:
           [id(1), propget, bindable, defaultbind, displaybind]
           long x();

           [id(1), propput, bindable, defaultbind, displaybind]
           void x(long rhs);
}
```

The enum Statement

Defines a C-style enumerated type.

Syntax

typedef [*attributes***] enum** [*tag*] **{**
    enumlist
 } *enumname***;**

Syntax Elements

attributes
The **helpstring**, **helpcontext**, **hidden**, and **uuid** attributes are accepted before an **enum**. The **helpstring** and **helpcontext** attributes are accepted on an enumeration element. See "Attribute Descriptions," earlier in this chapter, for more information on the attributes accepted before an enumeration definition. Attributes (including the brackets) may be omitted. If **uuid** is omitted, the enumeration is not uniquely specified in the system.

tag
An optional tag, as with a C **enum**.

enumlist
List of enumerated elements.

enumname
Name by which the enumeration is known in the type library.

Comments

The **enum** keyword must be preceded by **typedef**. The enumeration description must precede other references to the enumeration in the library. If = *value* is not specified for enumerators, the numbering progresses as with enumerations in C. The type of the **enum** elements is **int**, the system default integer, which depends on the target type library specification.

Examples

```
typedef [uuid(DEADF00D-C0DE-B1FF-F001-A100FF001ED),
           helpstring("Farm Animals are friendly"), helpcontext(234)]
enum {
     [helpstring("Moo")] cows = 1,
     pigs = 2
} ANIMALS;
```

The importlib Directive

Makes types that have already been compiled into another type library available to the library currently being created. All **importlib** directives must precede the other type descriptions in the library.

Syntax

importlib(*filename*);

Syntax Elements

filename
The location of the type library file when MkTypLib is executed.

Comments

The **importlib** directive makes any type defined in the imported library accessible from within the library being compiled. Ambiguity is resolved as follows. The current library is searched for the type. If the type can't be found, MkTypLib searches the imported library that is lexically first, then the next, and so on. To import a type name in code, you must enter the name as *libname.typename,* where *libname* is the library name as it appeared in the **library** statement when the library was compiled.

The imported type library should be distributed with the library being compiled.

Example

The following example imports the libraries STDOLE.TLB and MYDISP.TLB:

```
library BrowseHelper
{
     importlib("stdole.tlb");
     importlib("mydisp.tlb");
//Additional text omitted
}
```

The interface Statement

Defines an interface, which is a set of function definitions. An interface can inherit from any base interface.

Syntax

[*attributes***]**
  interface *interfacename* [**:***baseinterface*] **{**
    functionlist
  };

Syntax Elements

attributes
  The **dual**, **helpstring**, **helpcontext**, **hidden**, **odl**, **oleautomation**, **uuid**, and **version** attributes are accepted before **interface**. If the interface is a member of a coclass, the **source**, **default**, and **restricted** attributes are also accepted. See "Attribute Descriptions," earlier in this chapter, for more information on the attributes accepted before an interface definition.

  The **odl** and **uuid** attributes are required on all interface declarations.

interfacename
  The name by which the interface is known in the type library.

baseinterface
  The name of the interface that is the base class for this interface.

functionlist
  List of function prototypes for each function in the interface. Any number of function definitions can appear in the function list. A function in the function list has the following form:

  [*attributes***]** *returntype* **[***calling convention***]** *funcname***(***params***);**

  The following attributes are accepted on a function in an interface: **helpstring**, **helpcontext**, **string**, **propget**, **propput**, **propputref**, **bindable**, **defaultbind**, **displaybind**, and **vararg**. If **vararg** is specified, the last parameter must be a safe array of VARIANT type. The optional *calling convention* can be any one of **__pascal/_pascal/pascal** or **__cdecl/_cdecl/cdecl**, **__stdcall/_stdcall/stdcall**. In other words, the calling convention specification can include up to two leading underscores.

  The parameter list is a comma-delimited list as follows:

  [*attributes***]** *type paramname*

  The *type* can be any previously declared type, built-in type, a pointer to any type, or a pointer to a built-in type. Attributes on parameters are:

  in, **out**, **optional**, **string**

  If **optional** appears, it must only be specified on the rightmost parameters, and the types of those parameters must be VARIANT.

Comments

Because the functions described by the **interface** statement are in the VTBL, you can use **DispInvoke** and **CreateStdDispatch** to provide an implementation of **IDispatch::Invoke**. For this reason, **interface** is more commonly used than **dispinterface** to describe the properties and methods of an object.

Functions in interfaces are the same as described in "The **module** Statement," except that the **entry** attribute is not allowed.

Members of interfaces that need to raise exceptions should return an HRESULT and specify a **retval** parameter for the actual return value. The **retval** parameter is always the last parameter in the list.

Examples

The following example defines an interface named Hello with two member functions, HelloProc and Shutdown:

```
[uuid(BFB73347-822A-1068-8849-00DD011087E8), version(1.0)]
interface hello : IUnknown
{
void HelloProc([in, string] unsigned char * pszString);
void Shutdown(void);
};
```

The next example defines a **dual** interface named IMyInt, which has a pair of accessor functions for the MyMessage property and a method that returns a string.

```
[dual]
interface IMyInt : IDispatch
{
     //A property that is a string
     [propget] HRESULT MyMessage([in, lcid] LCID lcid,
                                              [out, retval] BSTR
*pbstrRetVal);
     [propput] HRESULT MyMessage([in] BSTR rhs, [in, lcid] DWORD lcid);

     //A method returning a string
     HRESULT SayMessage([in] long NumTimes,
                                 [in, lcid] DWORD lcid,
                                 [out, retval] BSTR *pbstrRetVal);
}
```

The members of this interface return error information and function return values through the HRESULT values and **retval** parameters, respectively. Tools that access the members can return the HRESULT to their users, or can simply expose the **retval** parameter as the return value and handle the HRESULT transparently.

Note that a **dual** interface must derive from **IDispatch**.

The library Statement

Describes a type library. This description contains all the other information in a MkTypLib input file.

Syntax

[attributes**] library** libname **{**
    definitions
};

Syntax Elements

attributes
The **helpstring**, **helpcontext**, **lcid, restricted, hidden**, **control**, and **uuid** attributes are accepted before a **library** statement. See "Attribute Descriptions," earlier in this chapter, for more information on the attributes accepted before a library definition. The **uuid** attribute is required.

libname
The name by which the type library is known.

definitions
Descriptions of any imported libraries, data types, modules, interfaces, dispinterfaces, and coclasses relevant to the object being exposed.

Comments

The **library** statement must precede any other type definitions.

Example

```
[
	uuid(F37C8060-4AD5-101B-B826-00DD01103DE1), // LIBID_Hello
	helpstring("Hello 2.0 Type Library"),
	lcid(0x0409),
	version(2.0)
]
library Hello
{
	importlib("stdole.tlb");
	[
		uuid(F37C8062-4AD5-101B-B826-00DD01103DE1),     // IID_IHello
		helpstring("Application object for the Hello application."),
		oleautomation,
		dual
	]
	interface IHello : IDispatch
	{
		[propget, helpstring("Returns the application of the object.")]
		HRESULT Application([in, lcid] long localeID,
			[out, retval] IHello** retval)
	}
}
```

The module Statement

Defines a group of functions, typically a set of DLL entry points.

Syntax

[attributes**]**
module modulename **{**
  elementlist
};

Syntax Elements

attributes
  The **uuid**, **version**, **helpstring**, **helpcontext**, **hidden**, and **dllname** attributes are accepted before a **module** statement. See "Attribute Descriptions," earlier in this chapter, for more information on the attributes accepted before a module definition. The **dllname** attribute is required. If **uuid** is omitted, the module is not uniquely specified in the system.

modulename
  The name of the module.

elementlist
  List of constant definitions and function prototypes for each function in the DLL. Any number of function definitions can appear in the function list. A function in the function list has the following form:

  [*attributes***]** *returntype* **[***calling convention***]** *funcname***(***params***);**
    [*attributes***]** **const** *constname* **=** *constval***;**

  Only the **helpstring** and **helpcontext** attributes are accepted for a **const**.

  The following attributes are accepted on a function in a module: **helpstring**, **helpcontext**, **string**, **entry**, **propget**, **propput**, **propputref**, **vararg**. If **vararg** is specified, the last parameter must be a safe array of VARIANT type.

  The optional *calling convention* can be one of **__pascal/_pascal**/**pascal**, **__cdecl/_cdecl**/**cdecl**, or **__stdcall/_stdcall**/**stdcall**. The *calling convention* can include up to two leading underscores.

  The parameter list is a comma-delimited list of:

  [*attributes***]** *type paramname*

  The *type* can be any previously declared type or built-in type, a pointer to any type, or a pointer to a built-in type. Attributes on parameters are:

  in, **out**, **optional**

  If **optional** appears, it must only be specified on the rightmost parameters, and the types of those parameters must be VARIANT.

Comments

The header file (.H) output for modules is a series of function prototypes. The **module** keyword and surrounding brackets are stripped from the header (.H) file output, but a comment (\\ **module** *modulename*) is inserted before the prototypes. The keyword **extern** is inserted before the declarations.

Example

```
[uuid(D00BED00-CEDE-B1FF-F001-A100FF001ED),
     helpstring("This is not GDI.EXE"), helpcontext(190),
     dllname("MATH.DLL")]
module somemodule{
     [helpstring("Color for the frame")] unsigned long const COLOR_FRAME
           = 0xH80000006;
     [helpstring("Not a rectangle but a square"), entry(1)] pascal double
square([in] double x);
};
```

The struct Statement

Defines a C-style structure.

Syntax

typedef [*attributes***]**
struct [*tag*] **{**
    memberlist
} *structname***;**

Syntax Elements

attributes
The **helpstring**, **helpcontext**, **uuid, hidden**, and **version** attributes are accepted before a **struct** statement. The **helpstring**, **helpcontext**, **string** attributes are accepted on a structure member. See "Attribute Descriptions," earlier in this chapter, for more information on the attributes accepted before a structure definition. Attributes (including the brackets) may be omitted. If **uuid** is omitted, the structure is not uniquely specified in the system.

tag
An optional tag, as with a C **struct**.

memberlist
List of structure members defined with C syntax.

structname
Name by which the structure is known in the type library.

Comments

The **struct** keyword must be preceded with **typedef**. The structure description must precede other references to the structure in the library. Members of a **struct** can be of any built-in type, or any type defined lexically as a **typedef** before the **struct**. See the sections "String Definitions" and "Array Definitions," earlier in this chapter, for a description of how strings and arrays can be entered.

Example

```
typedef [uuid(BFB7334B-822A-1068-8849-00DD011087E8),
     helpstring("A task"), helpcontext(1019)]
struct {
     DATE startdate;
     DATE enddate;
     BSTR ownername;
     SAFEARRAY (int) subtasks;
     int A_C_array[10];
} TASKS;
```

The typedef Statement

Creates an alias for a type.

Syntax

typedef [*attributes***]** *basetype aliasname***;**

Syntax Elements

attributes
Any attribute specifications must follow the **typedef** keyword. If no attributes and no other type (for example **enum**, **struct**, or **union**) are specified, the alias is treated as a **#define** and does not appear in the type library. If no other attribute is desired, **public** can be used to explicitly include the alias in the type library. The **helpstring**, **helpcontext**, and **uuid** attributes are accepted before a **typedef**. See "Attribute Descriptions," earlier in this chapter, for more information. If **uuid** is omitted, the typedef is not uniquely specified in the system.

aliasname
Name by which the type will be known in the type library.

basetype
The type for which the alias is defined.

Comments

The **typedef** keyword must also be used whenever a **struct** or **enum** is defined. The name recorded for the **enum** or **struct** is the typedef name, not the tag for the enumeration. No attributes are required to make sure the alias appears in the type library.

Enumerations, structures, and unions must be defined with the **typedef** keyword. The *attributes* for a type defined with **typedef** are enclosed in brackets following the **typedef** keyword. If a simple alias **typedef** has no attributes, it is treated like a #define and the *aliasname* does not appear in the library. Any attribute (use **public** if no others are desired) specified between the **typedef** keyword and the rest of a simple alias definition causes the alias to appear explicitly in the type library. The *attributes* typically include such items as a Help string and Help context.

Examples

```
typedef [public]  long DWORD;
```

The preceding example creates a type description for an alias type with the name DWORD.

```
typedef enum {
           TYPE_FUNCTION = 0,
           TYPE_PROPERTY = 1,
           TYPE_CONSTANT = 2,
           TYPE_PARAMETER = 3
     } OBJTYPE;
```

The second example creates a type description for an enumeration named OBJTYPE, which has four enumerator values.

The union Statement

Defines a C-style union.

Syntax

typedef [*attributes***] union** [*tag*] **{**
memberlist
} *unionname***;**

Syntax Elements

attributes
The **helpstring**, **helpcontext**, **uuid, hidden**, and **version** attributes are accepted before a **union**. The **helpstring**, **helpcontext**, and **string** attributes are accepted on a union member. See "Attribute Descriptions," earlier in this chapter, for more information on the attributes accepted before a union definition. Attributes (including the square brackets) may be omitted. If **uuid** is omitted, the union is not uniquely specified in the system.

tag
An optional tag, as with a C **union**.

memberlist
List of union members defined with C syntax.

unionname
Name by which the union is known in the type library.

Comments

The **union** keyword must be preceded with **typedef**. The union description must precede other references to the structure in the library. Members of a **union** can be of any built-in type, or any type defined lexically as a **typedef** before the **union**. See the sections "String Definitions" and "Array Definitions," earlier in this chapter, for a description of how strings and arrays can be entered.

Example

```
[uuid(BFB7334C-822A-1068-8849-00DD011087E8), helpstring("A task"),
helpcontext(1019)]
typedef union {
     COLOR polycolor;
     int   cVertices;
     boolean filled;
     SAFEARRAY (int) subtasks;
} UNIONSHOP;
```

Type Description Interfaces

The type description interfaces provide a way to read and bind to the descriptions of objects in a type library. The descriptions are used by OLE Automation controllers when they browse, create, and manipulate OLE Automation objects.

The type description interfaces include:

- **ITypeLib** –Used to retrieve information about a type library.
- **ITypeInfo** – Used to read the type information within the type library.
- **ITypeComp** – Used when creating compilers that use type information.

Overview of Type Description Interfaces

A type library is a container for type descriptions. **ITypeLib** provides access to information about a type description's containing library. **ITypeInfo** lets you access the type descriptions in a type library. The following table describes the member functions of each of the type description interfaces:

Category	Member name	Purpose
ITypeLib	**FindName**	Finds occurrences of a type description in a type library.
	GetDocumentation	Retrieves the library's documentation string, name of the complete Help file path and name, and the context ID for the library Help topic in the Help file.
	GetLibAttr	Retrieves the structure containing the library's attributes.
	GetTypeComp	Retrieves a pointer to the **ITypeComp** for a type library. This enables a client compiler to bind to the library's types, variables, constants and global functions.
	GetTypeInfo	Retrieves the specified type description in the library.
	GetTypeInfoCount	Retrieves the number of type descriptions in the library.
	GetTypeInfoType	Retrieves the type of a type description.
	GetTypeInfoOfGuid	Retrieves the type description corresponding to the specified GUID.
	IsName	Indicates whether a passed-in string contains the name of a type or member described in the library.
	ReleaseTLibAttr	Releases TLIBATTR originally obtained from **ITypeLib::GetLibAttr**.
ITypeInfo	**AddressOfMember**	Retrieves the addresses of static functions or variables, such as those defined in a DLL.
	CreateInstance	Creates a new instance of a type that describes a component object class (coclass).

GetContainingTypeLib	Retrieves both the type library that contains a specific type description and the index of the type description within the type library.
GetDllEntry	Retrieves a description or specification of an entry point for a function in a DLL.
GetDocumentation	Retrieves the documentation string, name of the complete Help file path and name, and the context ID for the Help topic for a specified type description.
GetFuncDesc	Retrieves the FUNCDESC structure containing information about a specified function.
GetIDsOfNames	Maps between member names and member IDs and parameter names and parameter IDs.
GetImplTypeFlags	Retrieves the IMPLTYPE flags for an interface.
GetMops	Retrieves marshaling information.
GetNames	Retrieves the variable with the specified member ID, or the name of the function and parameter names corresponding to the specified function ID.
GetRefTypeInfo	Retrieves the type descriptions referenced by a given type description.
GetRefTypeOfImplType	Retrieves the type description of the specified implemented interface types for a coclass, or an inherited interface.
GetTypeAttr	Retrieves a TYPEATTR structure containing the attributes of the type description.
GetTypeComp	Retrieves the **ITypeComp** interface for the type description, which enables a client compiler to bind to the type description's

		members.
	GetVarDesc	Retrieves a VARDESC structure describing the specified variable.
	Invoke	Invokes a method or accesses a property of an object that implements the interface described by the type description.
	ReleaseFuncDesc	Releases a FUNCDESC previously returned by **GetFuncDesc**.
	ReleaseTypeAttr	Releases a TYPEATTR previously returned by **GetTypeAttr**.
	ReleaseVarDesc	Releases a VARDESC previously returned by **GetVarDesc**.
ITypeComp	**Bind**	Maps a name to a member of a type, or binds global variables and functions contained in a type library.
	BindType	Binds to the type descriptions contained within a type library.

Overview of Type Description Functions

The functions for loading, registering, and querying type libraries are provided by OLEAUT32.DLL (for 32-bit systems) and TYPELIB.DLL (for 16-bit systems).

Category	Function name	Purpose
Library loading	**LoadTypeLib**	Loads and registers a type library.
	LoadRegTypeLib	Uses registry information to load a type library.
Library registration	**RegisterTypeLib**	Adds information about a type library to the system registry.
	QueryPathOfRegTypeLib	Retrieves the path of a registered type library.
Type compilation	**LHashValOfNameSys** **LHashValOfName**	Computes a hash value for a name that can then be passed to **ITypeComp::Bind**, **ITypeComp::BindType**, **ITypeLib::IsName**, or **ITypeLib::FindName**.

ITypeLib Interface

Implemented by	Used by	Header filename
OLEAUT32.DLL (32-bit systems) **TYPELIB.DLL** (16-bit systems)	Tools that need to access the descriptions of objects contained in type libraries.	**OLEAUTO.H** **DISPATCH.H**

Data describing a set of objects is stored in a type library. A type library may be a stand-alone binary file (.TLB), a resource in a DLL or .EXE file, or part of a compound document file.

A type library contains descriptions of one or more objects, and is accessed through the **ITypeLib** interface. The descriptions of individual objects are accessed through the **ITypeInfo** interface. The system registry contains a list of all the installed type libraries. Type library organization is illustrated in the following figure:

{ewc msdncd, EWGraphic, group10460 0 /a "SDK_01.bmp"}

The **ITypeLib** interface provides methods for accessing a library of type descriptions. This interface supports the following:

- Generalized containment for type information. **ITypeLib** allows iteration over the type descriptions contained in the library.
- Global functions and data. A type library can contain descriptions of a set of modules, each of which is the equivalent of a C or C++ source file that exports data and functions. The type library supports compiling references to the exported data and functions.
- General information, including a user-readable name for the library and Help for the library as a whole.

Using Structures and Enumerations with the ITypeLib Interface

The **ITypeLib** interface uses the following structures and enumerations:

Using LIBFLAGS with the ITypeLib Interface

The LIBFLAGS enumeration defines flags that apply to type libraries. LIBFLAGS is defined as follows:

```
typedef enum tagLIBFLAGS {
     LIBFLAG_FRESTRICTED = 0x01
     , LIBFLAG_FCONTROL = 0x02
     , LIBFLAG_FHIDDEN = 0x04
} LIBFLAGS;
```

Value	Description
LIBFLAG_FCONTROL	The type library describes controls and should not be displayed in type browsers intended for nonvisual objects.
LIBFLAG_FRESTRICTED	The type library is restricted and should not be displayed to users.
LIBFLAG_FHIDDEN	The type library should not be displayed to users, though its use is not restricted. It is intended for use by controls. Hosts need to create a new typelib that wraps the control with extended properties.

Using SYSKIND with the ITypeLib Interface

The SYSKIND identifies the target operating system platform.

```
typedef enum tagSYSKIND[
     SYS_WIN16,
     SYS_WIN32,
     SYS_MAC
] SYSKIND;
```

Value	Description
SYS_WIN16	The target operating system for the type library is 16-bit Windows systems. Data members are packed.
SYS_WIN32	The target operating system for the type library is 32-bit Windows systems. Data members are naturally aligned (for example, 2-byte integers are aligned on even-byte boundaries; 4-byte integers are aligned on quad-word boundaries, and so forth).
SYS_MAC	The target operating system for the type library is Macintosh. All data members are aligned on even-byte boundaries.

Using TLIBATTR with the ITypeLib Interface

The TLIBATTR structure contains information about a type library. Information from this structure is used to identify the type library and to provide national language support for member names.

```
typedef struct FARSTRUCT tagTLIBATTR {
     GUID guid;                                  // Unique ID of the library
     LCID lcid;                                  // Language/locale of the
library
     SYSKIND syskind;                            // Target hardware platform
     unsigned short wMajorVerNum;        // Major version number
     unsigned short wMinorVerNum;        // Minor version number
     unsigned short wLibFlags;           // Library flags
} TLIBATTR, FAR * LPTLIBATTR;
```

For more information on national language support, see "Supporting Multiple National Languages" in Chapter 2, and refer to the NLS API reference material in the Windows NT documentation.

ITypeLib::FindName

HRESULT ITypeLib::FindName(*szNameBuf*, *lHashVal*, *rgptinfo*, *rgmemid*, *pcFound***)**
OLECHAR FAR* *szNameBuf*
unsigned long *lHashVal*
ITypeInfo FAR* FAR* *rgptinfo*
MEMBERID FAR* *rgmemid*
unsigned int FAR* *pcFound*

Finds occurrences of a type description in a type library. This may be used to quickly verify that a name exists in a type library.

Parameter

szNameBuf
The name to search for.

lHashVal
A hash value to speed up the search, computed by **LHashValOfNameSys**. If *lHashVal* = 0, a value will be computed for you.

rgptinfo
On return, an array of pointers to the type descriptions that contain the name specified in *szNameBuf*. May not be NULL.

rgmemid
An array of the MEMBERIDs of the found items; *rgmemid*[*i*] is the MEMBERID which indexes into the type description specified by *rgptinfo*[*i*]. May not be NULL.

pcFound
On entry, indicates how many instances to look for. For example, you may call with **pcFound* = 1 in order to find the first occurrence. In this case, the search stops when one is found.

On exit, indicates the number of instances that were found. If the in and out values of **pcFound* are identical, there may be more type descriptions that contain the name.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not write to the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_CANTLOADLIBRARY	The library or DLL could not be loaded.
TYPE_E_ELEMENTNOTFOUND	The element was not found.

Comments

Passing **pcFound* = *n* indicates that there is enough room in the *rgptinfo* and *rgmemid* arrays for *n* (*ptinfo*, *memid*) pairs. The function returns MEMID_NIL in *rgmemid*[*i*] if the name in *szNameBuf* is the name of the typeinfo in *rgptinfo*[*i*].

ITypeLib::GetDocumentation

HRESULT ITypeLib::GetDocumentation(*index*, *lpbstrName*, *lpbstrDocString*, *lpdwHelpContext*, *lpbstrHelpFile***)**
int *index*
BSTR FAR* *lpbstrName*
BSTR FAR* *lpbstrDocString*
unsigned long FAR* *lpdwHelpContext*
BSTR FAR* *lpbstrHelpFile*

Retrieves the library's documentation string, name of the complete Help file path and name, and the context ID for the library Help topic in the Help file.

Parameter

index
Index of the type description whose documentation is to be returned; if index is -1, then the documentation for the library itself is returned.

lpbstrName
Returns a BSTR that contains the name of the specified item. If the caller does not need the item name, then *lpbstrName* can be NULL.

lpbstrDocString
Returns a BSTR that contains the documentation string for the specified item. If the caller does not need the documentation string, then *lpbstrDocString* can be NULL.

lpdwHelpContext
Returns the Help context ID associated with the specified item. If the caller does not need the Help context ID, then *lpdwHelpContext* can be NULL.

lpbstrHelpFile
Returns a BSTR that contains the fully qualified name of the Help file. If the caller does not need the Help filename, then *lpbstrHelpFile* can be NULL.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not write to the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_ELEMENTNOTFOUND	The element was not found.

Comments

The caller should free the BSTR parameters *lpbstrName*, *lpbstrDocString*, and *lpbstrHelpFile*.

Example

```
for (i = 0; i < utypeinfoCount; i++)
{
     CHECKRESULT(ptlib->GetDocumentation(i, &bstrName,
                                                              NULL, NULL,
NULL));
.
.
.
     SysFreeString(bstrName);
}
```

ITypeLib::GetLibAttr

HRESULT ITypeLib::GetLibAttr(*lplptlibattr***)**
TLIBATTR FAR* FAR* *lplptlibattr*

Retrieves the structure containing the library's attributes.

Parameter

lplptlibattr
Pointer to a structure containing the library's attributes.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not write to the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an unsupported format.
TYPE_E_INVALIDSTATE	The type library could not be opened.

Comments

Use **ITypeLib::ReleaseTLibAttr** to free the memory occupied by the TLIBATTR structure.

ITypeLib::GetTypeComp

HRESULT ITypeLib::GetTypeComp(*lplptcomp***)**
ITypeComp FAR* FAR* *lplptcomp*

Enables a client compiler to bind to the library's types, variables, constants and global functions.

Parameter

lplptcomp
Points to a pointer to the **ITypeComp** instance for this **ITypeLib** that a client compiler can use to bind to types in the **ITypeLib** and to the global functions, variables, and constants defined in the **ITypeLib**.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_WRONGTYPEKIND	Type mismatch.

Comments

The **Bind** function of the returned **TypeComp** binds to global functions, variables, constants, enumerated values, and coclass members. The **Bind** function also binds the names of the TYPEKINDs of TKIND_MODULE, TKIND_ENUM, and TKIND_COCLASS. These names shadow any global names defined within the type information. The members of TKIND_ENUM, TKIND_MODULE, and TKIND_COCLASS types marked as Application objects can be directly bound to from **ITypeComp**, without specifying the name of the module.

ITypeComp::Bind and **ITypeComp::BindType** accept only unqualified names. **ITypeLib::GetTypeComp** returns a pointer to the **ITypeComp** interface, which is then used to bind to global elements in the library. The names of some types (TKIND_ENUM, TKIND_MODULE, and TKIND_COCLASS) share the name space with variables, functions, constants, and enumerators. If a member requires qualification to differentiate it from other items in the name space, **GetTypeComp** can be called successively for each qualifier in order to bind to the desired member. This allows programming language compilers to access members of modules, enumerations, and coclasses, even though the member can't be bound to with a qualified name.

ITypeLib::GetTypeInfo

HRESULT ITypeLib::GetTypeInfo(*index*, *lplpitinfo***)**
   unsigned int   *index*
   ITypeInfo FAR* FAR* *lplpitinfo*

Retrieves the specified type description in the library.

Parameters

index
   Index of the **ITypeInfo** to be returned.

lplpitinfo
   If successful, returns a pointer to the **ITypeInfo**.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
TYPE_E_ELEMENTNOTFOUND	*Index* is outside the range of 0 to **GetTypeInfoCount**() -1.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_REGISTRYACCESS	There was an error accessing the system registration database.
TYPE_E_INVALIDSTATE	The type library could not be opened.

Comments

For dual interfaces, **ITypeLib::GetTypeInfo** returns only the TKIND_DISPATCH typeinfo. To get the TKIND_INTERFACE typeinfo, call **ITypeInfo::GetRefTypeOfImplType** on the TKIND_DISPATCH typeinfo, passing an index of -1. Then pass the returned typeinfo handle to **ITypeInfo::GetRefTypeInfo**.

Example

The following example gets the TKIND_INTERFACE typeinfo for a dual interface.

```
ptlib->GetTypeInfo((unsigned int) dwIndex, &ptypeinfoDisp);
ptypeinfoDisp->GetRefTypeOfImplType(-1, &phreftype);
ptypeinfoDisp->GetRefTypeInfo(phreftype, &ptypeinfoInt);
```

ITypeLib::GetTypeInfoCount

unsigned int ITypeLib::GetTypeInfoCount( )

Comments

Returns the number of type descriptions in the type library.

ITypeLib::GetTypeInfoOfGuid

HRESULT ITypeLib::GetTypeInfoOfGuid(*lpguid*, *lplpitinfo***)**
  REFGUID *lpguid*
  ITypeInfo FAR* FAR* *lplpitinfo*

Retrieves the type description corresponding to the specified GUID.

Parameters

lpguid
  Pointer to the globally unique ID of the type description.

lplpitinfo
  Pointer to a pointer to the **ITypeInfo**.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
TYPE_E_ELEMENTNOTFOUND	No type description was found in the library with the specified *guid*.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not write to the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_REGISTRYACCESS	There was an error accessing the system registration database.
TYPE_E_INVALIDSTATE	The type library could not be opened.

ITypeLib::GetTypeInfoType

HRESULT ITypeLib::GetTypeInfoType(*index*, *ptypekind***)**
unsigned int *index*
TYPEKIND FAR* *ptypekind*

Retrieves the type of a type description.

Parameters

index
The index of the type description within the type library.

ptypekind
A pointer to the TYPEKIND for the type description.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
TYPE_E_ELEMENTNOTFOUND	*Index* is outside the range of 0 to **GetTypeInfoCount**() -1.

ITypeLib::IsName

HResult ITypeLib::IsName(*szNameBuf*, *lHashVal*, *lpfName***)**
OLECHAR FAR* *szNameBuf*
unsigned long *lHashVal*
BOOL *lpfName*

Indicates whether a passed-in string contains the name of a type or member described in the library.

Parameter

szNameBuf
The string to test. If **IsName** is successful, *szNameBuf* is modified to match the case (capitalization) found in the type library.

lHashVal
The hash value of *szNameBuf*.

lpfName
Upon return, set to True if *szNameBuf* was found in the type library; otherwise False.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.

ITypeLib::ReleaseTLibAttr

void ITypeLib::ReleaseTLibAttr(*lptlibattr***)**
TLIBATTR FAR* *lptlibattr*

Releases TLIBATTR originally obtained from **ITypeLib::GetLibAttr**.

Parameter

lptlibattr
Pointer to the TLIBATTR to be freed.

Comments

Releases the specified TLIBATTR. This TLIBATTR was previously obtained with a call to **GetTypeLib::GetLibAttr**.

ITypeInfo Interface

Implemented by	Used by	Header filename
OLEAUT32.DLL (32-bit systems) TYPELIB.DLL (16-bit systems)	Tools that need to access the descriptions of objects contained in type libraries.	OLEAUTO.H DLL (32-bit systems) DISPATCH.H (16-bit systems)

This section describes **ITypeInfo**, an interface usually used for reading information about objects. For example, an object browser tool could use **ITypeInfo** to extract information about the characteristics and capabilities of objects from type libraries.

Type information interfaces are intended to describe the parts of the application that can be called by outside clients, rather than those that might be used internally to build an application.

The **ITypeInfo** interface provides access to the following:

- The set of function descriptions associated with the type; for interfaces, this contains the set of member functions in the interface.
- The set of data member descriptions associated with the type; for structures, this contains the set of fields of the type.
- The general attributes of the type such as whether it describes a structure, an interface, and so forth.

You can use the type description of an **IDispatch** interface to implement the **IDispatch** interface. See the description of **CreateStdDispatch** in Chapter 5, "Dispatch Interfaces," for more information.

An **ITypeInfo** instance provides various information about a type, and is used in different ways. A compiler could use an **ITypeInfo** to compile references to members of the type. A type interface browser could use it to find information about each member of the type. An **IDispatch** implementor could use it to provide automatic delegation of **IDispatch** calls to an interface.

Type Descriptions

The information associated with an object described by **ITypeInfo** can include a set of functions, a set of data members, and various type attributes. It is essentially the same as the information described by a C++ class declaration. A C++ class declaration can be used to define both interfaces and structures, as well as any combination of functions and data members. In addition to interfaces and structure definitions, the **ITypeInfo** interface is used to describe other types, including enumerations and aliases. Because the interface to a C file or library is simply a set of functions and variable declarations, **ITypeInfo** is used to describe these as well.

Type information comprises individual type descriptions. Each type description must have one of the following forms:

Category	ODL keyword	Description
alias	**typedef**	An alias for another type.
enumeration	**enum**	An enumeration.
structure	**struct**	A structure.
union	**union**	A single data item that can have one of a specified group of types.
module	**module**	Data and functions not accessed through VTBL entries.
IDispatch interface	**dispinterface**	**IDispatch** properties and methods accessed through **IDispatch::Invoke**.
OLE interface	**interface**	OLE member functions accessed through VTBL entries.
Component object class	**coclass**	A component object class. Specifies an implementation of one or more OLE interfaces and one or more **IDispatch** interfaces.

Note that all bit flags that are not specifically used should be set to zero for future compatibility.

Alias

An alias has TypeKind = TKIND_ALIAS. An alias is an empty set of functions, an empty set of data members, and a type description (located in the TYPEATTR) which gives the actual type of the alias.

Enumeration

An enumeration has TypeKind = TKIND_ENUM. An enumeration is an empty set of functions and a set of constant data members.

Structure

A structure description has TypeKind = TKIND_RECORD. A structure is an empty set of functions and a set of per-instance data members.

Union

A union description has TypeKind = TKIND_UNION. A union is an empty set of functions and a set of per-instance data members, each of which has an instance offset of zero.

Module

A module has TypeKind = TKIND_MODULE. A module is a set of static functions and a set of static data members.

OLE-compatible Interfaces

An interface definition has TypeKind = TKIND_INTERFACE. An interface is a set of pure virtual functions and an empty set of data members. If a type description contains any virtual functions, then the pointer to the VTBL is the first 4 bytes of the instance.

The type information fully describes the member functions in the vtable, including parameter names and types and function return types. It may inherit from no more than one other interface.

Note that with interfaces and dispinterfaces, all members should have different names, except the accessor functions of properties. For property functions having the same name, the documentation string and Help context should be set for only one of the functions (because they conceptually define the same property).

Dispatch Interfaces

These include objects (TypeKind = TKIND_DISPATCH) that support the **IDispatch** interface with a specification of the dispatch data members (for example, properties) and methods supported through the object's **Invoke** implementation. All members of the dispinterface should have different IDs, except for the accessor functions of properties.

Dual Interfaces

Dual interfaces have two different typeinfos for the same interface. The TKIND_INTERFACE typeInfo describes the interface as a standard OLE COM interface. The TKIND_DISPATCH typeinfo describes the interface as a standard Dispatch interface. The **lcid** and **retval** parameters and HRESULT return types are removed, and the return type of the member is specified to be the same type as the **retval** parameter.

By default, the TypeKind for a dual interface is TKIND_DISPATCH. Tools that bind to interfaces should check the type flags for TYPEFLAG_FDUAL. If this flag is set, the TKIND_INTERFACE typeinfo is available through a call to **ITypeInfo::GetRefTypeOfImplType** with an index of -1, followed by a call to **ITypeInfo::GetRefTypeInfo**.

Component Object Classes (coclasses)

These include objects (TypeKind = TKIND_COCLASS) that support a set of implemented interfaces, which can be of either TKIND_INTERFACE or TKIND_DISPATCH.

Using Structures and Enumerations with the ITypeInfo Interface

Structures and enumerations used by **ITypeInfo** include the following:

ARRAYDESC

A pointer to an ARRAYDESC is contained within the TYPEDESC, which describes a C-style array. The TYPEDESC describes the type of the array's elements and information describing the array's dimensions. It is defined as follows:

```
typedef struct tagARRAYDESC{
     TYPEDESC tdescElem;                  // Element type.
     unsigned short cDims;          // Dimension count.
     SAFEARRAYBOUND rgbounds[1];    // Variable length array containing one
                                          // element for each
dimension.
} ARRAYDESC;
```

ELEMDESC

An ELEMDESC structure includes the type description and process-transfer information for a variable, a function, or a function parameter. It is defined as follows:

```
typedef struct tagELEMDESC{
     TYPEDESC tdesc;         // Type of the element.
     IDLDESC idldesc;        // Information needed for transferring the
                                   // element between processes.
     }  ELEMDESC;
```

FUNCDESC

A FUNCDESC describes a function, and is defined as follows:

```
typedef struct tagFUNCDESC {
     MEMBERID memid;                // Function member ID
     SCODE FAR* lprgscode;    // Legal SCODES for the function.
     ELEMDESC FAR* lprgelemdescParam;  // Array of parameter types
     FUNCKIND funckind;             // specifies whether the function is
                                                  // virtual, static, or
dispatch-only.
     INVOKEKIND invkind;            // Invocation kind; indicates if this is
a                                         //property function and if so, what
kind.
     CALLCONV callconv;             // Specifies the function's calling
                                                        // convention.
     short cParams;                 // Count of total number of parameters.
     short cParamsOpt;        // Count of optional parameters (detailed
                                          // description below).
     short oVft;                    // For FUNC_VIRTUAL, specifies the offset
in                                        // the virtual function table.
     short cScodes;                 // Count of permitted Scodes.
     ELEMDESC elemdescFunc;  // Contains the return type of the function.
     unsigned short wFuncFlags; // See below for definition of flags.
}    FUNCDESC;
```

The field *cParams* specifies the total number of required and optional parameters.

The *cParamsOpt* field specifies the form of optional parameters accepted by the function, as follows:

- A value of 0 specifies that no optional arguments are supported.
- A value of -1 specifies that the method's last parameter is a pointer to a safe array of variants. Any number of variant arguments greater than *cParams* -1 must be packaged by the caller into a safe array and passed as the final parameter. This array of optional parameters must be freed by the caller after control is returned from the call.
- Any other number indicates that the last *n* parameters of the function are variants and need not be specified explicitly by the caller. The parameters left unspecified should be filled in by the compiler or interpreter as variants of type VT_ERROR with the value DISP_E_PARAMNOTFOUND.

The fields *cScodes* and *lprgscode* store the count and the set of errors that a function can return. If *cScodes* = -1 then the set of errors is unknown.
If *cScodes* = -1 or *cScodes* = 0, then *lprgscode* is undefined.

FUNCFLAGS

The FUNCFLAGS enumeration is defined as follows:

```
typedef enum tagFUNCFLAGS {
     FUNCFLAG_FRESTRICTED= 1
           , FUNCFLAG_FSOURCE= 0x2
           , FUNCFLAG_FBINDABLE= 0x4
           , FUNCFLAG_FREQUESTEDIT= 0x8
           , FUNCFLAG_FDISPLAYBIND= 0x10
           , FUNCFLAG_FDEFAULTBIND= 0x20
           , FUNCFLAG_FHIDDEN= 0x40
} FUNCFLAGS;
```

Value	Description
FUNCFLAG_FRESTRICTED	The function should not be accessible from macro languages. This flag is intended for system-level functions or functions that you do not want type browsers to display.
FUNCFLAG_FSOURCE	The function returns an object that is a source of events.
FUNCFLAG_FBINDABLE	The function that supports data binding.
FUNCFLAG_FDISPLAYBIND	The function that is displayed to the user as bindable; that is, FUNC_FBINDABLE must also be set.
FUNCFLAG_FDEFAULTBIND	The function that best represents the object. Only one function in a typeinfo may have this attribute.
FUNCFLAG_FHIDDEN	The function should not be displayed to the user, though it exists and is bindable.

FUNCKIND

The FUNCKIND enumeration is defined as follows:

```
typedef enum tagFUNCKIND {
     FUNC_VIRTUAL,
     FUNC_PUREVIRTUAL,
     FUNC_NONVIRTUAL,
     FUNC_STATIC,
     FUNC_DISPATCH,
} FUNCKIND;
```

Value	Description
FUNC_PUREVIRTUAL	The function is accessed through the virtual function table and takes an implicit *this* pointer.
FUNC_VIRTUAL	The function is accessed the same as PUREVIRTUAL, except the function has an implementation.
FUNC_NONVIRTUAL	The function is accessed by static address and takes an implicit *this* pointer.
FUNC_STATIC	The function is accessed by static address and does not take an implicit *this* pointer.
FUNC_DISPATCH	The function can be accessed only through **IDispatch**.

HREFTYPE

HREFTYPE is a handle that identifies a type description.

```
typedef unsigned long HREFTYPE;
```

IDLDESC

An IDLDESC contains information needed for transferring a structure element, parameter, or function return value between processes, and is defined as follows:

```
typedef struct FARSTRUCT tagIDLDESC {
     unsigned long dwReserved;
     unsigned short wIDLFlags;     /* IN, OUT, and so on */
} IDLDESCtypedef struct tagIDLDESC{
```

Note that *dwReserved* is reserved for future use and should be set to NULL.

IDLFLAGS

The IDLFLAGS are defined as follows:

```
#define IDLFLAG_NONE         0
#define IDLFLAG_FIN          0x1
#define IDLFLAG_FOUT         0x2
#define IDLFLAG_FLCID  0x4
#define IDLFLAG_FRETVAL0x8
```

Value	Description
IDLFLAG_NONE	Whether the parameter passes or receives information is unspecified. **IDispatch** interfaces can use this flag.
IDLFLAG_FIN	Parameter passes information from the caller to the callee.
IDLFLAG_FOUT	Parameter returns information from the callee to the caller.
IDLFLAG_FIN | IDLFLAG_FOUT	Parameter passes and returns information.
IDLFLAG_FLCID	Parameter is the LCID of a client application.
IDLFLAG_FRETVAL	Parameter is the return value of the member.

IMPLTYPEFLAGS

The IMPLTYPEFLAGS are defined as follows:

```
/* IMPLTYPE Flags */
#define IMPLTYPEFLAG_FDEFAULT0x1
#define IMPLTYPEFLAG_FSOURCE       0x2
#define IMPLTYPEFLAG_FRESTRICTED   0x4
```

Value	Description
IMPLTYPEFLAG_FDEFAULT	The interface or dispinterface represents the default for the source or sink.
IMPLTYPEFLAG_FSOURCE	This member of a coclass is called rather than implemented.
IMPLTYPEFLAG_FRESTRICTED	The member should not be displayed or programmable by users.

INVOKEKIND

The INVOKEKIND enumeration is defined as follows:

```
typedef enum tagINVOKEKIND {
     INVOKE_FUNC = DISPATCH_METHOD,
     INVOKE_PROPERTYGET = DISPATCH_PROPERTYGET,
     INVOKE_PROPERTYPUT = DISPATCH_PROPERTYPUT,
     INVOKE_PROPERTYPUTREF = DISPATCH_PROPERTYPUTREF
} INVOKEKIND;
```

Value	Description
INVOKE_FUNC	The member is called using normal function invocation syntax.
INVOKE_PROPERTYGET	The function is invoked by using normal property access syntax.
INVOKE_PROPERTYPUT	The function is invoked by using property value assignment syntax. Syntactically, a typical programming language might represent changing a property in the same way as assignment; for example: object.property := value.
INVOKE_PROPERTYPUTREF	The function is invoked by using property reference assignment syntax.

Note that in C, value assignment is written as *pobj1 = *pobj2, while reference assignment is written as pobj1 = pobj2. Other languages have other syntactic conventions. A property or data member may support value assignment only, reference assignment only, or both. For a more detailed description of property functions, see Chapter 5, "Dispatch Interfaces." These enumeration constants are the same constants that are passed to **IDispatch::Invoke** to specify the way in which a function is invoked.

MEMBERID

MEMBERID identifies the member in a type description. For **IDispatch** interfaces, this is the same as DISPID.

```
typedef DISPID MEMBERID;
```

This is a 32-bit integral value in the following format:

Bits	Value
0-15	Offset. Any value is permissible.
16-21	The nesting level of this typeinfo in the inheritance hierarchy. For example: `interface mydisp : IDispatch` The nesting level of **IUnknown** is 0, **IDispatch** is 1, and mydisp is 2.
22-25	Reserved; must be zero
26-28	DISPID value.
29	True if this is the member ID for a FuncDesc; otherwise False.
30-31	Must be 01.

Negative IDs are reserved for use by OLE Automation.

TYPEATTR

The TYPEATTR structure contains attributes of an **ITypeInfo**, and is defined as follows:

```
typedef struct FARSTRUCT tagTYPEATTR {
     GUID guid;                          // The GUID of the TypeInfo
     LCID lcid;                          // Locale of member names and doc
                                               // strings
     unsigned long dwReserved;
     MEMBERID memidConstructor;    // ID of constructor,MEMBERID_NIL if
                                               // None
     MEMBERID memidDestructor;     // ID of destructor, MEMBERID_NIL if
                                               // None
     OLECHAR FAR* lpstrSchema;     // Reserved for future use
     unsigned long cbSizeInstance;// The size of an instance of
                                                     // this type.
     TYPEKIND typekind;                  // The kind of type this typeinfo
                                                           // describes.
     unsigned short cFuncs;        // Number of functions
     unsigned short cVars;         // Number of variables/data members
     unsigned short cImplTypes;    // Number of implemented interfaces
     unsigned short cbSizeVft;     // The size of this type's virtual
                                                     // func table
     unsigned short cbAlignment;   // Byte alignment for an instance
                                                     // of this type
     unsigned short wTypeFlags;
     unsigned short wMajorVerNum;  // Major version number
     unsigned short wMinorVerNum;  // Minor version number
     TYPEDESC tdescAlias;                // If TypeKind == TKIND_ALIAS,
                                                                 //
specifies the type for which
                 // this type is an alias
     IDLDESC idldescType;                // IDL attributes of the
                                                                 //
described type
} TYPEATTR, FAR* LPTYPEATTR;
```

The *cbAlignment* field indicates how addresses are aligned. A value of 0 indicates alignment on the 64K boundary; 1 indicates no special alignment. For other values, *n* indicates aligned on byte *n*.

TYPEDESC

A TYPEDESC, which describes the type of a variable, the return type of a function, or the type of a function parameter, is defined as follows:

```
typedef struct FARSTRUCT tagTYPEDESC {
     union {
           /* VT_PTR|VT_SAFEAEEAY - the pointed-at type */
     struct FARSTRUCT tagTYPEDESC FAR* lptdesc;

           /* VT_CARRAY */
     struct FARSTRUCT tagARRAYDESC FAR* lpadesc;

           /* VT_USERDEFINED - this is used to get a TypeInfo for a user-
              defined type */
     HREFTYPE hreftype;

     }UNION_NAME(u);
     VARTYPE vt;
} TYPEDESC;
```

If the variable is VT_SAFEARRAY or VT_PTR, the union portion of the TYPEDESC contains a pointer to a TYPEDESC that specifies the element type.

TYPEFLAGS

The TYPEFLAGS enumeration is defined as follows:

```
typedef enum tagTYPEFLAGS {
     TYPEFLAG_FAPPOBJECT = 0x01
     , TYPEFLAG_FCANCREATE = 0x02
     , TYPEFLAG_FLICENSED = 0x04
     , TYPEFLAG_FPREDECLID = 0x08
     , TYPEFLAG_FHIDDEN = 0x10
     , TYPEFLAG_FCONTROL = 0x20
     , TYPEFLAG_FDUAL = 0x40
     , TYPEFLAG_FNONEXTENSIBLE = 0x80
     , TYPEFLAG_FOLEAUTOMATION = 0x100
} TYPEFLAGS;
```

Value	Description
TYPEFLAG_FAPPOBJECT	A type description that describes an Application object.
TYPEFLAG_FCANCREATE	Instances of the type can be created by **ITypeInfo::CreateInstance**.
TYPEFLAG_FLICENSED	The type is licensed.
TYPEFLAG_FPREDECLID	The type is predefined. The client application should automatically create a single instance of the object that has this attribute. The name of the variable that points to the object is the same as the class name of the object.
TYPEFLAG_FHIDDEN	The type should not be displayed to browsers.
TYPEFLAG_FCONTROL	The type is a control from which other types will be derived, and should not be displayed to users.
TYPEFLAG_FDUAL	The types in the interface derive from **IDispatch** and are fully compatible with OLE Automation. Not allowed on **Dispatch** interfaces.
TYPEFLAG_FNONEXTENSIBLE	The interface can't add members at run time.
TYPEFLAG_FOLEAUTOMATION	The types used in the interface are fully compatible with OLE, and may be displayed in an object browser. Setting **dual** on an interface sets this flag in addition to TYPEFLAG_FDUAL. Not allowed on dispinterfaces.

TYPEFLAG_FAPPOBJECT may be used on type descriptions with TypeKind = TKIND_COCLASS, and indicates that the type description describes an Application object.

Members of the Application object are globally accessible in that the Bind method of the **ITypeComp** instance associated with the library binds to the members of an Application object just as it does for type descriptions that have TypeKind = TKIND_MODULE.

The type description implicitly defines a global variable with the same name and type as the type described by the type description. This variable is also globally accessible. Specifically, when Bind is passed the name of an Application object, a VARDESC is returned describing the implicit variable. The ID of the implicitly created variable is always ID_DEFAULTINST.

When the **CreateInstance** function of an Application object type description is called, it uses **GetActiveObject** to retrieve the Application object. If **GetActiveObject** fails because the application is not running, then **CreateInstance** calls **CoCreateInstance** (which should start the application).

When TYPEFLAG_FCANCREATE is True, **ITypeInfo::CreateInstance** can create an instance of the type. Note that this is currently true only for component object classes for which a GUID has been specified.

TYPEKIND

The TYPEKIND enumeration is defined as follows:

```
typedef enum tagTYPEKIND {
       TKIND_ENUM = 0
     , TKIND_RECORD
     , TKIND_MODULE
     , TKIND_INTERFACE
     , TKIND_DISPATCH
     , TKIND_COCLASS
     , TKIND_ALIAS
     , TKIND_UNION
     , TKIND_MAX             /* end of enum marker */
} TYPEKIND;
```

Value	Description
TKIND_ALIAS	A type that is an alias for another type.
TKIND_COCLASS	A set of implemented component object interfaces.
TKIND_DISPATCH	A set of methods and properties that are accessible through **IDispatch::Invoke**. By default, dual interfaces return TKIND_DISPATCH.
TKIND_ENUM	A set of enumerators.
TKIND_INTERFACE	A type that has virtual functions, all of which are pure.
TKIND_MODULE	A module which can only have static functions and data (for instance, a DLL).
TKIND_RECORD	A struct with no methods.
TKIND_UNION	A union, all of whose members have offset zero.

VARDESC

A VARDESC structure describes a variable, constant, or data member, and is defined as follows:

```
typedef struct FARSTRUCT tagVARDESC {
     MEMBERID memid;
     OLECHAR FAR* lpstrSchema;/* reserved for future use */
     union {
                                         /* VAR_PERINSTANCE - the offset of
this                                                    variable within the
instance */
           unsigned long oInst;

                                         /* VAR_CONST—the value of the
constant */
     VARIANT FAR* lpvarValue;

     }UNION_NAME(u);
     ELEMDESC elemdescVar;
     unsigned short wVarFlags;
     VARKIND varkind;
} VARDESC
```

VARFLAGS

The VARFLAGS enumeration is defined as follows:

```
typedef enum tagVARFLAGS {
           VARFLAG_FREADONLY= 1
       , VARFLAG_FSOURCE= 0x2
       , VARFLAG_FBINDABLE= 0x4
       , VARFLAG_FREQUESTEDIT= 0x8
       , VARFLAG_FDISPLAYBIND= 0x10
       , VARFLAG_FDEFAULTBIND= 0x20
       , VARFLAG_FHIDDEN= 0x40
} VARFLAGS;
```

Value	Description
VARFLAG_READONLY	Assignment to the variable should not be allowed.
VARFLAG_FSOURCE	The variable returns an object that is a source of events.
VARFLAG_FBINDABLE	The variable supports data binding.
VARFLAG_FDISPLAYBIND	The variable is displayed to the user as bindable; that is, VARFLAG_FBINDABLE must also be set.
VARFLAG_FDEFAULTBIND	The variable is the single property that best represents the object. Only one variable in a typeinfo may have this attibute.
VARFLAG_FHIDDEN	The variable should not be displayed to the user in a browser, though it exists and is bindable.

VARKIND

The VARKIND enumeration is defined as follows:

```
typedef enum tagVARKIND {
     VAR_PERINSTANCE,
     VAR_STATIC,
     VAR_CONST,
     VAR_DISPATCH
} VARKIND;
```

Value	Description
VAR_PERINSTANCE	The variable is a field or member of the type; it exists at a fixed offset within each instance of the type.
VAR_STATIC	There is only one instance of the variable.
VAR_CONST	The VARDESC describes a symbolic constant. There is no memory associated with it.
VAR_DISPATCH	The variable can only be accessed through **IDispatch::Invoke**.

ITypeInfo::AddressOfMember

HRESULT ITypeInfo::AddressOfMember(*memid*, *invkind*, *lplpvoid***)**
MEMBERID *memid*
INVOKEKIND *invkind*
VOID FAR* FAR* *lplpvoid*

Retrieves the addresses of static functions or variables, such as those defined in a DLL.

Parameters

memid
Member ID of the static member whose address is to be retrieved.
The member ID is defined by DISPID.

invkind
Specifies whether the member is a property, and if so, what kind.

lplpvoid
Upon return, points to a pointer to the static member.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_WRONGTYPEKIND	Type mismatch.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_ELEMENTNOTFOUND	The element was not found.
TYPE_E_DLLFUNCTIONNOTFOUND	The function could not be found in the DLL.
TYPE_E_CANTLOADLIBRARY	The type library or DLL could not be loaded.

Comments

The addresses are valid until the caller releases its reference to the type description. Note that the *invkind* parameter can be ignored unless the address of a property function is being requested.

If the type description inherits from another type description, this function recurses on the base type description, if necessary, to find the item with the requested member ID.

ITypeInfo::CreateInstance

HRESULT ITypeInfo::CreateInstance(*punkOuter*, *riid*, *ppvObj***)**
  IUnknown FAR* *punkOuter*
  REFIID *riid*
  VOID FAR* FAR* *ppvObj*

Creates a new instance of a type that describes a component object class (coclass).

Parameters

punkOuter
  A pointer to the controlling IUnknown. If NULL, then a stand-alone instance is created. If valid, then an aggregate object is created.

riid
  An ID for the interface the caller will use to communicate with the resulting object.

ppvObj
  On return, points to a pointer to an instance of the created object.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
TYPE_E_WRONGTYPEKIND	Type mismatch.
E_INVALIDARG	One or more of the arguments is invalid.
E_NOINTERFACE	OLE could not find an implementation of one or more required interfaces.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
Other returns	Additional errors may be returned from **GetActiveObject** or **CoCreateInstance**.

Comments

For types that describe a component object class (coclass), **CreateInstance** creates a new instance of the class. Normally **CreateInstance** calls **CoCreateInstance** with the type description's GUID. For an Application object, it first calls **GetActiveObject**. If the application is active, **GetActiveObject** returns the active object; otherwise, if **GetActiveObject** fails, **CreateInstance** calls **CoCreateInstance**.

ITypeInfo::GetContainingTypeLib

HRESULT ITypeInfo::GetContainingTypeLib(*lplptlib*, *lpindex***)**
  ITypeLib FAR* FAR* *lplptlib*
  unsigned int FAR* *lpindex*

Retrieves the containing type library and the index of the type description within that type library.

Parameters

lplptlib
  Upon return, points to a pointer to the containing type library.

lpindex
  Upon return, points to the index of the type description within the containing type library.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
E_NOINTERFACE	OLE could not find an implementation of one or more required interfaces.
TYPE_E_IOERROR	The function could not write to the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.

ITypeInfo::GetDllEntry

HRESULT ITypeInfo::GetDllEntry(*memid*, *invkind*, *lpbstrDllName***,** *lpbstrName***,** *lpwOrdinal***)**
MEMBERID *memid*
INVOKEKIND *invkind*
BSTR FAR* *lpbstrDllName*
BSTR FAR* *lpbstrName*
unsigned short FAR* *lpwOrdinal*

Retrieves a description or specification of an entry point for a function in a DLL.

Parameters

memid
ID of the member function whose DLL entry description is to be returned.

invkind
Specifies the kind of member identified by *memid*. This is important for properties, because one *memid* can identify up to three separate functions.

lpbstrDllName
If not NULL, the function sets **lpbstrDllName* to a BSTR containing the DLL name.

lpbstrName
If not NULL, the function sets **lpbstrName* to a BSTR containing the name of the entry point; if the entry point is specified by an ordinal, **lpbstrName* is set to NULL.

lpwOrdinal
If not NULL, and if the function is defined by ordinal, then *lpwOrdinal* is set to point to the ordinal.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
E_NOINTERFACE	OLE could not find an implementation of one or more required interfaces.
TYPE_E_ELEMENTNOTFOUND	The element was not found.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_WRONGTYPEKIND	Type mismatch.

Comments

The caller passes in a MEMID representing the member function whose entry description is desired. If the function has a DLL entry point, the name of the DLL containing the function, and either its name or ordinal identifier are placed in the passed-in pointers allocated by the caller. If there is no DLL entry point for the function, an error is returned.

If the type description inherits from another type description, this function recurses on the base type description if necessary, to find the item with the requested member ID.

The caller should use **SysFreeString**() to free the BSTRs referenced by *lpbstrName* and *lpbstrDllName*.

ITypeInfo::GetDocumentation

HRESULT ITypeInfo::GetDocumentation(*memid*, *lpbstrName*, *lpbstrDocString*, *lpdwHelpContext*, *lpbstrHelpFile***)**
MEMBERID *memid*
BSTR FAR* *lpbstrName*
BSTR FAR* *lpbstrDocString*
unsigned long FAR* *lpdwHelpContext*
BSTR FAR* *lpbstrHelpFile*

Retrieves the documentation string, name of the complete Help file path and name, and the context ID for the Help topic for a specified type description.

Parameters

memid
ID of the member whose documentation is to be returned.

lpbstrName
Pointer to a BSTR allocated by the callee into which the name of the specified item is placed. If the caller does not need the item name, *lpbstrName* can be NULL.

lpbstrDocString
Pointer to a BSTR into which the documentation string for the specified item is placed. If the caller does not need the documentation string, *lpbstrDocString* can be NULL.

lpdwHelpContext
Pointer to the Help context associated with the specified item. If the caller does not need the Help context, the *lpdwHelpContext* can be NULL.

lpbstrHelpFile
Pointer to a BSTR into which the fully qualified name of the Help file is placed. If the caller does not need the Help filename, *lpbstrHelpFile* can be NULL.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_ELEMENTNOTFOUND	The element was not found.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_ELEMENTNOTFOUND	The element was not found.

Comments

Provides access to the documentation for the member specified by the *memid* parameter. If the passed-in *memid* is MEMBERID_NIL, then the documentation for the type description is returned.

If the type description inherits from another type description, this function recurses on the base type description, if necessary, to find the item with the requested member ID.

The caller should use **SysFreeString**() to free the BSTRs referenced by *lpbstrName*, *lpbstrDocString*, and *lpbstrHelpFile*.

Example

```
CHECKRESULT(ptypeinfo->GetDocumentation(idMember, &bstrName, NULL, NULL,
     NULL));
.
.
.
SysFreeString (bstrName);
```

ITypeInfo::GetFuncDesc

HRESULT ITypeInfo::GetFuncDesc(*index*, *lplpfuncdesc***)**
  unsigned int *index*
  FUNCDESC FAR* FAR* *lplpfuncdesc*

Retrieves the FUNCDESC structure containing information about a specified function.

Parameters

index
  Index of the function whose description is to be returned. The index should be in the range of 0 to 1 less than the number of functions in this type.

lplpfuncdesc
  Upon return, points to a pointer to a FUNCDESC that describes the specified function.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.

Comments

Provides access to a FUNCDESC that describes the function with the specified index. The FUNCDESC should be freed with **ITypeInfo::ReleaseFuncDesc**(). The number of functions in the type is one of the attributes contained in the TYPEATTR structure.

Example

```
CHECKRESULT(ptypeinfo->GetFuncDesc(i, &pfuncdesc));
idMember = pfuncdesc->elemdescFunc.ID;
CHECKRESULT(ptypeinfo->GetDocumentation(idMember,
        &bstrName, NULL, NULL, NULL));
ptypeinfo->ReleaseFuncDesc(pfuncdesc);
```

ITypeInfo::GetIDsOfNames

HRESULT ITypeInfo::GetIDsOfNames(*rgszNames*, *cNames*, *rgmemid***)**
OLECHAR FAR* FAR* *rgszNames*
unsigned int *cNames*
MEMBERID FAR* *rgmemid*

Maps between member names and member IDs, and parameter names and parameter IDs.

Parameters

rgszNames
Passed-in pointer to an array of names to be mapped.

cNames
Count of the names to be mapped.

rgmemid
Caller-allocated array in which name mappings are placed.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
DISP_E_UNKNOWNNAME	One or more of the names could not be found.
DISP_E_UNKNOWNLCID	The LCID could not be found in the OLE DLLs.
TYPE_E_IOERROR	The function could not write to the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_WRONGTYPEKIND	Type mismatch.

Comments

Maps the name of a member (*rgszNames*[0]) and its parameters (*rgszNames*[1] ...*rgszNames*[*cNames* - 1]) to the ID of the member (*rgid*[0]) and the IDs of the specified parameters (*rgid*[1] ... *rgid*[*cNames* - 1]). Note that the IDs of parameters are 0 for the first parameter in the member function's argument list, 1 for the second, and so on.

If the type description inherits from another type description, this function recurses on the base type description, if necessary, to find the item with the requested member ID.

ITypeInfo::GetImplTypeFlags

HRESULT ITypeInfo:: GetImplTypeFlags(*index*, *pimpltypeflags***)**
unsigned int *index*
MEMBERID FAR* *pimpltypeflags*

Retrieves the IMPLTYPEFLAGS for one implemented interface or base interface in a type description.

Parameters

index
Index of the implemented interface or base interface for which to get the flags.

pimpltypeflags
On return, pointer to the IMPLTYPEFLAGS.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_WRONGTYPEKIND	Type mismatch.

Comments

The flags are associated with the act of inheritance, not with the inherited interface.

ITypeInfo::GetMops

HRESULT ITypeInfo::GetMops(*memid*, *lpbstrMops*)
MEMBERID *memid*
BSTR FAR* *lpbstrMops*

Retrieves marshaling information.

Parameters

memid
Member ID indicating which marshaling information is sought.

lpbstrMops
Upon return, points to a pointer to the opcode string used in marshaling the fields of the structure described by the referenced type description, or NULL if there is no information to return.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_ELEMENTNOTFOUND	The element was not found.
TYPE_E_WRONGTYPEKIND	Type mismatch.

Comments

If the passed-in member ID is MEMBERID_NIL, the function returns the opcode string for marshaling the fields of the structure described by the type description. Otherwise, it returns the opcode string for marshaling the function specified by the index.

If the type description inherits from another type description, this function recurses on the base type description, if necessary, to find the item with the requested member ID.

ITypeInfo::GetNames

HRESULT ITypeInfo::GetNames(*memid*, *rgbstrNames*, *cNameMax*, *lpcName*)
MEMBERID *memid*
BSTR FAR* *rgbstrNames*
unsigned int *cNameMax*
unsigned int FAR* *lpcName*

Retrieves the variable with the specified member ID, or the name of the property or method and its parameters, corresponding to the specified function ID.

Parameters

memid
ID of member whose name (or names) is to be returned.

rgbstrNames
Pointer to caller-allocated array. On return, each of these *lpcName* elements is filled in to point to a BSTR containing the name (or names) associated with the member.

cNameMax
Length of the passed-in *rgbstrNames* array.

lpcName
On return, points to number representing the number of names in *rgbstrNames* array.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_WRONGTYPEKIND	Type mismatch.
TYPE_E_ELEMENTNOTFOUND	The element was not found.

Comments

The caller must release the returned BSTR array.

If the member ID identifies a property that is implemented with property functions, the property name is returned.

For property get functions, the names of the function and its parameters are always returned.

For property put and put reference functions, the right side of the assignment is unnamed. If *cNameMax* is less than is required to return all the names of parameters of a function, then only the names of the first *cNameMax* - 1 parameters are returned. The names of the parameters are returned in the array in the same order they appear elsewhere in the interface, for example, in the same order they appear in the parameter array associated with the FUNCDESC.

If the type description inherits from another type description, this function recurses on the base type description, if necessary, to find the item with the requested member ID.

ITypeInfo::GetRefTypeInfo

HRESULT ITypeInfo::GetRefTypeInfo(*hreftype*, *lplptinfo*)
HREFTYPE *hreftype*
ITypeInfo FAR* FAR* *lplptinfo*

If a type description references other type descriptions, this function retrieves the referenced type descriptions.

Parameters

hreftype
Handle to the referenced type description to be returned.

lplptinfo
Points to a pointer to the referenced type description.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_WRONGTYPEKIND	Type mismatch.
TYPE_E_ELEMENTNOTFOUND	The element was not found.
TYPE_E_REGISTRYACCESS	There was an error accessing the system registration database.
TYPE_E_LIBNOTREGISTERED	The type library was not found in the system registration database.

Comments

On return, the second parameter contains a pointer to a pointer to a type description that is referenced by this type description. A type description must have a reference to each type description that occurs as the type of any of its variables, function parameters, or function return types. For example, if the type of a data member is a record type, the typeinfo for that data member contains the *hreftype* of a referenced type description. To get a pointer to the type description, the reference is passed to **GetRefTypeInfo**.

ITypeInfo::GetRefTypeOfImplType

HRESULT ITypeInfo::GetRefTypeOfImplType(*index*, *lphreftype***)**
unsigned int *index*
HREFTYPE FAR* *lphreftype*

If this type description describes a component object class, the function retrieves the type description of the specified implemented interface types. For an interface, **GetRefTypeOfImplType** returns the type information for inherited interfaces, if any exist.

Parameters

index
Index of the implemented type whose handle is returned. The valid range is 0 to the *cImplTypes* field in the TYPEATTR structure.

lphreftype
Upon return, points to a handle for the implemented interface (if any). This handle can be passed to **ITypeInfo::GetRefTypeInfo** to get the type description.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
TYPE_E_ELEMENTNOTFOUND	Passed index is outside the range 0 to 1 less than the number of function descriptions.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_IOERROR	The function could not write to the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.

Comments

If you have the TKIND_DISPATCH typeinfo for a dual interface, you can get the TKIND_INTERFACE typeinfo by calling **GetRefTypeOfImplType** with an *Index* of -1, and passing the returned *lpHRefType* handle to **GetRefTypeInfo** to retrieve the typeinfo.

ITypeInfo::GetTypeAttr

HRESULT ITypeInfo::GetTypeAttr(*lplptypeattr***)**
  TYPEATTR FAR* FAR* *lplptypeattr*

Retrieves a TYPEATTR structure containing the attributes of the type description.

Parameter

lplptypeattr
  Upon return, points to a pointer to a structure that contains the attributes of this type description.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not write to the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.

Comments

To free the TYPEATTR structure, use **ITypeInfo::ReleaseTypeAttr.**

Example

```
CHECKRESULT(ptypeinfoCur->GetTypeAttr(&ptypeattrCur));
.
.
.
ptypeinfoCur->ReleaseTypeAttr(ptypeattrCur);
```

ITypeInfo::GetTypeComp

HRESULT ITypeInfo::GetTypeComp(*lplpcomp***)**
ITypeComp FAR* FAR* *lplpcomp*

Retrieves the **ITypeComp** interface for the type description, which enables a client compiler to bind to the type description's members.

Parameter

lplpcomp
Upon return, points to a pointer to the **ITypeComp** of the containing type library.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_WRONGTYPEKIND	Type mismatch.

Comments

A client compiler can use the **ITypeComp** interface to bind to members of the type.

ITypeInfo::GetVarDesc

HRESULT ITypeInfo::GetVarDesc(*index*, *lplpvardesc***)**
  unsigned int *index*
  VARDESC FAR* FAR* *lplpvardesc*

Retrieves a VARDESC structure describing the specified variable.

Parameters

index
  Index of the variable whose description is to be returned. The index should be in the range of 0 to 1 less than the number of variables in this type.

lplpvardesc
  Upon return, points to a pointer to a VARDESC that describes the specified variable.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.

Comments

To free the VARDESC structure, use **ReleaseVarDesc**.

Example

```
CHECKRESULT(ptypeinfo->GetVarDesc(i, &pvardesc));
idMember = pvardesc->memid;
CHECKRESULT(ptypeinfo->GetDocumentation(idMember, &bstrName, NULL, NULL,
           NULL));
ptypeinfo->ReleaseVarDesc(pvardesc);
```

ITypeInfo::Invoke

HRESULT ITypeInfo::Invoke(*lpvInstance*, *memid*, *wFlags*, *pdispparams*, *pvargResult*, *pexcepinfo*, *puArgErr***)**
VOID FAR* *lpvInstance*
MEMBERID *memid*
unsigned short *wFlags*
DISPPARAMS FAR* *pdispparams*
VARIANT FAR* *pvargResult*
EXCEPINFO FAR* *pexcepinfo*
unsigned int FAR* *puArgErr*

Invokes a method or accesses a property of an object that implements the interface described by the type description.

Parameters

lpvInstance
Pointer to an instance of the interface described by this type description.

memid
Identifies the interface member.

wFlags
Flags describing the context of the invoke call, as follows:

Value	**Description**
DISPATCH_METHOD	The member was accessed as a method. If there is ambiguity, both this and the DISPATCH_PROPERTYGET flag may be set.
DISPATCH_PROPERTYGET	The member is being retrieved as a property or data member.
DISPATCH_PROPERTYPUT	The member is being changed as a property or data member.
DISPATCH_PROPERTYPUTREF	The member is being changed by using a reference assignment, rather than a value assignment. This value is only valid when the property accepts a reference to an object.

pdispparams
Points to a structure containing an array of arguments, an array of DISPIDs for named arguments, and counts of the number of elements in each array.

pvargResult
Should be NULL if the caller expects no result; otherwise, it should be a pointer to the location at which the result is to be stored. If *wFlags* specifies DISPATCH_PROPERTYPUT or DISPATCH_PROPERTYPUTREF, *pvargResult* is ignored.

pexcepinfo
Points to an exception information structure, which is filled in only if DISP_E_EXCEPTION is returned. If *pexcepinfo* is NULL on input, only an HRESULT error will be returned.

puArgErr
If **Invoke** returns DISP_E_TYPEMISMATCH, *puArgErr* indicates the index (within *rgvarg*) of the argument with incorrect type. If more than one argument has an error, *puArgErr* indicates only the

first argument with an error. Note that arguments in *pdispparams*->*rgvarg* appear in reverse order, so the first argument is the one having the highest index in the array. Can't be NULL.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_INVALIDARG	One or more of the arguments is invalid.
DISP_E_EXCEPTION	The member being invoked has returned an error HRESULT. If the member implements **IErrorInfo**, details are available in the error object. Otherwise, the *pexcepinfo* parameter contains details.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_REGISTRYACCESS	There was an error accessing the system registration database.
TYPE_E_LIBNOTREGISTERED	The type library was not found in the system registration database.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_WRONGTYPEKIND	Type mismatch.
TYPE_E_ELEMENTNOTFOUND	The element was not found.
TYPE_E_BADMODULEKIND	The module does not support **Invoke**.
Other returns	Any of the **IDispatch::Invoke** errors may also be returned.

Comments

Use **ITypeInfo::Invoke** to access a member of an object or invoke a method that implements the interface described by this type description. For objects that support the **IDispatch** interface, **Invoke** can be used to implement **IDispatch::Invoke**.

ITypeInfo::Invoke takes a pointer to an instance of the class. Otherwise, its parameters are the same as **IDispatch::Invoke**, except that **ITypeInfo::Invoke** omits the REFIID and LCID parameters. When called, **ITypeInfo::Invoke** performs the actions described by the **IDispatch::Invoke** parameters on the specified instance.

For VTBL interface members, **ITypeInfo::Invoke** passes the LCID of the typeinfo into parameters tagged with the **lcid** attribute, and the returned value into the **retval** parameter.

If the type description inherits from another type description, this function recurses on the base type description if necessary, to find the item with the requested member ID.

ITypeInfo::ReleaseFuncDesc

VOID ITypeInfo::ReleaseFuncDesc(*lpfuncdesc***)**
    FUNCDESC FAR* *lpfuncdesc*

Releases a FUNCDESC previously returned by **GetFuncDesc**.

Parameter

lpfuncdes
    Pointer to the FUNCDESC to be freed.

Comments

ReleaseFuncDesc releases a FUNCDESC that was returned through **ITypeInfo::GetFuncDesc**.

Example

```
ptypeinfoCur->ReleaseFuncDesc(pfuncdesc);
```

ITypeInfo::ReleaseTypeAttr

VOID ITypeInfo::ReleaseTypeAttr(*lptypeattr***)**
    TYPEATTR FAR* *lptypeattr*

Releases a TYPEATTR previously returned by **GetTypeAttr**.

Parameter

lptypeattr
    Pointer to the TYPEATTR to be freed.

Comments

ReleaseTypeAttr releases a TYPEATTR that was returned through **ITypeInfo::GetTypeAttr**.

ITypeInfo::ReleaseVarDesc

VOID ITypeInfo::ReleaseVarDesc(*lpvardesc***)**
  VARDESC FAR* *lpvardesc*

Releases a VARDESC previously returned by **GetVarDesc**.

Parameter

lpvardesc
  Pointer to the VARDESC to be freed.

Comments

ReleaseVarDesc releases a VARDESC that was returned through **ITypeInfo::GetVarDesc**.

Example

```
VARDESC     FAR *pvardesc;
CHECKRESULT(ptypeinfo->GetVarDesc(i, &pvardesc));
idMember = pvardesc->memid;
CHECKRESULT(ptypeinfo->GetDocumentation(idMember, &bstrName, NULL, NULL,
     NULL));
ptypeinfo->ReleaseVarDesc(pvardesc);
```

ITypeComp Interface

Implemented by	Used by	Header filename
OLEAUT32.DLL (32-bit systems) **TYPELIB.DLL** (16-bit systems)	Tools that compile references to objects contained in type libraries.	**OLEAUTO.H** **DISPATCH.H**

Binding is the process of mapping names to types and type members. The **ITypeComp** interface provides a fast way to access information that compilers need when binding to and instantiating structures and interfaces.

Using Structures and Enumerations with the ITypeComp Interface

The **ITypeComp** interface uses the following structures and enumerations:

BINDPTR

A union containing a pointer to a FUNCDESC, VARDESC, or an **ITypeComp** interface.

```
typedef union tagBINDPTR {
     FUNCDESC FAR* lpfuncdesc;
     VARDESC FAR* lpvardesc;
     ITypeComp FAR* lptcomp;
} BINDPTR;
```

DESCKIND

Identifies the type of the type description being bound to.

```
typedef enum tagDESCKIND {
     DESCKIND_NONE,
     DESCKIND_FUNCDESC,
     DESCKIND_VARDESC,
     DESCKIND_TYPECOMP,
     DESCKIND_IMPLICITAPPOBJ
} DESCKIND;
```

Comments

Value	Description
DESCKIND_NONE	No match was found.
DESCKIND_FUNCDESC	A FUNCDESC was returned.
DESCKIND_VARDESC	A VARDESC was returned.
DESCKIND_TYPECOMP	A TYPECOMP was returned.
DESCKIND_IMPLICITAPPOBJ	An IMPLICITAPPOBJ was returned.

ITypeComp::Bind

HRESULT ITypeComp::Bind(*szName*, *lHashVal*, *wFlags*, *lplptinfo*, *lpdesckind*, *lpbindptr*)
OLECHAR FAR* *szName*
unsigned long *lHashVal*
unsigned short *wFlags*
ITypeInfo FAR* FAR* *lplptinfo*
DESCKIND FAR* *lpdesckind*
BINDPTR FAR* *lpbindptr*

Maps a name to a member of a type, or binds global variables and functions contained in a type library.

Parameters

szName
Name to be bound.

lHashVal
Hash value for the name computed by **LHashValOfNameSys**.

wFlags
Flags word containing one or more of the INVOKE flags defined in the INVOKEKIND enumeration. Specifies whether the name was referenced as a method or as a property. When binding to a variable, specify the INVOKE_PROPERTYGET flag. Specify 0 to bind to any type of member.

lplptinfo
If a FUNCDESC or VARDESC was returned, then *lplptinfo* points to a pointer to the type description that contains the item to which it is bound.

lpdesckind
Pointer to a DESCKIND enumerator that indicates whether the name bound to a VARDESC, FUNCDESC, or TYPECOMP. Points to DESCKIND_NONE if there was no match.

lpbindptr
Upon return, contains a pointer to the bound-to VARDESC, FUNCDESC, or **ITypeComp**.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_AMBIGUOUSNAME	More than one instance of this name occurs in the type library.

Comments

Used for binding to the variables and methods of a type, or for binding to the global variables and methods in a type library. The returned DESCKIND pointer *lpdesckind* indicates whether the name was bound to a VARDESC, a FUNCDESC, or an **ITypeComp** instance. The returned *lpbindptr* points to the

VARDESC, FUNCDESC, or **ITypeComp**.

If a data member or method is bound to, then *lplptinfo* points to the type description that contains the method or data member.

If **Bind** binds the name to a nested binding context, it returns a pointer to an **ITypeComp** instance in lpbindptr and a NULL type description pointer in lplptinfo. For example, if you pass the name of a type description for a module (TKIND_MODULE), enumeration (TKIND_ENUM), or coclass (TKIND_COCLASS) **Bind** returns the **ITypeComp** instance of the type description for the module, enumeration, or coclass. This feature supports languages like Visual Basic that allow references to the members of a type description to be qualified by the name of the type description. For example, a function in a module can be referenced by modulename**.**functionname.

The members of TKIND_ENUM, TKIND_MODULE, and TKIND_COCLASS types marked as Application objects can be directly bound to from **ITypeComp**, without specifying the name of the module. The **ITypeComp** of a coclass defers to the **ITypeComp** of its default interface.

As with other methods of **ITypeComp**, **ITypeInfo**, and **ITypeLib**, the calling code is responsible for releasing the returned object instances or structures. If a VARDESC or FUNCDESC is returned, the caller is responsible for deleting it via the returned type description and releasing the type description instance itself; otherwise, if an **ITypeComp** instance is returned, the caller must release it.

Special rules apply if you call a type library's **Bind** method, passing it the name of a member of an Application object class (that is, a class that has the TYPEFLAG_FAPPOBJECT flag set). In this case, **Bind** returns DESCKIND_IMPLICITAPPOBJ in *lpdesckind,* a VARDESC that describes the Application object in *lpbindptr*, and the **ITypeInfo** of the Application object class in *lplptinfo*. To bind to the object, you must call **ITypeInfo::GetTypeComp** to get the **ITypeComp** of the Application object class, and then reinvoke its **Bind** method with the name initially passed to the type library's **ITypeComp**.

The caller should use the returned **ITypeInfo** pointer (*lplptinfo*) to get the address of the member.

Note that the *wflags* parameter is the same as the *wflags* parameter in **IDispatch::Invoke**.

ITypeComp::BindType

HRESULT ITypeComp::BindType(*szName*, *lHashVal*, *lplpitinfo*, *lplpitcomp*
  OLECHAR FAR* *szName*
  unsigned long *lHashVal*
  ITypeInfo FAR* FAR* *lplptinfo*
  ITypeComp FAR* FAR* *lplptcomp*

Binds to the type descriptions contained within a type library.

Parameters

szName
  Name to be bound.

lHashVal
  Hash value for the name computed by **LHashValOfName**.

lplptinfo
  Upon return, contains a pointer to a pointer to an **ITypeInfo** of the type to which the name was bound.

lplptcomp
  Reserved for future use. Pass NULL.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_AMBIGUOUSNAME	More than one instance of this name occurs in the type library.

Comments

Used for binding a type name to the **ITypeInfo** that describes the type. This function is invoked on the **ITypeComp** returned by **ITypeLib::GetTypeComp** to bind to types defined within that library. It could also be used in the future for binding to nested types.

Type Compilation Functions

The type compilation functions are:

LHashValOfName
LHashValOfNameSys

LHashValOfName

unsigned long LHashValOfName(*lcid*, *szName***)**
LCID *lcid*
OLECHAR FAR* *szName*

Computes a hash value for a name that can then be passed to **ITypeComp::Bind**, **ITypeComp::BindType**, **ITypeLib::FindName**, or **ITypeLib::IsName**.

Parameters

lcid
The locale ID for the string.

szName
String whose hash value is to be computed.

Return Value

A 32-bit hash value representing the name passed in.

Comments

This function is equivalent to **LHashValOfNameSys**. The OLEAUTO.H header file contains macros that define **LHashValOfName** as **LHashValOfNameSys** with the target operating system (*syskind*), based on your build preprocessor flags.

LHashValOfName computes a 32-bit hash value for a name which can then be passed to **ITypeComp::Bind**, **ITypeComp::BindType**, **ITypeLib::FindName**, or **ITypeLib::IsName**. The returned hash value is independent of the case of the characters in *szName* as long as the language of the name is one of the languages supported by the OLE National Language Specification API. Specifically, for any two strings, if those strings match when a case-insensitive comparison is done using any language, then they will produce the same hash value.

LHashValOfNameSys

unsigned long LHashValOfName(*syskind*, *lcid*, *szName***)**
  SYSKIND *syskind*
  LCID *lcid*
  OLECHAR FAR* *szName*

Computes a hash value for a name that can then be passed to **ITypeComp::Bind**, **ITypeComp::BindType**, **ITypeLib::FindName**, or **ITypeLib::IsName**.

Parameters

syskind
  The SYSKIND of the target operating system.

lcid
  The locale ID for the string.

szName
  String whose hash value is to be computed.

Return Value

A 32-bit hash value representing the name passed in.

Type Library Loading and Registration Functions

The type library loading and registration functions are:

LoadTypeLib
LoadRegTypeLib
RegisterTypeLib
QueryPathOfRegTypeLib

LoadTypeLib

HRESULT LoadTypeLib(*szFileName*, *lplptlib***)**
  OLECHAR FAR* *szFileName*
  ITypeLib FAR* FAR* *lplptlib*

Loads and registers a type library.

Parameters

szFileName
  Contains the name of the file from which **LoadTypeLib** should attempt to load a type library.

lplptlib
  On return, contains a pointer to a pointer to the loaded type library.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not write to the file.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_UNKNOWNLCID	The LCID could not be found in the OLE support DLLs.
TYPE_E_CANTLOADLIBRARY	The type library or DLL could not be loaded.
Other returns	All FACILITY_STORAGE errors may be returned.

Comments

LoadTypeLib loads and registers a type library (usually created with MkTypLib) that is stored in the specified file. If *szFileName* specifies only a filename, with no path, **LoadTypeLib** searches for the file and proceeds as follows:

- If the file is a stand-alone type library implemented by TYPELIB.DLL, the library is loaded directly.
- If the file is a DLL or .EXE, the file is loaded. By default, the type library is extracted from the first resource of type ITypeLib. To load a different type library resource, append an integer index to *szFileName*. For example:

  ```
  LoadTypeLib("C:\MONTANA\EXE\MFA.EXE\3", lplptlib)
  ```

  This statement loads the type library resource 3 from the file MFA.EXE.
- If the file is none of the above, the filename is parsed into a moniker (an object that represents a file-based link source), then bound to the moniker. This approach allows **LoadTypeLib** to be used on foreign type libraries, including in-memory type libraries. Foreign type libraries can't reside in a DLL or .EXE file. See *OLE 2 Programmer's Reference, Vol. 1* for more information on monikers.

If the type library is already loaded, **LoadTypeLib** increments the type library's reference count and returns a pointer to the type library.

LoadRegTypeLib

HRESULT LoadRegTypeLib(*guid*, *wVerMajor*, *wVerMinor*, *lcid*, *lplptlib***)**
REFGUID *guid*
unsigned short *wVerMajor*
unsigned short *wVerMinor*
LCID *lcid*
ITypeLib FAR* FAR* *lplptlib*

Uses registry information to load a type library.

Parameters

guid
ID of the library being loaded.

wVerMajor
Major version number of library being loaded.

wVerMinor
Minor version number of library being loaded.

lcid
National language code of library being loaded.

lplptlib
On return, points to a pointer to the loaded type library.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not read from the file.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an old format.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_UNKNOWNLCID	The passed in LCID could not be found in the OLE support DLLs.
TYPE_E_CANTLOADLIBRARY	The type library or DLL could not be loaded.
Other returns	All FACILITY_STORAGE and system registry errors may also be returned.

Comments

LoadRegTypeLib defers to **LoadTypeLib** to load the file.

LoadRegTypeLib compares the requested version numbers against those found in the system registry and takes one of the following actions:

- If one of the registered libraries exactly matches both the requested major and minor version numbers, then that type library is loaded.
- If one or more registered type libraries exactly match the requested major version number and have a greater minor version number than that requested, the one with the greatest minor version number is loaded.
- If none of the registered type libraries exactly match the requested major version number or if none of those which do exactly match the major version number also have a minor version number greater than or equal to the requested minor version number, then **LoadRegTypeLib** returns an error.

RegisterTypeLib

HRESULT RegisterTypeLib(*ptlib*, *szFullPath*, *szHelpDir***)**
  ITypeLib FAR* *ptlib*
  OLECHAR FAR* *szFullPath*
  OLECHAR FAR* *szHelpDir*

Adds information about a type library to the system registry.

Parameters

ptlib
  Pointer to the type library being registered.

szFullPath
  Fully qualified path specification for the type library being registered.

szHelpDir
  Directory in which the Help file for the library being registered can be found. May be NULL.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not write to the file.
TYPE_E_REGISTRYACCESS	The system registration database could not be opened.
TYPE_E_INVALIDSTATE	The type library could not be opened.

Comments

RegisterTypeLib can be used during application initialization to correctly register the application's type library.

In addition to filling in a complete registry entry under the **TypeLib** key, **RegisterTypeLib** adds entries for each of the dispinterfaces and OLE Automation-compatible interfaces, including dual interfaces. This information is required in order to create instances of these interfaces.

QueryPathOfRegTypeLib

HRESULT QueryPathOfRegTypeLib(*guid*, *wVerMajor*, *wVerMinor*, *lcid*, *lpBstrPathName***)**
REFGUID *guid*
unsigned short *wVerMajor*
unsigned short *wVerMinor*
LCID *lcid*
LPBSTR *lpBstrPathName*

Retrieves the path of a registered type library.

Parameters

guid
ID of the library whose path is to be queried.

wVerMajor
Major version number of the library whose path is to be queried.

wVerMinor
Minor version number of the library whose path is to be queried.

lcid
National language code for the library whose path is to be queried.

lpBstrPathName
Caller-allocated BSTR in which the type library name is returned.

Return Value

The SCODE obtained from the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success

Comments

Returns the fully qualified filename specified for the type library in the registry. The caller allocates the BSTR that is passed in, and must free it after use.

Type Building Interfaces

The type building interfaces, **ICreateTypeInfo** and **ICreateTypeLib**, are used to build tools that automate the process of generating type descriptions and creating type libraries. The MkTypLib tool, for example, uses these interfaces to create type libraries. See Chapter 7, "MkTypLib and Object Description Language," for more information on MkTypLib.

Generally, you do not need to write custom implementations of these interfaces; MkTypLib itself uses the default implementations, which are returned by the **CreateTypeLib** function. If you want to create tools similar to MkTypLib, calling the default implementations should suffice.

Implemented by	**Used by**	**Header filename**	**Import library name**
OLEAUT32.DLL (32-bit systems) TYPELIB.DLL (16-bit systems)	Applications that expose programmable objects.	OLEAUTO.H DISPATCH.H	OLEAUT32.LIB TYPELIB.LIB

ICreateTypeInfo Interface

The type building interfaces include the following member functions:

Interface	Member name	Purpose
ICreateTypeInfo	**AddFuncDesc**	Adds a function description as a type description.
	AddImplType	Specifies an inherited interface.
	AddRefTypeInfo	Adds a type description to those referenced by the type description being created.
	AddVarDesc	Adds a data member description as a type description.
	DefineFuncAsDllEntry	Associates a DLL entry point with a function that has a specified index.
	LayOut	Assigns VTBL offsets for virtual functions and instance offsets for per-instance data members.
	SetAlignment	Specifies data alignment for types of TKIND_RECORD.
	SetDocString	Sets the documentation string displayed by type browsers.
	SetFuncAndParamNames	Sets the function name and names of its parameters.
	SetFuncDocString	Sets the documentation string for a function.
	SetFuncHelpContext	Sets the Help context for a function.
	SetGuid	Sets the globally unique ID for the type library.
	SetHelpContext	Sets the Help context of the type description.
	SetImplTypeFlags	Sets the attributes for an implemented or inherited interface of a type.
	SetMops	Sets the opcode string for a type description.
	SetSchema	Reserved for future use.
	SetTypeDescAlias	Sets the type description for which this type description is an alias, if TYPEKIND=TKIND_ALIAS.
	SetTypeFlags	Sets type flags of the type description being created.
	SetTypeIdlDesc	Reserved for future use.
	SetVarDocString	Sets the documentation string for a variable.
	SetVarHelpContext	Sets the Help context for a variable.
	SetVarName	Sets the name of a variable.
	SetVersion	Sets version numbers for the type description.
ICreateTypeLib	**CreateTypeInfo**	Creates a new type description

		instance within the type library.
	SaveAllChanges	Saves the **ICreateTypeLib** instance.
	SetDocString	Sets the documentation string for the type library.
	SetHelpContext	Sets the Help context for general information about the type library in the Help file.
	SetHelpFileName	Sets the Help filename.
	SetLcid	Sets the locale code indicating the national language associated with the library.
	SetLibFlags	Sets library flags, such as LIBFLAG_FRESTRICTED.
	SetName	Sets the name of the type library.
	SetGuid	Sets the globally unique ID for the type library.
	SetVersion	Sets major and minor version numbers for the type library.
Library creation functions	**CreateTypeLib**	Gives access to a new object instance that supports the **ICreateTypeLib** interface.

Enumerations and Structures

The type building interfaces use the following structures and enumerations.

Using LIBFLAGS with the ICreateTypeInfo Interface

The LIBFLAGS enumeration defines flags that apply to type libraries. LIBFLAGS is defined as follows:

```
typedef enum tagLIBFLAGS {
     LIBFLAG_FRESTRICTED = 0x01
     , LIBFLAG_FCONTROL = 0x02
     , LIBFLAG_FHIDDEN = 0x04
} LIBFLAGS;
```

Value	Description
LIBFLAG_FCONTROL	The type library describes controls and should not be displayed in type browsers intended for nonvisual objects.
LIBFLAG_FRESTRICTED	The type library is restricted and should not be displayed to users.
LIBFLAG_FHIDDEN	The type library should not be displayed to users, although its use is not restricted. To be used by controls; hosts should create a new type library that wraps the control with extended properties.

Using SYSKIND with the ICreateTypeInfo Interface

The SYSKIND identifies the target operating system platform.

```
typedef enum tagSYSKIND[
     SYS_WIN16,
     SYS_WIN32,
     SYS_MAC
] SYSKIND;
```

Value	Description
SYS_WIN16	The target operating system for the type library is 16-bit Windows systems. By default, data members are packed.
SYS_WIN32	The target operating system for the type library is 32-bit Windows systems. By default, data members are naturally aligned (for example, 2-byte integers are aligned on even-byte boundaries; 4-byte integers are aligned on quad-word boundaries, and so forth).
SYS_MAC	The target operating system for the type library is Macintosh. By default, all data members are aligned on even-byte boundaries.

ICreateTypeInfo::AddFuncDesc

HRESULT ICreateTypeInfo::AddFuncDesc(*index*, *lpFuncDesc***)**
  unsigned int *index*
  FUNCDESC FAR* *lpFuncDesc*

Parameters

index

Index of the new FUNCDESC in the type information.

lpFuncDesc

Pointer to a FUNCDESC structure that describes the function. The *bstrIDLInfo* field in the FUNCDESC should be set to NULL for future compatibility.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
E_ACCESSDENIED	Can't write to destination.
TYPE_E_WRONGTYPEKIND	Type mismatch.

Comments

AddFuncDesc is used to add a function description to the type description. The index specifies the order of the functions within the type information. The first function has an index of zero. If an index is specified that exceeds one less than the number of functions in the type information, then an error is returned. Calling this function does not pass ownership of the FUNCDESC structure to **ICreateTypeInfo**. Therefore, the caller must still deallocate the FUNCDESC structure.

The passed-in VTBL field (*oVft*) of the FUNCDESC is ignored. This attribute is set when **ICreateTypeInfo::LayOut** is called.

AddFuncDesc uses the passed-in member ID fields within each FUNCDESC for classes with TYPEKIND = TKIND_DISPATCH or TKIND_INTERFACE. If the member IDs are set to MEMID_NIL, **AddFuncDesc** assigns member IDs to the functions. Otherwise, the member ID fields within each FUNCDESC are ignored.

Note that any HREFTYPE fields in the FUNCDESC structure must have been produced by the same instance of **ITypeInfo** for which **AddFuncDesc** is called.

The get and put accessor functions for the same property must have the same DISPID.

ICreateTypeInfo::AddImplType

HRESULT ICreateTypeInfo::AddImplType(*index, hreftype***)**
  unsigned int *index*
  HREFTYPE *hreftype*

Parameters

index
  Index of the implementation class to be added; specifies the order of the type relative to the other type.

hreftype
  Handle to the referenced type description obtained from **AddRefType** description.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_ACCESSDENIED	Can't write to destination.
TYPE_E_WRONGTYPEKIND	Type mismatch.

Comments

AddImplType is used only for specifying an inherited interface or an interface implemented by a component object class.

ICreateTypeInfo::AddRefTypeInfo

HRESULT ICreateTypeInfo::AddRefTypeInfo(*lptinfo*, *lphreftype***)**
ITypeInfo FAR* *lptinfo*
HREFTYPE FAR* *lphreftype*

Parameters

lptinfo
Pointer to the type description to be referenced.

lphreftype
On return, pointer to the handle that this type description associates with the referenced type information.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
E_ACCESSDENIED	Can't write to destination.
TYPE_E_WRONGTYPEKIND	Type mismatch.

Comments

Adds a type description to those referenced by the type description being created. The second parameter returns a pointer to the handle of the added type information. If **AddRefTypeInfo** has previously been called for the same type information, the index that was returned by the previous call is returned in *lphreftype*.

ICreateTypeInfo::AddVarDesc

HRESULT ICreateTypeInfo::AddVarDesc(*index*, *lpVarDesc***)**
unsigned int *index*
VARDESC FAR* *lpVarDesc*

Parameters

index
Index of the variable or data member to be added to the type description.

lpVarDesc
Pointer to the variable or data member description to be added.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
E_ACCESSDENIED	Can't write to destination.
TYPE_E_WRONGTYPEKIND	Type mismatch.

Comments

Adds a variable or data member description to the type description. The index specifies the order of the variables. The first variable has an index of zero. **ICreateTypeInfo::AddVarDesc** returns an error if the specified index is greater than the number of variables currently in the type information. Calling this function does not pass ownership of the VARDESC structure to **ICreateTypeInfo**. Note that the Instance field (*oInst*) of the VARDESC structure is ignored; this attribute is set when **ICreateTypeInfo::LayOut** is called. Also, the member ID fields within the VARDESCs are ignored unless the TYPEKIND of the class is TKIND_DISPATCH.

Any HREFTYPE fields in the VARDESC structure must have been produced by the same instance of **ITypeInfo** for which **AddVarDesc** is called.

AddVarDesc ignores the contents of the *idldesc* field of the ELEMDESC.

ICreateTypeInfo::DefineFuncAsDllEntry

HRESULT ICreateTypeInfo::DefineFuncAsDllEntry(*index*, *szDllName*, *szProcName***)**
  unsigned int *index*
  OLECHAR FAR* *szDllName*
  OLECHAR FAR* *szProcName*

Parameters

index
  Index of the function.

szDllName
  Name of the DLL containing the entry point.

szProcName
  Name of the entry point or an ordinal (if the high word is zero).

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_ELEMENTNOTFOUND	The element can't be found.
TYPE_E_WRONGTYPEKIND	Type mismatch.

Comments

Associates a DLL entry point with the function that has the specified index. If the high word of *szProcName* is zero, then the low word must contain the ordinal of the entry point; otherwise, *szProcName* points to the zero-terminated name of the entry point.

ICreateTypeInfo::LayOut

HRESULT ICreateTypeInfo::LayOut()

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_ACCESSDENIED	Can't write to destination.
TYPE_E_UNDEFINEDTYPE	Bound to unrecognized type.
TYPE_E_INVALIDSTATE	The type library's state is not valid for this operation.
TYPE_E_WRONGTYPEKIND	Type mismatch.
TYPE_E_ELEMENTNOTFOUND	The element can't be found.
TYPE_E_AMBIGUOUSNAME	More than one item exists with this name.
TYPE_E_SIZETOOBIG	The type information is too large to lay out.
TYPE_E_TYPEMISMATCH	Type mismatch.

Comments

LayOut assigns VTBL offsets for virtual functions and instance offsets for per-instance data members, and creates the two typeinfos for dual interfaces. **LayOut** also assigns member ID numbers to the functions and variables unless the TYPEKIND of the class is TKIND_DISPATCH. **LayOut** should be called after all members of the type information are defined and before the type library is saved.

Use **SaveAllChanges** to save the type information after calling **LayOut**. Don't call other members of the **ICreateTypeInfo** interface after calling **LayOut**.

Note that different implementations of **ICreateTypeInfo** or other interfaces that create type information are free to assign any member ID numbers, provided that all members, including inherited members, have unique IDs.

ICreateTypeInfo::SetAlignment

HRESULT ICreateTypeInfo::SetAlignment(*cbAlignment***)**
unsigned short *cbAlignment*

Parameters

cbAlignment
Alignment method for the type. A value of 0 indicates alignment on the 64K boundary; 1 indicates no special alignment. For other values, *n* indicates alignment on byte *n*.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_ACCESSDENIED	Can't write to destination.
TYPE_E_INVALIDSTATE	The type library's state is not valid for this operation.

Comments

Specifies the data alignment for an item of TYPEKIND=TKIND_RECORD. The alignment is the minimum of the natural alignment (for example, byte data on byte boundaries, word data on word boundaries, and so on) and the alignment denoted by *cbAlignment*.

ICreateTypeInfo::SetDocString

HRESULT ICreateTypeInfo::SetDocString(*szDoc***)**
OLECHAR FAR* *szDoc*

Parameters

szDoc
Pointer to the documentation string.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_ACCESSDENIED	Can't write to destination.
TYPE_E_INVALIDSTATE	The type library's state is not valid for this operation.

Comments

Sets the documentation string displayed by type browsers. The documentation string is a brief description of the type description being created.

ICreateTypeInfo::SetFuncAndParamNames

HRESULT ICreateTypeInfo::SetFuncAndParamNames(*index*, *rgszNames*, *cNames***)**
unsigned int *index*
OLECHAR FAR* FAR* *rgszNames*
unsigned int *cNames*

Parameters

index
Index of the function whose function name and parameter names are to be set.

rgszNames
Array of pointers to names. The first element is the function name; subsequent elements are names of parameters.

cNames
Number of elements in the *rgszNames* array.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
E_ACCESSDENIED	Can't write to destination.
TYPE_E_ELEMENTNOTFOUND	The element can't be found.

Comments

Sets the name of a function and the names of its parameters to the names in the array of pointers *rgszNames*. You only need to use **SetFuncAndParamNames** once for each property, because all property accessor functions are identified by one name. For property functions, provide names for the named parameters only; the last parameter for put and putref accessor functions is unnamed.

ICreateTypeInfo::SetFuncDocString

HRESULT ICreateTypeInfo::SetFuncDocString(*index*, *szDocString***)**
unsigned int *index*
OLECHAR FAR* *szDocString*

Parameters

index
Index of the function.

szDocString
Pointer to the documentation string.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
E_ACCESSDENIED	Can't write to destination.
TYPE_E_ELEMENTNOTFOUND	The element can't be found.

Comments

Sets the documentation string for the function with the specified *index*. The documentation string is a brief description of the function intended for use by tools like type browsers. You only need to use **SetFuncDocString** once for each property, because all property accessor functions are identified by one name.

ICreateTypeInfo::SetFuncHelpContext

HRESULT ICreateTypeInfo::SetFuncHelpContext(*index, dwHelpContext***)**
  unsigned int *index*
  unsigned long *dwHelpContext*

Parameters

index
  Index of the function.

dwHelpContext
  A Help context ID for the Help topic.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_ACCESSDENIED	Can't write to destination.
TYPE_E_ELEMENTNOTFOUND	The element can't be found.
E_INVALIDARG	One or more of the arguments is invalid.

Comments

Sets the Help context for the function with the specified index. You only need to use **SetFuncHelpContext** once for each property, because all property accessor functions are identified by one name.

ICreateTypeInfo::SetGuid

HRESULT ICreateTypeInfo::SetGuid(*guid***)**
  REFGUID *guid*

Parameters

guid
  Globally unique ID to be associated with the type description.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_ACCESSDENIED	Can't write to destination.

Comments

Sets the globally unique ID (GUID) associated with the type description. For an interface, this is an interface ID; for a coclass, it is a class ID. See Chapter 7, "MkTypLib and Object Description Language," for information on GUIDs.

ICreateTypeInfo::SetHelpContext

HRESULT ICreateTypeInfo::SetHelpContext(*dwHelpContext***)**
unsigned long *dwHelpContext*

Parameters

dwHelpContext
Handle to the Help context.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
E_ACCESSDENIED	Can't write to destination.

Comments

Sets the Help context of the type information.

ICreateTypeInfo::SetImplTypeFlags

HRESULT ICreateTypeInfo::SetImplTypeFlags(*index, impltypeflags***)**
 unsigned int *index*
 int *impltypeflags*

Parameters

index
 Index of the interface for which to set type flags.

impltypeflags
 IMPLTYPE flags to set.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
E_ACCESSDENIED	Can't write to destination.

Comments

Sets the IMPLTYPE flags for the indexed interface. See the "IMPLTYPEFLAGS" section in Chapter 8, "Type Description Interfaces."

ICreateTypeInfo::SetMops

HRESULT ICreateTypeInfo::SetMops(*index, bstrMops***)**
unsigned int *index*
BSTR *bstrMops*

Parameters

index

Index of the member for which to set the opcode string. If index is -1, sets the opcode string for the type description.

bstrMops

The marshaling opcode string.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
E_ACCESSDENIED	Can't write to destination.

Comments

Sets the marshaling opcode string associated with the type description or the function.

ICreateTypeInfo::SetTypeDescAlias

HRESULT ICreateTypeInfo::SetTypeDescAlias(*lptDescAlias***)**
TYPEDESC FAR* *lptDescAlias*

Parameters

lptDescAlias
Pointer to a type description that describes the type for which this is an alias.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
E_ACCESSDENIED	Can't write to destination.
TYPE_E_WRONGTYPEKIND	Type mismatch.

Comments

Call **SetTypeDescAlias** for a type description whose TYPEKIND is TKIND_ALIAS to set the type for which it is an alias.

ICreateTypeInfo::SetTypeFlags

HRESULT ICreateTypeInfo::SetTypeFlags(*uTypeFlags***)**
unsigned int *uTypeFlags*

Parameter

uTypeFlags
Settings for the type flags.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
E_ACCESSDENIED	Can't write to destination.
TYPE_E_WRONGTYPEKIND	Type mismatch.

Comments

Use **SetTypeFlags** to set the flags for the type description. See the "TYPEFLAGS" section in Chapter 8, "Type Description Interfaces," for details.

ICreateTypeInfo::SetVarDocString

HRESULT ICreateTypeInfo::SetVarDocString(*index*, *szDocString***)**
unsigned int *index*
OLECHAR FAR* *szDocString*

Parameters

index
Index of the variable being documented.

szDocString
The documentation string to be set.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_ACCESSDENIED	Can't write to destination.
TYPE_E_ELEMENTNOTFOUND	The element was not found.

Comments

Sets the documentation string for the variable with the specified *index*.

ICreateTypeInfo::SetVarHelpContext

HRESULT ICreateTypeInfo::SetVarHelpContext(*index*, *dwHelpContext***)**
unsigned int *index*
unsigned long *dwHelpContext*

Parameters

index
Index of the variable described by the type description.

dwHelpContext
Handle to the Help context for the Help topic on the variable.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_ACCESSDENIED	Can't write to destination.
TYPE_E_ELEMENTNOTFOUND	The element can't be found.

Comments

Sets the Help context for the variable with the specified *index*.

ICreateTypeInfo::SetVarName

HRESULT ICreateTypeInfo::SetVarName(*index*, *szName***)**
unsigned int *index*
OLECHAR FAR* *szName*

Parameters

index
Index of the variable whose name is being set.

szName
Name for the variable.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_ACCESSDENIED	Can't write to destination.
TYPE_E_ELEMENTNOTFOUND	The element can't be found.

Comments

Sets the name of a variable.

ICreateTypeInfo::SetVersion

HRESULT ICreateTypeInfo::SetVersion(*wMajorVerNum*, *wMinorVerNum***)**
  unsigned short *wMajorVerNum*
  unsigned short *wMinorVerNum*

Parameters

wMajorVerNum
  Major version number for the type.

wMinorVerNum
  Minor version number for the type.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
E_ACCESSDENIED	Can't write to destination.
TYPE_E_INVALIDSTATE	The type library's state is not valid for this operation.

Comments

Sets the major and minor version number of the type information.

ICreateTypeLib Interface

The ICreateTypeLib interface methods are:

ICreateTypeLib::CreateTypeInfo
ICreateTypeLib::SaveAllChanges
ICreateTypeLib::SetDocString
ICreateTypeLib::SetGuid
ICreateTypeLib::SetHelpContext
ICreateTypeLib::SetHelpFileName
ICreateTypeLib::SetLibFlags
ICreateTypeLib::SetLcid
ICreateTypeLib::SetName
ICreateTypeLib::SetVersion
CreateTypeLib

ICreateTypeLib::CreateTypeInfo

HRESULT ICreateTypeLib::CreateTypeInfo(*szName*, *tkind*, *lplpctinfo***)**
OLECHAR FAR* *szName*
TYPEKIND *tkind*
ICreateTypeInfo FAR* **FAR*** *lplpctinfo*

Parameters

szName
Name of the new type.

tkind
TYPEKIND of the type description to be created.

lplpctinfo
On return, contains a pointer to the type description.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_INVALIDSTATE	The type library's state is not valid for this operation.
TYPE_E_NAMECONFLICT	The provided name is not unique.
TYPE_E_WRONGTYPEKIND	Type mismatch.

Comments

Use **CreateTypeInfo** to create a new type description instance within the library. An error is returned if the specified name already appears in the library. Valid *tkind* values are described in the "TYPEKIND" section in Chapter 8, "Type Description Interfaces."

ICreateTypeLib::SaveAllChanges

HRESULT ICreateTypeLib::SaveAllChanges()

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function can't write to the file.
TYPE_E_INVALIDSTATE	The type library's state is not valid for this operation.
Other returns	All FACILITY_STORAGE errors.

Comments

Saves the **ICreateTypeLib** instance following the layout of the type information. Do not call any other **ICreateTypeLib** methods after calling **SaveAllChanges**.

ICreateTypeLib::SetDocString

HRESULT ICreateTypeLib::SetDocString(*szDoc***)**
OLECHAR FAR* *szDoc*

Parameters

szDoc
A documentation string that briefly describes the type library.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.

Comments

Sets the documentation string associated with the library. The documentation string is a brief description of the library intended for use by type information browsing tools.

ICreateTypeLib::SetGuid

HRESULT ICreateTypeLib::SetGuid(*guid***)**
REFGUID *guid*

Parameters

guid
The universal unique ID to be assigned to the library.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_INVALIDSTATE	The type library's state is not valid for this operation.

Comments

Sets the universal unique ID (UUID) associated with the type library. UUIDs are described in Chapter 7, "MkTypLib and Object Description Language."

ICreateTypeLib::SetHelpContext

HRESULT ICreateTypeLib::SetHelpContext(*dwHelpContext***)**
unsigned long *dwHelpContext*

Parameters

dwHelpContext
Help context to be assigned to the library.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_INVALIDSTATE	The type library's state is not valid for this operation.

Comments

Sets the Help context for retrieving general Help information for the type library. Note that calling **SetHelpContext** with a Help context of zero is equivalent to not calling it at all, because zero indicates a NULL Help context.

ICreateTypeLib::SetHelpFileName

HRESULT ICreateTypeLib::SetHelpFileName(*szFileName***)**
OLECHAR FAR* *szFileName*

Parameters

szFileName
The name of the Help file for the library.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_INVALIDSTATE	The type library's state is not valid for this operation.

Comments

Sets the name of the Help file. Each type library can reference a single Help file.

The **GetDocumentation** method of the created **ITypeLib** returns a fully qualified path for the Help file, which is formed by appending the name passed into *szFileName* to the registered Help directory for the type library. The Help directory is registered under:

\TYPELIB\<*guid of library*>\<*Major.Minor version* >\HELPDIR

ICreateTypeLib::SetLibFlags

HRESULT ICreateTypeLib::SetLibFlags(*uLibFlags***)**
unsigned int *uLibFlags*

Parameters

uLibFlags
The flags to set for the library.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_INVALIDSTATE	The type library's state is not valid for this operation.

Comments

Valid *uLibFlags* values are listed in "Using LIBFLAGS with the ICreateTypeInfo Interface," earlier in this chapter.

ICreateTypeLib::SetLcid

HRESULT ICreateTypeLib::SetLcid(*lcid***)**
LCID *lcid*

Parameters

lcid
An LCID representing the locale ID for the type library.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_INVALIDSTATE	The type library's state is not valid for this operation.

Comments

Sets the binary Microsoft national language ID associated with the library. For more information on national language IDs, see "Supporting Multiple National Languages," in Chapter 2, "Exposing OLE Automation Objects." For additional information for 16-bit systems, refer to Appendix A, "National Language Support Functions." For 32-bit systems, refer to the Windows NT documentation on the NLS API.

ICreateTypeLib::SetName

HRESULT ICreateTypeLib::SetName(*szName***)**
OLECHAR FAR* *szName*

Parameters

szName
Name to be assigned to the library.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_INVALIDSTATE	The type library's state is not valid for this operation.

Comments

Sets the name of the type library.

ICreateTypeLib::SetVersion

HRESULT ICreateTypeLib::SetVersion(*wMajorVerNum*, *wMinorVerNum***)**
unsigned short *wMajorVerNum*
unsigned short *wMinorVerNum*

Parameters

wMajorVerNum
Major version number for the library.

wMinorVerNum
Minor version number for the library.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	**Meaning**
S_OK	Success.
TYPE_E_INVALIDSTATE	The type library's state is not valid for this operation.

Comments

Sets the major and minor version numbers of the type library.

CreateTypeLib

HRESULT CreateTypeLib(*syskind*, *szFile*, *lplpctlib***)**
  SYSKIND *syskind*
  OLECHAR FAR* *szFile*
  ICreateTypeLib FAR* FAR* *lplpctlib*

Parameters

syskind
  The target operating system for which to create a type library.

szFile
  The name of the file to create.

lplpctlib
  Pointer to an instance supporting the **ICreateTypeLib** interface.

Return Value

The SCODE value of the returned HRESULT is one of the following:

SCODE	Meaning
S_OK	Success.
STG_E_INSUFFICIENTMEMORY	Out of memory.
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not create the file.
Other returns	All FACILITY_STORAGE errors.

Comments

CreateTypeLib sets its output parameter (*lplpctlib*) to point to a newly created object that supports the **ICreateTypeLib** interface.

Glossary

A

accessor function
A function that sets or retrieves the value of a property. Most properties have a pair of accessor functions; properties that are read-only may have only one accessor function.

Application object
The top-level object in an application's object hierarchy. The Application object identifies the application to the system and typically becomes active when the application starts. Specified by the **appobj** attribute in the type library.

C

class identifier (CLSID)
A UUID that identifies an OLE object. An object registers its CLSID in the system registration database so that it can be loaded and programmed by other applications.

class factory
An object that implements the **IClassFactory** interface, which allows it to create other objects of a specific class.

coclass
Component object model class; a top-level object in the object hierarchy.

collection object
A grouping of exposed objects. A collection object enables you to address multiple occurrences of an object as a unit, for example, to draw a set of points.

component object
A Windows object identified by a unique class identifier (CLSID), which associates the object with a DLL or an .EXE file.

compound document
A document that contains data of different formats, such as sound clips, spreadsheets, text, and bitmaps, created by different applications. Compound documents are stored by container applications.

container application
An application that provides storage, a display site, and access to a compound document object.

D

Dispatch identifier (DISPID)
The number by which a member function, parameter, or data member of an object is known internally to **IDispatch**.

dispinterface
An **IDispatch** interface that responds only to a certain fixed set of names. The properties and methods of the dispinterface are not in the virtual function table (VTBL) for the object.

dual interface
An interface that supports both **IDispatch** and VTBL binding.

E

event
An action recognized by an object, such as clicking the mouse or pressing a key, and for which you can write code to respond. In OLE Automation, an event is a method that is called, rather than implemented, by an OLE Automation object.

event sink
A function that handles events. The code associated with a Visual Basic form, which contains event handlers for one or more controls, is an event sink.

event source
A control that experiences events and calls an event handler to dispose of them.

exposed object
See OLE Automation object.

H

HRESULT
A value returned from a function call to an interface, consisting of a severity code, context information, a facility code, and a status code that describes the result. For 16-bit Windows systems, the HRESULT is an opaque result handle defined to be zero for a successful return from a function, and nonzero if error or status information is to be returned. To convert an HRESULT into the more detailed SCODE, applications call **GetSCode()**. *See SCODE.*

I

ID binding
The ability to bind member names to DISPIDs at compile time, for example, by obtaining the IDs from a type library. This approach eliminates the need for calls to **IDispatch::GetIDsOfNames** and results in improved performance over late-bound calls. *See also late binding.*

in-place activation
The ability to activate an object from within an OLE control and to associate a verb with that activation (for example, edit, play, change). Sometimes referred to as in-place editing or visual editing.

in-process server
An object application that runs in the same process space as the OLE Automation controller.

interface
One or more well-defined base classes providing member functions that, when implemented in an application, provide a specific service. Interfaces may include compiled support functions to simplify their implementation.

L

late binding

The ability to bind names to IDs at run time, rather than compile time.

local server
An object application implemented in an .EXE file that runs in a separate process space from the OLE Automation controller.

locale identifier (LCID)
A 32-bit value that identifies the human language preferred by a user, region, or application.

M

marshaling
The process of packaging and sending interface parameters across process boundaries.

member function
One of a group of related functions that make up an interface. *See also method and property.*

method
Member functions of an exposed object that perform some action on the object, such as saving it to disk.

multiple-document interface (MDI) application
An application that can support multiple documents from one application instance. MDI object applications can simultaneously service a user and one or more embedding containers. *See also single-document interface (SDI) application.*

O

object
A unit of information that resides in a compound document and whose behavior is constant no matter where it is located or used.

Object Description Language (ODL)
A scripting language used to describe exposed libraries, objects, types, and interfaces. ODL scripts are compiled into type libraries by the MkTypLib tool.

OLE
An acronym for Object Linking and Embedding.

OLE Automation
A technology that provides a way to manipulate objects defined by an application or library from outside the application. OLE Automation enables programmability.

OLE Automation controller
An application, programming tool, or scripting language that accesses OLE Automation objects. Visual Basic is an OLE Automation controller.

OLE Automation object
An instance of a class defined within an application that is exposed for access by other applications or programming tools by means of OLE Automation interfaces.

OLE Automation server
An application, type library, or other source that makes OLE Automation objects available for programming by other applications, programming tools, or scripting languages.

P

programmable object

See OLE Automation object. property
A data member of an exposed object. Properties are set or returned by means of get and put accessor functions.

proxy
An interface-specific object that packages parameters for that interface in preparation for a remote method call. A proxy runs in the address space of the sender and communicates with a corresponding stub in the receiver's address space. *See also stub, marshaling, and unmarshaling.*

S

safe array
An array that contains information about the number of dimensions and the bounds of its dimensions. Safe arrays are passed by **IDispatch::Invoke** within VARIANTARGs. Their base type is VT_tag | VT_ARRAY.

SCODE
A DWORD value that is used to pass detailed information to the caller of an interface member or API function. The status codes for OLE interfaces and API's are defined in FACILITY_ITF. *See HRESULT.*

single-document interface (SDI) application
An application that can support only one document at a time. Multiple instances of an SDI application must be started to service both an embedded object and a user. *See also multiple-document interface (MDI) application*.

stub
An interface-specific object that unpackages the parameters for that interface after they are marshaled across the process boundary, and makes the requested method call. The stub runs in the address space of the receiver and communicates with a corresponding proxy in the sender's address space. *See proxy, marshaling, and unmarshaling.*

type description
The information used to build the type information for one or more aspects of an application's interface. Type descriptions are written in Object Description Language (ODL) and include both programmable and nonprogrammable interfaces.

type information
Information that describes the interfaces of an application. Type information is created from type descriptions using OLE Automation tools, such as MkTypLib or the **CreateDispTypeInfo** function. Type information may be accessed through the **ITypeInfo** interface.

type information element
A unit of information identified by one of these statements in a type description: typedef, enum,

struct, module, interface, dispinterface, or coclass.

type library
A file or component within another file that contains type information. Type libraries are created from type descriptions using MkTypLib, and may be accessed through the **ITypeLib** interface.

U

unmarshaling
The process of unpackaging parameter that have been sent across process boundaries.

V

Value property
The property that defines the default behavior of an object when no other methods or properties are specified. You indicate the Value property by specifying the **default** attribute in ODL.

virtual function table (VTBL)
A table of function pointers, such as an implementation of a class in C++. The pointers in the VTBL point to the members of the interfaces that an object supports.

Introduction

This book provides procedural and reference information for OLE Automation. While OLE Automation runs on other platforms, such as the Apple® Macintosh® system, the focus of this document is applications that use the Windows® 32-bit operating system.

To get the most out of this book, you should be familiar with:

- The C++ programming language.
- The Microsoft® Windows programming environment, version 3.1 or later. The OLE protocols are implemented through dynamic-link libraries (DLLs) that are used in conjunction with other Microsoft Windows programs.
- The OLE 2 Component Object Model. This model is the foundation of OLE. It is explained in depth by the books listed in "Other Books and Technical Support."

About This Book

The book contains three parts:

Part 1 About OLE Automation

Chapter 1 "Overview of OLE Automation," introduces the basic concepts of OLE Automation and identifies the components you'll use.

Chapter 2, "Exposing OLE Automation Objects," shows how to write and expose programmable objects for use by OLE Automation controllers.

Chapter 3, "Accessing OLE Automation Objects," explains how to write applications and programming tools that access exposed objects.

Chapter 4, "Standards and Guidelines," lists the standard OLE Automation objects that are recommended for most applications, and describes naming conventions for objects.

Part 2 Reference Information

Chapter 5, "Dispatch Interfaces," describes the interfaces and functions that support access to exposed objects.

Chapter 6, "Data Manipulation Functions," describes functions that manipulate arrays, strings, and variant types of data within OLE Automation.

Chapter 7, "MkTypLib and Object Description Language," describes the MkTypLib tool and its source file language. MkTypLib creates type libraries according to the description you provide.

Chapter 8, "Type Description Interfaces," describes the interfaces and functions that allow programs to read and bind to the descriptions of objects in a type library.

Chapter 9, "Type Building Interfaces," describes the interfaces and functions that are used by tools like MkTypLib to describe objects and build type libraries.

Appendixes

Appendix A, "National Language Support Functions," describes functions for 16-bit systems that support multiple national languages.

Appendix B, "Files You Need," lists the files you and your customers need to run OLE Automation applications.

Appendix C, "Information for Visual Basic Programmers," lists the OLE API's called by Visual Basic statements.

Appendix D, "How OLE Automation Compares Strings," describes the string comparison rules applied by OLE Automation.

Appendix E, "Handling GUIDs," provides supplemental information on GUIDs.

The **Glossary** defines terms useful in understanding OLE Automation.

Other Books and Technical Support

OLE Automation is part of OLE, which provides mechanisms for in-place activation, structured file storage, and many other application features. These other parts of OLE 2 are fully described by two books:

- *OLE 2 Programmer's Reference, Volume 1* describes the component object model, in-place activation, visual editing, structured file storage, and application registration in terms of the APIs and interfaces provided by OLE.
- *Inside OLE 2* by Kraig Brockschmidt provides introductory and how-to information about implementing OLE objects and containers.

If you are developing C++ applications, you may also find *The CDK Programmer's Guide* useful. This book is included with the Microsoft OLE Custom Controls Developer's Kit, which is part of Microsoft C++, version 2.0.

For technical support, see the OLE 2 SDK and the documentation for the product with which you received OLE. Support for OLE Automation is also provided through a forum on Compuserve. To access this forum, type **GO WINOBJ** at any Compuserve prompt. Once you are in the forum, ask your questions in Section 8.

Document Conventions

The following typographical conventions are used throughout this book:

Convention	**Meaning**
bold	Indicates a word that is a function name or other fixed part of a programming language, the Microsoft Windows operating system, or the OLE Application Programming Interface (API). For example, **DispInvoke** is an OLE-specific function. These words must always be typed exactly as they are printed.
italic	Indicates a word that is a placeholder or variable. For example, *ClassName* would be a placeholder for any OLE object class name. Function parameters in API reference material are italic to indicate that any variable name can be used. In addition, OLE terms are italicized at first use to highlight their definition.
UPPERCASE	Indicates a constant or an MS-DOS® path and name. For example, E_INVALIDARG is a constant. C:\OLE2\INCLUDE\OLE2.H is an MS-DOS path and name.
InitialCaps	Indicates the name of an object, method, property, user-defined function or event. For example, the Application object has a Visible property.
`monospace`	Indicates source code and syntax spacing. For example: `*pdwRegisterCF = 0;`

Note   The interface syntax in this book follows the variable-naming convention known as Hungarian notation, invented by programmer Charles Simonyi. Variables are prefixed with lowercase letters indicating their data type. For example, *lpszNewDocname* would be a long pointer to a zero-terminated string named *NewDocname*. See *Programming Windows* by Charles Petzold for more information about Hungarian notation.

BROWSE

SAMPLE: BROWSE: OLE Automation controller

BROWSE is an automation controller that controls the automation objects of the BROWSEH (Browse Helper) inproc server to browse a type library. BROWSE is a developer-oriented type library browser.

In addition to displaying type library information, this sample shows how to control an automation object by accessing it's properties and methods.

(BROWSEH is an Automation object that exposes the type information of a type library. BROWSEH is shipped as sample code with this release).

To compile: -----------

Requires OLE 2.02 or later. Use the external makefile called makefile to compile.

To run: -------

Run browse.exe and use the menu to select a type library to browse.

Files: ------

INVHELP.CPP, INVHELP.H contains two helper functions that are useful to create an automation object (CreateObject()) and to invoke a method or property of that object (Invoke()). These functions are general enough to be used by any Automation controller.

BROWSE.CPP uses the helper functions in invhelp.cpp to access the properties and methods of the automation objects of BROWSEH.

MAKEFILE makefile.

==
==

MAKEFILE   (BROWSE OLE Sample)

```
####
#makefile - makefile for browse.exe
#
#       Copyright (C) 1994, Microsoft Corporation
#
#Purpose:
#  Builds the OLE 2.0 Automation controller, browse.exe.
#
#
#  Usage: NMAKE                 ; build with defaults
#     or: NMAKE option          ; build with the given option(s)
#     or: NMAKE clean           ; erase all compiled files
#
#     option: dev = [win16 | win32]    ; dev=win32 is the default
#             DEBUG=[0 | 1]            ; DEBUG=1 is the default
#             HOST=[DOS | NT | WIN95]  ; HOST=DOS (for win16)
#                                      ; HOST=NT (for win32 on NT)
#                                      ; HOST=WIN95 (for win32 on Win95)
#
#Notes:
#  This makefile assumes that the PATH, INCLUDE and LIB environment
#  variables are setup properly.
#
############################################################################
##


##########################################################################
#
# Default Settings
#

CPU = i386

!if "$(dev)" == ""
dev = win32
HOST = NT
!endif

!if !("$(dev)" == "win16" || "$(dev)" == "win32")
!error Invalid dev option, choose from [win16 | win32]
!endif

!if "$(dev)" == "win16"
TARGET  = WIN16
!if "$(HOST)" == ""
HOST  = DOS
!endif
!endif

!if "$(dev)" == "win32"
TARGET  = WIN32
```

```
!if "$(HOST)" == ""
HOST  = NT
!endif
!endif

!ifdef NODEBUG
DEBUG = 0
!endif

!if "$(DEBUG)" == "0"
NODEBUG = 1
!endif

!if "$(DEBUG)" == ""
DEBUG = 1
!endif


##########################################################################
#
# WIN16 Settings
#
!if "$(TARGET)" == "WIN16"

CC   = cl
LINK = link
!if "$(HOST)" == "DOS"
WX   = wx /w
!else
WX   =
!endif

RCFLAGS = -dWIN16
CFLAGS = -c -W3 -AM -GA -GEs -DWIN16
LINKFLAGS = /NOD /NOI /BATCH /ONERROR:NOEXE

LIBS = libw.lib mlibcew.lib

!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS) -Od -Zi -D_DEBUG $(CL)
LINKFLAGS = $(LINKFLAGS) /COD
!else
CFLAGS = $(CFLAGS) -Ox $(CL)
LINKFLAGS = $(LINKFLAGS) /FAR /PACKC
!endif
!endif


##########################################################################
#
# WIN32 Settings
#
!if "$(TARGET)" == "WIN32"
```

```
WX =

!include <ntwin32.mak>

CC = $(cc)
CFLAGS = $(cflags) $(cvarsmt) -DINC_OLE2 $(cdebug)

!if "$(HOST)" == "NT"
CFLAGS = $(CFLAGS) -DUNICODE
!endif

!ifndef NODEBUG
CFLAGS = $(CFLAGS) -D_DEBUG
!endif

LINK = $(link)
LINKFLAGS = $(linkdebug) $(guilflags)
RCFLAGS = -DWIN32

!endif

############################################################################
#
# Build rules
#

.cpp.obj:
    @echo Compiling $<...
    $(CC) $<

.c.obj:

    @echo Compiling $<...
    $(CC) $<


############################################################################
#
# Application Settings
#

APPS = browse


!if "$(TARGET)" == "WIN16"
LIBS = commdlg.lib ole2.lib compobj.lib ole2disp.lib typelib.lib $(LIBS)
!endif
!if "$(TARGET)" == "WIN32"
LIBS = $(ole2libsmt)
!endif

OBJS = browse.obj invhelp.obj


############################################################################
```

```
#
# Default Goal
#

goal : setflags $(APPS).exe

setflags :
    set CL=$(CFLAGS)


##########################################################################
#
# Clean (erase) generated files
#
clean :
    if exist *.obj       del *.obj
    if exist $(APPS).exe del $(APPS).exe
    if exist $(APPS).map del $(APPS).map
    if exist $(APPS).res del $(APPS).res
    if exist *.pdb       del *.pdb


##########################################################################
#
# Application Build (WIN16 Specific)
#

!if "$(TARGET)" == "WIN16"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
    link $(LINKFLAGS) @<<
$(OBJS),
$@,,
$(LIBS),
$(APPS).def
<<
    rc -k -t $(APPS).res $@
!endif


##########################################################################
#
# Application Build (WIN32 Specific)
#
!if "$(TARGET)" == "WIN32"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
      $(LINK) @<<
        $(LINKFLAGS)
        -out:$@
        -map:$*.map
        $(OBJS)
        $(APPS).res
        $(LIBS)
<<
!endif
```

```
##########################################################################
#
# Application Build (Common)
#

$(APPS).res : $(APPS).rc
    rc $(RCFLAGS) -r -fo$@ $?


##########################################################################
#
# Dependencies
#

browse.obj : browse.cpp browse.h resource.h
     $(CC) browse.cpp
invhelp.obj : invhelp.cpp invhelp.h
     $(CC) invhelp.cpp
```

BROWSE.H (BROWSE OLE Sample)

```
#ifdef WIN32

#ifdef UNICODE
    #define FROM_OLE_STRING(str) str
    #define TO_OLE_STRING(str) str
#else
    #define FROM_OLE_STRING(str) ConvertToAnsi(str)
    char* ConvertToAnsi(OLECHAR FAR* szW);
    #define TO_OLE_STRING(str) ConvertToUnicode(str)
    OLECHAR* ConvertToUnicode(char FAR* szA);
    // Maximum length of string that can be converted between Ansi & Unicode
    #define STRCONVERT_MAXLEN 500
#endif

#else  // WIN16
  #define APIENTRY far pascal
  #define TCHAR char
  #define TEXT(sz) sz
  #define FROM_OLE_STRING(str) str
  #define TO_OLE_STRING(str) str
  #define LPTSTR LPSTR
  #define LPCTSTR LPCSTR

  // Windows NT defines the following in windowsx.h
  #define GET_WM_COMMAND_ID(w,l) (w)
  #define GET_WM_COMMAND_CMD(w,l) HIWORD(l)
  #define GET_WM_COMMAND_HWND(w,l) LOWORD(l)
#endif

// Function prototypes
int APIENTRY WinMain (HINSTANCE, HINSTANCE, LPSTR, int);
#ifdef WIN16
BOOL __export CALLBACK MainDialogFunc(HWND hwndDlg, UINT msg, WPARAM wParam,
LPARAM lParam);
#else
BOOL CALLBACK MainDialogFunc(HWND hwndDlg, UINT msg, WPARAM wParam, LPARAM
lParam);
#endif

typedef enum {
    TYPE_FUNCTION = 0,
    TYPE_PROPERTY = 1,
    TYPE_CONSTANT = 2,
    TYPE_PARAMETER = 3
} OBJTYPE;

class CBrowseApp
{
public:
    void Init(HWND hwndMain);
    void Cleanup();
    void ClearTypeLibStaticFields();
    void ClearTypeInfoStaticFields();
    void ClearElementStaticFields();
    void ClearParamStaticFields();
    HRESULT BrowseTypeLibrary();
    HRESULT ChangeTypeInfosSelection();
    HRESULT ChangeElementsSelection();
    HRESULT ChangeParametersSelection();
    void OpenElementHelpFile();
    void EmptyList(HWND hwndList);
    HRESULT LoadList(LPDISPATCH pdispItems, int nListID);
    HRESULT TypeToString(LPDISPATCH pdispTypeDesc, LPTSTR pszTypeName);
    void IDLFlagsToString(int n, LPTSTR psz);
    void CallConvToString(int n, LPTSTR psz);
    void FuncKindToString(int n, LPTSTR psz);
    void InvokeKindToString(int n, LPTSTR psz);
    void VariantToString(VARIANT v, LPTSTR psz);

private:
   HINSTANCE m_hinst;           // App's HINSTANCE.
   HWND m_hwndMain;             // App's main window which is a dialog..
   HFONT m_hfont;               // Non-bold font used for dialog fields.
   TCHAR m_szHelpFile[128];     // Help file used by type library being
browsed.
   long m_lElemHelpCtx;         // Help context of TypeInfo element being
browsed.
};
```

INVHELP.H (BROWSE OLE Sample)

```
HRESULT CreateObject(LPOLESTR pszProgID, IDispatch FAR* FAR* ppdisp);
HRESULT
Invoke(LPDISPATCH pdisp,
    WORD wFlags,
    LPVARIANT pvRet,
    EXCEPINFO FAR* pexcepinfo,
    UINT FAR* pnArgErr,
    LPOLESTR pszName,
    LPCTSTR pszFmt,
    ...);
```

RESOURCE.H   (BROWSE OLE Sample)

```
//{{NO_DEPENDENCIES}}
// App Studio generated include file.
// Used by BROWSER.RC
//
#define IDD_MAINDIALOG                  101
#define ID_MENU                         102
#define ID_ICON                         103

#define IDC_TYPEINFOSLIST               1001
#define IDC_ELEMENTSLIST                1002
#define IDC_PARAMETERSLIST              1003
#define IDC_TYPELIB_NAME                1004
#define IDC_TYPELIB_DOC                 1005
#define IDC_TYPELIB_VERSION             1006
#define IDC_TYPELIB_HELPFILE            1007
#define IDC_TYPELIB_LCID                1008
#define IDC_TYPEINFO_TYPE               1009
#define IDC_TYPEINFO_HELPCTX            1010
#define IDC_TYPEINFO_DOC                1011
#define IDC_TYPEINFO_ALIASTYPE          1012
#define IDC_ELEM_TYPE                   1013
#define IDC_ELEM_HELPCTX                1014
#define IDC_ELEM_FUNCKIND               1015
#define IDC_ELEM_CALLCONV               1016
#define IDC_ELEM_INVOKEKIND             1017
#define IDC_ELEM_MEMID                  1018
#define IDC_ELEM_CONST_VALUE            1019
#define IDC_ELEM_DOC                    1020
#define IDC_PARAM_TYPE                  1021
#define IDC_PARAM_INOUT                 1022
#define IDC_ELEM_OPEN_HELPFILE          1023

#define IDC_TYPELIB_FIRST               1004
#define IDC_TYPELIB_LAST                1008
#define IDC_TYPEINFO_FIRST              1009
#define IDC_TYPEINFO_LAST               1012
#define IDC_ELEM_FIRST                  1013
#define IDC_ELEM_LAST                   1020
#define IDC_PARAM_FIRST                 1021
#define IDC_PARAM_LAST                  1022

#define IDM_FILEOPEN                    40001
#define IDM_EXIT                        40002
#define IDC_STATIC                      -1
```

BROWSE.CPP (BROWSE OLE Sample)

```
/*************************************************************************
**
**    Type Library Browser
**
**    browseex.cpp
**
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
  #include <commdlg.h>
#endif
#include "resource.h"
#include "browse.h"
#include "invhelp.h"

CBrowseApp BrowseApp;          // Application

int APIENTRY WinMain (HINSTANCE hinst, HINSTANCE hinstPrev, LPSTR lpCmdLine,
int nCmdShow)
{
   WNDCLASS wc;
   HRESULT hr;

   //  It is recommended that all OLE applications set
   //  their message queue size to 96. This improves the capacity
   //  and performance of OLE's LRPC mechanism.
   int cMsg = 96;                  // Recommend msg queue size for OLE
   while (cMsg && !SetMessageQueue(cMsg))  // take largest size we can get.
       cMsg -= 8;
   if (!cMsg)
       return -1;

   // Register class for the dialog that is the main window.
   wc.style = 0;
   wc.lpfnWndProc = DefDlgProc;
   wc.cbClsExtra = 0;
   wc.cbWndExtra = DLGWINDOWEXTRA;
   wc.hInstance = hinst;
   wc.hIcon = LoadIcon(hinst, MAKEINTRESOURCE(ID_ICON));
   wc.hCursor = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground = HBRUSH(COLOR_WINDOW + 1);
   wc.lpszMenuName = NULL;
   wc.lpszClassName = TEXT("DlgClass");
   RegisterClass(&wc);

   hr = OleInitialize(NULL);
   if FAILED(hr)
   {
       MessageBox(NULL, TEXT("Could not Intialize OLE"), TEXT("Error"),
MB_ICONEXCLAMATION|MB_OK);
       return 0;
   }
   DialogBox(hinst, MAKEINTRESOURCE(IDD_MAINDIALOG), NULL,
(DLGPROC)MainDialogFunc);
   OleUninitialize();

   return 0;
}


BOOL CALLBACK MainDialogFunc(HWND hwndDlg, UINT msg, WPARAM wParam, LPARAM
lParam)
{
   HRESULT hr;

   switch (msg)
   {
      case WM_INITDIALOG:
          BrowseApp.Init(hwndDlg);
          return FALSE;

      case WM_COMMAND:
         switch(GET_WM_COMMAND_ID(wParam,lParam))
         {
             case IDM_EXIT:
             case IDCANCEL:
                 BrowseApp.Cleanup();
                 EndDialog(hwndDlg, IDOK);
                 CoFreeUnusedLibraries();
                 return TRUE;

             case IDM_FILEOPEN:
                 hr = BrowseApp.BrowseTypeLibrary();
                 if (FAILED(hr))
                     MessageBox(hwndDlg, TEXT("Failed to browse type
library"), TEXT("Error"), MB_ICONEXCLAMATION|MB_OK);
                 return TRUE;

             case IDC_TYPEINFOSLIST:
                 switch (GET_WM_COMMAND_CMD(wParam, lParam))
                 {
                     case LBN_SELCHANGE:
                         hr = BrowseApp.ChangeTypeInfosSelection();
                         if (FAILED(hr))
```

```
                             MessageBox(hwndDlg, TEXT("Failed to get
TypeInfo information"), TEXT("Error"), MB_ICONEXCLAMATION|MB_OK);
                         return TRUE;
                 }
                 break;

             case IDC_ELEMENTSLIST:
                 switch (GET_WM_COMMAND_CMD(wParam, lParam))
                 {
                     case LBN_SELCHANGE:
                         hr = BrowseApp.ChangeElementsSelection();
                         if (FAILED(hr))
                             MessageBox(hwndDlg, TEXT("Failed to get Element
information"), TEXT("Error"), MB_ICONEXCLAMATION|MB_OK);
                         return TRUE;
                 }
                 break;

             case IDC_PARAMETERSLIST:
                 switch (GET_WM_COMMAND_CMD(wParam, lParam))
                 {
                     case LBN_SELCHANGE:
                         hr = BrowseApp.ChangeParametersSelection();
                         if (FAILED(hr))
                             MessageBox(hwndDlg, TEXT("Failed to get
Parameter information"), TEXT("Error"), MB_ICONEXCLAMATION|MB_OK);
                         return TRUE;

                 }
                 break;

             case IDC_ELEM_OPEN_HELPFILE:
                 BrowseApp.OpenElementHelpFile();
                 return TRUE;

         }
         break;
   }
   return FALSE;
}

/*
 * CBrowseApp::BrowseTypeLibrary
 *
 * Purpose:
 *  Prompts user for type library. Fills the TypeInfo list with the
TypeInfos of
 *  the type library.
 *
 */
HRESULT CBrowseApp::BrowseTypeLibrary()
{
    OPENFILENAME   ofn;
    TCHAR szFileName[128];
    BOOL bRes;
    VARIANT vRet;
    EXCEPINFO excepinfo;
    UINT nArgErr;
    HRESULT hr;
    HCURSOR hcursorOld;
    LPDISPATCH pdispBrowseHelper = NULL;
    LPDISPATCH pdispTypeLibrary = NULL;
    LPDISPATCH pdispTypeInfos = NULL;
    TCHAR szTemp[100];

    // Prompt user for type library
    szFileName[0] = '\0';
    _fmemset(&ofn, 0, sizeof(ofn));
    ofn.lStructSize = sizeof(ofn);
    ofn.hwndOwner = m_hwndMain;
    ofn.lpstrFile = szFileName;
    ofn.nMaxFile  = sizeof(szFileName);
    ofn.hInstance = m_hinst;
    ofn.lpstrFilter  = TEXT("Type Libraries *.tlb,*.olb\0*.tlb;*.olb\0All
Files *.*\0*.*\0\0");
    ofn.nFilterIndex = 1;
    ofn.Flags= OFN_FILEMUSTEXIST;
    bRes = GetOpenFileName(&ofn);
    if (!bRes)
        return ResultFromScode(S_FALSE);

    // Empty lists and clear fields.
    EmptyList(GetDlgItem(m_hwndMain, IDC_TYPEINFOSLIST));
    EmptyList(GetDlgItem(m_hwndMain, IDC_ELEMENTSLIST));
    EmptyList(GetDlgItem(m_hwndMain, IDC_PARAMETERSLIST));
    ClearTypeLibStaticFields();
    ClearTypeInfoStaticFields();
    ClearElementStaticFields();
    ClearParamStaticFields();
    EnableWindow(GetDlgItem(m_hwndMain, IDC_ELEM_OPEN_HELPFILE), FALSE);

    hcursorOld = SetCursor(LoadCursor(NULL, IDC_WAIT));
    // Create BrowseHelper object
    hr = CreateObject(OLESTR("BrowseHelper.Browser"), &pdispBrowseHelper);
    if (FAILED(hr))
    {
        MessageBox(m_hwndMain, TEXT("Failed to create BrowseHelper object"),
TEXT("Error"), MB_ICONEXCLAMATION|MB_OK);
        goto error;
    }

    // Invoke IBrowseHelper.BrowseTypeLibrary(szFileName). Returns
ITypeLibrary.
    hr = Invoke(pdispBrowseHelper, DISPATCH_METHOD, &vRet, &excepinfo,
&nArgErr,
                      OLESTR("BrowseTypeLibrary"), TEXT("s"),
(LPOLESTR)TO_OLE_STRING(szFileName));
    if (FAILED(hr))
        goto error;
    pdispTypeLibrary = V_DISPATCH(&vRet);
    // Invoke ITypeLibrary.Name. Returns BSTR.
    hr = Invoke(pdispTypeLibrary, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("Name"), TEXT(""));
    if (FAILED(hr))
        goto error;
    SetDlgItemText(m_hwndMain, IDC_TYPELIB_NAME,
FROM_OLE_STRING(V_BSTR(&vRet)));
    VariantClear(&vRet);

    // Invoke ITypeLibrary.HelpFile. Returns BSTR.
    hr = Invoke(pdispTypeLibrary, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("HelpFile"), TEXT(""));
    if (FAILED(hr))
        goto error;
    SetDlgItemText(m_hwndMain, IDC_TYPELIB_HELPFILE,
FROM_OLE_STRING(V_BSTR(&vRet)));
    if (NULL == V_BSTR(&vRet))
        m_szHelpFile[0] = '\0';
    else lstrcpy(m_szHelpFile, FROM_OLE_STRING(V_BSTR(&vRet)));
    VariantClear(&vRet);

    // Invoke ITypeLibrary.Documentation. Returns BSTR.
    hr = Invoke(pdispTypeLibrary, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("Documentation"), TEXT(""));
    if (FAILED(hr))
        goto error;
    SetDlgItemText(m_hwndMain, IDC_TYPELIB_DOC,
FROM_OLE_STRING(V_BSTR(&vRet)));
    VariantClear(&vRet);

    // Invoke ITypeLibrary.LocaleID. Returns VT_I4.
    hr = Invoke(pdispTypeLibrary, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("LocaleID"), TEXT(""));
    if (FAILED(hr))
        goto error;
    wsprintf(szTemp, TEXT("0x%lx"), V_I4(&vRet));
    SetDlgItemText(m_hwndMain, IDC_TYPELIB_LCID, szTemp);

    // Invoke ITypeLibrary.MajorVersion. Returns VT_I2.
    hr = Invoke(pdispTypeLibrary, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("MajorVersion"), TEXT(""));
    if (FAILED(hr))
        goto error;
    wsprintf(szTemp, TEXT("%d"), V_I2(&vRet));
    // Invoke ITypeLibrary.MinorVersion. Returns VT_I2.
    hr = Invoke(pdispTypeLibrary, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("MinorVersion"), TEXT(""));
    if (FAILED(hr))
```

```
        goto error;
    wsprintf(szTemp, TEXT("%s.%d"), (LPSTR)szTemp, V_I2(&vRet));
    SetDlgItemText(m_hwndMain, IDC_TYPELIB_VERSION, szTemp);


    // Invoke ITypeLibrary.TypeInfos. Returns ICollection of ITypeInfo.
    hr = Invoke(pdispTypeLibrary, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("TypeInfos"), TEXT(""));
    if (FAILED(hr))
        goto error;
    pdispTypeInfos = V_DISPATCH(&vRet);

    // Load TypeInfo list with the names of the type infos.
    hr = LoadList(pdispTypeInfos, IDC_TYPEINFOSLIST);
    if (FAILED(hr))
        goto error;

    pdispBrowseHelper->Release();
    pdispTypeLibrary->Release();
    pdispTypeInfos->Release();
    SetCursor(hcursorOld);
    return NOERROR;

error:
    if (pdispBrowseHelper) pdispBrowseHelper->Release();
    if (pdispTypeLibrary) pdispTypeLibrary->Release();
    if (pdispTypeInfos) pdispTypeLibrary->Release();
    VariantClear(&vRet);
    SetCursor(hcursorOld);
    return hr;
}

/*
 * CBrowseApp::ChangeTypeInfosSelection
 *
 * Purpose:
 *  Called when user changes selection in TypeInfos list box. Information
 *  about the the selected TypeInfo is dispayed. Elements of the TypeInfo
 *  are placed in the Elements list box.
 *
 */
HRESULT CBrowseApp::ChangeTypeInfosSelection()
{
    int nCurSel;
    VARIANT vRet;
    EXCEPINFO excepinfo;
    UINT nArgErr;
    HRESULT hr;
    HCURSOR hcursorOld;
    LPDISPATCH pdispTypeInfo;
    TYPEKIND typekind;
    TCHAR szTemp[100];
    LPDISPATCH pdispElements = NULL;
    LPDISPATCH pdispMembers = NULL;
    LPDISPATCH pdispFunctions = NULL;
    LPDISPATCH pdispProperties = NULL;
    LPDISPATCH pdispMethods = NULL;
    LPDISPATCH pdispInterfaces = NULL;
    LPDISPATCH pdispTypeDesc = NULL;

    // Get current selection in TypeInfos list.
    nCurSel = (int)SendDlgItemMessage(m_hwndMain, IDC_TYPEINFOSLIST,
LB_GETCURSEL, 0, 0L);
    if (nCurSel == LB_ERR)
        return NOERROR;
    // Get IDispatch* of selected TypeInfo.
    pdispTypeInfo = (LPDISPATCH) SendDlgItemMessage(m_hwndMain,
IDC_TYPEINFOSLIST, LB_GETITEMDATA,
                (WPARAM)nCurSel, 0L);

    // Empty Elements & Parameters lists.
    EmptyList(GetDlgItem(m_hwndMain, IDC_ELEMENTSLIST));
    EmptyList(GetDlgItem(m_hwndMain, IDC_PARAMETERSLIST));
    ClearElementStaticFields();
    ClearParamStaticFields();

    ClearTypeInfoStaticFields();

    hcursorOld = SetCursor(LoadCursor(NULL, IDC_WAIT));
    // Invoke ITypeInformation.HelpContext. Returns VT_I4.
    hr = Invoke(pdispTypeInfo, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("HelpContext"), TEXT(""));
    if (FAILED(hr))
        goto error;
    wsprintf(szTemp, TEXT("%ld"), V_I4(&vRet));
    SetDlgItemText(m_hwndMain, IDC_TYPEINFO_HELPCTX, szTemp);

    // Invoke ITypeInformation.Documentation. Returns BSTR.
    hr = Invoke(pdispTypeInfo, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("Documentation"), TEXT(""));
    if (FAILED(hr))
        goto error;
    SetDlgItemText(m_hwndMain, IDC_TYPEINFO_DOC,
FROM_OLE_STRING(V_BSTR(&vRet)));
    VariantClear(&vRet);

    // Invoke ITypeInformation.TypeInfoKind. Returns TYPEKIND.
    hr = Invoke(pdispTypeInfo, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("TypeInfoKind"), TEXT(""));
    if (FAILED(hr))
        goto error;
    typekind = (TYPEKIND)V_I2(&vRet);

    switch (typekind)
    {
        case TKIND_ENUM:
            SetDlgItemText(m_hwndMain, IDC_TYPEINFO_TYPE,
TEXT("TKIND_ENUM"));
            // Invoke IEnum.Elements. Returns ICollection of IConstant.
            hr = Invoke(pdispTypeInfo, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                        OLESTR("Elements"), TEXT(""));
            if (FAILED(hr))
                goto error;
            pdispElements = V_DISPATCH(&vRet);
            // Put the Enumerator constants in the Elements list.
            hr = LoadList(pdispElements, IDC_ELEMENTSLIST);
            if (FAILED(hr))
                goto error;
            pdispElements->Release();
            break;

        case TKIND_RECORD:
            SetDlgItemText(m_hwndMain, IDC_TYPEINFO_TYPE,
TEXT("TKIND_RECORD"));
            // Invoke IStruct.Members. Returns ICollection of IProperty.
            hr = Invoke(pdispTypeInfo, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                        OLESTR("Members"), TEXT(""));
            if (FAILED(hr))
                goto error;
            pdispMembers = V_DISPATCH(&vRet);
            // Put the Structure memebers in the Elements list.
            hr = LoadList(pdispMembers, IDC_ELEMENTSLIST);
            if (FAILED(hr))
                goto error;
            pdispMembers->Release();
            break;

        case TKIND_MODULE:
            SetDlgItemText(m_hwndMain, IDC_TYPEINFO_TYPE,
TEXT("TKIND_MODULE"));
            // Invoke IModule.Functions. Returns ICollection of IFunction.
            hr = Invoke(pdispTypeInfo, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                        OLESTR("Functions"), TEXT(""));
            if (FAILED(hr))
                goto error;
            pdispFunctions = V_DISPATCH(&vRet);
            // Put the Module functions in the Elements list.
            hr = LoadList(pdispFunctions, IDC_ELEMENTSLIST);
            if (FAILED(hr))
                goto error;
            pdispFunctions->Release();
            break;

        case TKIND_INTERFACE:
            SetDlgItemText(m_hwndMain, IDC_TYPEINFO_TYPE,
TEXT("TKIND_INTERFACE"));
```

```
            // Invoke IInterface.Functions. Returns ICollection of
IFunction.
            hr = Invoke(pdispTypeInfo, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                        OLESTR("Functions"), TEXT(""));
            if (FAILED(hr))
                goto error;
            pdispFunctions = V_DISPATCH(&vRet);
            // Put the Interface functions in the Elements list.
            hr = LoadList(pdispFunctions, IDC_ELEMENTSLIST);
            if (FAILED(hr))
                goto error;
            pdispFunctions->Release();
            break;

        case TKIND_DISPATCH:
            SetDlgItemText(m_hwndMain, IDC_TYPEINFO_TYPE,
TEXT("TKIND_DISPATCH"));
            // Invoke Dispinterface.Properties. Returns ICollection of
IProperty.
            hr = Invoke(pdispTypeInfo, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                        OLESTR("Properties"), TEXT(""));
            if (FAILED(hr))
                goto error;
            pdispProperties = V_DISPATCH(&vRet);
            // Invoke IDispinterface.Methods. Returns ICollection of
IFunction.
            hr = Invoke(pdispTypeInfo, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                        OLESTR("Methods"), TEXT(""));
            if (FAILED(hr))
                goto error;
            pdispMethods = V_DISPATCH(&vRet);
            // Put the Dispinterface properties and methods in the Elements
list.
            hr = LoadList(pdispProperties, IDC_ELEMENTSLIST);
            if (FAILED(hr))
                goto error;
            pdispProperties->Release();
            hr = LoadList(pdispMethods, IDC_ELEMENTSLIST);
            if (FAILED(hr))
                goto error;
            pdispMethods->Release();
            break;

        case TKIND_COCLASS:
            SetDlgItemText(m_hwndMain, IDC_TYPEINFO_TYPE,
TEXT("TKIND_COCLASS"));
            // Invoke ICoClass.Interfaces. Returns ICollection of IInterface
& IDispinterface.
            hr = Invoke(pdispTypeInfo, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                        OLESTR("Interfaces"), TEXT(""));
            if (FAILED(hr))
```

```
                goto error;
            pdispInterfaces = V_DISPATCH(&vRet);
            // Put the CoClass interfaces and dispinterfaces in the Elements
list.
            hr = LoadList(pdispInterfaces, IDC_ELEMENTSLIST);
            if (FAILED(hr))
                goto error;
            pdispInterfaces->Release();
            break;

        case TKIND_ALIAS:
            // An alias does not have elements. Display the base type.
            SetDlgItemText(m_hwndMain, IDC_TYPEINFO_TYPE,
TEXT("TKIND_ALIAS"));
            // Invoke IAlias.BaseType. Returns ITypeDesc.
            hr = Invoke(pdispTypeInfo, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                        OLESTR("BaseType"), TEXT(""));
            if (FAILED(hr))
                goto error;
            pdispTypeDesc = V_DISPATCH(&vRet);
            szTemp[0] = '\0'; // Must initialize to empty string before
calling TypeToString.
            hr = TypeToString(pdispTypeDesc, szTemp);
            if (FAILED(hr))
                goto error;
            SetDlgItemText(m_hwndMain, IDC_TYPEINFO_ALIASTYPE, szTemp);
            pdispTypeDesc->Release();
            break;

        case TKIND_UNION:
            SetDlgItemText(m_hwndMain, IDC_TYPEINFO_TYPE,
TEXT("TKIND_UNION"));
            // Invoke IUnion.Members. Returns ICollection of IProperty.
            hr = Invoke(pdispTypeInfo, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                        OLESTR("Members"), TEXT(""));
            if (FAILED(hr))
                goto error;
            pdispMembers = V_DISPATCH(&vRet);
            // Put the Union members in the Elements list.
            hr = LoadList(pdispMembers, IDC_ELEMENTSLIST);
            if (FAILED(hr))
                goto error;
            pdispMembers->Release();
            break;
    }
    SetCursor(hcursorOld);
    return NOERROR;
error:
    if (pdispElements) pdispElements->Release();
    if (pdispMembers) pdispMembers->Release();
    if (pdispFunctions) pdispFunctions->Release();
    if (pdispProperties) pdispProperties->Release();
    if (pdispMethods) pdispElements->Release();
    if (pdispInterfaces) pdispInterfaces->Release();
    if (pdispTypeDesc) pdispTypeDesc->Release();
    VariantClear(&vRet);
    SetCursor(hcursorOld);
    return hr;
}

/*
 * CBrowseApp::ChangeElementsSelection
 *
 * Purpose:
 *  Called when user changes selection in Elements list box. Information
 *  about the the selected Element is dispayed. If the element is a
function,
 *  the parameters are placed in the the Parameters list box.
 *
 */
HRESULT CBrowseApp::ChangeElementsSelection()
{
    int nCurSel;
    VARIANT vRet;
    EXCEPINFO excepinfo;
    UINT nArgErr;
    HRESULT hr;
    int nKind;
    TYPEKIND typekind;
    LPDISPATCH pdispElement;
    LPDISPATCH pdispParameters = NULL;
    LPDISPATCH pdispTypeDesc = NULL;
    TCHAR szTemp[100];

    // Disable the help file button.
    EnableWindow(GetDlgItem(m_hwndMain, IDC_ELEM_OPEN_HELPFILE), FALSE);

    // Get current selection in Elements list.
    nCurSel = (int)SendDlgItemMessage(m_hwndMain, IDC_ELEMENTSLIST,
LB_GETCURSEL, 0, 0L);
    if (nCurSel == LB_ERR)
        return NOERROR;
    // Get IDispatch* of selected Element.
    pdispElement = (LPDISPATCH) SendDlgItemMessage(m_hwndMain,
IDC_ELEMENTSLIST, LB_GETITEMDATA,
                (WPARAM)nCurSel, 0L);

    // Empty Parameters lists.
    EmptyList(GetDlgItem(m_hwndMain, IDC_PARAMETERSLIST));
    ClearParamStaticFields();

    ClearElementStaticFields();

    // Invoke
IFunction/IConstant/IProperty/IInterface/IDispinterface.HelpContext. Returns
VT_I4.
```

```
    hr = Invoke(pdispElement, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("HelpContext"), TEXT(""));
    if (FAILED(hr))
        goto error;
    wsprintf(szTemp, TEXT("%ld"), V_I4(&vRet));
    SetDlgItemText(m_hwndMain, IDC_ELEM_HELPCTX, szTemp);
    m_lElemHelpCtx = V_I4(&vRet);

    // Invoke
IFunction/IConstant/IProperty/IInterface/IDispinterface.Documentation.
Returns BSTR.
    hr = Invoke(pdispElement, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("Documentation"), TEXT(""));
    if (FAILED(hr))
        goto error;
    SetDlgItemText(m_hwndMain, IDC_ELEM_DOC,
FROM_OLE_STRING(V_BSTR(&vRet)));
    VariantClear(&vRet);

    // Invoke IFunction/IConstant/IProperty.MemberID. Returns VT_I4.
    hr = Invoke(pdispElement, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("MemberID"), TEXT(""));
    if (FAILED(hr))
    {
        // This maybe a interface/dispinterface element of a coclass
        hr = Invoke(pdispElement, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("TypeInfoKind"), TEXT(""));
        if (FAILED(hr))
            goto error;
        typekind = (TYPEKIND)V_I2(&vRet);
        if (typekind == TKIND_INTERFACE)
            SetDlgItemText(m_hwndMain, IDC_ELEM_TYPE,
TEXT("TKIND_INTERFACE"));
        else if (typekind == TKIND_DISPATCH)
            SetDlgItemText(m_hwndMain, IDC_ELEM_TYPE,
TEXT("TKIND_DISPATCH"));
        else SetDlgItemText(m_hwndMain, IDC_ELEM_TYPE, TEXT("Unknown
element"));
        return NOERROR;
    }
    wsprintf(szTemp, TEXT("0x%lx"), V_I4(&vRet));
    SetDlgItemText(m_hwndMain, IDC_ELEM_MEMID, szTemp);

    // Invoke IFunction/IConstant/IProperty.Kind property. Returns OBJKIND.
    hr = Invoke(pdispElement, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("Kind"), TEXT(""));
    if (FAILED(hr))
        goto error;
    nKind = V_I2(&vRet);
    switch (nKind)
    {
        case TYPE_FUNCTION: //Function
            // Invoke IFunction.Parameters. Returns ICollection of
IParameter..
            hr = Invoke(pdispElement, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                        OLESTR("Parameters"), TEXT(""));
            if (FAILED(hr))
                goto error;
            pdispParameters = V_DISPATCH(&vRet);
            hr = LoadList(pdispParameters, IDC_PARAMETERSLIST);
            if FAILED(hr)
                goto error;
            pdispParameters->Release(); pdispParameters = NULL;

            // Invoke IFunction.CallConvention. Returns CALLCONV.
            hr = Invoke(pdispElement, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                OLESTR("CallConvention"), TEXT(""));
            if (FAILED(hr))
                goto error;
            CallConvToString(V_I2(&vRet), szTemp);
            SetDlgItemText(m_hwndMain, IDC_ELEM_CALLCONV, szTemp);

            // Invoke IFunction.FuncKind. Returns FUNCKIND.
            hr = Invoke(pdispElement, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                OLESTR("FuncKind"), TEXT(""));
            if (FAILED(hr))
                goto error;
            FuncKindToString(V_I2(&vRet), szTemp);
            SetDlgItemText(m_hwndMain, IDC_ELEM_FUNCKIND, szTemp);

            // Invoke IFunction.InvocationKind. Returns INVOKEKIND.
            hr = Invoke(pdispElement, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                OLESTR("InvocationKind"), TEXT(""));
            if (FAILED(hr))
                goto error;
            InvokeKindToString(V_I2(&vRet), szTemp);
            SetDlgItemText(m_hwndMain, IDC_ELEM_INVOKEKIND, szTemp);

            // Invoke IFunction.ReturnType. Returns ITypeDesc.
            hr = Invoke(pdispElement, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                OLESTR("ReturnType"), TEXT(""));
            if (FAILED(hr))
                goto error;
            break;

        case TYPE_CONSTANT:
            // Invoke IConstant.Value. Returns VARIANT.
            hr = Invoke(pdispElement, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                OLESTR("Value"), TEXT(""));
            if (FAILED(hr))
                goto error;
            VariantToString(vRet, szTemp);
            SetDlgItemText(m_hwndMain, IDC_ELEM_CONST_VALUE, szTemp);
            // Fall through to default.

        default:
            // Invoke IConstant/IProperty.Type. Returns ITypeDesc.
            hr = Invoke(pdispElement, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                OLESTR("Type"), TEXT(""));
            if (FAILED(hr))
                goto error;
            break;
    }
    // Get string that describes type and display it.
    pdispTypeDesc = V_DISPATCH(&vRet);
    szTemp[0] = '\0'; // Must initialize to empty string before calling
TypeToString.
    hr = TypeToString(pdispTypeDesc, szTemp);
    if (FAILED(hr))
        goto error;
    SetDlgItemText(m_hwndMain, IDC_ELEM_TYPE, szTemp);
    pdispTypeDesc->Release();

    // Enable the helpfile button if a helpfile and help context is
specified.
    if (m_lElemHelpCtx && m_szHelpFile[0])
        EnableWindow(GetDlgItem(m_hwndMain, IDC_ELEM_OPEN_HELPFILE), TRUE);
    return NOERROR;
error:
    if (pdispParameters) pdispParameters->Release();
    VariantClear(&vRet);
    return hr;
}

/*
 * CBrowseApp::ChangeParametersSelection
 *
 * Purpose:
 *  Called when user changes selection in Parameters list box. Information
 *  about the the selected Parameter is dispayed.
 *
 */
HRESULT CBrowseApp::ChangeParametersSelection()
{
    int nCurSel;
    VARIANT vRet;
    EXCEPINFO excepinfo;
    UINT nArgErr;
    HRESULT hr;
    LPDISPATCH pdispParameter;
    LPDISPATCH pdispTypeDesc = NULL;
    TCHAR szTemp[100];

    // Get current selection in Parameters list.
    nCurSel = (int)SendDlgItemMessage(m_hwndMain, IDC_PARAMETERSLIST,
LB_GETCURSEL, 0, 0L);
    if (nCurSel == LB_ERR)
        return NOERROR;
    // Get IDispatch* of selected Parameter.
    pdispParameter = (LPDISPATCH) SendDlgItemMessage(m_hwndMain,
IDC_PARAMETERSLIST, LB_GETITEMDATA,
                (WPARAM)nCurSel, 0L);

    ClearParamStaticFields();

    // Invoke IParameter.IDLFlags. Returns VT_I2.
    hr = Invoke(pdispParameter, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("IDLFlags"), TEXT(""));
    if (FAILED(hr))
        goto error;
    IDLFlagsToString(V_I2(&vRet), szTemp);
    SetDlgItemText(m_hwndMain, IDC_PARAM_INOUT, szTemp);

    // Invoke IParameter.Type. Returns ITypeDesc.
    hr = Invoke(pdispParameter, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("Type"), TEXT(""));
    if (FAILED(hr))
        goto error;

   // Get string that describes type and display it.
    pdispTypeDesc = V_DISPATCH(&vRet);
    szTemp[0] = '\0'; // Must initialize to empty string before calling
TypeToString.
    hr = TypeToString(pdispTypeDesc, szTemp);
    if (FAILED(hr))
        goto error;
    SetDlgItemText(m_hwndMain, IDC_PARAM_TYPE, szTemp);
    pdispTypeDesc->Release();
    return NOERROR;
error:
    if (pdispTypeDesc) pdispTypeDesc->Release();
    VariantClear(&vRet);
    return hr;
}

/*
 * CBrowseApp::OpenElementHelpFile
 *
 * Purpose:
 *  Opens the helpfile for the element.
 *
 */
void CBrowseApp::OpenElementHelpFile()
```

```
{
    BOOL b;

    b = WinHelp(m_hwndMain, m_szHelpFile, HELP_CONTEXT, m_lElemHelpCtx);
    if (!b)
        MessageBox(m_hwndMain, TEXT("Failed to open help file"),
TEXT("Error"),
            MB_ICONEXCLAMATION|MB_OK);
}

void CBrowseApp::EmptyList(HWND hwndList)
{
    int nItems, i;
    LPDISPATCH pdispItem;

    nItems = (int)SendMessage(hwndList, LB_GETCOUNT, 0, 0L);
    for (i=0; i<nItems; i++)
    {
        pdispItem = (LPDISPATCH)SendMessage(hwndList, LB_GETITEMDATA,
(WPARAM)i, 0L);
        if (pdispItem)
            pdispItem->Release();
    }
    SendMessage(hwndList, LB_RESETCONTENT, 0, 0L);
}

/*
 * CBrowseApp::LoadList
 *
 * Parameters:
 *  pdispItems IDispatch* of collection of items to be put into the list.
 *  nListID    Identifies the list to be filled.
 *
 * Purpose:
 *  Loads the list with the items in the collection.
 *
 */
HRESULT CBrowseApp::LoadList(LPDISPATCH pdispItems, int nListID)
{
    VARIANT vRet, v;
    HRESULT hr;
    EXCEPINFO excepinfo;
    UINT nArgErr;
    LPUNKNOWN punkEnum;
    IEnumVARIANT FAR* penum = NULL;
    LPDISPATCH pdispItem = NULL;
    int nIndex;

    // Get _NewEnum property. Returns enumerator's IUnknown.
    hr = Invoke(pdispItems, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                        OLESTR("_NewEnum"), TEXT(""));
    if (FAILED(hr))
        goto error;
    punkEnum = V_UNKNOWN(&vRet);
    hr = punkEnum->QueryInterface(IID_IEnumVARIANT, (LPVOID FAR*)&penum);
    if (FAILED(hr))
        goto error;
    punkEnum->Release();

    VariantInit(&v);
    // Enumerate the Items.
    while (S_OK == penum->Next(1, &v, NULL))
    {
        pdispItem = V_DISPATCH(&v);
        pdispItem->AddRef();
        VariantClear(&v);

        // Get Name of Item.
        hr = Invoke(pdispItem, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                    OLESTR("Name"), TEXT(""));
        if (FAILED(hr))
            goto error;
        // Add name to ItemsList.
        nIndex = (int)SendDlgItemMessage(m_hwndMain, nListID, LB_ADDSTRING,
            0, (LPARAM)(LPTSTR)FROM_OLE_STRING(V_BSTR(&vRet)));
        VariantClear(&vRet);
        // Save IDispatch* of Item in the list.
        SendDlgItemMessage(m_hwndMain, nListID, LB_SETITEMDATA,
            nIndex, (LPARAM)pdispItem);
    }
    penum->Release();
    return NOERROR;

error:
    if (penum) penum->Release();
    if (pdispItem) pdispItem->Release();
    return hr;
}

void CBrowseApp::Init(HWND hwndMain)
{
    HFONT hfontDlg;
    LOGFONT lfont;
    int nID;

    m_hwndMain = hwndMain;
    m_hfont = NULL;
#ifdef WIN16
    m_hinst = (HINSTANCE)GetWindowWord(hwndMain, GWW_HINSTANCE);
#else
    m_hinst = (HINSTANCE)GetWindowLong(hwndMain, GWL_HINSTANCE);
#endif
    EnableWindow(GetDlgItem(m_hwndMain, IDC_ELEM_OPEN_HELPFILE), FALSE);

    // Change to the font of the fields to a non-bold font.
    hfontDlg = (HFONT)SendMessage(m_hwndMain, WM_GETFONT, NULL, NULL);
    if (!hfontDlg)
```

```
        return;
    GetObject(hfontDlg, sizeof(LOGFONT), (LPVOID)&lfont);
    lfont.lfWeight = FW_NORMAL;
    if (m_hfont = CreateFontIndirect(&lfont))
        for (nID=IDC_TYPELIB_FIRST; nID<=IDC_PARAM_LAST; nID++)
            SendDlgItemMessage(m_hwndMain, nID, WM_SETFONT, (WPARAM)m_hfont,
0L);

}

void CBrowseApp::Cleanup()
{
    EmptyList(GetDlgItem(m_hwndMain, IDC_TYPEINFOSLIST));
    EmptyList(GetDlgItem(m_hwndMain, IDC_ELEMENTSLIST));
    EmptyList(GetDlgItem(m_hwndMain, IDC_PARAMETERSLIST));
    ClearTypeLibStaticFields();
    ClearTypeInfoStaticFields();
    ClearElementStaticFields();
    ClearParamStaticFields();
    if (m_hfont)
        DeleteObject(m_hfont);
}

void CBrowseApp::ClearTypeLibStaticFields()
{
    int nID;

    for (nID=IDC_TYPELIB_FIRST; nID<=IDC_TYPELIB_LAST; nID++)

        SetDlgItemText(m_hwndMain, nID, TEXT(""));
}
void CBrowseApp::ClearTypeInfoStaticFields()
{
    int nID;

    for (nID=IDC_TYPEINFO_FIRST; nID<=IDC_TYPEINFO_LAST; nID++)
        SetDlgItemText(m_hwndMain, nID, TEXT(""));
}
void CBrowseApp::ClearElementStaticFields()
{
    int nID;

    for (nID=IDC_ELEM_FIRST; nID<=IDC_ELEM_LAST; nID++)
        SetDlgItemText(m_hwndMain, nID, TEXT(""));
}
void CBrowseApp::ClearParamStaticFields()
{
    int nID;

    for (nID=IDC_PARAM_FIRST; nID<=IDC_PARAM_LAST; nID++)
        SetDlgItemText(m_hwndMain, nID, TEXT(""));
}

#define CASE_VT(vt)   \
        case vt: \
            lstrcat(pszTypeName, TEXT(#vt)); \
            break;
HRESULT CBrowseApp::TypeToString(LPDISPATCH pdispTypeDesc, LPTSTR
pszTypeName)
{
    VARIANT vRet;
    VARTYPE vartype;
    HRESULT hr;
    EXCEPINFO excepinfo;
    UINT nArgErr;
    LPDISPATCH pdispTypeDesc2;

    // Get Type property
    hr = Invoke(pdispTypeDesc, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("Type"), TEXT(""));
    if (FAILED(hr))
        goto error;
    vartype = V_I2(&vRet);

    if (vartype & 0x2000)  // If SafeArray
      lstrcat(pszTypeName, TEXT("SAFEARRAY("));

    switch (vartype & ~0x7000)
    {
    CASE_VT(VT_EMPTY)
    CASE_VT(VT_NULL)
    CASE_VT(VT_I2)
    CASE_VT(VT_I4)
    CASE_VT(VT_R4)
    CASE_VT(VT_R8)
    CASE_VT(VT_CY)
    CASE_VT(VT_DATE)
    CASE_VT(VT_BSTR)
    CASE_VT(VT_DISPATCH)
    CASE_VT(VT_ERROR)
    CASE_VT(VT_BOOL)
    CASE_VT(VT_VARIANT)
    CASE_VT(VT_UNKNOWN)

    CASE_VT(VT_I1)
    CASE_VT(VT_UI1)
    CASE_VT(VT_UI2)
    CASE_VT(VT_UI4)
    CASE_VT(VT_I8)
    CASE_VT(VT_UI8)
    CASE_VT(VT_INT)
    CASE_VT(VT_UINT)
    CASE_VT(VT_VOID)
    CASE_VT(VT_HRESULT)
    CASE_VT(VT_SAFEARRAY)
    CASE_VT(VT_CARRAY)
    CASE_VT(VT_LPSTR)
    CASE_VT(VT_LPWSTR)
    CASE_VT(VT_FILETIME)
    CASE_VT(VT_BLOB)
    CASE_VT(VT_STREAM)
    CASE_VT(VT_STORAGE)
    CASE_VT(VT_STREAMED_OBJECT)
    CASE_VT(VT_STORED_OBJECT)
    CASE_VT(VT_BLOB_OBJECT)
    CASE_VT(VT_CF)
    CASE_VT(VT_CLSID)

    case VT_PTR:
         // Get ITypeDesc.PointerDesc property. Returns ITypeDesc.
         hr = Invoke(pdispTypeDesc, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("PointerDesc"), TEXT(""));
         if (FAILED(hr))
             goto error;
         pdispTypeDesc2 = V_DISPATCH(&vRet);
         TypeToString(pdispTypeDesc2, pszTypeName);
         lstrcat(pszTypeName, TEXT(" *"));
         pdispTypeDesc2->Release();
         break;

    case VT_USERDEFINED:
         // Get ITypeDesc.UserDefinedDesc property. Returns ITypeInformation
         hr = Invoke(pdispTypeDesc, DISPATCH_PROPERTYGET, &vRet, &excepinfo,
&nArgErr,
                OLESTR("UserDefinedDesc"), TEXT(""));
         if (FAILED(hr))
             goto error;
         pdispTypeDesc2 = V_DISPATCH(&vRet);
         // Get ITypeInformation.Name property. Returns BSTR.
         hr = Invoke(pdispTypeDesc2, DISPATCH_PROPERTYGET, &vRet,
&excepinfo, &nArgErr,
                OLESTR("Name"), TEXT(""));
         if (FAILED(hr))
             goto error;
         lstrcat(pszTypeName, FROM_OLE_STRING(V_BSTR(&vRet)));
         VariantClear(&vRet);
         pdispTypeDesc2->Release();
         break;
    }

    if (vartype & 0x2000) // If SafeArray
        lstrcat(pszTypeName, TEXT(")"));

    return NOERROR;
error:
    return ERROR;
}

void CBrowseApp::IDLFlagsToString(int n, LPTSTR psz)
{
   psz[0] = '\0';
   if (n & IDLFLAG_FIN)
   {
       lstrcpy(psz, TEXT("IN"));
       if (n & IDLFLAG_FOUT)
           lstrcat(psz, TEXT("|OUT"));
   }
   else if (n & IDLFLAG_FOUT)
       lstrcpy(psz, TEXT("OUT"));
}

#define CASE_INVOKE(invokekind)  \
        case invokekind: \
            lstrcpy(psz, TEXT(#invokekind)); \
            break;
void CBrowseApp::CallConvToString(int n, LPTSTR psz)
{
    CALLCONV c = (CALLCONV)n;
    switch (c)
    {
        CASE_INVOKE(CC_CDECL)
        CASE_INVOKE(CC_PASCAL)
        CASE_INVOKE(CC_MACPASCAL)
        CASE_INVOKE(CC_STDCALL)
        CASE_INVOKE(CC_SYSCALL)
        default:
            lstrcpy(psz, TEXT("Unknown"));
    }
}

#define CASE_FUNC(funckind)  \
        case funckind: \
            lstrcpy(psz, TEXT(#funckind)); \
            break;
void CBrowseApp::FuncKindToString(int n, LPTSTR psz)
{
    FUNCKIND f = (FUNCKIND)n;
    switch (f)
    {
        CASE_FUNC(FUNC_VIRTUAL)
        CASE_FUNC(FUNC_PUREVIRTUAL)
        CASE_FUNC(FUNC_NONVIRTUAL)
        CASE_FUNC(FUNC_STATIC)
        CASE_FUNC(FUNC_DISPATCH)
        default:
            lstrcpy(psz, TEXT("Unknown"));
    }
}

#define CASE_INVOKE(invokekind)  \
        case invokekind: \
            lstrcpy(psz, TEXT(#invokekind)); \
            break;
void CBrowseApp::InvokeKindToString(int n, LPTSTR psz)
{
    INVOKEKIND i = (INVOKEKIND)n;
    switch (i)
    {
        CASE_INVOKE(INVOKE_FUNC)
        CASE_INVOKE(INVOKE_PROPERTYGET)
        CASE_INVOKE(INVOKE_PROPERTYPUT)
        CASE_INVOKE(DISPATCH_PROPERTYPUTREF)
        default:
            lstrcpy(psz, TEXT("Unknown"));
    }
}
void CBrowseApp::VariantToString(VARIANT v, LPTSTR psz)
{
    HRESULT hr;
    VARIANT vTemp;

    VariantInit(&vTemp);
    hr = VariantChangeType(&vTemp, &v, 0, VT_BSTR);
    if (FAILED(hr))
        lstrcpy(psz, TEXT("Cannot Display Value"));
    else lstrcpy(psz, FROM_OLE_STRING(V_BSTR(&vTemp)));
    VariantClear(&vTemp);
}

#ifdef WIN32

#ifndef UNICODE
char* ConvertToAnsi(OLECHAR FAR* szW)
{
  static char achA[STRCONVERT_MAXLEN];

  WideCharToMultiByte(CP_ACP, 0, szW, -1, achA, STRCONVERT_MAXLEN, NULL,
NULL);
  return achA;
}

OLECHAR* ConvertToUnicode(char FAR* szA)
{
  static OLECHAR achW[STRCONVERT_MAXLEN];

  MultiByteToWideChar(CP_ACP, 0, szA, -1, achW, STRCONVERT_MAXLEN);
  return achW;
}
#endif

#endif
```

INVHELP.CPP (BROWSE OLE Sample)

```
/*************************************************************************
**
**    Automation Controller helper functions
**
**    invhelp.cpp
**
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include <stdarg.h>

#ifdef WIN16
  #define LPCTSTR LPCSTR
  #define LPOLESTR LPSTR
#endif


HRESULT CountArgsInFormat(LPCTSTR pszFmt, UINT FAR *pn);
LPCTSTR GetNextVarType(LPCTSTR pszFmt, VARTYPE FAR* pvt);

/*
 * CreateObject
 *
 * Purpose:
 *  Creates an instance of the Automation object and obtains it's IDispatch
interface.
 *  Uses Unicode with OLE.
 *
 * Parameters:
 *  pszProgID         ProgID of Automation object
 *  ppdisp             Returns IDispatch of Automation object
 *
 * Return Value:
 *  HRESULT indicating success or failure
 */
HRESULT CreateObject(LPOLESTR pszProgID, IDispatch FAR* FAR* ppdisp)
{
    CLSID clsid;                     // CLSID of automation object
    HRESULT hr;
    LPUNKNOWN punk = NULL;         // IUnknown of automation object
    LPDISPATCH pdisp = NULL;       // IDispatch of automation object

    *ppdisp = NULL;

    // Retrieve CLSID from the progID that the user specified
    hr = CLSIDFromProgID(pszProgID, &clsid);
    if (FAILED(hr))
        goto error;

    // Create an instance of the automation object and ask for the IDispatch
interface
    hr = CoCreateInstance(clsid, NULL, CLSCTX_SERVER,
                          IID_IUnknown, (void FAR* FAR*)&punk);
    if (FAILED(hr))
        goto error;

    hr = punk->QueryInterface(IID_IDispatch, (void FAR* FAR*)&pdisp);
    if (FAILED(hr))
        goto error;

    *ppdisp = pdisp;
    punk->Release();
    return NOERROR;

error:
    if (punk) punk->Release();
    if (pdisp) pdisp->Release();
    return hr;
}

/*
 * Invoke
 *
 * Purpose:
 *  Invokes a property accessor function or method of an automation object.
Uses Unicode with OLE.
 *
 * Parameters:
 *  pdisp         IDispatch* of automation object.
 *  wFlags        Specfies if property is to be accessed or method to be
invoked.
 *                Can hold DISPATCH_PROPERTYGET, DISPATCH_PROPERTYPUT,
DISPATCH_METHOD,
 *                DISPATCH_PROPERTYPUTREF or DISPATCH_PROPERTYGET|
DISPATCH_METHOD.
 *  pvRet         NULL if caller excepts no result. Otherwise returns
result.
 *  pexcepinfo    Returns exception info if DISP_E_EXCEPTION is returned.
Can be NULL if
 *                caller is not interested in exception information.
 *  pnArgErr      If return is DISP_E_TYPEMISMATCH, this returns the index
(in reverse
 *                order) of argument with incorrect type. Can be NULL if
caller is not interested
```

```
 *                 in this information.
 *  pszName        Name of property or method.
 *  pszFmt         Format string that describes the variable list of
parameters that
 *                 follows. The format string can contain the follwoing
characters.
 *                 & = mark the following format character as VT_BYREF
 *                 b = VT_BOOL
 *                 i = VT_I2
 *                 I = VT_I4
 *                 r = VT_R2
 *                 R = VT_R4
 *                 c = VT_CY
 *                 s = VT_BSTR (far string pointer can be passed, BSTR will
be allocated by this function).
 *                 e = VT_ERROR
 *                 d = VT_DATE
 *                 v = VT_VARIANT. Use this to pass data types that are not
described in
 *                                 the format string. (For example
SafeArrays).
 *                 D = VT_DISPATCH
 *                 U = VT_UNKNOWN
 *
 *  ...            Arguments of the property or method. Arguments are
described by pszFmt.
 *                 ****FAR POINTERS MUST BE PASSED FOR POINTER ARGUMENTS in
Win16.****
 * Return Value:
 *  HRESULT indicating success or failure
 *
 * Usage examples:
 *
 *  HRESULT hr;
 *  LPDISPATCH pdisp;
 *  BSTR bstr;
 *  short i;
 *  BOOL b;
 *  VARIANT v, v2;
 *
 *1. bstr = SysAllocString(OLESTR(""));
 *   hr = Invoke(pdisp, DISPATCH_METHOD, NULL, NULL, NULL,
OLESTR("method1"),
 *        TEXT("bis&b&i&s"), TRUE, 2, (LPOLESTR)OLESTR("param"), (BOOL
FAR*)&b, (short FAR*)&i, (BSTR FAR*)&bstr);
 *
 *2. VariantInit(&v);
 *   V_VT(&v) = VT_R8;
 *   V_R8(&v) = 12345.6789;
 *   VariantInit(&v2);
 *   hr = Invoke(pdisp, DISPATCH_METHOD, NULL, NULL, NULL,
OLESTR("method2"),
 *         TEXT("v&v"), v, (VARIANT FAR*)&v2);
 */
```

```
HRESULT
Invoke(LPDISPATCH pdisp,
    WORD wFlags,
    LPVARIANT pvRet,
    EXCEPINFO FAR* pexcepinfo,
    UINT FAR* pnArgErr,
    LPOLESTR pszName,
    LPCTSTR pszFmt,
    ...)
{
    va_list argList;
    va_start(argList, pszFmt);
    DISPID dispid;
    HRESULT hr;
    VARIANTARG* pvarg = NULL;

    if (pdisp == NULL)
        return ResultFromScode(E_INVALIDARG);

    // Get DISPID of property/method
    hr = pdisp->GetIDsOfNames(IID_NULL, &pszName, 1, LOCALE_USER_DEFAULT,
&dispid);
    if(FAILED(hr))
        return hr;

    DISPPARAMS dispparams;
    _fmemset(&dispparams, 0, sizeof dispparams);

    // determine number of arguments
    if (pszFmt != NULL)
        CountArgsInFormat(pszFmt, &dispparams.cArgs);

    // Property puts have a named argument that represents the value that
the property is
    // being assigned.
    DISPID dispidNamed = DISPID_PROPERTYPUT;
    if (wFlags & DISPATCH_PROPERTYPUT)
    {
        if (dispparams.cArgs == 0)
            return ResultFromScode(E_INVALIDARG);
        dispparams.cNamedArgs = 1;
        dispparams.rgdispidNamedArgs = &dispidNamed;
    }

    if (dispparams.cArgs != 0)
    {
        // allocate memory for all VARIANTARG parameters
        pvarg = new VARIANTARG[dispparams.cArgs];
        if(pvarg == NULL)
            return ResultFromScode(E_OUTOFMEMORY);
        dispparams.rgvarg = pvarg;
        _fmemset(pvarg, 0, sizeof(VARIANTARG) * dispparams.cArgs);

        // get ready to walk vararg list
        LPCTSTR psz = pszFmt;
        pvarg += dispparams.cArgs - 1;   // params go in opposite order

        while (psz = GetNextVarType(psz, &pvarg->vt))
        {
            if (pvarg < dispparams.rgvarg)
            {
                hr = ResultFromScode(E_INVALIDARG);
                goto cleanup;
            }
            switch (pvarg->vt)
            {
            case VT_I2:
                V_I2(pvarg) = va_arg(argList, short);
                break;
            case VT_I4:
                V_I4(pvarg) = va_arg(argList, long);
                break;
            case VT_R4:
                V_R4(pvarg) = va_arg(argList, float);
                break;
            case VT_DATE:
            case VT_R8:
                V_R8(pvarg) = va_arg(argList, double);
                break;
            case VT_CY:
                V_CY(pvarg) = va_arg(argList, CY);
                break;
            case VT_BSTR:
                V_BSTR(pvarg) = SysAllocString(va_arg(argList, OLECHAR
FAR*));
                if (pvarg->bstrVal == NULL)
                {
                    hr = ResultFromScode(E_OUTOFMEMORY);
                    pvarg->vt = VT_EMPTY;
                    goto cleanup;
                }
                break;
            case VT_DISPATCH:
                V_DISPATCH(pvarg) = va_arg(argList, LPDISPATCH);
                break;
            case VT_ERROR:
                V_ERROR(pvarg) = va_arg(argList, SCODE);
                break;
            case VT_BOOL:
                V_BOOL(pvarg) = va_arg(argList, BOOL) ? -1 : 0;
                break;
            case VT_VARIANT:
                *pvarg = va_arg(argList, VARIANTARG);
                break;
            case VT_UNKNOWN:
                V_UNKNOWN(pvarg) = va_arg(argList, LPUNKNOWN);
                break;

            case VT_I2|VT_BYREF:
                V_I2REF(pvarg) = va_arg(argList, short FAR*);
                break;
            case VT_I4|VT_BYREF:
                V_I4REF(pvarg) = va_arg(argList, long FAR*);
                break;
            case VT_R4|VT_BYREF:
                V_R4REF(pvarg) = va_arg(argList, float FAR*);
                break;
            case VT_R8|VT_BYREF:
                V_R8REF(pvarg) = va_arg(argList, double FAR*);
                break;
            case VT_DATE|VT_BYREF:
                V_DATEREF(pvarg) = va_arg(argList, DATE FAR*);
                break;
            case VT_CY|VT_BYREF:
                V_CYREF(pvarg) = va_arg(argList, CY FAR*);
                break;
            case VT_BSTR|VT_BYREF:
                V_BSTRREF(pvarg) = va_arg(argList, BSTR FAR*);
                break;
            case VT_DISPATCH|VT_BYREF:
                V_DISPATCHREF(pvarg) = va_arg(argList, LPDISPATCH FAR*);
                break;
            case VT_ERROR|VT_BYREF:
                V_ERRORREF(pvarg) = va_arg(argList, SCODE FAR*);
                break;
            case VT_BOOL|VT_BYREF:
                {
                    BOOL FAR* pbool = va_arg(argList, BOOL FAR*);
                    *pbool = 0;
                    V_BOOLREF(pvarg) = (VARIANT_BOOL FAR*)pbool;
                }
                break;
            case VT_VARIANT|VT_BYREF:
                V_VARIANTREF(pvarg) = va_arg(argList, VARIANTARG FAR*);
                break;
            case VT_UNKNOWN|VT_BYREF:
                V_UNKNOWNREF(pvarg) = va_arg(argList, LPUNKNOWN FAR*);
                break;

            default:
                {
                    hr = ResultFromScode(E_INVALIDARG);
                    goto cleanup;
                }
                break;
            }

            --pvarg; // get ready to fill next argument
        } //while
    } //if

    // Initialize return variant, in case caller forgot. Caller can pass
NULL if return
    // value is not expected.
    if (pvRet)
        VariantInit(pvRet);
    // make the call
    hr = pdisp->Invoke(dispid, IID_NULL, LOCALE_USER_DEFAULT, wFlags,
        &dispparams, pvRet, pexcepinfo, pnArgErr);

cleanup:
    // cleanup any arguments that need cleanup
    if (dispparams.cArgs != 0)
    {
        VARIANTARG FAR* pvarg = dispparams.rgvarg;
        UINT cArgs = dispparams.cArgs;

        while (cArgs--)
        {
            switch (pvarg->vt)
            {
            case VT_BSTR:
                VariantClear(pvarg);
                break;
            }
            ++pvarg;
        }
    }
    delete dispparams.rgvarg;
    va_end(argList);
    return hr;
}

HRESULT CountArgsInFormat(LPCTSTR pszFmt, UINT FAR *pn)
{
    *pn = 0;

    if(pszFmt == NULL)
      return NOERROR;

    while (*pszFmt)
    {
       if (*pszFmt == '&')
           pszFmt++;

       switch(*pszFmt)
       {
           case 'b':
           case 'i':
           case 'I':
           case 'r':
           case 'R':
           case 'c':
           case 's':
           case 'e':
           case 'd':
           case 'v':
           case 'D':
           case 'U':
               ++*pn;
               pszFmt++;
               break;
           case '\0':
           default:
               return ResultFromScode(E_INVALIDARG);
        }
    }
    return NOERROR;
}


LPCTSTR GetNextVarType(LPCTSTR pszFmt, VARTYPE FAR* pvt)
{
    *pvt = 0;
    if (*pszFmt == '&')
    {
         *pvt = VT_BYREF;
         pszFmt++;
         if (!*pszFmt)
             return NULL;
    }
    switch(*pszFmt)
    {
        case 'b':
            *pvt |= VT_BOOL;
            break;
        case 'i':
            *pvt |= VT_I2;
            break;
        case 'I':
            *pvt |= VT_I4;
            break;
        case 'r':
            *pvt |= VT_R4;
            break;
        case 'R':
            *pvt |= VT_R8;
            break;
        case 'c':
            *pvt |= VT_CY;
            break;
        case 's':
            *pvt |= VT_BSTR;
            break;
        case 'e':
            *pvt |= VT_ERROR;
            break;
        case 'd':
            *pvt |= VT_DATE;
            break;
        case 'v':
            *pvt |= VT_VARIANT;
            break;
        case 'U':
            *pvt |= VT_UNKNOWN;
            break;
        case 'D':
            *pvt |= VT_DISPATCH;
            break;
        case '\0':
             return NULL;     // End of Format string
        default:
            return NULL;
    }
    return ++pszFmt;
}
```

BROWSEH

SAMPLE: BROWSEH: InProc OLE Automation Server that browses a type library

BROWSEH is a inproc OLE Automation Server that browses a type library. It shows how to create an inproc Automation server and how to use the ITypeLib and ITypeInfo interfaces to browse a type library.

BROWSEH exposes a number of automation objects that correspond to components of a type library. This allows an automation controller like BROWSE (shipped with the OLE 2.02 SDK) to access the properties and methods of these objects to browse the type library instead of directly using ITypeLib and ITypeInfo. So BROWSEH is a type library browse helper.

BROWSEH does not support vtable binding. Controllers must use IDispatch to control this server.

ProgID: BrowseHelper.Browser

See main.cpp for code to create an inproc automation server. See other source files for code that uses ITypeLib and ITypeInfo to browse various components of a type library.

To compile: -----------

Requires OLE 2.02 or later. Use the external makefile called makefile to compile. In Win16, run the WXSRVER.EXE from \OLE2\BIN before running the makefile. The makefile invokes mktyplib.exe that reads mydisp.odl and browseh.odl and creates the type libraries, mydisp.tlb and browseh.tlb. It then compliles the source files.

To run: -------

Change browseh.reg to provide the full path of browseh.dll and browseh.tlb. Register browseh.reg in the registration database by double-clicking it. Run the Automation controller BROWSE that uses BROWSEH to browse a type library.

Files: ------ MYDISP.ODL Contains the description of a base class that implements IDispatch. All automation objects in this server derive from this class. This the implementation of IDispatch in the base class to be inherited in the derived classes.
BROWSEH.ODL Object description language that describes the objects that BROWSEH exposes.
TLB.H Header file generated by mktyplib.exe MAKEFILE Makefile for project. MAIN.CPP Code to create an inproc server.

Other files implement the BROWSEH automation server. For example coclass.cpp demonstates how to use ITypeInfo and ITypeLib to browse a coclass in a type library.
==

MAKEFILE (BROWSEH OLE Sample)

```
####
#makefile - makefile for browseh.dll
#
#       Copyright (C) 1994, Microsoft Corporation
#
#Purpose:
#  Builds the Inproc OLE 2.0 Automation object, browseh.dll.
#
#
#  Usage: NMAKE                 ; build with defaults
#     or: NMAKE option          ; build with the given option(s)
#     or: NMAKE clean           ; erase all compiled files
#
#     option: dev = [win16 | win32]    ; dev=win32 is the default
#             DEBUG=[0 | 1]            ; DEBUG=1 is the default
#             HOST=[DOS | NT | WIN95]  ; HOST=DOS (for win16)
#                                      ; HOST=NT (for win32 on NT)
#                                      ; HOST=WIN95 (for win32 on Win95)
#
#Notes:
#  This makefile assumes that the PATH, INCLUDE and LIB environment
#  variables are setup properly.
#
############################################################################
##


##########################################################################
#
# Default Settings
#

CPU = i386

!if "$(dev)" == ""
dev = win32
HOST = NT
!endif

!if !("$(dev)" == "win16" || "$(dev)" == "win32")
!error Invalid dev option, choose from [win16 | win32]
!endif

!if "$(dev)" == "win16"
TARGET  = WIN16
!if "$(HOST)" == ""
HOST  = DOS
!endif
!endif

!if "$(dev)" == "win32"
TARGET  = WIN32
```

```
!if "$(HOST)" == ""
HOST  = NT
!endif
!endif

!ifdef NODEBUG
DEBUG = 0
!endif

!if "$(DEBUG)" == "0"
NODEBUG = 1
!endif

!if "$(DEBUG)" == ""
DEBUG = 1
!endif


############################################################################
#
# WIN16 Settings
#
!if "$(TARGET)" == "WIN16"

CC   = cl
LINK = link
!if "$(HOST)" == "DOS"
WX   = wx /w
!else
WX   =
!endif

RCFLAGS = -dWIN16
CFLAGS = -c -W3 -AM -GD -DWIN16
LINKFLAGS = /NOD /BATCH /ONERROR:NOEXE

LIBS = libw.lib mdllcew.lib

!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS) -Od -Zi -D_DEBUG $(CL)
LINKFLAGS = $(LINKFLAGS) /COD
!else
CFLAGS = $(CFLAGS) -Ox $(CL)
LINKFLAGS = $(LINKFLAGS) /FAR /PACKC
!endif
!endif


############################################################################
#
# WIN32 Settings
#
!if "$(TARGET)" == "WIN32"
```

```
WX =

!include <ntwin32.mak>

CC = $(cc)
CFLAGS = $(cflags) $(cvarsmt) -DINC_OLE2 $(cdebug)

!if "$(HOST)" == "NT"
CFLAGS = $(CFLAGS) -DUNICODE
!endif

!ifndef NODEBUG
CFLAGS = $(CFLAGS) -D_DEBUG
!endif

LINK = $(link)
LINKFLAGS = $(linkdebug) $(guilflags) -dll -entry:_DllMainCRTStartup$
(DLLENTRY)\
      -export:DllGetClassObject -export:DllCanUnloadNow
RCFLAGS = -DWIN32

!endif

##########################################################################
#
# Build rules
#

.cpp.obj:
    @echo Compiling $<...
    $(CC) $<

.c.obj:
    @echo Compiling $<...
    $(CC) $<


##########################################################################
#
# Application Settings
#

APPS = browseh


!if "$(TARGET)" == "WIN16"
LIBS = $(LIBS) ole2.lib compobj.lib ole2disp.lib typelib.lib
!endif
!if "$(TARGET)" == "WIN32"
LIBS = $(ole2libsmt)
!endif

OBJS = alias.obj main.obj mydisp.obj browseh.obj browsecf.obj collect.obj \
       enumvar.obj typelib.obj typeinfo.obj intface.obj dispface.obj
module.obj \
       coclass.obj function.obj property.obj param.obj enum.obj constant.obj
\
       type.obj union.obj struct.obj



###########################################################################
#
# Default Goal
#

goal : setflags $(APPS).dll

setflags :
    set CL=$(CFLAGS)


###########################################################################
#
# Clean (erase) generated files
#
clean :
    if exist *.obj       del *.obj
    if exist $(APPS).dll del $(APPS).dll
    if exist $(APPS).tlb del $(APPS).tlb
    if exist $(APPS).map del $(APPS).map
    if exist $(APPS).res del $(APPS).res
    if exist tlb.h       del tlb.h
    if exist *.log       del *.log
    if exist *.pdb       del *.pdb


###########################################################################
#
# Application Build (WIN16 Specific)
#

!if "$(TARGET)" == "WIN16"
$(APPS).dll : $(OBJS) $(APPS).def $(APPS).res
    link @<<
$(LINKFLAGS)+
$(OBJS),
$@,,
$(LIBS),
$(APPS).def
<<
    rc -k -t $(APPS).res $@
!endif


###########################################################################
#
# Application Build (WIN32 Specific)
```

```
#
!if "$(TARGET)" == "WIN32"
$(APPS).dll : $(OBJS) $(APPS).def $(APPS).res
      $(LINK) @<<
        $(LINKFLAGS)
        -out:$@
        -map:$*.map
        $(OBJS)
        $(APPS).res
        $(LIBS)
<<
!endif


##########################################################################
#
# Application Build (Common)
#

$(APPS).res : $(APPS).rc
    rc $(RCFLAGS) -r -fo$@ $?


##########################################################################
#
# Dependencies
#

mydisp.tlb :: mydisp.odl
     if exist mydisp.tlb  del mydisp.tlb
     $(WX) mktyplib /D$(TARGET) /o mydisp.log /tlb mydisp.tlb mydisp.odl
     type mydisp.log

tlb.h : browseh.odl mydisp.tlb
     if exist tlb.h  del tlb.h
     if exist browseh.tlb  del browseh.tlb
     $(WX) mktyplib /D$(TARGET) /h tlb.h /o browseh.log /tlb browseh.tlb
browseh.odl
     type browseh.log

main.obj : main.cpp browseh.h mydisp.h tlb.h
     $(CC) main.cpp
mydisp.obj : mydisp.cpp browseh.h mydisp.h tlb.h
     $(CC) mydisp.cpp
browseh.obj : browseh.cpp browseh.h mydisp.h tlb.h
     $(CC) browseh.cpp
browsecf.obj : browsecf.cpp browseh.h mydisp.h tlb.h
     $(CC) browsecf.cpp
typelib.obj : typelib.cpp browseh.h mydisp.h tlb.h
     $(CC) typelib.cpp
typeinfo.obj : typeinfo.cpp browseh.h mydisp.h tlb.h
     $(CC) typeinfo.cpp
intface.obj : intface.cpp browseh.h mydisp.h tlb.h
     $(CC) intface.cpp
dispface.obj : dispface.cpp browseh.h mydisp.h tlb.h
```

```
     $(CC) dispface.cpp
module.obj : module.cpp browseh.h mydisp.h tlb.h
     $(CC) module.cpp
coclass.obj : coclass.cpp browseh.h mydisp.h tlb.h
     $(CC) coclass.cpp
function.obj : function.cpp browseh.h mydisp.h tlb.h
     $(CC) function.cpp

property.obj : property.cpp browseh.h mydisp.h tlb.h
     $(CC) property.cpp
param.obj : param.cpp browseh.h mydisp.h tlb.h
     $(CC) param.cpp
collect.obj : collect.cpp browseh.h mydisp.h tlb.h
     $(CC) collect.cpp
enumvar.obj : enumvar.cpp browseh.h mydisp.h tlb.h
     $(CC) enumvar.cpp
enum.obj : enum.cpp browseh.h mydisp.h tlb.h
     $(CC) enum.cpp
constant.obj : constant.cpp browseh.h mydisp.h tlb.h
     $(CC) constant.cpp
alias.obj : alias.cpp browseh.h mydisp.h tlb.h
     $(CC) alias.cpp
struct.obj : struct.cpp browseh.h mydisp.h tlb.h
     $(CC) struct.cpp
union.obj : union.cpp browseh.h mydisp.h tlb.h
     $(CC) union.cpp
type.obj : type.cpp browseh.h mydisp.h tlb.h
     $(CC) type.cpp
```

BROWSEH.H (BROWSEH OLE Sample)

```
#ifdef WIN32

#ifdef UNICODE
    #define FROM_OLE_STRING(str) str
    #define TO_OLE_STRING(str) str
#else
    #define FROM_OLE_STRING(str) ConvertToAnsi(str)
    char FAR* ConvertToAnsi(OLECHAR FAR* szW);
    #define TO_OLE_STRING(str) ConvertToUnicode(str)
    OLECHAR FAR* ConvertToUnicode(char FAR* szA);
    // Maximum length of string that can be converted between Ansi & Unicode
    #define STRCONVERT_MAXLEN 300
#endif

#else  // WIN16
  #define TCHAR char
  #define TEXT(sz) sz
  #define FROM_OLE_STRING(str) str
  #define TO_OLE_STRING(str) str
  #define LPTSTR LPSTR
#endif

#include "mydisp.h"
#include "tlb.h"

// MAX len of string table entries
#define STR_LEN   200

// String table constants
#define IDS_SERVERNAME              1
// These exception strings IDs that is will used in EXCEPINFO.wCode
#define IDS_Unexpected              1001
#define IDS_OutOfMemory             1002
#define IDS_InvalidIndex            1003
#define IDS_CollectionFull          1004
#define IDS_CannotFindTypeLibrary  1005
#define IDS_TypeLibraryCreationFailed 1006
#define IDS_WrongType               1007
#define IDS_InvalidProgid           1008
#define IDS_CouldNotCreateObject    1009
#define IDS_ObjectDoesNotSupportAutomation   1010
#define IDS_ObjectDoesNotProvideTypeInfo  1011

// Function prototypes
BOOL InitDLL (HINSTANCE hInstance);
extern "C" STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid, LPVOID FAR
*ppv);
extern "C" STDAPI DllCanUnloadNow(void);

//Forward declarations.
class CArrayBound;
class FAR CBrowseHelperCF : public IClassFactory
{
public:
    // IUnknown methods
    STDMETHOD(QueryInterface)(REFIID riid, LPVOID FAR* ppvObj);
    STDMETHOD_(ULONG, AddRef)(void);
    STDMETHOD_(ULONG, Release)(void);

    // IClassFactory methods
    STDMETHOD(CreateInstance)(IUnknown FAR* punkOuter, REFIID riid, void
FAR* FAR* ppv);
    STDMETHOD(LockServer)(BOOL fLock);

    CBrowseHelperCF();

private:
    ULONG m_cRef;                   // Reference count
};


class FAR CBrowseHelper : public IBrowseHelper
{
public:
    // IBrowseHelper automation exposed methods
    STDMETHOD_(ITypeLibrary FAR*, BrowseTypeLibrary)(BSTR bstrPath);

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(CBrowseHelper FAR* FAR* ppBrowseHelper); //
Creates and intializes Applicaton object
    CBrowseHelper();
    ~CBrowseHelper();
};

class FAR CTypeLibrary : public ITypeLibrary
{
public:
    // ITypeLibrary automation exposed properties & methods
    STDMETHOD_(BSTR, get_Name)();
    STDMETHOD_(BSTR, get_Documentation)();
    STDMETHOD_(long, get_HelpContext)();
    STDMETHOD_(BSTR, get_HelpFile)();
    STDMETHOD_(long, get_LocaleID)();
    STDMETHOD_(int, get_MajorVersion)();
    STDMETHOD_(int, get_MinorVersion)();
    STDMETHOD_(ICollection FAR*, get_TypeInfos)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(LPTYPELIB ptlib, CTypeLibrary FAR* FAR*
ppTypeLibrary); // Creates and intializes TypeLibrary object
    CTypeLibrary();
    ~CTypeLibrary();

private:
    BSTR m_bstrName;               // Type Library name
    BSTR m_bstrDocumentation;      // Documentation
    unsigned long m_ulHelpContext; // HelpContext
    BSTR m_bstrHelpFile;           // HelpFile
    GUID m_guid;                   // GUID
    LCID m_lcid;                   // Locale ID
    unsigned short m_wMajorVer;    // Major version
    unsigned short m_wMinorVer;    // Minor version
    LPTYPELIB m_ptlib;             // ITypeLib* of type library.
    LPDISPATCH m_pdispTypeInfos;   // Collection of typeinfos contained by
this typelib
};

// CTypeInfo is the base class for all the TypeInfos (CInterface,
CDispinterface,
// CModule, CCoClass, CEnum, CStruct, CUnion, CAlias)
class FAR CTypeInfo : public ITypeInformation
{
public:
    // ITypeInfo automation exposed properties & methods
    STDMETHOD_(BSTR, get_Name)();
    STDMETHOD_(BSTR, get_Documentation)();
    STDMETHOD_(long, get_HelpContext)();
    STDMETHOD_(BSTR, get_HelpFile)();
    STDMETHOD_(TYPEKIND, get_TypeInfoKind)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    STDMETHOD(_InitTypeInfo)(LPTYPEINFO ptinfo); // Called by derived class
to intialize this base class
    static HRESULT Create(LPTYPEINFO ptinfo, CTypeInfo FAR* FAR*
ppTypeInfo); // Creates and intializes TypeInfo object
    CTypeInfo();
    ~CTypeInfo();

private:
    BSTR m_bstrName;               // TypeInfo name

    BSTR m_bstrDocumentation;      // Documentation
    unsigned long m_ulHelpContext; // Help context
    BSTR m_bstrHelpFile;           // Help file
    TYPEKIND m_typekind;           // Type of TypeInfo. See TYPEKIND
enumeration.
    GUID m_guid;                   // GUID
};

class FAR CInterface : public CTypeInfo
{
public:
    // IInterface automation exposed properties & methods
    STDMETHOD_(ICollection FAR*, get_Functions)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(LPTYPEINFO ptinfo, CInterface FAR* FAR*
ppInterface); // Creates and intializes Interface object
    CInterface();
    ~CInterface();
```

```
private:
    LPTYPEINFO m_ptinfo;             // ITypeInfo* of interface typeinfo.
    LPDISPATCH m_pdispFunctions;     // Collection of functions in
interface.
};

class FAR CDispinterface : public CTypeInfo
{
public:
    // IDispinterface automation exposed properties & methods
    STDMETHOD_(ICollection FAR*, get_Properties)();
    STDMETHOD_(ICollection FAR*, get_Methods)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(LPTYPEINFO ptinfo, CDispinterface FAR* FAR*
ppDispinterface); // Creates and intializes Dispinterface object
    CDispinterface();
    ~CDispinterface();

private:
    LPTYPEINFO m_ptinfo;              // ITypeInfo* of dispinterface
typeinfo.
    LPDISPATCH m_pdispProperties;     // Collection of properties in
dispinterface
    LPDISPATCH m_pdispMethods;        // Collection of methods in
dispinterface
};

class FAR CModule : public CTypeInfo
{
public:
    // IModule automation exposed properties & methods
    STDMETHOD_(ICollection FAR*, get_Functions)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(LPTYPEINFO ptinfo, CModule FAR* FAR* ppInterface);
// Creates and intializes Module object
    CModule();
    ~CModule();

private:
    LPTYPEINFO m_ptinfo;              // ITypeInfo* of module typeinfo.
    LPDISPATCH m_pdispFunctions;      // Collection of functions in module
};

class FAR CCoClass : public CTypeInfo
{
public:
    // ICoClass automation exposed properties & methods
    STDMETHOD_(ICollection FAR*, get_Interfaces)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(LPTYPEINFO ptinfo, CCoClass FAR* FAR*
ppCoClass); // Creates and intializes CoClass object
    CCoClass();
    ~CCoClass();

private:
    LPTYPEINFO m_ptinfo;              // ITypeInfo* of coclass typeinfo.
    LPDISPATCH m_pdispInterfaces;     // Collection of
interfaces/dispinterfaces in coclass
};

class FAR CEnum : public CTypeInfo
{
public:
    // IEnum automation exposed properties & methods
    STDMETHOD_(ICollection FAR*, get_Elements)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(LPTYPEINFO ptinfo, CEnum FAR* FAR* ppEnum); //
Creates and intializes Enum object
    CEnum();
    ~CEnum();

private:
    LPTYPEINFO m_ptinfo;             // ITypeInfo* of enum typeinfo.
    LPDISPATCH m_pdispElements;      // Collection of elements in enum.
};

class FAR CStruct : public CTypeInfo
{
public:
     // IStruct automation exposed properties & methods
    STDMETHOD_(ICollection FAR*, get_Members)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(LPTYPEINFO ptinfo, CStruct FAR* FAR* ppStruct); //
Creates and intializes Struct object
    CStruct();
    ~CStruct();

private:
    LPTYPEINFO m_ptinfo;            // ITypeInfo* of struct typeinfo.
    LPDISPATCH m_pdispMembers;      // Collection of members in struct.
};

class FAR CUnion : public CTypeInfo
{
public:
    // IUnion automation exposed properties & methods
    STDMETHOD_(ICollection FAR*, get_Members)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(LPTYPEINFO ptinfo, CUnion FAR* FAR* ppUnion); //
Creates and intializes Struct object
    CUnion();
    ~CUnion();

private:
    LPTYPEINFO m_ptinfo;            // ITypeInfo* of union typeinfo.
    LPDISPATCH m_pdispMembers;      // Collection of members in union.
};


class FAR CAlias : public CTypeInfo
{
public:
    // IAlias automation exposed properties & methods
    STDMETHOD_(ITypeDesc FAR*, get_BaseType)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(LPTYPEINFO ptinfo, CAlias FAR* FAR* ppAlias); //
Creates and intializes Alias object
    CAlias();
    ~CAlias();

private:
    LPTYPEINFO m_ptinfo;              // ITypeInfo* of alias typeinfo.
    LPDISPATCH m_pdispTypeDescBase;  // IDispatch of ITypeDesc which
describes the base type of this alias.
};

// CFunction represents all functions (including methods).
class FAR CFunction : public IFunction
{
public:
    // IFunction automation exposed properties & methods
    STDMETHOD_(BSTR, get_Name)();
    STDMETHOD_(BSTR, get_Documentation)();
    STDMETHOD_(long, get_HelpContext)();
    STDMETHOD_(BSTR, get_HelpFile)();
    STDMETHOD_(ITypeDesc FAR*, get_ReturnType)();
    STDMETHOD_(ICollection FAR*, get_Parameters)();
    STDMETHOD_(MEMBERID, get_Memberid)();
    STDMETHOD_(CALLCONV, get_CallConvention)();
    STDMETHOD_(FUNCKIND, get_FuncKind)();
    STDMETHOD_(INVOKEKIND, get_InvocationKind)();
    STDMETHOD_(OBJTYPE, get_Kind)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(LPTYPEINFO ptinfo, unsigned short nIndex,
CFunction FAR* FAR* ppFunction); // Creates and intializes Function object
    CFunction();
    ~CFunction();

private:
    BSTR m_bstrName;                      // Function name.
    BSTR m_bstrDocumentation;             // Documentation.
    unsigned long m_ulHelpContext;        // Help context.
    BSTR m_bstrHelpFile;                  // Help file.
    LPDISPATCH m_pdispTypeDesc;           // IDispatch of ITypeDesc that
described return type.
```

```
    LPDISPATCH m_pdispParameters;        // Collection of parameters
function.
    LPFUNCDESC m_pfuncdesc;              // FUNCDESC of function.
    LPTYPEINFO m_ptinfoFunction;         // TypeInfo of which this function
is an element.
};

// CProperty represents dispinterface properties and structure & union
members.
class FAR CProperty : public IProperty
{
public:
    // IProperty automation exposed properties & methods
    STDMETHOD_(BSTR, get_Name)();
    STDMETHOD_(BSTR, get_Documentation)();
    STDMETHOD_(long, get_HelpContext)();
    STDMETHOD_(BSTR, get_HelpFile)();
    STDMETHOD_(ITypeDesc FAR*, get_Type)();
    STDMETHOD_(MEMBERID, get_Memberid)();
    STDMETHOD_(OBJTYPE, get_Kind)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(LPTYPEINFO ptinfo, LPVARDESC pvardesc, CProperty
FAR* FAR* ppProperty); // Creates and intializes Property object
    CProperty();
    ~CProperty();

private:
    BSTR m_bstrName;                     // Property name.
    BSTR m_bstrDocumentation;            // Documentation.
    unsigned long m_ulHelpContext;       // Help Context.
    BSTR m_bstrHelpFile;                 // Helpfile.
    MEMBERID m_memid;                    // MEMBERID of property.
    LPDISPATCH m_pdispTypeDesc;          // ITypeDesc that describes type of
property.
};

class FAR CConstant : public IConstant
{
public:
    // IConstant automation exposed properties & methods
    STDMETHOD_(BSTR, get_Name)();
    STDMETHOD_(BSTR, get_Documentation)();
    STDMETHOD_(long, get_HelpContext)();
    STDMETHOD_(BSTR, get_HelpFile)();
    STDMETHOD_(ITypeDesc FAR*, get_Type)();
    STDMETHOD_(VARIANT, get_Value)();
    STDMETHOD_(MEMBERID, get_Memberid)();
    STDMETHOD_(OBJTYPE, get_Kind)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(LPTYPEINFO ptinfo, LPVARDESC pvardesc, CConstant
FAR* FAR* ppConstant); // Creates and intializes Constant object
    CConstant();
    ~CConstant();
```

```
private:
    BSTR m_bstrName;                          // Constant name
    BSTR m_bstrDocumentation;                 // Documentation
    unsigned long m_ulHelpContext;            // Help context
    BSTR m_bstrHelpFile;                      // Helpfile
    LPDISPATCH m_pdispTypeDesc;               // ITypeDesc that describes type
of constant.
    MEMBERID m_memid;                         // MEMBERID.
    VARIANT m_vValue;                         // Constant value.
};

class FAR CParameter : public IParameter
{
public:
    // IParameter automation exposed properties & methods
    STDMETHOD_(BSTR, get_Name)();
    STDMETHOD_(ITypeDesc FAR*, get_Type)();
    STDMETHOD_(int, get_IDLFlags)();
    STDMETHOD_(OBJTYPE, get_Kind)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(LPTYPEINFO ptinfo, BSTR bstrName, TYPEDESC FAR*
ptypedesc,
         IDLDESC FAR* pidldesc, CParameter FAR* FAR* ppParameter);  //
Creates and intializes Parameter object
    CParameter();
    ~CParameter();

private:
    BSTR m_bstrName;                         // Parameter name.
    unsigned short m_wIDLFlags;              // specifies in/out/in-out/none
    LPDISPATCH m_pdispTypeDesc;              // ITypeDesc* that describes type
of parameter.
};

// Describes a type
class FAR CTypeDesc : public ITypeDesc
{
public:
    // ITypeDesc automation exposed properties & methods
    STDMETHOD_(short, get_Type)();
    STDMETHOD_(ITypeInformation FAR*, get_UserDefinedDesc)();
    STDMETHOD_(IArrayDesc FAR*, get_ArrayDesc)();
    STDMETHOD_(ITypeDesc FAR*, get_PointerDesc)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(LPTYPEINFO ptinfo, TYPEDESC FAR* ptypedesc,
CTypeDesc FAR* FAR* ppTypeDesc); // Creates and intializes TypeDesc object
    CTypeDesc();
    ~CTypeDesc();

private:
    VARTYPE m_vartype;                     // Type
    LPDISPATCH m_pdispUserDefinedDesc;  // If m_vartype==VT_USERDEFINED,
contains ITypeInformation* of user-defined type.,
    LPDISPATCH m_pdispArrayDesc;        // if m_vartype==VT_CARRAY, contains
IArrayDesc*
    LPDISPATCH m_pdispPointerDesc;      // if m_vartype==VT_PTR contains
ITypeDesc* of type pointed to.
};

// Describes C-Style array
class FAR CArrayDesc : public IArrayDesc
{
public:
    // IArrayDesc automation exposed properties & methods
    STDMETHOD_(ITypeDesc FAR*, get_ElementType)();
    STDMETHOD_(ICollection FAR*, get_ArrayBounds)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(LPTYPEINFO ptinfo, ARRAYDESC FAR* parraydesc,
CArrayDesc FAR* FAR* ppArrayDesc); // Creates and intializes ArrayDesc
object
    CArrayDesc();
    ~CArrayDesc();

private:
    LPDISPATCH m_pdispTypeDescElement;     // ITypeDesc that descibed type
of array element.
    LPDISPATCH m_pdispArrayBounds;         // IArrayBound that describes
array bounds.
};

class FAR CArrayBound : public IArrayBound
{
public:
    // IArrayBound automation exposed properties & methods
    STDMETHOD_(long, get_ElementsCount)();
    STDMETHOD_(long, get_LowerBound)();

    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(unsigned long cElements, long lLBound, CArrayBound
FAR* FAR* ppArrayBound); // Creates and intializes ArrayBound object

private:
    unsigned long m_cElements;             // Count of elements in array.
    long m_lLBound;                        // Lower bound of array.
};

// Generic collection class that holds all collections.
class FAR CCollection : public ICollection
{
public:
    // ICollection automation exposed properties & methods
    STDMETHOD_(long, get_Count)();
    STDMETHOD_(LPDISPATCH, get_Item)(long lIndex);
    STDMETHOD_(LPUNKNOWN, get__NewEnum)();
    // CCollection methods
    STDMETHOD_(void, Add)(LPDISPATCH pdispItem);
    STDMETHOD_(REFCLSID, GetInterfaceID)();
    static HRESULT Create(ULONG lMaxSize, long lLBound, CCollection FAR*
FAR* ppCollection); // Creates and intializes Collection object
    CCollection();
    ~CCollection();

private:
    SAFEARRAY FAR *m_psa;          // Safe array that holds Collection
collection items.
    ULONG m_cElements;             // Number of items in Collection
collection.
    ULONG m_cMax;                  // Maximum number of items Collection
collection can hold.
    long m_lLBound;                // Lower bound of index of Collection
collection.
};

class FAR CEnumVariant : public IEnumVARIANT
{
public:
    // IUnknown methods
    STDMETHOD(QueryInterface)(REFIID riid, LPVOID FAR* ppvObj);
    STDMETHOD_(ULONG, AddRef)();
    STDMETHOD_(ULONG, Release)();

    // IEnumVARIANT methods
    STDMETHOD(Next)(ULONG cElements, VARIANT FAR* pvar, ULONG FAR*
pcElementFetched);
    STDMETHOD(Skip)(ULONG cElements);
    STDMETHOD(Reset)();
    STDMETHOD(Clone)(IEnumVARIANT FAR* FAR* ppenum);

    static HRESULT Create(SAFEARRAY FAR*, ULONG, CEnumVariant FAR* FAR*); //
Creates and intializes Enumerator
    CEnumVariant();
    ~CEnumVariant();

private:
    ULONG m_cRef;           // Reference count
    ULONG m_cElements;      // Number of elements in enumerator.
    long m_lLBound;         // Lower bound of index.
    long m_lCurrent;        // Current index.
    SAFEARRAY FAR* m_psa;   // Safe array that holds elements.
};
```

MYDISP.H   (BROWSEH OLE Sample)

```
class FAR CMyDispatch : public IDispatch
{
public:
    /* IUnknown methods */
    STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj);
    STDMETHOD_(ULONG, AddRef)(THIS);
    STDMETHOD_(ULONG, Release)(THIS);

    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(THIS_ UINT FAR* pctinfo);

    STDMETHOD(GetTypeInfo)(
      THIS_
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo);

    STDMETHOD(GetIDsOfNames)(
      THIS_
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid);

    STDMETHOD(Invoke)(
      THIS_
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr);

      CMyDispatch();
      virtual ~CMyDispatch();

    /* CMyDispatch methods */
    STDMETHOD(LoadTypeInfo)(REFCLSID clsid);
    STDMETHOD_(void, RaiseException)(int nID);
    STDMETHOD_(REFCLSID, GetInterfaceID)() PURE;

private:
    LPTYPEINFO m_ptinfo;         // Type information of interface.
    ULONG m_cRef;                // Reference count.
    EXCEPINFO m_excepinfo;       // Information to raise an exception on
error.
    BOOL m_bRaiseException;      // Properties and methods use this to
signal an exception to be raised.
#ifdef _DEBUG
public:
    TCHAR m_szClassName[100];
#endif
};
```

TLB.H (BROWSEH OLE Sample)

```
/* This header file machine-generated by mktyplib.exe */
/* Interface to type library: BrowseHelper */

#ifndef _BrowseHelper_H_
#define _BrowseHelper_H_

DEFINE_GUID(LIBID_BrowseHelper,0x19FEEEC0,0x4104,0x101B,0xB8,0x26,0x00,0xDD,
0x01,0x10,0x3D,0xE1);

interface IArrayDesc;

typedef enum {
    TYPE_FUNCTION = 0,
    TYPE_PROPERTY = 1,
    TYPE_CONSTANT = 2,
    TYPE_PARAMETER = 3
} OBJTYPE;

DEFINE_GUID(IID_ICollection,0x19FEEECC,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0x0
1,0x10,0x3D,0xE1);

/* Definition of interface: ICollection */
DECLARE_INTERFACE_(ICollection, CMyDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */
#endif

    /* ICollection methods */
    STDMETHOD_(long, get_Count)(THIS) PURE;
    STDMETHOD_(IDispatch *, get_Item)(THIS_ long Index) PURE;
    STDMETHOD_(IUnknown *, get__NewEnum)(THIS) PURE;
};

DEFINE_GUID(IID_ITypeLibrary,0x19FEEEC3,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0x
01,0x10,0x3D,0xE1);

/* Definition of interface: ITypeLibrary */
DECLARE_INTERFACE_(ITypeLibrary, CMyDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */
#endif

    /* ITypeLibrary methods */
    STDMETHOD_(BSTR, get_Name)(THIS) PURE;
    STDMETHOD_(BSTR, get_Documentation)(THIS) PURE;
    STDMETHOD_(long, get_HelpContext)(THIS) PURE;
    STDMETHOD_(BSTR, get_HelpFile)(THIS) PURE;
    STDMETHOD_(long, get_LocaleID)(THIS) PURE;
    STDMETHOD_(int, get_MajorVersion)(THIS) PURE;
    STDMETHOD_(int, get_MinorVersion)(THIS) PURE;
    STDMETHOD_(ICollection FAR*, get_TypeInfos)(THIS) PURE;
};

DEFINE_GUID(IID_ITypeInformation,0x19FEEEC4,0x4104,0x101B,0xB8,0x26,0x00,0xD
D,0x01,0x10,0x3D,0xE1);

/* Definition of interface: ITypeInformation */
DECLARE_INTERFACE_(ITypeInformation, CMyDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */
#endif

    /* ITypeInformation methods */
    STDMETHOD_(BSTR, get_Name)(THIS) PURE;
    STDMETHOD_(BSTR, get_Documentation)(THIS) PURE;
    STDMETHOD_(long, get_HelpContext)(THIS) PURE;
    STDMETHOD_(BSTR, get_HelpFile)(THIS) PURE;
    STDMETHOD_(TYPEKIND, get_TypeInfoKind)(THIS) PURE;
    STDMETHOD(_InitTypeInfo)(THIS_ LPTYPEINFO ptinfo) PURE;
};

DEFINE_GUID(IID_IBrowseHelper,0x19FEEEC2,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0
x01,0x10,0x3D,0xE1);

/* Definition of interface: IBrowseHelper */
DECLARE_INTERFACE_(IBrowseHelper, CMyDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */
#endif

    /* IBrowseHelper methods */
    STDMETHOD_(ITypeLibrary FAR*, BrowseTypeLibrary)(THIS_ BSTR
TypeLibraryPath) PURE;
};

DEFINE_GUID(IID_ITypeDesc,0x19FEEED1,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0x01,
0x10,0x3D,0xE1);

/* Definition of interface: ITypeDesc */
DECLARE_INTERFACE_(ITypeDesc, CMyDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */
```

```
#endif

    /* ITypeDesc methods */
    STDMETHOD_(short, get_Type)(THIS) PURE;
    STDMETHOD_(ITypeInformation FAR*, get_UserDefinedDesc)(THIS) PURE;
    STDMETHOD_(IArrayDesc FAR*, get_ArrayDesc)(THIS) PURE;
    STDMETHOD_(ITypeDesc FAR*, get_PointerDesc)(THIS) PURE;
};

DEFINE_GUID(IID_IArrayBound,0x19FEEED3,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0x0
1,0x10,0x3D,0xE1);

/* Definition of interface: IArrayBound */
DECLARE_INTERFACE_(IArrayBound, CMyDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */
#endif

    /* IArrayBound methods */
    STDMETHOD_(long, get_ElementsCount)(THIS) PURE;
    STDMETHOD_(long, get_LowerBound)(THIS) PURE;
};

DEFINE_GUID(IID_IArrayDesc,0x19FEEED2,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0x01
,0x10,0x3D,0xE1);

/* Definition of interface: IArrayDesc */
DECLARE_INTERFACE_(IArrayDesc, CMyDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */
#endif


    /* IArrayDesc methods */
    STDMETHOD_(ITypeDesc FAR*, get_ElementType)(THIS) PURE;
    STDMETHOD_(ICollection FAR*, get_ArrayBounds)(THIS) PURE;
};

DEFINE_GUID(IID_IInterface,0x19FEEEC5,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0x01
,0x10,0x3D,0xE1);

/* Definition of interface: IInterface */
DECLARE_INTERFACE_(IInterface, ITypeInformation)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */
```

```
    /* ITypeInformation methods */
    STDMETHOD_(BSTR, get_Name)(THIS) PURE;
    STDMETHOD_(BSTR, get_Documentation)(THIS) PURE;
    STDMETHOD_(long, get_HelpContext)(THIS) PURE;
    STDMETHOD_(BSTR, get_HelpFile)(THIS) PURE;
    STDMETHOD_(TYPEKIND, get_TypeInfoKind)(THIS) PURE;
    STDMETHOD(_InitTypeInfo)(THIS_ LPTYPEINFO ptinfo) PURE;
#endif

    /* IInterface methods */
    STDMETHOD_(IDispatch *, get_Functions)(THIS) PURE;
};

DEFINE_GUID(IID_IDispinterface,0x19FEEEC6,0x4104,0x101B,0xB8,0x26,0x00,0xDD,
0x01,0x10,0x3D,0xE1);

/* Definition of interface: IDispinterface */
DECLARE_INTERFACE_(IDispinterface, ITypeInformation)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */

    /* ITypeInformation methods */
    STDMETHOD_(BSTR, get_Name)(THIS) PURE;
    STDMETHOD_(BSTR, get_Documentation)(THIS) PURE;
    STDMETHOD_(long, get_HelpContext)(THIS) PURE;
    STDMETHOD_(BSTR, get_HelpFile)(THIS) PURE;
    STDMETHOD_(TYPEKIND, get_TypeInfoKind)(THIS) PURE;
    STDMETHOD(_InitTypeInfo)(THIS_ LPTYPEINFO ptinfo) PURE;
#endif

    /* IDispinterface methods */
    STDMETHOD_(ICollection FAR*, get_Properties)(THIS) PURE;
    STDMETHOD_(ICollection FAR*, get_Methods)(THIS) PURE;
};

DEFINE_GUID(IID_IModule,0x19FEEEC7,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x
10,0x3D,0xE1);

/* Definition of interface: IModule */
DECLARE_INTERFACE_(IModule, ITypeInformation)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */

    /* ITypeInformation methods */
    STDMETHOD_(BSTR, get_Name)(THIS) PURE;
    STDMETHOD_(BSTR, get_Documentation)(THIS) PURE;
    STDMETHOD_(long, get_HelpContext)(THIS) PURE;
    STDMETHOD_(BSTR, get_HelpFile)(THIS) PURE;
    STDMETHOD_(TYPEKIND, get_TypeInfoKind)(THIS) PURE;
    STDMETHOD(_InitTypeInfo)(THIS_ LPTYPEINFO ptinfo) PURE;
#endif

    /* IModule methods */
    STDMETHOD_(ICollection FAR*, get_Functions)(THIS) PURE;
};

DEFINE_GUID(IID_ICoClass,0x19FEEEC8,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0x01,0
x10,0x3D,0xE1);

/* Definition of interface: ICoClass */
DECLARE_INTERFACE_(ICoClass, ITypeInformation)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */

    /* ITypeInformation methods */
    STDMETHOD_(BSTR, get_Name)(THIS) PURE;
    STDMETHOD_(BSTR, get_Documentation)(THIS) PURE;
    STDMETHOD_(long, get_HelpContext)(THIS) PURE;
    STDMETHOD_(BSTR, get_HelpFile)(THIS) PURE;
    STDMETHOD_(TYPEKIND, get_TypeInfoKind)(THIS) PURE;
    STDMETHOD(_InitTypeInfo)(THIS_ LPTYPEINFO ptinfo) PURE;
#endif

    /* ICoClass methods */
    STDMETHOD_(ICollection FAR*, get_Interfaces)(THIS) PURE;
};

DEFINE_GUID(IID_IEnum,0x19FEEECD,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x10
,0x3D,0xE1);

/* Definition of interface: IEnum */
DECLARE_INTERFACE_(IEnum, ITypeInformation)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */

    /* ITypeInformation methods */
    STDMETHOD_(BSTR, get_Name)(THIS) PURE;
    STDMETHOD_(BSTR, get_Documentation)(THIS) PURE;
    STDMETHOD_(long, get_HelpContext)(THIS) PURE;
    STDMETHOD_(BSTR, get_HelpFile)(THIS) PURE;
    STDMETHOD_(TYPEKIND, get_TypeInfoKind)(THIS) PURE;
    STDMETHOD(_InitTypeInfo)(THIS_ LPTYPEINFO ptinfo) PURE;
#endif

    /* IEnum methods */
    STDMETHOD_(ICollection FAR*, get_Elements)(THIS) PURE;
};
```

```
DEFINE_GUID(IID_IStruct,0x19FEEECE,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x
10,0x3D,0xE1);

/* Definition of interface: IStruct */
DECLARE_INTERFACE_(IStruct, ITypeInformation)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */


    /* ITypeInformation methods */
    STDMETHOD_(BSTR, get_Name)(THIS) PURE;
    STDMETHOD_(BSTR, get_Documentation)(THIS) PURE;
    STDMETHOD_(long, get_HelpContext)(THIS) PURE;
    STDMETHOD_(BSTR, get_HelpFile)(THIS) PURE;
    STDMETHOD_(TYPEKIND, get_TypeInfoKind)(THIS) PURE;
    STDMETHOD(_InitTypeInfo)(THIS_ LPTYPEINFO ptinfo) PURE;
#endif

    /* IStruct methods */
    STDMETHOD_(ICollection FAR*, get_Members)(THIS) PURE;
};

DEFINE_GUID(IID_IUnion,0x19FEEECF,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x1
0,0x3D,0xE1);

/* Definition of interface: IUnion */
DECLARE_INTERFACE_(IUnion, ITypeInformation)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */

    /* ITypeInformation methods */
    STDMETHOD_(BSTR, get_Name)(THIS) PURE;
    STDMETHOD_(BSTR, get_Documentation)(THIS) PURE;
    STDMETHOD_(long, get_HelpContext)(THIS) PURE;
    STDMETHOD_(BSTR, get_HelpFile)(THIS) PURE;
    STDMETHOD_(TYPEKIND, get_TypeInfoKind)(THIS) PURE;
    STDMETHOD(_InitTypeInfo)(THIS_ LPTYPEINFO ptinfo) PURE;
#endif

    /* IUnion methods */
    STDMETHOD_(ICollection FAR*, get_Members)(THIS) PURE;
};

DEFINE_GUID(IID_IAlias,0x19FEEED0,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x1
0,0x3D,0xE1);

/* Definition of interface: IAlias */
DECLARE_INTERFACE_(IAlias, ITypeInformation)
{
```

```
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */

    /* ITypeInformation methods */
    STDMETHOD_(BSTR, get_Name)(THIS) PURE;
    STDMETHOD_(BSTR, get_Documentation)(THIS) PURE;
    STDMETHOD_(long, get_HelpContext)(THIS) PURE;
    STDMETHOD_(BSTR, get_HelpFile)(THIS) PURE;
    STDMETHOD_(TYPEKIND, get_TypeInfoKind)(THIS) PURE;
    STDMETHOD(_InitTypeInfo)(THIS_ LPTYPEINFO ptinfo) PURE;
#endif

    /* IAlias methods */
    STDMETHOD_(ITypeDesc FAR*, get_BaseType)(THIS) PURE;
};

DEFINE_GUID(IID_IFunction,0x19FEEEC9,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0x01,
0x10,0x3D,0xE1);

/* Definition of interface: IFunction */
DECLARE_INTERFACE_(IFunction, CMyDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */
#endif

    /* IFunction methods */
    STDMETHOD_(BSTR, get_Name)(THIS) PURE;
    STDMETHOD_(BSTR, get_Documentation)(THIS) PURE;
    STDMETHOD_(long, get_HelpContext)(THIS) PURE;
    STDMETHOD_(BSTR, get_HelpFile)(THIS) PURE;
    STDMETHOD_(ITypeDesc FAR*, get_ReturnType)(THIS) PURE;
    STDMETHOD_(ICollection FAR*, get_Parameters)(THIS) PURE;
    STDMETHOD_(MEMBERID, get_Memberid)(THIS) PURE;
    STDMETHOD_(CALLCONV, get_CallConvention)(THIS) PURE;
    STDMETHOD_(FUNCKIND, get_FuncKind)(THIS) PURE;
    STDMETHOD_(INVOKEKIND, get_InvocationKind)(THIS) PURE;
    STDMETHOD_(OBJTYPE, get_Kind)(THIS) PURE;
};

DEFINE_GUID(IID_IProperty,0x19FEEECA,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0x01,
0x10,0x3D,0xE1);

/* Definition of interface: IProperty */
DECLARE_INTERFACE_(IProperty, CMyDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */
#endif
```

```
    /* IProperty methods */
    STDMETHOD_(BSTR, get_Name)(THIS) PURE;
    STDMETHOD_(BSTR, get_Documentation)(THIS) PURE;
    STDMETHOD_(long, get_HelpContext)(THIS) PURE;
    STDMETHOD_(BSTR, get_HelpFile)(THIS) PURE;
    STDMETHOD_(ITypeDesc FAR*, get_Type)(THIS) PURE;
    STDMETHOD_(MEMBERID, get_Memberid)(THIS) PURE;
    STDMETHOD_(OBJTYPE, get_Kind)(THIS) PURE;
};

DEFINE_GUID(IID_IConstant,0x19FEEED4,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0x01,
0x10,0x3D,0xE1);

/* Definition of interface: IConstant */
DECLARE_INTERFACE_(IConstant, CMyDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */
#endif

    /* IConstant methods */
    STDMETHOD_(BSTR, get_Name)(THIS) PURE;
    STDMETHOD_(BSTR, get_Documentation)(THIS) PURE;
    STDMETHOD_(long, get_HelpContext)(THIS) PURE;
    STDMETHOD_(BSTR, get_HelpFile)(THIS) PURE;
    STDMETHOD_(ITypeDesc FAR*, get_Type)(THIS) PURE;
    STDMETHOD_(VARIANT, get_Value)(THIS) PURE;
    STDMETHOD_(MEMBERID, get_Memberid)(THIS) PURE;
    STDMETHOD_(OBJTYPE, get_Kind)(THIS) PURE;
};

DEFINE_GUID(IID_IParameter,0x19FEEECB,0x4104,0x101B,0xB8,0x26,0x00,0xDD,0x01
,0x10,0x3D,0xE1);

/* Definition of interface: IParameter */
DECLARE_INTERFACE_(IParameter, CMyDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* CMyDispatch methods */
/* You must describe methods for this interface here */

#endif

    /* IParameter methods */
    STDMETHOD_(BSTR, get_Name)(THIS) PURE;
    STDMETHOD_(ITypeDesc FAR*, get_Type)(THIS) PURE;
    STDMETHOD_(int, get_IDLFlags)(THIS) PURE;
    STDMETHOD_(OBJTYPE, get_Kind)(THIS) PURE;
};
```

```
DEFINE_GUID(CLSID_BrowseHelper,0x19FEEEC1,0x4104,0x101B,0xB8,0x26,0x00,0xDD,
0x01,0x10,0x3D,0xE1);

class BrowseHelper;

#endif
```

ALIAS.CPP (BROWSEH OLE Sample)

```
/*************************************************************************
**
**    OLE Automation TypeLibrary Browse Helper Sample
**
**    alias.cpp
**
**    CAlias implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "browseh.h"

/*
 * CAlias::Create
 *
 * Purpose:
 *  Creates an instance of the Alias automation object and initializes it.
 *
 * Parameters:
 *  ptinfo     Typeinfo of Alias.
 *  ppAlias    Returns Alias automation object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CAlias::Create(LPTYPEINFO ptinfo, CAlias FAR* FAR* ppAlias)
{
    HRESULT hr;
    CAlias FAR* pAlias = NULL;
    LPTYPEATTR ptypeattr = NULL;
    CTypeDesc FAR* pTypeDesc;

    *ppAlias = NULL;

    // Create alias object.
    pAlias = new CAlias();
    if (pAlias == NULL)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }
    // Load type information for the object from type library.
    hr = pAlias->LoadTypeInfo(IID_IAlias);
    if (FAILED(hr))
        goto error;

    // Ask base class (CTypeInfo) to initialize
    hr = pAlias->_InitTypeInfo(ptinfo);
    if (FAILED(hr))
        goto error;

    // Get base type of this alias.
    hr = ptinfo->GetTypeAttr(&ptypeattr);
    if (FAILED(hr))
        return NULL;
    hr = CTypeDesc::Create(ptinfo, &ptypeattr->tdescAlias, &pTypeDesc);
    if (FAILED(hr))
        goto error;
    pTypeDesc->QueryInterface(IID_IDispatch, (LPVOID FAR*)&pAlias-
>m_pdispTypeDescBase);
    ptinfo->ReleaseTypeAttr(ptypeattr);

#ifdef _DEBUG
    lstrcpyn(pAlias->m_szClassName, TEXT("Alias"), 100);
#endif

    *ppAlias = pAlias;
    return NOERROR;

error:
    if (pAlias == NULL) return ResultFromScode(E_OUTOFMEMORY);
    if (pAlias->m_pdispTypeDescBase) pAlias->m_pdispTypeDescBase->Release();
    if (ptypeattr) ptinfo->ReleaseTypeAttr(ptypeattr);

    // Set to NULL to prevent destructor from attempting to free again
    pAlias->m_pdispTypeDescBase = NULL;

    delete pAlias;
    return hr;
}

/*
 * CAlias::CAlias
 *
 * Purpose:
 *  Constructor for CAlias object. Initializes members to NULL.
 *
 */
CAlias::CAlias()
{
    m_pdispTypeDescBase = NULL;
}
```

```
/*
 * CAlias::~CAlias
 *
 * Purpose:
 *  Destructor for CAlias object.
 *
 */
CAlias::~CAlias()
{
    if (m_pdispTypeDescBase) m_pdispTypeDescBase->Release();
}

STDMETHODIMP_(REFCLSID)
CAlias::GetInterfaceID()
{
    return IID_IAlias;
}

STDMETHODIMP_(ITypeDesc FAR*)
CAlias::get_BaseType()
{
    m_pdispTypeDescBase->AddRef();
    return (ITypeDesc FAR*)m_pdispTypeDescBase;
}
```

BROWSECF.CPP   (BROWSEH OLE Sample)

```
/*************************************************************************
**
**    OLE Automation TypeLibrary Browse Helper Sample
**
**    BrowseHelpercf.cpp
**
**    CBrowseHelperCF (class factory) implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/
#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "browseh.h"

CBrowseHelperCF::CBrowseHelperCF(void)
{
    m_cRef = 0;
}

/*
 * CBrowseHelperCF::QueryInterface, AddRef, Release
 *
 * Purpose:
 *  Implements IUnknown::QueryInterface, AddRef, Release
 *
 */
STDMETHODIMP
CBrowseHelperCF::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
    *ppv = NULL;

    if (iid == IID_IUnknown || iid == IID_IClassFactory)
        *ppv = this;
    else
        return ResultFromScode(E_NOINTERFACE);

    AddRef();
    return NOERROR;
}


STDMETHODIMP_(ULONG)
```

```
CBrowseHelperCF::AddRef(void)
{
    return ++m_cRef;
}


STDMETHODIMP_(ULONG)
CBrowseHelperCF::Release(void)
{
    if(--m_cRef == 0)
    {
        delete this;
        return 0;
    }
    return m_cRef;
}

/*
 * CBrowseHelperCF::CreateInstance, LockServer
 *
 * Purpose:
 *  Implements IClassFactory::CreateInstance, LockServer
 *
 */
STDMETHODIMP
CBrowseHelperCF::CreateInstance(IUnknown FAR* punkOuter,
                         REFIID riid,
                         void FAR* FAR* ppv)
{
    CBrowseHelper FAR* pBrowseHelper;
    HRESULT hr;

    *ppv = NULL;

    // This implementation does not allow aggregation
    if (punkOuter)
        return ResultFromScode(CLASS_E_NOAGGREGATION);

    // Create an instance of the BrowseHelper automation object.
    hr = CBrowseHelper::Create(&pBrowseHelper);
    if (FAILED(hr))
        return hr;

    hr = pBrowseHelper->QueryInterface(riid, ppv);
    if (FAILED(hr))
    {
        delete pBrowseHelper;
        return hr;
    }
    return NOERROR;
}

STDMETHODIMP
CBrowseHelperCF::LockServer(BOOL fLock)
{
    extern ULONG g_cLock;

    if (fLock)
        g_cLock++;
    else
        g_cLock--;

    return NOERROR;
}
```

BROWSEH.CPP   (BROWSEH OLE Sample)

```
/*************************************************************************
**
**    OLE Automation TypeLibrary Browse Helper Sample
**
**    browse.cpp
**
**    CBrowseHelper implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "browseh.h"

/*
 * CBrowseHelper::Create
 *
 * Purpose:
 *  Creates an instance of the BrowseHelper automation object and
initializes it.
 *
 * Parameters:
 *  ppBrowseHelper   Returns BrowseHelper automation object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CBrowseHelper::Create(CBrowseHelper FAR* FAR* ppBrowseHelper )
{
    HRESULT hr;
    CBrowseHelper FAR* pBrowseHelper = NULL;

    *ppBrowseHelper = NULL;

    // Create object.
    pBrowseHelper = new CBrowseHelper();
    if (pBrowseHelper == NULL)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }

    // Load type information for the application object from type library.
    hr = pBrowseHelper->LoadTypeInfo(IID_IBrowseHelper);
    if (FAILED(hr))
        goto error;

#ifdef _DEBUG
    lstrcpyn(pBrowseHelper->m_szClassName, TEXT("BrowseHelper"), 100);
#endif

    *ppBrowseHelper = pBrowseHelper;
    return NOERROR;

error:
    if (pBrowseHelper == NULL)
        return ResultFromScode(E_OUTOFMEMORY);

    delete pBrowseHelper;
    return hr;
}

/*
 * CBrowseHelper::CBrowseHelper
 *
 * Purpose:
 *  Constructor for CBrowseHelper object. Initializes members to NULL.
 *
 */
CBrowseHelper::CBrowseHelper()
{
    extern ULONG g_cObj;
    g_cObj++;
}

/*
 * CBrowseHelper::~CBrowseHelper
 *
 * Purpose:
 *  Destructor for CBrowseHelper object. Frees BrowseHelper message BSTR and
default
 *  IDispatch implementation. Closes the aplication.
 *
 */
CBrowseHelper::~CBrowseHelper()
{
     extern ULONG g_cObj;

     g_cObj--;
}

STDMETHODIMP_(REFCLSID)
CBrowseHelper::GetInterfaceID()
{
    return IID_IBrowseHelper;
}

/*
 * CBrowseHelper::BrowseTypeLibrary
 *
 * Purpose:
 *  Opens and browses type library. Creates and returns a TypeLibrary
object.
 *
 */
STDMETHODIMP_(ITypeLibrary FAR*)
CBrowseHelper::BrowseTypeLibrary(BSTR bstrPath)
{
    LPTYPELIB ptlib = NULL;
    LPDISPATCH pdisp;
    HRESULT hr;
    CTypeLibrary FAR* pTypeLibrary;

    hr = LoadTypeLib(bstrPath, &ptlib);
    if (FAILED(hr))
        {RaiseException(IDS_CannotFindTypeLibrary); return NULL;}

    hr = CTypeLibrary::Create(ptlib, &pTypeLibrary);
    if (FAILED(hr))
         {RaiseException(IDS_TypeLibraryCreationFailed); goto error;}
    hr = pTypeLibrary->QueryInterface(IID_IDispatch, (void FAR*
FAR*)&pdisp);
    if (FAILED(hr))
        {RaiseException(IDS_Unexpected); goto error;}

    ptlib->Release();
    return (ITypeLibrary FAR*)pdisp;


error:
    if (ptlib)
        ptlib->Release();
    return NULL;
}
```

COCLASS.CPP   (BROWSEH OLE Sample)

```
/*************************************************************************
**
**    OLE Automation TypeLibrary Browse Helper Sample
**
**    coclass.cpp
**
**    CCoClass implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "browseh.h"

/*
 * CCoClass::Create
 *
 * Purpose:
 *  Creates an instance of the CoClass automation object and initializes it.
 *
 * Parameters:
 *  ptinfo       TypeInfo of coclass.
 *  ppCoClass    Returns CoClass automation object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CCoClass::Create(LPTYPEINFO ptinfo, CCoClass FAR* FAR* ppCoClass)
{
    HRESULT hr;
    CCoClass FAR* pCoClass = NULL;

    *ppCoClass = NULL;

    // Create object.
    pCoClass = new CCoClass();
    if (pCoClass == NULL)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }
    // Load type information for the object from type library.
    hr = pCoClass->LoadTypeInfo(IID_ICoClass);
    if (FAILED(hr))
        goto error;

    // Ask base class (CTypeInfo) to initialize.
    hr = pCoClass->_InitTypeInfo(ptinfo);
    if (FAILED(hr))
        goto error;

    ptinfo->AddRef();
    pCoClass->m_ptinfo = ptinfo;

#ifdef _DEBUG
    lstrcpyn(pCoClass->m_szClassName, TEXT("CoClass"), 100);
#endif

    *ppCoClass = pCoClass;
    return NOERROR;

error:
    if (pCoClass == NULL) return ResultFromScode(E_OUTOFMEMORY);
    if (pCoClass->m_ptinfo) pCoClass->m_ptinfo->Release();

    // Set to NULL to prevent destructor from attempting to free again
    pCoClass->m_ptinfo = NULL;

    delete pCoClass;
    return hr;
}

/*
 * CCoClass::CCoClass
 *
 * Purpose:
 *  Constructor for CCoClass object. Initializes members to NULL.
 *
 */
CCoClass::CCoClass()
{
    m_pdispInterfaces = NULL;
    m_ptinfo = NULL;
}

/*
 * CCoClass::~CCoClass
 *
 * Purpose:
 *  Destructor for CCoClass object.
 *
 */
CCoClass::~CCoClass()
{
    if (m_pdispInterfaces) m_pdispInterfaces->Release();
    if (m_ptinfo) m_ptinfo->Release();
}

STDMETHODIMP_(REFCLSID)
CCoClass::GetInterfaceID()
{
    return IID_ICoClass;
}

STDMETHODIMP_(ICollection FAR*)
CCoClass::get_Interfaces()
{
    HRESULT hr;
    CCollection FAR* pCollection = NULL;
    CInterface FAR* pInterface;
    CCoClass FAR* pCoClass;
    LPDISPATCH pdisp;
    HREFTYPE hreftype;
    LPTYPEATTR ptypeattr = NULL;
    LPTYPEINFO ptinfoInterface = NULL;
    LPTYPELIB ptlib = NULL;
    unsigned int nIndex;
    unsigned short n;
    TYPEKIND typekind;

    if (m_pdispInterfaces == NULL)
    {
        hr = m_ptinfo->GetTypeAttr(&ptypeattr);
        if (FAILED(hr))
            {RaiseException(IDS_Unexpected); return NULL;}
        hr = CCollection::Create(ptypeattr->cImplTypes, 0, &pCollection);
        if (FAILED(hr))
            {RaiseException(IDS_Unexpected); goto error;}

        // Enumerate interfaces/dispinterfaces in coclass and return a
collection of these.
        for (n=0; n<ptypeattr->cImplTypes; n++)
        {
            hr = m_ptinfo->GetRefTypeOfImplType(n, &hreftype);
            if (FAILED(hr))
                {RaiseException(IDS_Unexpected); goto error;}
            hr = m_ptinfo->GetRefTypeInfo(hreftype, &ptinfoInterface);
            if (FAILED(hr))
                {RaiseException(IDS_Unexpected); goto error;}
            hr = ptinfoInterface->GetContainingTypeLib(&ptlib, &nIndex);
            if (FAILED(hr))
                {RaiseException(IDS_Unexpected); goto error;}
            hr = ptlib->GetTypeInfoType(nIndex, &typekind);
            if (FAILED(hr))
                {RaiseException(IDS_Unexpected); goto error;}
            ptlib->Release();
            ptlib = NULL;
            switch (typekind)
            {
                case TKIND_INTERFACE:
                    hr = CInterface::Create(ptinfoInterface, &pInterface);
                    if (FAILED(hr))
                        {RaiseException(IDS_Unexpected); goto error;}
                    pInterface->QueryInterface(IID_IDispatch, (void FAR*
FAR*)&pdisp);
                    break;

                case TKIND_DISPATCH:
                    hr = CCoClass::Create(ptinfoInterface, &pCoClass);
                    if (FAILED(hr))
                        {RaiseException(IDS_Unexpected); goto error;}
                    pCoClass->QueryInterface(IID_IDispatch, (void FAR*
FAR*)&pdisp);
                    break;
            }

            ptinfoInterface->Release();
            ptinfoInterface = NULL;
            pCollection->Add(pdisp);
            pdisp->Release();
        }
        pCollection->QueryInterface(IID_IDispatch, (void FAR* FAR*)&pdisp);
        m_pdispInterfaces = pdisp;
        m_ptinfo->ReleaseTypeAttr(ptypeattr);
    }
    m_pdispInterfaces->AddRef();
    return (ICollection FAR*)m_pdispInterfaces;

error:
    if (ptypeattr) m_ptinfo->ReleaseTypeAttr(ptypeattr);
    if (pCollection) delete pCollection;
    if (ptlib) ptlib->Release();
    if (ptinfoInterface) ptinfoInterface->Release();
    return NULL;
}
```

COLLECT.CPP (BROWSEH OLE Sample)

```
/*************************************************************************
**
**    OLE Automation TypeLibrary Browse Helper Sample
**
**    collect.cpp
**
**    CCollection implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "browseh.h"

/*
 * CCollection::Create
 *
 * Purpose:
 *  Creates an instance of a collection object and initializes it.
 *
 * Parameters:
 *  lMaxSize   Maximum number of items that can added to collection.
 *  lLBound    Lower bound of index of collection.
 *  ppCollection    Returns collection object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CCollection::Create(ULONG lMaxSize, long lLBound, CCollection FAR* FAR*
ppCollection)
{
    HRESULT hr;
    CCollection FAR* pCollection = NULL;
    SAFEARRAYBOUND sabound[1];

    *ppCollection = NULL;

    // Create new collection
    pCollection = new CCollection();
    if (pCollection == NULL)
        goto error;
    // Load type information for the collection from type library.
    hr = pCollection->LoadTypeInfo(IID_ICollection);
    if (FAILED(hr))
        goto error;

    // If lMaxSize is 0 increment it to 1 or SafeArrayCreate will fail
    if (lMaxSize == 0)
       lMaxSize = 1;
    pCollection->m_cMax = lMaxSize;
    pCollection->m_cElements = 0;
    pCollection->m_lLBound = lLBound;

    // Create a safe array which is used to implement the collection
    sabound[0].cElements = lMaxSize;
    sabound[0].lLbound = lLBound;
    pCollection->m_psa = SafeArrayCreate(VT_VARIANT, 1, sabound);
    if (pCollection->m_psa == NULL)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }

#ifdef _DEBUG
    lstrcpyn(pCollection->m_szClassName, TEXT("Collection"), 100);
#endif

    *ppCollection = pCollection;
    return NOERROR;

error:
    if (pCollection == NULL)
        return ResultFromScode(E_OUTOFMEMORY);
    if (pCollection->m_psa)
        SafeArrayDestroy(pCollection->m_psa);

    pCollection->m_psa = NULL;

    delete pCollection;
    return hr;
}

/*
 * CCollection::CCollection
 *
 * Purpose:
 *  Constructor for CCollection object. Initializes members to NULL.
 *
 */
CCollection::CCollection()
{
    m_psa = NULL;
}
```

```
/*
 * CCollection::~CCollection
 *
 * Purpose:
 *  Destructor for CCollection object.
 *
 */
CCollection::~CCollection()
{
     if (m_psa) SafeArrayDestroy(m_psa);
}

STDMETHODIMP_(REFCLSID)
CCollection::GetInterfaceID()
{
    return IID_ICollection;
}

/*
 * CCollection::get_Count
 *
 * Purpose:
 *  Returns number of items in collection.
 *
 */
STDMETHODIMP_(long)
CCollection::get_Count(void)
{
    return m_cElements;
}

/*

 * CCollection::get_Item
 *
 * Purpose:
 *  Retrieves item from collection, given an index.
 *
 * Parameters:
 *   lIndex   Index of item to be retrieved.
 *
 * Returns
 *  IDispatch of item retrieved from collection.
 *
 */
STDMETHODIMP_(LPDISPATCH)
CCollection::get_Item(long lIndex)
{
    HRESULT hr;
    VARIANT v;
    LPDISPATCH pdisp = NULL;

    // Check if integer index is within range
    if (lIndex < m_lLBound || lIndex >= (long)(m_lLBound+m_cElements))
        {RaiseException(IDS_InvalidIndex); return NULL;}
    // Retrieve and return item. Note that SafeArrayGetElement AddRefs so an
additional AddRef
    // is not required.
    VariantInit(&v);
    hr = SafeArrayGetElement(m_psa, &lIndex, &v);
    if (FAILED(hr))
        {RaiseException(IDS_Unexpected); return NULL;}
    return V_DISPATCH(&v);
}

/*
 * CCollection::get_NewEnum
 *
 * Purpose:
 *  Returns an enumerator (IEnumVARIANT) for the items curently in the
collection.
 *  The NewEnum property is restricted and so is invisible to users of an
 *  automation controller's scripting language. Automation controllers that
support
 *  a 'For Each' statement to iterate through the elements of a collection
will use
 *  the enumerator returned by NewEnum. The enumerator creates a snapshot of
the
 *  the current state of the collection.
 *
 */
STDMETHODIMP_(LPUNKNOWN)
CCollection::get__NewEnum(void)
{
    CEnumVariant FAR* penum = NULL;;
    LPUNKNOWN punkEnumVariant = NULL;
    HRESULT hr;

    // Create new enumerator for items currently in collection and QI for
IUnknown
    hr = CEnumVariant::Create(m_psa, m_cElements, &penum);
    if (FAILED(hr))
        {RaiseException(IDS_OutOfMemory); goto error;}
    hr = penum->QueryInterface(IID_IUnknown, (VOID FAR*
FAR*)&punkEnumVariant);
    if (FAILED(hr))
        {RaiseException(IDS_Unexpected); goto error;}
    return punkEnumVariant;

error:
    if (penum)
        delete penum;
    return NULL;
}

/*
 * CCollection::Add
 *
 * Purpose:
```

```
 *  Adds an item to the collection.
 *
 * Parameters:
 *  pdispItemAdd    Item to be added to collection.
 *
 */
STDMETHODIMP_(void)
CCollection::Add(LPDISPATCH pdispItem)
{
    HRESULT hr;
    LONG l;
    VARIANT v;

    // Is the collection full?
    if (m_cElements == m_cMax)
        {RaiseException(IDS_CollectionFull); return;}

    l = m_lLBound+m_cElements;
    VariantInit(&v);
    V_VT(&v) = VT_DISPATCH;
    V_DISPATCH(&v) = pdispItem;

    hr = SafeArrayPutElement(m_psa, &l, &v);
    if (FAILED(hr))
        {RaiseException(IDS_Unexpected); return;}

    m_cElements++;
}
```

CONSTANT.CPP (BROWSEH OLE Sample)

```
/*************************************************************************
**
**    OLE Automation Constant Browse Helper Sample
**
**    Constant.cpp
**
**    CConstant implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "browseh.h"

/*
 * CConstant::Create
 *
 * Purpose:
 *  Creates an instance of the Constant automation object and initializes
it.
 *
 * Parameters:
 *  ptinfo        TypeInfo of which this constant is an element.
 *  pvardesc      VARDESC that describes this constant.
 *  ppConstant    Returns Constant automation object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CConstant::Create(LPTYPEINFO ptinfo, LPVARDESC pvardesc, CConstant FAR* FAR*
ppConstant)
{
    HRESULT hr;
    CConstant FAR* pConstant = NULL;
    BSTR bstr;
    unsigned int cNames;
    CTypeDesc FAR* pTypeDesc = NULL;

    *ppConstant = NULL;
    // Create object.
    pConstant = new CConstant();
    if (pConstant == NULL)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }

    // Load type information for the application object from type library.
    hr = pConstant->LoadTypeInfo(IID_IConstant);
    if (FAILED(hr))
        goto error;

    pConstant->m_memid = pvardesc->memid;
    ptinfo->GetNames(pvardesc->memid, &bstr, 1, &cNames);
    pConstant->m_bstrName = bstr;

    // Type of constant.
    hr = CTypeDesc::Create(ptinfo, &pvardesc->elemdescVar.tdesc,
&pTypeDesc);
    if (FAILED(hr))
        goto error;
    pTypeDesc->QueryInterface(IID_IDispatch, (LPVOID FAR*)&pConstant-
>m_pdispTypeDesc);

    // Constant value.
    if (pvardesc->varkind == VAR_CONST)
        pConstant->m_vValue = *pvardesc->lpvarValue;
    else goto error;

    hr = ptinfo->GetDocumentation(pvardesc->memid, NULL, &pConstant-
>m_bstrDocumentation,
             &pConstant->m_ulHelpContext, &pConstant->m_bstrHelpFile);
    if (FAILED(hr))
        goto error;

#ifdef _DEBUG
    lstrcpyn(pConstant->m_szClassName, TEXT("Constant"), 100);
#endif

    *ppConstant = pConstant;
    return NOERROR;

error:
    if (pConstant == NULL) return ResultFromScode(E_OUTOFMEMORY);
    if (pConstant->m_bstrName) SysFreeString(pConstant->m_bstrName);
    if (pConstant->m_bstrDocumentation) SysFreeString(pConstant-
>m_bstrDocumentation);
    if (pConstant->m_bstrHelpFile) SysFreeString(pConstant->m_bstrHelpFile);

    // Set to NULL to prevent destructor from attempting to free again
    pConstant->m_bstrName = NULL;
    pConstant->m_bstrDocumentation = NULL;
    pConstant->m_bstrHelpFile = NULL;
    delete pConstant;
    return hr;
}

/*
 * CConstant::CConstant
 *
 * Purpose:
 *  Constructor for CConstant object. Initializes members to NULL.
 *
 */
CConstant::CConstant()
{
    m_bstrName = NULL;
    m_bstrDocumentation = NULL;
    m_bstrHelpFile = NULL;
    m_pdispTypeDesc = NULL;
}

/*
 * CConstant::~CConstant
 *
 * Purpose:
 *  Destructor for CConstant object.
 *
 */
CConstant::~CConstant()
{
     if (m_bstrName) SysFreeString(m_bstrName);
     if (m_bstrDocumentation) SysFreeString(m_bstrDocumentation);
     if (m_bstrHelpFile) SysFreeString(m_bstrHelpFile);
     if (m_pdispTypeDesc) m_pdispTypeDesc->Release();
}

STDMETHODIMP_(REFCLSID)
CConstant::GetInterfaceID()
{
    return IID_IConstant;
}


STDMETHODIMP_(BSTR)
CConstant::get_Name()
{
    return SysAllocString(m_bstrName);
}

STDMETHODIMP_(BSTR)
CConstant::get_Documentation()
{
    return SysAllocString(m_bstrDocumentation);
}
```

```
STDMETHODIMP_(long)
CConstant::get_HelpContext()
{
    return (long)m_ulHelpContext;
}

STDMETHODIMP_(BSTR)
CConstant::get_HelpFile()
{
    return SysAllocString(m_bstrHelpFile);
}

STDMETHODIMP_(ITypeDesc FAR*)
CConstant::get_Type()
{
    m_pdispTypeDesc->AddRef();
    return (ITypeDesc FAR*)m_pdispTypeDesc;
}

STDMETHODIMP_(VARIANT)
CConstant::get_Value()
{
    return m_vValue;
}

STDMETHODIMP_(MEMBERID)
CConstant::get_Memberid()
{
    return m_memid;
}

STDMETHODIMP_(OBJTYPE)
CConstant::get_Kind()
{
    return TYPE_CONSTANT;
}
```

DISPFACE.CPP (BROWSEH OLE Sample)

```
/*************************************************************************
**
**    OLE Automation TypeLibrary Browse Helper Sample
**
**    dispface.cpp
**
**    CDispinterface implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "browseh.h"

/*
 * CDispinterface::Create
 *
 * Purpose:
 *  Creates an instance of the Dispinterface automation object and
initializes it.
 *
 * Parameters:
 *  ptinfo             TypeInfo of dispinterface.
 *  ppDispinterface    Returns Dispinterface automation object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CDispinterface::Create(LPTYPEINFO ptinfo, CDispinterface FAR* FAR*
ppDispinterface)
{
    HRESULT hr;
    CDispinterface FAR* pDispinterface = NULL;

    *ppDispinterface = NULL;

    // Create object.
    pDispinterface = new CDispinterface();
    if (pDispinterface == NULL)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }
    // Load type information for the object from type library.
    hr = pDispinterface->LoadTypeInfo(IID_IDispinterface);
    if (FAILED(hr))
        goto error;

    // Ask base class (CTypeInfo) to initialize.
    hr = pDispinterface->_InitTypeInfo(ptinfo);
    if (FAILED(hr))
        goto error;

    ptinfo->AddRef();
    pDispinterface->m_ptinfo = ptinfo;

#ifdef _DEBUG
    lstrcpyn(pDispinterface->m_szClassName, TEXT("Dispinterface"), 100);
#endif

    *ppDispinterface = pDispinterface;
    return NOERROR;

error:
    if (pDispinterface == NULL) return ResultFromScode(E_OUTOFMEMORY);
    if (pDispinterface->m_ptinfo) pDispinterface->m_ptinfo->Release();

    // Set to NULL to prevent destructor from attempting to free again
    pDispinterface->m_ptinfo = NULL;

    delete pDispinterface;
    return hr;
}

/*
 * CDispinterface::CDispinterface
 *
 * Purpose:
 *  Constructor for CDispinterface object. Initializes members to NULL.
 *
 */
CDispinterface::CDispinterface()
{
    m_pdispProperties = NULL;
    m_pdispMethods = NULL;
    m_ptinfo = NULL;
}

/*
 * CDispinterface::~CDispinterface
 *
 * Purpose:
 *  Destructor for CDispinterface object.
 *
 */
CDispinterface::~CDispinterface()
{
    if (m_pdispProperties) m_pdispProperties->Release();
    if (m_pdispMethods) m_pdispMethods->Release();
    if (m_ptinfo) m_ptinfo->Release();
}

STDMETHODIMP_(REFCLSID)
CDispinterface::GetInterfaceID()
{
    return IID_IDispinterface;
}

STDMETHODIMP_(ICollection FAR*)
CDispinterface::get_Methods()
{
    HRESULT hr;
    CFunction FAR* pFunction;
    CCollection FAR* pCollection = NULL;
    LPDISPATCH pdisp;
    LPTYPEATTR ptypeattr = NULL;
    unsigned short n;

    if (m_pdispMethods == NULL)
    {
        hr = m_ptinfo->GetTypeAttr(&ptypeattr);
        if (FAILED(hr))
            {RaiseException(IDS_Unexpected); return NULL;}
        hr = CCollection::Create(ptypeattr->cFuncs, 0, &pCollection);
        if (FAILED(hr))
            {RaiseException(IDS_Unexpected); goto error;}
        // Enumerate methods and return a collection of these.
        for (n=0; n<ptypeattr->cFuncs; n++)
        {
            hr = CFunction::Create(m_ptinfo, n, &pFunction);
            if (FAILED(hr))
                {RaiseException(IDS_Unexpected); goto error;}
            pFunction->QueryInterface(IID_IDispatch, (void FAR*
FAR*)&pdisp);
            pCollection->Add(pdisp);
            pdisp->Release();
        }
        pCollection->QueryInterface(IID_IDispatch, (void FAR* FAR*)&pdisp);
        m_pdispMethods = pdisp;
        m_ptinfo->ReleaseTypeAttr(ptypeattr);
    }
    m_pdispMethods->AddRef();
    return (ICollection FAR*)m_pdispMethods;

error:
    if (ptypeattr) m_ptinfo->ReleaseTypeAttr(ptypeattr);
    if (pCollection) delete pCollection;
    return NULL;
}

STDMETHODIMP_(ICollection FAR*)
CDispinterface::get_Properties()
{
    HRESULT hr;
    CProperty FAR* pProperty;
    CCollection FAR* pCollection = NULL;
    LPDISPATCH pdisp;
    LPVARDESC pvardesc = NULL;
    LPTYPEATTR ptypeattr = NULL;
    unsigned short n;

    if (m_pdispProperties == NULL)
    {
        hr = m_ptinfo->GetTypeAttr(&ptypeattr);
        if (FAILED(hr))
            {RaiseException(IDS_Unexpected); return NULL;}
        hr = CCollection::Create(ptypeattr->cVars, 0, &pCollection);
        if (FAILED(hr))
            {RaiseException(IDS_Unexpected); goto error;}
        // Enumerate properties and return a collection of these.
        for (n=0; n<ptypeattr->cVars; n++)
        {
            hr = m_ptinfo->GetVarDesc(n, &pvardesc);
            if (FAILED(hr))
                {RaiseException(IDS_Unexpected); goto error;}
            hr = CProperty::Create(m_ptinfo, pvardesc, &pProperty);
            if (FAILED(hr))
                {RaiseException(IDS_Unexpected); goto error;}
            m_ptinfo->ReleaseVarDesc(pvardesc);
            pvardesc = NULL;
            pProperty->QueryInterface(IID_IDispatch, (void FAR*
FAR*)&pdisp);
            pCollection->Add(pdisp);
            pdisp->Release();
        }
        pCollection->QueryInterface(IID_IDispatch, (void FAR* FAR*)&pdisp);
        m_pdispProperties = pdisp;
        m_ptinfo->ReleaseTypeAttr(ptypeattr);
    }
    m_pdispProperties->AddRef();
    return (ICollection FAR*)m_pdispProperties;

error:
    if (ptypeattr) m_ptinfo->ReleaseTypeAttr(ptypeattr);
    if (pCollection) delete pCollection;
    if (pvardesc) m_ptinfo->ReleaseVarDesc(pvardesc);
    return NULL;
}
```

ENUM.CPP (BROWSEH OLE Sample)

```
/*************************************************************************
**
**    OLE Automation TypeLibrary Browse Helper Sample
**
**    enum.cpp
**
**    CEnum implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "browseh.h"

/*
 * CEnum::Create
 *
 * Purpose:
 *  Creates an instance of the Enum automation object and initializes it.
 *
 * Parameters:
 *  ptinfo    TypeInfo of Enum.
 *  ppEnum    Returns Enum automation object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CEnum::Create(LPTYPEINFO ptinfo, CEnum FAR* FAR* ppEnum)
{
    HRESULT hr;
    CEnum FAR* pEnum = NULL;

    *ppEnum = NULL;

    // Create object.
    pEnum = new CEnum();
    if (pEnum == NULL)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }
    // Load type information for the object from type library.
    hr = pEnum->LoadTypeInfo(IID_IEnum);
    if (FAILED(hr))
        goto error;
    // Ask base class (CTypeInfo) to initialize.
    hr = pEnum->_InitTypeInfo(ptinfo);
    if (FAILED(hr))
        goto error;

    ptinfo->AddRef();
    pEnum->m_ptinfo = ptinfo;

#ifdef _DEBUG
    lstrcpyn(pEnum->m_szClassName, TEXT("Enum"), 100);
#endif

    *ppEnum = pEnum;
    return NOERROR;

error:
    if (pEnum == NULL) return ResultFromScode(E_OUTOFMEMORY);
    if (pEnum->m_ptinfo) pEnum->m_ptinfo->Release();

    // Set to NULL to prevent destructor from attempting to free again
    pEnum->m_ptinfo = NULL;

    delete pEnum;
    return hr;
}

/*
 * CEnum::CEnum
 *
 * Purpose:
 *  Constructor for CEnum object. Initializes members to NULL.
 *
 */
CEnum::CEnum()
{
    m_pdispElements = NULL;
    m_ptinfo = NULL;
}

/*
 * CEnum::~CEnum
 *
 * Purpose:
 *  Destructor for CEnum object.
 *
 */
CEnum::~CEnum()
{
    if (m_pdispElements) m_pdispElements->Release();
    if (m_ptinfo) m_ptinfo->Release();
}

STDMETHODIMP_(REFCLSID)
CEnum::GetInterfaceID()
{
    return IID_IEnum;
}

STDMETHODIMP_(ICollection FAR*)
CEnum::get_Elements()
{
    HRESULT hr;
    CConstant FAR* pConstant;
    CCollection FAR* pCollection = NULL;
    LPDISPATCH pdisp;
    LPVARDESC pvardesc = NULL;
    LPTYPEATTR ptypeattr = NULL;
    unsigned short n;

    if (m_pdispElements == NULL)
    {
        hr = m_ptinfo->GetTypeAttr(&ptypeattr);
        if (FAILED(hr))
            {RaiseException(IDS_Unexpected); return NULL;}
        hr = CCollection::Create(ptypeattr->cVars, 0, &pCollection);
        if (FAILED(hr))
            {RaiseException(IDS_Unexpected); goto error;}
        // Enumerate enum constants and return a collection of these.
        for (n=0; n<ptypeattr->cVars; n++)
        {

            hr = m_ptinfo->GetVarDesc(n, &pvardesc);
            if (FAILED(hr))
                {RaiseException(IDS_Unexpected); goto error;}
            hr = CConstant::Create(m_ptinfo, pvardesc, &pConstant);
            if (FAILED(hr))
                {RaiseException(IDS_Unexpected); goto error;}
            m_ptinfo->ReleaseVarDesc(pvardesc);
            pvardesc = NULL;
            pConstant->QueryInterface(IID_IDispatch, (void FAR*
FAR*)&pdisp);
            pCollection->Add(pdisp);
            pdisp->Release();
        }
        pCollection->QueryInterface(IID_IDispatch, (void FAR* FAR*)&pdisp);
        m_pdispElements = pdisp;
        m_ptinfo->ReleaseTypeAttr(ptypeattr);
    }
    m_pdispElements->AddRef();
    return (ICollection FAR*)m_pdispElements;

error:
    if (ptypeattr) m_ptinfo->ReleaseTypeAttr(ptypeattr);
    if (pCollection) delete pCollection;
    if (pvardesc) m_ptinfo->ReleaseVarDesc(pvardesc);
    return NULL;
}
```

ENUMVAR.CPP   (BROWSEH OLE Sample)

```
/*************************************************************************
**
**    OLE Automation TypeLibrary Browse Helper Sample
**
**    enumvar.cpp
**
**    CEnumVariant implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "browseh.h"

/*
 * CEnumVariant::Create
 *
 * Purpose:
 *  Creates an instance of the IEnumVARIANT enumerator object and
initializes it.
 *
 * Parameters:
 *  psa         Safe array containing items to be enumerated.
 *  cElements  Number of items to be enumerated.
 *  ppenumvariant    Returns enumerator object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CEnumVariant::Create(SAFEARRAY FAR* psa, ULONG cElements, CEnumVariant FAR*
FAR* ppenumvariant)
{
    HRESULT hr;
    CEnumVariant FAR* penumvariant = NULL;
    long lLBound;

    *ppenumvariant = NULL;

    penumvariant = new CEnumVariant();
    if (penumvariant == NULL)
        goto error;

    penumvariant->m_cRef = 0;

    // Copy elements into safe array that is used in enumerator
implemenatation and
    // initialize state of enumerator.
    hr = SafeArrayGetLBound(psa, 1, &lLBound);
    if (FAILED(hr))
        goto error;
    penumvariant->m_cElements = cElements;
    penumvariant->m_lLBound = lLBound;
    penumvariant->m_lCurrent = lLBound;
    hr = SafeArrayCopy(psa, &penumvariant->m_psa);
    if (FAILED(hr))
       goto error;

    *ppenumvariant = penumvariant;
    return NOERROR;

error:
    if (penumvariant == NULL)
        return ResultFromScode(E_OUTOFMEMORY);

    if (penumvariant->m_psa)
        SafeArrayDestroy(penumvariant->m_psa);
    penumvariant->m_psa = NULL;
    delete penumvariant;
    return hr;
}

/*
 * CEnumVariant::CEnumVariant
 *
 * Purpose:
 *  Constructor for CEnumVariant object. Initializes members to NULL.
 *
 */
CEnumVariant::CEnumVariant()
{
    m_psa = NULL;
}

/*
 * CEnumVariant::~CEnumVariant
 *
 * Purpose:
 *  Destructor for CEnumVariant object.
 *
 */
CEnumVariant::~CEnumVariant()
{
    if (m_psa) SafeArrayDestroy(m_psa);
}
```

```
/*
 * CEnumVariant::QueryInterface, AddRef, Release
 *
 * Purpose:
 *  Implements IUnknown::QueryInterface, AddRef, Release
 *
 */
STDMETHODIMP
CEnumVariant::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
    *ppv = NULL;

    if (iid == IID_IUnknown || iid == IID_IEnumVARIANT)
        *ppv = this;
    else return ResultFromScode(E_NOINTERFACE);

    AddRef();
    return NOERROR;
}


STDMETHODIMP_(ULONG)
CEnumVariant::AddRef(void)
{

#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, Enum\r\n"), m_cRef+1);
    OutputDebugString(ach);
#endif

    return ++m_cRef;  // AddRef Application Object if enumerator will
outlive application object
}



STDMETHODIMP_(ULONG)
CEnumVariant::Release(void)
{

#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, Enum\r\n"), m_cRef-1);
    OutputDebugString(ach);
#endif

    if(--m_cRef == 0)
    {
        delete this;
        return 0;
    }
    return m_cRef;
}
```

```
/*
 * CEnumVariant::Next
 *
 * Purpose:
 *  Retrieves the next cElements elements. Implements IEnumVARIANT::Next.
 *
 */
STDMETHODIMP
CEnumVariant::Next(ULONG cElements, VARIANT FAR* pvar, ULONG FAR*
pcElementFetched)
{
    HRESULT hr;
    ULONG l;
    long l1;
    ULONG l2;

    if (pcElementFetched != NULL)
        *pcElementFetched = 0;

    for (l=0; l<cElements; l++)
        VariantInit(&pvar[l]);

    // Retrieve the next cElements elements.
    for (l1=m_lCurrent, l2=0; l1<(long)(m_lLBound+m_cElements) &&
l2<cElements; l1++, l2++)
    {
       hr = SafeArrayGetElement(m_psa, &l1, &pvar[l2]);
       if (FAILED(hr))
           goto error;
    }
    // Set count of elements retrieved
    if (pcElementFetched != NULL)
        *pcElementFetched = l2;
    m_lCurrent = l1;

    return  (l2 < cElements) ? ResultFromScode(S_FALSE) : NOERROR;

error:
    for (l=0; l<cElements; l++)
        VariantClear(&pvar[l]);
    return hr;
}

/*
 * CEnumVariant::Skip
 *
 * Purpose:
 *  Skips the next cElements elements. Implements IEnumVARIANT::Skip.
 *
 */
STDMETHODIMP
CEnumVariant::Skip(ULONG cElements)
{
    m_lCurrent += cElements;
    if (m_lCurrent > (long)(m_lLBound+m_cElements))
    {
        m_lCurrent =  m_lLBound+m_cElements;
        return ResultFromScode(S_FALSE);
    }
    else return NOERROR;
}

/*
 * CEnumVariant::Reset
 *
 * Purpose:
 *  Resets the current element in the enumerator to the beginning.
Implements IEnumVARIANT::Reset.
 *
 */
STDMETHODIMP
CEnumVariant::Reset()
{
    m_lCurrent = m_lLBound;
    return NOERROR;
}

/*
 * CEnumVariant::Clone
 *
 * Purpose:
 *  Creates a copy of the current enumeration state. Implements
IEnumVARIANT::Clone.
 *
 */
STDMETHODIMP
CEnumVariant::Clone(IEnumVARIANT FAR* FAR* ppenum)
{
    CEnumVariant FAR* penum = NULL;
    HRESULT hr;

    *ppenum = NULL;

    hr = CEnumVariant::Create(m_psa, m_cElements, &penum);
    if (FAILED(hr))
        goto error;
    penum->AddRef();
    penum->m_lCurrent = m_lCurrent;

    *ppenum = penum;
    return NOERROR;

error:
    if (penum)
        penum->Release();
    return hr;
}
```

FUNCTION.CPP (BROWSEH OLE Sample)

```
/*************************************************************************
**
**    OLE Automation TypeLibrary Browse Helper Sample
**
**    function.cpp
**
**    CFunction implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "browseh.h"

/*
 * CFunction::Create
 *
 * Purpose:
 *  Creates an instance of the Function automation object and initializes
it.
 *
 * Parameters:
 *  ptinfo        TypeInfo of which this Function is an element of.
 *  nIndex       Index of function in containing TypeInfo.
 *  ppFunction    Returns Function automation object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CFunction::Create(LPTYPEINFO ptinfo, unsigned short nIndex, CFunction FAR*
FAR* ppFunction)
{
    HRESULT hr;
    CFunction FAR* pFunction = NULL;
    CTypeDesc FAR* pTypeDesc = NULL;

    *ppFunction = NULL;

    // Create object.
    pFunction = new CFunction();
    if (pFunction == NULL)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }

    // Load type information for the application object from type library.
    hr = pFunction->LoadTypeInfo(IID_IFunction);
    if (FAILED(hr))
        goto error;

    hr = ptinfo->GetFuncDesc(nIndex, &pFunction->m_pfuncdesc);
    if (FAILED(hr))
        goto error;
    // Function return type.
    hr = CTypeDesc::Create(ptinfo, &pFunction->m_pfuncdesc-
>elemdescFunc.tdesc, &pTypeDesc);
    if (FAILED(hr))
        goto error;
    pTypeDesc->QueryInterface(IID_IDispatch, (LPVOID FAR*)&pFunction-
>m_pdispTypeDesc);

    hr = ptinfo->GetDocumentation(pFunction->m_pfuncdesc->memid, &pFunction-
>m_bstrName, &pFunction->m_bstrDocumentation,
             &pFunction->m_ulHelpContext, &pFunction->m_bstrHelpFile);
    if (FAILED(hr))
        goto error;

    ptinfo->AddRef();
    pFunction->m_ptinfoFunction = ptinfo;

#ifdef _DEBUG
    lstrcpyn(pFunction->m_szClassName, TEXT("Function"), 100);
#endif

    *ppFunction = pFunction;
    return NOERROR;

error:
    if (pFunction == NULL) return ResultFromScode(E_OUTOFMEMORY);
    if (pFunction->m_pfuncdesc) ptinfo->ReleaseFuncDesc(pFunction-
>m_pfuncdesc);
    if (pFunction->m_ptinfoFunction) pFunction->m_ptinfoFunction->Release();
    if (pFunction->m_bstrName) SysFreeString(pFunction->m_bstrName);
    if (pFunction->m_bstrDocumentation) SysFreeString(pFunction-
>m_bstrDocumentation);
    if (pFunction->m_bstrHelpFile) SysFreeString(pFunction->m_bstrHelpFile);

    // Set to NULL to prevent destructor from attempting to free again
    pFunction->m_ptinfoFunction = NULL;
    pFunction->m_pfuncdesc = NULL;
    pFunction->m_bstrName = NULL;
    pFunction->m_bstrDocumentation = NULL;
    pFunction->m_bstrHelpFile = NULL;
    delete pFunction;
    return hr;
}

/*
 * CFunction::CFunction
 *
 * Purpose:
 *  Constructor for CFunction object. Initializes members to NULL.
 *
 */
CFunction::CFunction()
{
    m_bstrName = NULL;
    m_bstrDocumentation = NULL;
    m_bstrHelpFile = NULL;
    m_pdispParameters = NULL;
    m_ptinfoFunction = NULL;
    m_pfuncdesc = NULL;
    m_pdispTypeDesc = NULL;
}

/*
 * CFunction::~CFunction
 *
 * Purpose:
 *  Destructor for CFunction object.
 *
 */
CFunction::~CFunction()
{
     if (m_bstrName) SysFreeString(m_bstrName);
     if (m_bstrDocumentation) SysFreeString(m_bstrDocumentation);
     if (m_bstrHelpFile) SysFreeString(m_bstrHelpFile);
     if (m_pdispParameters) m_pdispParameters->Release();
     if (m_pfuncdesc && m_ptinfoFunction) m_ptinfoFunction-
>ReleaseFuncDesc(m_pfuncdesc);
     if (m_ptinfoFunction) m_ptinfoFunction->Release();
     if (m_pdispTypeDesc) m_pdispTypeDesc->Release();
}

STDMETHODIMP_(REFCLSID)
CFunction::GetInterfaceID()
{
    return IID_IFunction;
}

STDMETHODIMP_(BSTR)
CFunction::get_Name()
{
    return SysAllocString(m_bstrName);
}
```

```
STDMETHODIMP_(BSTR)
CFunction::get_Documentation()
{
    return SysAllocString(m_bstrDocumentation);
}

STDMETHODIMP_(long)
CFunction::get_HelpContext()
{
    return (long)m_ulHelpContext;
}

STDMETHODIMP_(BSTR)
CFunction::get_HelpFile()
{
    return SysAllocString(m_bstrHelpFile);
}


STDMETHODIMP_(ITypeDesc FAR*)
CFunction::get_ReturnType()
{
    m_pdispTypeDesc->AddRef();
    return (ITypeDesc FAR*)m_pdispTypeDesc;
}


STDMETHODIMP_(ICollection FAR*)
CFunction::get_Parameters()
{
    HRESULT hr;
    CParameter FAR* pParameter;
    CCollection FAR* pCollection = NULL;
    LPDISPATCH pdisp;
    BSTR FAR* rgbstrNames = NULL;
    unsigned int cNames;
    unsigned short n = 0;

    if (m_pdispParameters == NULL)
    {
        // Create a collection of parameters and return it.
        rgbstrNames = new BSTR[m_pfuncdesc->cParams+1];
        if (rgbstrNames == NULL)
            {RaiseException(IDS_OutOfMemory); return NULL;}
        hr = m_ptinfoFunction->GetNames(m_pfuncdesc->memid, rgbstrNames,
m_pfuncdesc->cParams+1, &cNames);
        if (FAILED(hr))
            {RaiseException(IDS_Unexpected); goto error;}
        SysFreeString(rgbstrNames[0]);

        hr = CCollection::Create(cNames-1, 0, &pCollection);
        if (FAILED(hr))
            {RaiseException(IDS_Unexpected); goto error;}

        for (n=0; n<cNames-1; n++)
        {
            hr = CParameter::Create(m_ptinfoFunction, rgbstrNames[n+1],
                   &m_pfuncdesc->lprgelemdescParam[n].tdesc,
                   &m_pfuncdesc->lprgelemdescParam[n].idldesc,
                   &pParameter);
            if (FAILED(hr))
                {RaiseException(IDS_Unexpected); goto error;}
            SysFreeString(rgbstrNames[n+1]);
            pParameter->QueryInterface(IID_IDispatch, (void FAR*
FAR*)&pdisp);
            pCollection->Add(pdisp);
            pdisp->Release();
        }
        pCollection->QueryInterface(IID_IDispatch, (void FAR* FAR*)&pdisp);
        m_pdispParameters = pdisp;
        delete rgbstrNames;
    }
    m_pdispParameters->AddRef();
    return (ICollection FAR*)m_pdispParameters;

error:
    if (pCollection) delete pCollection;
    if (rgbstrNames)
        for (; n<cNames-1; n++)
            SysFreeString(rgbstrNames[n+1]);
    return NULL;
}

STDMETHODIMP_(MEMBERID)
CFunction::get_Memberid()
{
    return m_pfuncdesc->memid;
}

STDMETHODIMP_(CALLCONV)
CFunction::get_CallConvention()
{
    return m_pfuncdesc->callconv;
}

STDMETHODIMP_(FUNCKIND)
CFunction::get_FuncKind()
{
    return m_pfuncdesc->funckind;
}

STDMETHODIMP_(INVOKEKIND)
CFunction::get_InvocationKind()
{
    return m_pfuncdesc->invkind;
}

STDMETHODIMP_(OBJTYPE)
CFunction::get_Kind()
{
    return TYPE_FUNCTION;
}
```

INTFACE.CPP   (BROWSEH OLE Sample)

```
/**************************************************************************
**
**    OLE Automation TypeLibrary Browse Helper Sample
**
**    intface.cpp
**
**    CInterface implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
**************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "browseh.h"

/*
 * CInterface::Create
 *
 * Purpose:
 *  Creates an instance of the Interface automation object and initializes
it.
 *
 * Parameters:
 *  ptinfo          TypeInfo of interface.
 *  ppInterface    Returns Interface automation object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CInterface::Create(LPTYPEINFO ptinfo, CInterface FAR* FAR* ppInterface)
{
    HRESULT hr;
    CInterface FAR* pInterface = NULL;

    *ppInterface = NULL;

    // Create object.
    pInterface = new CInterface();
    if (pInterface == NULL)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }
    // Load type information for the object from type library.
    hr = pInterface->LoadTypeInfo(IID_IInterface);
    if (FAILED(hr))
        goto error;
    // Intialize base class, CTypeInfo
    hr = pInterface->_InitTypeInfo(ptinfo);
    if (FAILED(hr))
        goto error;

    ptinfo->AddRef();
    pInterface->m_ptinfo = ptinfo;

#ifdef _DEBUG
    lstrcpyn(pInterface->m_szClassName, TEXT("Interface"), 100);
#endif

    *ppInterface = pInterface;
    return NOERROR;

error:
    if (pInterface == NULL) return ResultFromScode(E_OUTOFMEMORY);
    if (pInterface->m_ptinfo) pInterface->m_ptinfo->Release();

    // Set to NULL to prevent destructor from attempting to free again
    pInterface->m_ptinfo = NULL;

    delete pInterface;
    return hr;
}

/*
 * CInterface::CInterface
 *
 * Purpose:
 *  Constructor for CInterface object. Initializes members to NULL.
 *
 */
CInterface::CInterface()
{
    m_pdispFunctions = NULL;
    m_ptinfo = NULL;
}

/*
 * CInterface::~CInterface
 *
 * Purpose:
 *  Destructor for CInterface object.
 *
 */
CInterface::~CInterface()
{
    if (m_pdispFunctions) m_pdispFunctions->Release();
    if (m_ptinfo) m_ptinfo->Release();
}

STDMETHODIMP_(REFCLSID)
CInterface::GetInterfaceID()
{
    return IID_IInterface;
}

STDMETHODIMP_(ICollection FAR*)
CInterface::get_Functions()
{
    HRESULT hr;
    CFunction FAR* pFunction;
    CCollection FAR* pCollection = NULL;
    LPDISPATCH pdisp;
    LPTYPEATTR ptypeattr = NULL;
    unsigned short n;

    if (m_pdispFunctions == NULL)
    {
        // Create collection of functions in interface.
        hr = m_ptinfo->GetTypeAttr(&ptypeattr);
        if (FAILED(hr))
            {RaiseException(IDS_Unexpected); return NULL;}
        hr = CCollection::Create(ptypeattr->cFuncs, 0, &pCollection);
        if (FAILED(hr))
            {RaiseException(IDS_Unexpected); goto error;}
        for (n=0; n<ptypeattr->cFuncs; n++)
        {
            hr = CFunction::Create(m_ptinfo, n, &pFunction);
            if (FAILED(hr))
                {RaiseException(IDS_Unexpected); goto error;}
            pFunction->QueryInterface(IID_IDispatch, (void FAR*
FAR*)&pdisp);
            pCollection->Add(pdisp);
            pdisp->Release();
        }
        pCollection->QueryInterface(IID_IDispatch, (void FAR* FAR*)&pdisp);
        m_pdispFunctions = pdisp;
        m_ptinfo->ReleaseTypeAttr(ptypeattr);
    }
    m_pdispFunctions->AddRef();
    return (ICollection FAR*)m_pdispFunctions;

error:
    if (ptypeattr) m_ptinfo->ReleaseTypeAttr(ptypeattr);
    if (pCollection) delete pCollection;
    return NULL;
}
```

MAIN.CPP (BROWSEH OLE Sample)

```
/*************************************************************************
**
**    OLE Automation TypeLibrary Browse Helper Sample
**
**    main.cpp
**
**    new and delete operator redefinition to use memory manager of calling
**    task.
**    LibMain, WEP, DllGetClassObject, DllCanUnloadNow, DLL initialization.
**    Procedure to create standard dispatch implementation.
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/
#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include <initguid.h>
#include "browseh.h"

// Globals
HINSTANCE   g_hinst;                  // Instance of application
                                      //Count number of objects and number
of locks.
ULONG       g_cObj=0;
ULONG       g_cLock=0;

// String resource buffers
TCHAR g_szServerName[STR_LEN];


/*
 * new
 *
 * Purpose:
 *   Since this is an InProcServer object, the memory allocator used by
 *   the calling task must be used.
 *   This is done by redefining the global new operator and using new
 *   for all memory allocations.
 */
void FAR* operator new(size_t size)
{
    IMalloc FAR* pMalloc;
    LPVOID lpv;
    if (CoGetMalloc(MEMCTX_TASK, &pMalloc) == NOERROR)
    {
        lpv = pMalloc->Alloc(size);
        pMalloc->Release();
        return lpv;
    }
    return NULL;
}

/*
 * delete
 *
 * Purpose:
 *   Use the memory manager of the calling task to free memory.
 */
void operator delete(void FAR* lpv)
{
    IMalloc FAR* pMalloc;

    if (lpv == NULL) return;

    if( CoGetMalloc(MEMCTX_TASK, &pMalloc) == NOERROR)
    {
        if (pMalloc->DidAlloc(lpv))
            pMalloc->Free(lpv);
        pMalloc->Release();
    }
}


#ifdef WIN16
/*
 * LibMain
 *
 * Purpose:
 *  Called by Win16 on DLL load. Does any one-time initializations.
 *
 */
int PASCAL LibMain (HINSTANCE hinst, WORD wDataSeg, WORD cbHeapSize, LPSTR
lpCmdLine)
{
   if (cbHeapSize != 0)
       UnlockData(0);

   g_hinst = hinst;

   if (!InitDLL(hinst))
         return FALSE;

   return TRUE;
}

/*
 * WEP
 *
 * Purpose:
 *  Called by Windows on DLL unload.
 *
 */
extern "C" void FAR PASCAL _WEP(int bSystemExit)
{
    return;
}

#else //Win 32

BOOL WINAPI DllMain (HINSTANCE hinst, DWORD dwReason, LPVOID lpReserved)
{
    switch (dwReason)
    {
        case DLL_PROCESS_ATTACH:
            if (!InitDLL(hinst))
                return FALSE;
               else return TRUE;

     default:
               return TRUE;
    }
}

#endif

/*
 * DLLGetClassObject
 *

 * Purpose:
 *  OLE calls this funtion to obtain the class factory. Note that the class
 *  factory is not registered by the inproc server.
 *
 */
STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid,
                                   LPVOID FAR *ppv)
{
    LPCLASSFACTORY pcf;
    HRESULT hr;

    *ppv =  NULL;

    // Check if this CLSID is supported.
    if (rclsid != CLSID_BrowseHelper)
       return ResultFromScode(E_FAIL);

    // Create class factory and return it
    pcf = new CBrowseHelperCF;
    if (!pcf)
        return ResultFromScode(E_OUTOFMEMORY);
    hr = pcf->QueryInterface(riid, ppv);
    if (FAILED(hr))
    {
        delete pcf;
        return hr;
    }
    return NOERROR;
}

/*
 * DLLCanUnloadNow
 *
 * Purpose:
 *  DllCanUnloadNow is called by OLE to determine if the DLL can be unloded.
 *
 */
STDAPI DllCanUnloadNow(void)
{
    if (g_cObj==0L && g_cLock==0L)
        return ResultFromScode(S_OK);
    else return ResultFromScode(S_FALSE);
}

/*
 * InitDLL
 *
 * Purpose:
 *  Load strings & Registers the window class
 *
 * Parameters:
 *  hinstance       hinstance of application
 *
 */
BOOL InitDLL (HINSTANCE hinst)
{
   return LoadString(hinst, IDS_SERVERNAME, g_szServerName,
sizeof(g_szServerName));
}
```

MODULE.CPP   (BROWSEH OLE Sample)

```
/*************************************************************************
**
**    OLE Automation TypeLibrary Browse Helper Sample
**
**    module.cpp
**
**    CModule implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "browseh.h"

/*
 * CModule::Create
 *
 * Purpose:
 *  Creates an instance of the Module automation object and initializes it.
 *
 * Parameters:
 *  ptinfo      TypeInfo of module.
 *  ppModule    Returns Module automation object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CModule::Create(LPTYPEINFO ptinfo, CModule FAR* FAR* ppModule)
{
    HRESULT hr;
    CModule FAR* pModule = NULL;

    *ppModule = NULL;

    // Create object.
    pModule = new CModule();
    if (pModule == NULL)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }
    // Load type information for the object from type library.
    hr = pModule->LoadTypeInfo(IID_IModule);
    if (FAILED(hr))
        goto error;
    // Intialize base class, CTypeInfo
    hr = pModule->_InitTypeInfo(ptinfo);
    if (FAILED(hr))
        goto error;

    ptinfo->AddRef();
    pModule->m_ptinfo = ptinfo;

#ifdef _DEBUG
    lstrcpyn(pModule->m_szClassName, TEXT("Module"), 100);
#endif

    *ppModule = pModule;
    return NOERROR;

error:
    if (pModule == NULL) return ResultFromScode(E_OUTOFMEMORY);
    if (pModule->m_ptinfo) pModule->m_ptinfo->Release();

    // Set to NULL to prevent destructor from attempting to free again
    pModule->m_ptinfo = NULL;

    delete pModule;
    return hr;
}

/*
 * CModule::CModule
 *
 * Purpose:
 *  Constructor for CModule object. Initializes members to NULL.
 *
 */
CModule::CModule()
{
    m_pdispFunctions = NULL;
    m_ptinfo = NULL;
}

/*
 * CModule::~CModule
 *
 * Purpose:
 *  Destructor for CModule object.
 *
 */
CModule::~CModule()
{
    if (m_pdispFunctions) m_pdispFunctions->Release();
    if (m_ptinfo) m_ptinfo->Release();
}

STDMETHODIMP_(REFCLSID)
CModule::GetInterfaceID()
{
    return IID_IModule;
}

STDMETHODIMP_(ICollection FAR*)
CModule::get_Functions()
{
    HRESULT hr;
    CFunction FAR* pFunction;
    CCollection FAR* pCollection = NULL;
    LPDISPATCH pdisp;
    LPTYPEATTR ptypeattr = NULL;
    unsigned short n;

    if (m_pdispFunctions == NULL)
    {
        // Create collection of functions of the interface.
        hr = m_ptinfo->GetTypeAttr(&ptypeattr);
        if (FAILED(hr))
            {RaiseException(IDS_Unexpected); return NULL;}
        hr = CCollection::Create(ptypeattr->cFuncs, 0, &pCollection);
        if (FAILED(hr))
            {RaiseException(IDS_Unexpected); goto error;}
        for (n=0; n<ptypeattr->cFuncs; n++)
        {
            hr = CFunction::Create(m_ptinfo, n, &pFunction);
            if (FAILED(hr))
                {RaiseException(IDS_Unexpected); goto error;}
            pFunction->QueryInterface(IID_IDispatch, (void FAR*
FAR*)&pdisp);
            pCollection->Add(pdisp);
            pdisp->Release();
        }
        pCollection->QueryInterface(IID_IDispatch, (void FAR* FAR*)&pdisp);
        m_pdispFunctions = pdisp;
        m_ptinfo->ReleaseTypeAttr(ptypeattr);
    }
    m_pdispFunctions->AddRef();
    return (ICollection FAR*)m_pdispFunctions;

error:
    if (ptypeattr) m_ptinfo->ReleaseTypeAttr(ptypeattr);
    if (pCollection) delete pCollection;
    return NULL;
}
```

MYDISP.CPP (BROWSEH OLE Sample)

```
/*************************************************************************
**
**    OLE Automation TypeLibrary Browse Helper Sample
**
**    mydisp.cpp
**
**    CMyDispatch implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
**    The CMyDispatch object implement IUnknown & IDispatch for all the
automation
**    objects in this sample. All objects derive from CMyDispatch.
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "browseh.h"

/*
 * CMyDispatch::CMyDispatch
 *
 * Purpose:
 *  Constructor for CMyDispatch object. Initializes members to NULL.
 *
 */
CMyDispatch::CMyDispatch()
{
    m_ptinfo = NULL;
    m_cRef = 0;
}

/*
 * CMyDispatch::~CMyDispatch
 *
 * Purpose:
 *  Destructor for CMyDispatch object.
 *
 */
CMyDispatch::~CMyDispatch()
{
     if (m_ptinfo) m_ptinfo->Release();
```

```
}

/*
 * CMyDispatch::QueryInterface, AddRef, Release
 *
 * Purpose:
 *  Implements IUnknown::QueryInterface, AddRef, Release
 *
 */

STDMETHODIMP
CMyDispatch::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
    *ppv = NULL;

    if (iid == IID_IUnknown)
        *ppv = this;
    else if (iid == IID_IDispatch)
        *ppv = this;
    else if (iid == GetInterfaceID())
        *ppv = this;
    else return ResultFromScode(E_NOINTERFACE);

    AddRef();
    return NOERROR;
}


STDMETHODIMP_(ULONG)
CMyDispatch::AddRef(void)
{
#ifdef _DEBUG
    TCHAR ach[150];
    wsprintf(ach, TEXT("Ref = %ld, Object = %s\r\n"), m_cRef+1,
m_szClassName);
    OutputDebugString(ach);
#endif
    return ++m_cRef;
}


STDMETHODIMP_(ULONG)
CMyDispatch::Release(void)
{
#ifdef _DEBUG
    TCHAR ach[150];
    wsprintf(ach, TEXT("Ref = %ld, Object = %s\r\n"), m_cRef-1,
m_szClassName);
    OutputDebugString(ach);
#endif
    if(--m_cRef == 0)
    {
        delete this;
        return 0;
    }
    return m_cRef;
}

/*
 * CMyDispatch::GetTypeInfoCount
 *
 * Purpose:
 *  Implements IDispatch::GetTypeInfoCount.
 *
 */
STDMETHODIMP
CMyDispatch::GetTypeInfoCount(UINT FAR* pctinfo)
{
    *pctinfo = 1;
    return NOERROR;
}

/*
 * CMyDispatch::GetTypeInfo
 *
 * Purpose:
 *  Implements IDispatch::GetTypeInfo.
 *
 */
STDMETHODIMP
CMyDispatch::GetTypeInfo(
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo)
{
    *pptinfo = NULL;

    if(itinfo != 0)
        return ResultFromScode(DISP_E_BADINDEX);

    m_ptinfo->AddRef();
    *pptinfo = m_ptinfo;


    return NOERROR;
}

/*
 * CMyDispatch::GetIDsOfNames
 *
 * Purpose:
 *  Implements IDispatch::GetIDsOfNames.  The standard implementation,
DispGetIDsOfNames,
 *  is used.
 *
 */
STDMETHODIMP
CMyDispatch::GetIDsOfNames(
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid)
{
    return DispGetIDsOfNames(m_ptinfo, rgszNames, cNames, rgdispid);
}

/*
 * CMyDispatch::Invoke
 *
 * Purpose:
 *  Implements IDispatch::Invoke.  The standard implementation, DispInvoke,
 *  is used. Properties and methods  will
 *  set m_bRaiseException to raise an exception.
 *
 */
STDMETHODIMP
CMyDispatch::Invoke(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr)
{
    HRESULT hr;

    m_bRaiseException = FALSE;
    hr =  DispInvoke(
        this, m_ptinfo,
        dispidMember, wFlags, pdispparams,
        pvarResult, pexcepinfo, puArgErr);
    if (m_bRaiseException)
    {
       if (NULL != pexcepinfo)
           _fmemcpy(pexcepinfo, &m_excepinfo, sizeof(EXCEPINFO));
       return ResultFromScode(DISP_E_EXCEPTION);
    }
    else return hr;
}

/*
 * CMyDispatch::LoadTypeInfo
 *
 *  Purpose:
 *   Gets type information of an object's interface from type library.
 *
 * Parameters:
 *  clsid               Interface id of object in type library.
 *
 * Return Value:
 *  HRESULT
 *
 */
STDMETHODIMP
CMyDispatch::LoadTypeInfo(REFCLSID clsid)
{
    HRESULT hr;
    LPTYPELIB ptlib = NULL;
    LPTYPEINFO ptinfo = NULL;

    // Load Type Library. If required, notify user on failure.
    hr = LoadRegTypeLib(LIBID_BrowseHelper, 1, 0, 0x0409, &ptlib);
    if (FAILED(hr))
        return hr;

    // Get type information for interface of the object.
    hr = ptlib->GetTypeInfoOfGuid(clsid, &ptinfo);
    if (FAILED(hr))
    {
        ptlib->Release();
        return hr;
    }

    ptlib->Release();
    m_ptinfo = ptinfo;
    return NOERROR;
}

/*
 * CMyDispatch::RaiseException
 *
 *  Purpose:
 *   Raises exception so CMyDispatch::Invoke will return DISP_E_EXCEPTION
 *
 * Parameters:
 *  nID               ID of exception to be raised.
 *
 */
STDMETHODIMP_(void)
CMyDispatch::RaiseException(int nID)
{
    extern HINSTANCE g_hinst;
    extern TCHAR g_szServerName[];
    TCHAR szError[STR_LEN];

    _fmemset(&m_excepinfo, 0, sizeof(EXCEPINFO));

    m_excepinfo.wCode = nID;
    if (LoadString(g_hinst, nID, szError, sizeof(szError)))
        m_excepinfo.bstrDescription =
SysAllocString(TO_OLE_STRING(szError));
    m_excepinfo.bstrSource = SysAllocString(TO_OLE_STRING(g_szServerName));

    m_bRaiseException = TRUE;
}
```

PARAM.CPP (BROWSEH OLE Sample)

```
/*************************************************************************
**
**    OLE Automation TypeLibrary Browse Helper Sample
**
**    param.cpp
**
**    CParameter implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "browseh.h"

/*
 * CParameter::Create
 *
 * Purpose:
 *  Creates an instance of the Parameter automation object and initializes
it.
 *
 * Parameters:
 *  ptinfo         TypeInfo in which the function of this parameter is
contained.
 *  bstrName       Name of parameter
 *  ptypedesc      Type of parameter.
 *  pidldesc       IDLDESC of parameter.
 *  ppParameter    Returns Parameter automation object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CParameter::Create(LPTYPEINFO ptinfo, BSTR bstrName, TYPEDESC FAR*
ptypedesc,
     IDLDESC FAR* pidldesc, CParameter FAR* FAR* ppParameter)
{
    HRESULT hr;
    CParameter FAR* pParameter = NULL;
    CTypeDesc FAR* pTypeDesc = NULL;
    *ppParameter = NULL;

    // Create object.
    pParameter = new CParameter();
    if (pParameter == NULL)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }
   // Load type information for the object from type library.
    hr = pParameter->LoadTypeInfo(IID_IParameter);
    if (FAILED(hr))
        goto error;

    pParameter->m_bstrName = SysAllocString(bstrName);

    // Parameter type.
    hr = CTypeDesc::Create(ptinfo, ptypedesc, &pTypeDesc);
    if (FAILED(hr))
        goto error;
    pTypeDesc->QueryInterface(IID_IDispatch, (LPVOID FAR*)&pParameter-
>m_pdispTypeDesc);

    pParameter->m_wIDLFlags = pidldesc->wIDLFlags;
#ifdef _DEBUG
    lstrcpyn(pParameter->m_szClassName, TEXT("Parameter"), 100);
#endif

    *ppParameter = pParameter;
    return NOERROR;

error:
    if (pParameter == NULL) return ResultFromScode(E_OUTOFMEMORY);

    delete pParameter;
    return hr;
}

/*
 * CParameter::CParameter
 *
 * Purpose:
 *  Constructor for CParameter object. Initializes members to NULL.
 *
 */
CParameter::CParameter()
{
    m_bstrName = NULL;
    m_pdispTypeDesc = NULL;
}

/*
 * CParameter::~CParameter
 *
 * Purpose:
 *  Destructor for CParameter object.
 *
 */
CParameter::~CParameter()
{
     if (m_bstrName) SysFreeString(m_bstrName);
     if (m_pdispTypeDesc) m_pdispTypeDesc->Release();
}

STDMETHODIMP_(REFCLSID)
CParameter::GetInterfaceID()
{
    return IID_IParameter;
}

STDMETHODIMP_(BSTR)
CParameter::get_Name()
{
    return SysAllocString(m_bstrName);
}


STDMETHODIMP_(ITypeDesc FAR*)
CParameter::get_Type()
{
    m_pdispTypeDesc->AddRef();
    return (ITypeDesc FAR*)m_pdispTypeDesc;
}

STDMETHODIMP_(int)
CParameter::get_IDLFlags()
{
    return (int)m_wIDLFlags;

}

STDMETHODIMP_(OBJTYPE)
CParameter::get_Kind()
{
    return TYPE_PARAMETER;
}
```

HELLCTRL

HELLCTRL is a vtable binding automation controller that controls the HELLO 2.0 sample.

HELLCTRL uses CoCreateInstance to create an instance of Hello and obtains its IHello interface using QueryInterface. It then calls the methods of the IHello interface to access the properties and methods of the Hello object. Rich Error Information is obtained by accessing the IErrorInfo interface.

FILES: ------

tlb.h File copied from the HELLO 2.0 sample. This file was

generated by mktyplib in the HELLO 2.0 sample.

hellctrl.cpp, hellctrl.h Controller source files.

TO RUN: -------

Select the CreateHello menu after registering the HELLO 2.0 sample. This was launch Hello invisibly. Use the SetVisible menu to make it visible.

MAKEFILE (HELLCTRL OLE Sample)

```
####
#makefile - makefile for browse.exe
#
#       Copyright (C) 1994, Microsoft Corporation
#
#Purpose:
#  Builds the OLE 2.0 Automation controller, browse.exe.
#
#
#  Usage: NMAKE                  ; build with defaults
#     or: NMAKE option           ; build with the given option(s)
#     or: NMAKE clean            ; erase all compiled files
#
#     option: dev = [win16 | win32]    ; dev=win32 is the default
#             DEBUG=[0 | 1]            ; DEBUG=1 is the default
#             HOST=[DOS | NT | WIN95]  ; HOST=DOS (for win16)
#                                      ; HOST=NT (for win32 on NT)
#                                      ; HOST=WIN95 (for win32 on Win95)
#
#Notes:
#  This makefile assumes that the PATH, INCLUDE and LIB environment
#  variables are setup properly.
#
############################################################################
##


###########################################################################
#
# Default Settings
#

CPU = i386

!if "$(dev)" == ""
dev = win32
HOST = NT
!endif

!if !("$(dev)" == "win16" || "$(dev)" == "win32")
!error Invalid dev option, choose from [win16 | win32]
!endif

!if "$(dev)" == "win16"
TARGET  = WIN16
!if "$(HOST)" == ""
HOST  = DOS
!endif
!endif

!if "$(dev)" == "win32"
TARGET  = WIN32
```

```
!if "$(HOST)" == ""
HOST  = NT
!endif
!endif

!ifdef NODEBUG
DEBUG = 0
!endif

!if "$(DEBUG)" == "0"
NODEBUG = 1
!endif

!if "$(DEBUG)" == ""
DEBUG = 1
!endif


##########################################################################
#
# WIN16 Settings
#
!if "$(TARGET)" == "WIN16"

CC   = cl
LINK = link
!if "$(HOST)" == "DOS"
WX   = wx /w
!else
WX   =
!endif

RCFLAGS = -dWIN16
CFLAGS = -c -W3 -AM -GA -GEs -DWIN16
LINKFLAGS = /NOD /NOI /BATCH /ONERROR:NOEXE

LIBS = libw.lib mlibcew.lib

!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS) -Od -Zi -D_DEBUG $(CL)
LINKFLAGS = $(LINKFLAGS) /COD
!else
CFLAGS = $(CFLAGS) -Ox $(CL)
LINKFLAGS = $(LINKFLAGS) /FAR /PACKC
!endif
!endif


##########################################################################
#
# WIN32 Settings
#
!if "$(TARGET)" == "WIN32"
```

```
WX =

!include <ntwin32.mak>

CC = $(cc)
CFLAGS = $(cflags) $(cvarsmt) -DINC_OLE2 $(cdebug)

!if "$(HOST)" == "NT"
CFLAGS = $(CFLAGS) -DUNICODE
!endif

!ifndef NODEBUG
CFLAGS = $(CFLAGS) -D_DEBUG
!endif

LINK = $(link)
LINKFLAGS = $(linkdebug) $(guilflags)
RCFLAGS = -DWIN32

!endif

##########################################################################
#
# Build rules
#

.cpp.obj:
    @echo Compiling $<...
    $(CC) $<

.c.obj:

    @echo Compiling $<...
    $(CC) $<


##########################################################################
#
# Application Settings
#

APPS = hellctrl


!if "$(TARGET)" == "WIN16"
LIBS = commdlg.lib ole2.lib compobj.lib ole2disp.lib typelib.lib $(LIBS)
!endif
!if "$(TARGET)" == "WIN32"
LIBS = $(ole2libsmt)
!endif

OBJS = hellctrl.obj


##########################################################################
```

```
#
# Default Goal
#

goal : setflags $(APPS).exe

setflags :
    set CL=$(CFLAGS)


##########################################################################
#
# Clean (erase) generated files
#
clean :
    if exist *.obj       del *.obj
    if exist $(APPS).exe del $(APPS).exe
    if exist $(APPS).map del $(APPS).map
    if exist $(APPS).res del $(APPS).res
    if exist *.pdb       del *.pdb


##########################################################################
#
# Application Build (WIN16 Specific)
#

!if "$(TARGET)" == "WIN16"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
    link $(LINKFLAGS) @<<
$(OBJS),
$@,,
$(LIBS),
$(APPS).def
<<
    rc -k -t $(APPS).res $@
!endif


##########################################################################
#
# Application Build (WIN32 Specific)
#
!if "$(TARGET)" == "WIN32"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
      $(LINK) @<<
        $(LINKFLAGS)
        -out:$@
        -map:$*.map
        $(OBJS)
        $(APPS).res
        $(LIBS)
<<
!endif
```

```
##########################################################################
#
# Application Build (Common)
#

$(APPS).res : $(APPS).rc
    rc $(RCFLAGS) -r -fo$@ $?


##########################################################################
#
# Dependencies
#

hellctrl.obj : hellctrl.cpp hellctrl.h tlb.h
     $(CC) hellctrl.cpp
```

HELLCTRL.H (HELLCTRL OLE Sample)

```
#ifdef WIN32

#ifdef UNICODE
    #define FROM_OLE_STRING(str) str
    #define TO_OLE_STRING(str) str
#else
    #define FROM_OLE_STRING(str) ConvertToAnsi(str)
    char* ConvertToAnsi(OLECHAR FAR* szW);
    #define TO_OLE_STRING(str) ConvertToUnicode(str)
    OLECHAR* ConvertToUnicode(char FAR* szA);
    // Maximum length of string that can be converted between Ansi & Unicode
    #define STRCONVERT_MAXLEN 500
#endif

#else  // WIN16
  #define APIENTRY far pascal
  #define TCHAR char
  #define TEXT(sz) sz
  #define FROM_OLE_STRING(str) str
  #define TO_OLE_STRING(str) str
  #define LPTSTR LPSTR
  #define LPCTSTR LPCSTR

  // Windows NT defines the following in windowsx.h
  #define GET_WM_COMMAND_ID(w,l) (w)
  #define GET_WM_COMMAND_CMD(w,l) HIWORD(l)
  #define GET_WM_COMMAND_HWND(w,l) LOWORD(l)
#endif

// Menu-item IDs
#define IDM_CREATEHELLO  300
#define IDM_SETVISIBLE   301
#define IDM_SETINVISIBLE   302
#define IDM_GETHELLOMESSAGE   303
#define IDM_SAYHELLO  304
#define IDM_RELEASEHELLO 305

// MAX len of string table entries
#define STR_LEN   100

// String table constants
#define IDS_PROGNAME                  1
#define IDS_RESULT                    2
#define IDS_ERROR                     3

// Function prototypes
int PASCAL WinMain (HINSTANCE, HINSTANCE, LPSTR, int);
BOOL InitApplication (HINSTANCE);
BOOL InitInstance (HINSTANCE, int);
void DisplayError(IHello FAR* phello);
#ifdef WIN16
```

```
LRESULT __export CALLBACK MainWndProc (HWND hwnd, UINT msg, WPARAM wParam,
LPARAM lParam);
#else
LRESULT CALLBACK MainWndProc (HWND hwnd, UINT msg, WPARAM wParam, LPARAM
lParam);
#endif
```

RESOURCE.H (HELLCTRL OLE Sample)

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by hellctrl.ico
//
#define IDI_ICON1                       101

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        102
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

TLB.H (HELLCTRL OLE Sample)

```
/* This header file machine-generated by mktyplib.exe */
/* Interface to type library: Hello */

#ifndef _Hello_H_
#define _Hello_H_

DEFINE_GUID(LIBID_Hello,0xF37C8060,0x4AD5,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x
10,0x3D,0xE1);

DEFINE_GUID(IID_IHello,0xF37C8062,0x4AD5,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x1
0,0x3D,0xE1);

/* Definition of interface: IHello */
DECLARE_INTERFACE_(IHello, IDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* IUnknown methods */
    STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
    STDMETHOD_(ULONG, AddRef)(THIS) PURE;
    STDMETHOD_(ULONG, Release)(THIS) PURE;

    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(THIS_ UINT FAR* pctinfo) PURE;

    STDMETHOD(GetTypeInfo)(
      THIS_
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo) PURE;

    STDMETHOD(GetIDsOfNames)(
      THIS_
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid) PURE;

    STDMETHOD(Invoke)(
      THIS_
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr) PURE;
#endif

    /* IHello methods */
    STDMETHOD(get_Application)(THIS_ IHello FAR* FAR* retval) PURE;
    STDMETHOD(get_FullName)(THIS_ BSTR FAR* retval) PURE;
    STDMETHOD(get_Name)(THIS_ BSTR FAR* retval) PURE;
    STDMETHOD(get_Parent)(THIS_ IHello FAR* FAR* retval) PURE;
    STDMETHOD(put_Visible)(THIS_ VARIANT_BOOL VisibleFlag) PURE;
    STDMETHOD(get_Visible)(THIS_ VARIANT_BOOL FAR* retval) PURE;
    STDMETHOD(Quit)(THIS) PURE;
    STDMETHOD(put_HelloMessage)(THIS_ BSTR Message) PURE;
    STDMETHOD(get_HelloMessage)(THIS_ BSTR FAR* retval) PURE;
    STDMETHOD(SayHello)(THIS) PURE;
};

DEFINE_GUID(CLSID_Hello,0xF37C8061,0x4AD5,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x
10,0x3D,0xE1);

class Hello;

#endif
```

HELLCTRL.CPP   (HELLCTRL OLE Sample)

```
/*************************************************************************
**
**    Automation Controller that uses vtable binding.
**    Controls the HELLO automation object.
**
**    hellctrl.cpp
**
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/
#define STRICT

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
  #include <commdlg.h>
#endif
#include <initguid.h>
#include "tlb.h"
#include "hellctrl.h"

// Globals
HINSTANCE g_hinst;                          // Instance of application
HWND      g_hwnd;                           // Toplevel window handle

// String resource buffers
TCHAR g_szTitle[STR_LEN];                    // Main window caption
TCHAR g_szResult[STR_LEN];                   // "Result"
TCHAR g_szError[STR_LEN];                    // "Error"

/*
 * WinMain
 *
 * Purpose:
 *  Main entry point of application. Should register the app class
 *  if a previous instance has not done so and do any other one-time
 *  initializations.
 *
 */
int APIENTRY WinMain (HINSTANCE hinst, HINSTANCE hinstPrev, LPSTR lpCmdLine,
int nCmdShow)
{
   MSG msg;
   //  It is recommended that all OLE applications set
   //  their message queue size to 96. This improves the capacity
   //  and performance of OLE's LRPC mechanism.
   int cMsg = 96;                  // Recommend msg queue size for OLE
   while (cMsg && !SetMessageQueue(cMsg))  // take largest size we can get.
       cMsg -= 8;
   if (!cMsg)
       return -1;                  // ERROR: we got no message queue

   // Load string constants
   LoadString(hinst, IDS_PROGNAME, g_szTitle, STR_LEN);
   LoadString(hinst, IDS_RESULT, g_szResult, STR_LEN);
   LoadString(hinst, IDS_ERROR, g_szError, STR_LEN);

   if (!hinstPrev)
      if (!InitApplication(hinst))
         return (FALSE);

   if(OleInitialize(NULL) != NOERROR)
      return FALSE;

   if (!InitInstance(hinst, nCmdShow))
      return (FALSE);

   while (GetMessage(&msg, NULL, NULL, NULL))
   {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
   }

   OleUninitialize();

   return (msg.wParam); // Returns the value from PostQuitMessage
}

/*
 * InitApplication
 *
 * Purpose:
 *  Registers window class
 *
 * Parameters:
 *  hinst       hInstance of application
 *
 * Return Value:
 *  TRUE if initialization succeeded, FALSE otherwise.
 */
BOOL InitApplication (HINSTANCE hinst)
{
   WNDCLASS wc;

   wc.style = CS_DBLCLKS;
   wc.lpfnWndProc = MainWndProc;
   wc.cbClsExtra = 0;
   wc.cbWndExtra = 0;
   wc.hInstance = hinst;
   wc.hIcon = LoadIcon(hinst, TEXT("ControlIcon"));
   wc.hCursor = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
   wc.lpszMenuName = TEXT("ControlMenu");
   wc.lpszClassName = TEXT("MainWndClass");

   return RegisterClass(&wc);
 }

/*
 * InitInstance
 *
 * Purpose:
 *  Creates and shows main window
 *
 * Parameters:
 *  hinst           hInstance of application
 *  nCmdShow        specifies how window is to be shown
 *
 * Return Value:
 *  TRUE if initialization succeeded, FALSE otherwise.
 */
BOOL InitInstance (HINSTANCE hinst, int nCmdShow)
{

   g_hinst = hinst;
   // Create Main Window
   g_hwnd = CreateWindow(TEXT("MainWndClass"), g_szTitle,
                       WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       400, 200,
                       NULL, NULL, hinst, NULL);
   if (!g_hwnd)
      return FALSE;

   ShowWindow(g_hwnd, nCmdShow);
   UpdateWindow(g_hwnd);
   return TRUE;
}

/*
 * MainWndProc
 *
 * Purpose:
 *  Window procedure for main window
 *
 */
LRESULT CALLBACK MainWndProc (HWND hwnd, UINT msg, WPARAM wParam, LPARAM
lParam)
{
   static IHello FAR* phello = NULL;
   HRESULT hr;
   LPUNKNOWN punk;
   switch (msg)
   {
      case WM_COMMAND:
         switch (wParam)
         {
            case IDM_CREATEHELLO:
                // Create Hello object and QueryInterface for IHello
interface.
                hr = CoCreateInstance(CLSID_Hello, NULL, CLSCTX_SERVER,
                     IID_IUnknown, (void FAR* FAR*)&punk);
                if (FAILED(hr))
                {
                    MessageBox(NULL, TEXT("CoCreateInstance"), g_szError,
MB_OK);
                    return 0L;
                }
                hr = punk->QueryInterface(IID_IHello,  (void FAR*
FAR*)&phello);
                if (FAILED(hr))
                {
                    MessageBox(NULL, TEXT("QueryInterface(IID_IHello)"),
g_szError, MB_OK);
                    punk->Release();
                    return 0L;
                }
                punk->Release();
                return 0L;

            case IDM_SETVISIBLE:
                // Set Visible property to TRUE
                hr = phello->put_Visible(TRUE);
                if (FAILED(hr))
                    DisplayError(phello);
                return 0L;

            case IDM_SETINVISIBLE:
                // Set visible property to FALSE
                hr = phello->put_Visible(FALSE);
                if (FAILED(hr))
                    DisplayError(phello);
                return 0L;

            case IDM_GETHELLOMESSAGE:
            {
                // Access Hello Message property and display it
                // in a MessageBox
                BSTR bstr = NULL;   // BSTR must be intialized before
passing
                                    // to get_HelloMessage.
                hr = phello->get_HelloMessage(&bstr);
                if (FAILED(hr))
                    DisplayError(phello);
                else MessageBox(NULL, FROM_OLE_STRING(bstr), g_szResult,
MB_OK);
```

```
                // Caller is responsible for freeing parameters and return
values.
                if (bstr)
                    SysFreeString(bstr);
                return 0L;
            }

            case IDM_SAYHELLO:
                // Invoke SayHello method
                hr = phello->SayHello();
                if (FAILED(hr))
                    DisplayError(phello);
                return 0L;

            case IDM_RELEASEHELLO:
                // Release the Hello object
                phello->Release();
                phello = NULL;
                return 0L;
         }
         break;

      case WM_INITMENUPOPUP:
      {
         HMENU hmenu = (HMENU)wParam;

         if (LOWORD(lParam) != 0)
            return 0L;

         // Enable or gray the appropriate menu items. phello indicates if
an automation object
         //  is currently being controlled.
         EnableMenuItem(hmenu, IDM_CREATEHELLO,  MF_BYCOMMAND | (phello?
MF_GRAYED:MF_ENABLED));
         EnableMenuItem(hmenu, IDM_SETVISIBLE,   MF_BYCOMMAND | (phello?
MF_ENABLED:MF_GRAYED));
         EnableMenuItem(hmenu, IDM_SETINVISIBLE,   MF_BYCOMMAND | (phello?
MF_ENABLED:MF_GRAYED));
         EnableMenuItem(hmenu, IDM_GETHELLOMESSAGE,   MF_BYCOMMAND |
(phello?MF_ENABLED:MF_GRAYED));
         EnableMenuItem(hmenu, IDM_SAYHELLO,  MF_BYCOMMAND | (phello?
MF_ENABLED:MF_GRAYED));
         EnableMenuItem(hmenu, IDM_RELEASEHELLO, MF_BYCOMMAND | (phello?
MF_ENABLED:MF_GRAYED));
         return 0L;
      }

      case WM_DESTROY:
         if (phello)
            phello->Release();
         PostQuitMessage(0);
         break;

      default:
         return DefWindowProc(hwnd, msg, wParam, lParam);
   }

   return NULL;
}

/*
 * DisplayError
 *
 * Purpose:
 *  Obtains Rich Error Information about the automation error from
 *  the IErrorInfo interface.
 *
 */
void DisplayError(IHello FAR* phello)
{
   IErrorInfo FAR* perrinfo;
   BSTR bstrDesc;
   HRESULT hr;

 /*
   // This is commented out because the OLE 2.02 release did not
   // include remoting code for ISupportErrorInfo.

   ISupportErrorInfo FAR* psupporterrinfo;

   hr = phello->QueryInterface(IID_ISupportErrorInfo, (LPVOID
FAR*)&psupporterrinfo);
   if (FAILED(hr))
   {
      MessageBox(NULL, TEXT("QueryInterface(IID_ISupportErrorInfo)"),
g_szError, MB_OK);
      return;
   }

   hr = psupporterrinfo->InterfaceSupportsErrorInfo(IID_IHello);
   if (hr != NOERROR)
   {
       psupporterrinfo->Release();
       return;
   }
   psupporterrinfo->Release();
*/
   // In this example only the error description is obtained and displayed.
   // See the IErrorInfo interface fo other information that is available.
   hr = GetErrorInfo(0, &perrinfo);
   if (FAILED(hr))
       return;
   hr = perrinfo->GetDescription(&bstrDesc);
   if (FAILED(hr))
   {
       perrinfo->Release();
       return;
   }
   MessageBox(NULL, FROM_OLE_STRING(bstrDesc), g_szError, MB_OK);
   SysFreeString(bstrDesc);
}

#ifdef WIN32

#ifndef UNICODE
char* ConvertToAnsi(OLECHAR FAR* szW)
{
  static char achA[STRCONVERT_MAXLEN];

  WideCharToMultiByte(CP_ACP, 0, szW, -1, achA, STRCONVERT_MAXLEN, NULL,
NULL);
  return achA;
}

OLECHAR* ConvertToUnicode(char FAR* szA)
{
  static OLECHAR achW[STRCONVERT_MAXLEN];

  MultiByteToWideChar(CP_ACP, 0, szA, -1, achW, STRCONVERT_MAXLEN);
  return achW;
}
#endif

#endif
```

HELLO

SAMPLE: HELLO: Simple OLE Automation Server

HELLO is a simple OLE Automation server application. This is a good sample to use to learn to create your first OLE Automation server. This has one object that supports the following properties and methods:

Hello Object:

```
 Properties
      HelloMessage
 Methods
      SayHello
```

The sample the following features:

· Supports dual interfaces which allows access of automation properties and methods through vtable binding and through IDispatch.
· Provides Rich Error information for vtable-binding controllers. This is does by implementing ISupportErrorInfo and using IErrorInfo.
· Implements active object registration using RegisterActiveObject and RevokeActiveObject. · Implements correct shut-down behavior. · Includes a .reg file which contains Hello.Application and Hello.Application.2 as progIDs. · When created for automation, is invisible initially.

To compile: -----------

Requires OLE 2.02 or later. Use the external makefile called makefile to compile. In Win16, run the WXSRVER.EXE from \OLE2\BIN before running the makefile. The makefile invokes mktyplib.exe that reads hello.odl and creates the type library, hello.tlb. It then compliles the source files.

To run: -------

Change hello.reg to provide the full path of hello.exe and hello.tlb. Register hello.reg in the registration database by double-clicking it. The HELLCTRL directory contains a controller that will control HELLO using vtable-binding.

Files: ------

HELLO.ODL Object description language that describes the property and method that HELLO exposes.
TLB.H Header file generated by mktyplib.exe MAKEFILE Makefile for project.

Other files implement the HELLO automation server.
==

MAKEFILE (HELLO OLE Sample)

```
####
#makefile - makefile for hello.exe
#
#       Copyright (C) 1994, Microsoft Corporation
#
#Purpose:
#  Builds the OLE 2.0 Automation object, hello.exe.
#
#
#  Usage: NMAKE                 ; build with defaults
#     or: NMAKE option          ; build with the given option(s)
#     or: NMAKE clean           ; erase all compiled files
#
#     option: dev = [win16 | win32]    ; dev=win32 is the default
#             DEBUG=[0 | 1]            ; DEBUG=1 is the default
#             HOST=[DOS | NT | WIN95]  ; HOST=DOS (for win16)
#                                      ; HOST=NT (for win32 on NT)
#                                      ; HOST=WIN95 (for win32 on Win95)
#
#Notes:
#  This makefile assumes that the PATH, INCLUDE and LIB environment
#  variables are setup properly.
#
############################################################################
##


##########################################################################
#
# Default Settings
#

CPU = i386

!if "$(dev)" == ""
dev = win32
HOST = NT
!endif

!if !("$(dev)" == "win16" || "$(dev)" == "win32")
!error Invalid dev option, choose from [win16 | win32]
!endif

!if "$(dev)" == "win16"
TARGET  = WIN16
!if "$(HOST)" == ""
HOST  = DOS
!endif
!endif

!if "$(dev)" == "win32"
TARGET  = WIN32
```

```
!if "$(HOST)" == ""
HOST  = NT
!endif
!endif

!ifdef NODEBUG
DEBUG = 0
!endif

!if "$(DEBUG)" == "0"
NODEBUG = 1
!endif

!if "$(DEBUG)" == ""
DEBUG = 1
!endif


##########################################################################
#
# WIN16 Settings
#
!if "$(TARGET)" == "WIN16"

CC   = cl
LINK = link
!if "$(HOST)" == "DOS"
WX   = wx /w
!else
WX   =
!endif

RCFLAGS = -dWIN16
CFLAGS = -c -W3 -AM -GA -GEs -DWIN16
LINKFLAGS = /NOD /NOI /BATCH /ONERROR:NOEXE

LIBS = libw.lib mlibcew.lib

!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS) -Od -Zi -D_DEBUG $(CL)
LINKFLAGS = $(LINKFLAGS) /COD
!else
CFLAGS = $(CFLAGS) -Ox $(CL)
LINKFLAGS = $(LINKFLAGS) /FAR /PACKC
!endif
!endif


##########################################################################
#
# WIN32 Settings
#
!if "$(TARGET)" == "WIN32"
```

```
WX =

!include <ntwin32.mak>

CC = $(cc)
CFLAGS = $(cflags) $(cvarsmt) -DINC_OLE2 $(cdebug)

!if "$(HOST)" == "NT"
CFLAGS = $(CFLAGS) -DUNICODE
!endif

!ifndef NODEBUG
CFLAGS = $(CFLAGS) -D_DEBUG
!endif

LINK = $(link)
LINKFLAGS = $(linkdebug) $(guilflags)
RCFLAGS = -DWIN32

!endif

##########################################################################
#
# Build rules
#

.cpp.obj:
    @echo Compiling $<...
    $(CC) $<

.c.obj:

    @echo Compiling $<...
    $(CC) $<


##########################################################################
#
# Application Settings
#

APPS = hello


!if "$(TARGET)" == "WIN16"
LIBS = ole2.lib compobj.lib ole2disp.lib typelib.lib $(LIBS)
!endif
!if "$(TARGET)" == "WIN32"
LIBS = $(ole2libsmt)
!endif

OBJS = main.obj hello.obj hellocf.obj


##########################################################################
```

```
#
# Default Goal
#

goal : setflags $(APPS).exe

setflags :
    set CL=$(CFLAGS)


###########################################################################
#
# Clean (erase) generated files
#
clean :
    if exist *.obj       del *.obj
    if exist $(APPS).exe del $(APPS).exe
    if exist $(APPS).tlb del $(APPS).tlb
    if exist $(APPS).map del $(APPS).map
    if exist $(APPS).res del $(APPS).res
    if exist tlb.h       del tlb.h
    if exist *.log       del *.log
    if exist *.pdb       del *.pdb


###########################################################################
#
# Application Build (WIN16 Specific)
#

!if "$(TARGET)" == "WIN16"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
    link $(LINKFLAGS) @<<
$(OBJS),
$@,,
$(LIBS),
$(APPS).def
<<
    rc -k -t $(APPS).res $@
!endif


###########################################################################
#
# Application Build (WIN32 Specific)
#
!if "$(TARGET)" == "WIN32"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
      $(LINK) @<<
        $(LINKFLAGS)
        -out:$@
        -map:$*.map
        $(OBJS)
        $(APPS).res
        $(LIBS)
```

```
<<
!endif


##########################################################################
#
# Application Build (Common)
#

$(APPS).res : $(APPS).rc
    rc $(RCFLAGS) -r -fo$@ $?


##########################################################################
#
# Dependencies
#

tlb.h : hello.odl
     if exist tlb.h  del tlb.h
     if exist hello.tlb  del hello.tlb
     $(WX) mktyplib /D$(TARGET) /h tlb.h /o hello.log /tlb hello.tlb
hello.odl
     type hello.log

main.obj : main.cpp hello.h tlb.h
     $(CC) main.cpp
hello.obj : hello.cpp hello.h tlb.h
     $(CC) hello.cpp
hellocf.obj : hellocf.cpp hello.h tlb.h
     $(CC) hellocf.cpp
```

HELLO.H (HELLO OLE Sample)

```
#ifdef WIN32

#ifdef UNICODE
    #define FROM_OLE_STRING(str) str
    #define TO_OLE_STRING(str) str
#else
    #define FROM_OLE_STRING(str) ConvertToAnsi(str)
    char* ConvertToAnsi(OLECHAR FAR* szW);
    #define TO_OLE_STRING(str) ConvertToUnicode(str)
    OLECHAR* ConvertToUnicode(char FAR* szA);
    // Maximum length of string that can be converted between Ansi & Unicode
    #define STRCONVERT_MAXLEN 300
#endif

#else  // WIN16
  #define APIENTRY far pascal
  #define TCHAR char
  #define TEXT(sz) sz
  #define FROM_OLE_STRING(str) str
  #define TO_OLE_STRING(str) str
  #define LPTSTR LPSTR

  // Windows NT defines the following in windowsx.h
  #define GET_WM_COMMAND_ID(w,l) (w)
  #define GET_WM_COMMAND_CMD(w,l) HIWORD(l)
  #define GET_WM_COMMAND_HWND(w,l) LOWORD(l)
#endif

#define STRICT

// Resource IDs
#define IDD_MAINWINDOW   101
#define IDI_ICON         102

// Child window IDs
#define IDC_HELLODISPLAY 101
#define IDC_SAYHELLO     102

// MAX len of string table entries
#define STR_LEN   100

// String table constants
#define IDS_Name                 1
#define IDS_HelloMessage         2
#define IDS_ErrorLoadingTypeLib  3
#define IDS_SayHello             4
#define IDS_ProgID               5
#define IDS_Unexpected           1001

// SCODEs for the Hello Application. This is required for vtable-binding
// Automation objects want to return custom HRESULTs. All the OLE-defined
// FACILITY_ITF codes have a code value which lies in the region 0x0000-
0x01FFF.
// While it is legal for the definer to use any code, it is highly
recommended
// that only code values in the range 0x0200-0xFFFF be used, as this will
reduce the
// possiblity of accidental confusion with any OLE-defined errors.
#define HELLO_E_FIRST       MAKE_SCODE(SEVERITY_ERROR, FACILITY_ITF, 0x0200)

#define HELLO_E_UNEXPECTED  (HELLO_E_FIRST + 0x0)
// Unexpected error

// Number of SCODEs
#define SCODE_COUNT 1

// Function prototypes
int APIENTRY WinMain (HINSTANCE, HINSTANCE, LPSTR, int);
BOOL InitApplication (HINSTANCE);
BOOL InitInstance (HINSTANCE);
BOOL ProcessCmdLine(LPSTR lpCmdLine, LPDWORD pdwRegisterCF, LPDWORD
pdwRegisterActiveObject, int nCmdShow);
void Uninitialize(DWORD dwRegisterCF, DWORD dwRegisterActiveObject);
HRESULT LoadTypeInfo(ITypeInfo FAR* FAR* pptinfo, REFCLSID clsid);
#ifdef WIN16
extern "C" LRESULT __export CALLBACK MainWndProc (HWND, UINT, WPARAM,
LPARAM);
#else
extern "C" LRESULT CALLBACK MainWndProc (HWND, UINT, WPARAM, LPARAM);
#endif

#include "tlb.h"

// ISupportErrorInfo interface implementation
interface CSupportErrorInfo : public ISupportErrorInfo
{
public:
    // IUnknown methods
    STDMETHOD(QueryInterface)(REFIID riid, LPVOID FAR* ppvObj);
    STDMETHOD_(ULONG, AddRef)(void);
    STDMETHOD_(ULONG, Release)(void);

    // ISupportErrorInfo method
    STDMETHOD(InterfaceSupportsErrorInfo)(REFIID riid);

    CSupportErrorInfo(IUnknown FAR* punkObject, REFIID riid);
private:
    LPUNKNOWN m_punkObject;  // IUnknown of Object that implements this
interface
    GUID m_iid;
};

class FAR CHello : public IHello
{
public:
    // IUnknown methods
    STDMETHOD(QueryInterface)(REFIID riid, LPVOID FAR* ppvObj);
    STDMETHOD_(ULONG, AddRef)(void);
    STDMETHOD_(ULONG, Release)(void);

    // IDispatch methods
    STDMETHOD(GetTypeInfoCount)(UINT FAR* pctinfo);
    STDMETHOD(GetTypeInfo)(
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo);
    STDMETHOD(GetIDsOfNames)(
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid);
    STDMETHOD(Invoke)(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr);

    // IHello methods
    STDMETHOD(get_Application)(IHello FAR* FAR* ppHello);
    STDMETHOD(get_FullName)(BSTR FAR* pbstr);
    STDMETHOD(get_Name)(BSTR FAR* pbstr);
    STDMETHOD(get_Parent)(IHello FAR* FAR* ppHello);
    STDMETHOD(put_Visible)(VARIANT_BOOL bVisible);
    STDMETHOD(get_Visible)(VARIANT_BOOL FAR* pbool);
    STDMETHOD(Quit)();
    STDMETHOD(put_HelloMessage)(BSTR bstrMessage);
    STDMETHOD(get_HelloMessage)(BSTR FAR* pbstrMessage);
    STDMETHOD(SayHello)();

    // CHello methods
    STDMETHOD(RaiseException)(int nID);
    STDMETHOD_(void, ShowWindow)(int nCmdShow);
    static HRESULT Create(HINSTANCE hinst, LPTSTR lpszHelloMessage, CHello
FAR* FAR* pphello); // Creates and intializes Hello object
    CHello();
    ~CHello();

public:
    HWND m_hwnd;                   // Application window.
    HINSTANCE m_hinst;             // Hinstance of application.
    BSTR m_bstrName;               // Name of application.
     BSTR m_bstrProgID;             // ProgID of application.

private:
    LPTYPEINFO m_ptinfo;           // Type information of Hello application 
interface.
    BSTR m_bstrHelloMsg;           // Hello message
    BSTR m_bstrFullName;           // Full name of application.
    BOOL m_bVisible;               // Is window visible?
    EXCEPINFO m_excepinfo;         // Information to raise an exception on 
error.
    BOOL m_bRaiseException;        // Properties and methods use this to 
signal that an exception is to be raised.
    ULONG m_cRef;                  // Reference count
    CSupportErrorInfo m_SupportErrorInfo; // ISupportErrorInfo interface 
implementation
};

class FAR CHelloCF : public IClassFactory
{
public:
    // IUnknown methods
    STDMETHOD(QueryInterface)(REFIID riid, LPVOID FAR* ppvObj);
    STDMETHOD_(ULONG, AddRef)(void);
    STDMETHOD_(ULONG, Release)(void);

    // IClassFactory methods
    STDMETHOD(CreateInstance)(IUnknown FAR* punkOuter, REFIID riid,
                              void FAR* FAR* ppv);
    STDMETHOD(LockServer)(BOOL fLock);

    CHelloCF();

private:
    ULONG m_cRef;                   // Reference count
};

extern CHello FAR* g_phello;
```

TLB.H (HELLO OLE Sample)

```
/* This header file machine-generated by mktyplib.exe */
/* Interface to type library: Hello */

#ifndef _Hello_H_
#define _Hello_H_

DEFINE_GUID(LIBID_Hello,0xF37C8060,0x4AD5,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x
10,0x3D,0xE1);

DEFINE_GUID(IID_IHello,0xF37C8062,0x4AD5,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x1
0,0x3D,0xE1);

/* Definition of interface: IHello */
DECLARE_INTERFACE_(IHello, IDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* IUnknown methods */
    STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
    STDMETHOD_(ULONG, AddRef)(THIS) PURE;
    STDMETHOD_(ULONG, Release)(THIS) PURE;

    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(THIS_ UINT FAR* pctinfo) PURE;

    STDMETHOD(GetTypeInfo)(
      THIS_
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo) PURE;

    STDMETHOD(GetIDsOfNames)(
      THIS_
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid) PURE;

    STDMETHOD(Invoke)(
      THIS_
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr) PURE;
#endif

    /* IHello methods */
    STDMETHOD(get_Application)(THIS_ IHello FAR* FAR* retval) PURE;
    STDMETHOD(get_FullName)(THIS_ BSTR FAR* retval) PURE;
    STDMETHOD(get_Name)(THIS_ BSTR FAR* retval) PURE;
    STDMETHOD(get_Parent)(THIS_ IHello FAR* FAR* retval) PURE;
    STDMETHOD(put_Visible)(THIS_ VARIANT_BOOL VisibleFlag) PURE;
    STDMETHOD(get_Visible)(THIS_ VARIANT_BOOL FAR* retval) PURE;
    STDMETHOD(Quit)(THIS) PURE;
    STDMETHOD(put_HelloMessage)(THIS_ BSTR Message) PURE;
    STDMETHOD(get_HelloMessage)(THIS_ BSTR FAR* retval) PURE;
    STDMETHOD(SayHello)(THIS) PURE;
};

DEFINE_GUID(CLSID_Hello,0xF37C8061,0x4AD5,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x
10,0x3D,0xE1);

class Hello;

#endif
```

HELLO.CPP   (HELLO OLE Sample)

```
/*************************************************************************
**
**    OLE Automation Hello 2.0 Application.
**
**    hello.cpp
**
**    CHello implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "hello.h"

/*
 * CHello::Create
 *
 * Purpose:
 *  Creates an instance of the Hello Application object and initializes it.
 *
 * Parameters:
 *  hinst      HINSTANCE of application.
 *  lpszHelloMessage Initial Hello message.
 *  pphello    Returns Hello automation object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CHello::Create(HINSTANCE hinst, LPTSTR lpszHelloMessage, CHello FAR* FAR*
pphello)
{
    TCHAR ach[STR_LEN];
    TCHAR achFullName[260];
    HRESULT hr;
    CHello FAR* phello = NULL;
    HWND hwnd;

    *pphello = NULL;
    // Create Hello Application object
    phello = new CHello();
    if (phello == NULL)
        goto error;

    // Create Main Window
    hwnd = CreateDialog(hinst, MAKEINTRESOURCE(IDD_MAINWINDOW), NULL, NULL);
    if (!hwnd)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }

    phello->m_cRef = 0;
    phello->m_bVisible = FALSE;
    phello->m_hwnd = hwnd;
    phello->m_hinst = hinst;

    // FullName
    GetModuleFileName(hinst, achFullName, sizeof(achFullName));
    phello->m_bstrFullName = SysAllocString(TO_OLE_STRING(achFullName));
    if (NULL == phello->m_bstrFullName)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }
    // Name
    LoadString(hinst, IDS_Name, ach, sizeof(ach));
    phello->m_bstrName = SysAllocString(TO_OLE_STRING(ach));
    if (NULL == phello->m_bstrName)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }
    // ProgID
    LoadString(hinst, IDS_ProgID, ach, sizeof(ach));
    phello->m_bstrProgID = SysAllocString(TO_OLE_STRING(ach));
    if (NULL == phello->m_bstrProgID)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }
    // Hello Message
    phello->m_bstrHelloMsg =
SysAllocString(TO_OLE_STRING(lpszHelloMessage));
    if (NULL == phello->m_bstrHelloMsg)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }

    // Load type information from type library. If required, notify user on
failure.
    hr = LoadTypeInfo(&phello->m_ptinfo, IID_IHello);
    if (FAILED(hr))
    {
        LoadString(hinst, IDS_ErrorLoadingTypeLib, ach, sizeof(ach));
        MessageBox(NULL, ach, FROM_OLE_STRING(phello->m_bstrName), MB_OK);
        goto error;
    }

    *pphello = phello;
    return NOERROR;

error:
    if (phello == NULL) return ResultFromScode(E_OUTOFMEMORY);
    if (phello->m_bstrFullName) SysFreeString(phello->m_bstrFullName);
    if (phello->m_bstrName) SysFreeString(phello->m_bstrName);
    if (phello->m_bstrProgID) SysFreeString(phello->m_bstrProgID);
    if (phello->m_bstrHelloMsg) SysFreeString(phello->m_bstrHelloMsg);
    if (phello->m_ptinfo) phello->m_ptinfo->Release();

    // Set to NULL to prevent destructor from attempting to free again
    phello->m_bstrFullName = NULL;
    phello->m_bstrName = NULL;
     phello->m_bstrProgID = NULL;
    phello->m_bstrHelloMsg = NULL;
    phello->m_ptinfo = NULL;

    delete phello;

    return hr;
}

/*
 * CHello::CHello
 *
 * Purpose:
 *  Constructor for CHello object. Initializes members to NULL.
 *
 */
#pragma warning (disable : 4355)
CHello::CHello() : m_SupportErrorInfo(this, IID_IHello)
#pragma warning (default : 4355)
{
    extern ULONG g_cObj;

    m_hwnd = NULL;
    m_bstrFullName = NULL;
    m_bstrName = NULL;
     m_bstrProgID = NULL;
    m_bstrHelloMsg = NULL;
    m_ptinfo = NULL;
    m_bVisible = 0;
}

/*
 * CHello::~CHello
 *
 * Purpose:
 *  Destructor for CHello object.
 *
 */
CHello::~CHello()
{
     extern ULONG g_cObj;

     if (m_bstrFullName) SysFreeString(m_bstrFullName);
     if (m_bstrName) SysFreeString(m_bstrName);
      if (m_bstrProgID) SysFreeString(m_bstrProgID);
     if (m_bstrHelloMsg) SysFreeString(m_bstrHelloMsg);
     if (m_ptinfo) m_ptinfo->Release();
     if (IsWindow(m_hwnd)) DestroyWindow(m_hwnd);
}

/*
 * CHello::QueryInterface, AddRef, Release
 *
 * Purpose:
 *  Implements IUnknown::QueryInterface, AddRef, Release
 *
 */
STDMETHODIMP
CHello::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
    *ppv = NULL;

    if (iid == IID_IUnknown || iid == IID_IDispatch || iid == IID_IHello  )
        *ppv = this;
    else if (iid == IID_ISupportErrorInfo)
        *ppv = &m_SupportErrorInfo;
    else return ResultFromScode(E_NOINTERFACE);

    AddRef();
    return NOERROR;
}


STDMETHODIMP_(ULONG)
CHello::AddRef(void)
{
#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("AddRef: Ref = %ld, Hello\r\n"), m_cRef+1);
    OutputDebugString(ach);
#endif

    return ++m_cRef;
}

STDMETHODIMP_(ULONG)
CHello::Release(void)
{
#ifdef _DEBUG
```

```
    TCHAR ach[50];
    wsprintf(ach, TEXT("Release: Ref = %ld, Hello\r\n"), m_cRef-1);
    OutputDebugString(ach);
#endif

    if(--m_cRef == 0)
    {
        delete this;
        return 0;
    }
    return m_cRef;
}

/*
 * CHello::GetTypeInfoCount
 *
 * Purpose:
 *  Implements IDispatch::GetTypeInfoCount.
 *
 */
STDMETHODIMP
CHello::GetTypeInfoCount(UINT FAR* pctinfo)
{
    *pctinfo = 1;
    return NOERROR;
}

/*
 * CHello::GetTypeInfo
 *
 * Purpose:
 *  Implements IDispatch::GetTypeInfo.
 *
 */
STDMETHODIMP
CHello::GetTypeInfo(
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo)
{
    *pptinfo = NULL;

    if(itinfo != 0)
        return ResultFromScode(DISP_E_BADINDEX);

    m_ptinfo->AddRef();
    *pptinfo = m_ptinfo;

    return NOERROR;
}

/*
 * CHello::GetIDsOfNames
 *
 * Purpose:
```

```
 *  Implements IDispatch::GetIDsOfNames.  The standard implementation,
DispGetIDsOfNames,
 *  is used.
 *
 */
STDMETHODIMP
CHello::GetIDsOfNames(
      REFIID riid,

      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid)
{
    return DispGetIDsOfNames(m_ptinfo, rgszNames, cNames, rgdispid);
}

/*
 * CHello::Invoke
 *
 * Purpose:
 *  Implements IDispatch::Invoke.  The standard implementation, DispInvoke,
 *  is used.
 *
 */
STDMETHODIMP
CHello::Invoke(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr)
{
    HRESULT hr;

    m_bRaiseException = FALSE;
    hr =  DispInvoke(
        this, m_ptinfo,
        dispidMember, wFlags, pdispparams,
        pvarResult, pexcepinfo, puArgErr);
    if (m_bRaiseException)
    {
        if (NULL != pexcepinfo)
           _fmemcpy(pexcepinfo, &m_excepinfo, sizeof(EXCEPINFO));
       return ResultFromScode(DISP_E_EXCEPTION);
    }
    else return hr;
}

/*
 * CHello::put_HelloMessage, get_HelloMessage, SayHello
 *
 * Purpose:
 *  Implements the standard Application, FullName, Name, Parent & Visible
properties
 *  and the Quit method.
 *
 */
STDMETHODIMP
CHello::get_Application(IHello FAR* FAR* ppHello)
{
    HRESULT hr;

    *ppHello = NULL;

    hr = QueryInterface(IID_IDispatch, (void FAR* FAR*)ppHello);
    if (FAILED(hr))
        return RaiseException(IDS_Unexpected);
    return hr;
}

STDMETHODIMP
CHello::get_FullName(BSTR FAR* pbstr)
{
    *pbstr = SysAllocString(m_bstrFullName);
    return NOERROR;
}

STDMETHODIMP
CHello::get_Name(BSTR FAR* pbstr)
{
    *pbstr = SysAllocString(m_bstrName);
    return NOERROR;
}

STDMETHODIMP
CHello::get_Parent(IHello FAR* FAR* ppHello)
{
    HRESULT hr;

    *ppHello = NULL;

    hr = QueryInterface(IID_IDispatch, (void FAR* FAR*)ppHello);
    if (FAILED(hr))
        return RaiseException(IDS_Unexpected);
    return NOERROR;
}

STDMETHODIMP
CHello::put_Visible(VARIANT_BOOL bVisible)
{
    ShowWindow(bVisible ? SW_SHOW : SW_HIDE);
    return NOERROR;
}

STDMETHODIMP
CHello::get_Visible(VARIANT_BOOL FAR* pbool)
```

```
{
    *pbool =  m_bVisible;
    return NOERROR;
}

STDMETHODIMP
CHello::Quit()
{
    // CoDisconnectObject has no effect for an inproc server.  So the
controller
    // will GP fault if it attempts to access the object (including calling
IUnknown::Release())
    // after Quit has been called. For a local server, CoDisconnectObject
will disconnect
    // the object from external connections. So the controller will get an
RPC error if
    // it accesses the object after calling Quit and controller will not GP
fault.
    CoDisconnectObject((LPUNKNOWN)this, 0);
    PostMessage(m_hwnd, WM_CLOSE, 0, 0L);
    return NOERROR;
}

/*
 * CHello::put_HelloMessage, get_HelloMessage, SayHello
 *
 * Purpose:
 *  Implements the HelloMessage property and the SayHello method.
 *
 */
STDMETHODIMP
CHello::put_HelloMessage(BSTR bstrMessage)
{
    SysReAllocString(&m_bstrHelloMsg, bstrMessage);
    return NOERROR;
}

STDMETHODIMP
CHello::get_HelloMessage(BSTR FAR* pbstrMessage)
{
    *pbstrMessage = SysAllocString(m_bstrHelloMsg);
    return NOERROR;
}


STDMETHODIMP
CHello::SayHello()
{
    SetDlgItemText(m_hwnd, IDC_HELLODISPLAY,
FROM_OLE_STRING(m_bstrHelloMsg));
    return NOERROR;
}

STDMETHODIMP_(void)
CHello::ShowWindow(int nCmdShow)
```

```
{
    // Return if curently hidden and asked to hide or currently visible
    // and asked to show.
    if ((!m_bVisible && nCmdShow == SW_HIDE) || (m_bVisible && nCmdShow !=
SW_HIDE))
        return;

    m_bVisible = (nCmdShow == SW_HIDE) ? FALSE : TRUE;

    // The Automation object shutdown behavior is as follows:
    // 1. If the object application is visible, it shuts down only in
response to an
    // explicit user command (File/Exit) or it's programmatic equivalent
(for example
    // the Quit method of the Application object).
    // 2. If the object application is not visible, it goes away when it's
last
    // object is released.
    //
    // CoLockObjectExternal can be used to increment the ref count of the
application object
    // when it is visible. This will implement shutdown behavior 1. When the
application
    // goes invisible, CoLockObjectExternal is used to decrement the ref
count. This will
    // implement shutdown behavior 2.

    if (m_bVisible)
        CoLockObjectExternal(this, TRUE /*fLock*/, TRUE/*ignored when
fLock==TRUE*/);
    else CoLockObjectExternal(this, FALSE/*fLock*/,
TRUE/*fLastLockReleases*/);
    ::ShowWindow (m_hwnd, nCmdShow);
}


/*
 * CHello::RaiseException
 *
 * Purpose:
 *  Fills the EXCEPINFO structure and signal IDispatch::Invoke to return
DISP_E_EXCEPTION.
 *  Sets ErrorInfo object for vtable-binding controllers.
 *
 */
STDMETHODIMP
CHello::RaiseException(int nID)
{
    extern SCODE g_scodes[];
    TCHAR szError[STR_LEN];
    ICreateErrorInfo *pcerrinfo;
    IErrorInfo *perrinfo;
    HRESULT hr;

    _fmemset(&m_excepinfo, 0, sizeof(EXCEPINFO));
    m_excepinfo.wCode = nID;
    if (LoadString(m_hinst, nID, szError, sizeof(szError)))
        m_excepinfo.bstrDescription =
SysAllocString(TO_OLE_STRING(szError));
    m_excepinfo.bstrSource = SysAllocString(m_bstrProgID);

    m_bRaiseException = TRUE;

    // Set ErrInfo object so that vtable binding containers can get
    // rich error information.
    hr = CreateErrorInfo(&pcerrinfo);
    if (SUCCEEDED(hr))
    {
       pcerrinfo->SetGUID(IID_IHello);
       if (m_excepinfo.bstrSource)
           pcerrinfo->SetSource(m_excepinfo.bstrSource);
       if (m_excepinfo.bstrDescription)
           pcerrinfo->SetDescription(m_excepinfo.bstrDescription);
       hr = pcerrinfo->QueryInterface(IID_IErrorInfo, (LPVOID FAR*)
&perrinfo);
       if (SUCCEEDED(hr))
       {
          SetErrorInfo(0, perrinfo);
          perrinfo->Release();
       }
       pcerrinfo->Release();
    }

    return ResultFromScode(g_scodes[nID-1001]);
}


/*
 * ISupportErrorInfo implementation
 *
 */
CSupportErrorInfo::CSupportErrorInfo(IUnknown FAR* punkObject, REFIID riid)
{
    m_punkObject = punkObject;
    m_iid = riid;
}

STDMETHODIMP
CSupportErrorInfo::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
    return m_punkObject->QueryInterface(iid, ppv);
}


STDMETHODIMP_(ULONG)
CSupportErrorInfo::AddRef(void)
{
    return m_punkObject->AddRef();
}
```

```
STDMETHODIMP_(ULONG)
CSupportErrorInfo::Release(void)
{
    return m_punkObject->Release();
}

STDMETHODIMP
CSupportErrorInfo::InterfaceSupportsErrorInfo(REFIID riid)
{
    return (riid == m_iid) ? NOERROR : ResultFromScode(S_FALSE);
}
```

HELLOCF.CPP (HELLO OLE Sample)

```
/*************************************************************************
**
**    OLE Automation Hello 2.0 Application.
**
**    hellocf.cpp
**
**    CHelloCF (class factory) implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/
#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "hello.h"

CHelloCF::CHelloCF(void)
{
    m_cRef = 0;
}

/*
 * CHelloCF::QueryInterface, AddRef, Release
 *
 * Purpose:
 *  Implements IUnknown::QueryInterface, AddRef, Release
 *
 */
STDMETHODIMP
CHelloCF::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
    *ppv = NULL;

    if (iid == IID_IUnknown || iid == IID_IClassFactory)
        *ppv = this;
    else
        return ResultFromScode(E_NOINTERFACE);

    AddRef();
    return NOERROR;
}


STDMETHODIMP_(ULONG)
```

```
CHelloCF::AddRef(void)
{
#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, Hello Class Factory\r\n"), m_cRef+1);
    OutputDebugString(ach);
#endif

    return ++m_cRef;
}


STDMETHODIMP_(ULONG)
CHelloCF::Release(void)
{
#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, Hello Class Factory\r\n"), m_cRef-1);
    OutputDebugString(ach);
#endif

    if(--m_cRef == 0)
    {
        delete this;
        return 0;
    }
    return m_cRef;
}

/*
 * CHelloCF::CreateInstance, LockServer
 *
 * Purpose:
 *  Implements IClassFactory::CreateInstance, LockServer
 *
 */
STDMETHODIMP
CHelloCF::CreateInstance(IUnknown FAR* punkOuter,
                         REFIID riid,
                         void FAR* FAR* ppv)
{
     HRESULT hr;

    *ppv = NULL;

    // This implementation does'nt allow aggregation
    if (punkOuter)
        return ResultFromScode(CLASS_E_NOAGGREGATION);

    // This is REGCLS_SINGLEUSE class factory, so CreateInstance will be
    // called atmost once. An application objects has a REGCLS_SINGLEUSE
class
    // factory. The global application object has already been created when
    // CreateInstance is called. A REGCLS_MULTIPLEUSE class factory's
    // CreateInstance would be called multiple times and would create a new
    // object each time. An MDI application would have a REGCLS_MULTIPLEUSE
    // class factory for it's document objects.
    hr = g_phello->QueryInterface(riid, ppv);
    if (FAILED(hr))
    {
        g_phello->Quit();
        return hr;
    }
    return NOERROR;
}

STDMETHODIMP
CHelloCF::LockServer(BOOL fLock)
{
    CoLockObjectExternal(g_phello, fLock, TRUE);
    return NOERROR;
}
```

MAIN.CPP   (HELLO OLE Sample)

```
/*************************************************************************
**
**    OLE Automation Hello 2.0 Application.
**
**    main.cpp
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/
#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include <initguid.h>
#include "hello.h"

// Globals
CHello FAR* g_phello;

SCODE g_scodes[SCODE_COUNT] =          // Array of SCODEs for easy lookup
{
    HELLO_E_UNEXPECTED
};


/*
 * WinMain
 *
 * Purpose:
 *  Main entry point of application.
 *
 */
int APIENTRY WinMain (HINSTANCE hinst, HINSTANCE hinstPrev, LPSTR pCmdLine,
int nCmdShow)
{
    MSG msg;
    DWORD dwRegisterCF;
    DWORD dwRegisterActiveObject;

    //  It is recommended that all OLE applications set
    //  their message queue size to 96. This improves the capacity
    //  and performance of OLE's LRPC mechanism.
    int cMsg = 96;                  // Recommend msg queue size for OLE
    while (cMsg && !SetMessageQueue(cMsg))  // take largest size we can get.
        cMsg -= 8;
    if (!cMsg)
        return -1;                  // ERROR: we got no message queue

    if (!hinstPrev)
       if (!InitApplication(hinst)) // Register window class
          return FALSE;

    if (!InitInstance(hinst))   // Initialize OLE and create Hello
application object
        return (FALSE);

    // Determine if /Automation was specified in command line and register
class
    // factory and active object. Show window if application was started
stand alone.
    if (!ProcessCmdLine(pCmdLine, &dwRegisterCF, &dwRegisterActiveObject,
nCmdShow))
    {
       Uninitialize(dwRegisterCF, dwRegisterActiveObject);
       return (FALSE);
    }

    while (GetMessage(&msg, NULL, 0, 0))
    {
       TranslateMessage(&msg);
       DispatchMessage(&msg);
    }

    Uninitialize(dwRegisterCF, dwRegisterActiveObject);
    return (msg.wParam);
}

/*
 * InitApplication
 *
 * Purpose:
 *  Registers window class
 *
 * Parameters:
 *  hinst       hInstance of application
 *
 * Return Value:
 *  TRUE if initialization succeeded, FALSE otherwise.
 */
BOOL InitApplication (HINSTANCE hinst)
{
   WNDCLASS wc;

   wc.style = CS_DBLCLKS | CS_SAVEBITS | CS_BYTEALIGNWINDOW;
   wc.lpfnWndProc = MainWndProc;
   wc.cbClsExtra = 0;
   wc.cbWndExtra = DLGWINDOWEXTRA;
   wc.hInstance = hinst;
   wc.hIcon = LoadIcon(hinst, MAKEINTRESOURCE(IDI_ICON));
   wc.hCursor = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground = HBRUSH(COLOR_WINDOW + 1);
   wc.lpszMenuName = NULL;
   wc.lpszClassName = TEXT("MainWndClass");

   return RegisterClass(&wc);
}

/*
 * InitInstance
 *
 * Purpose:
 *  Intializes OLE and creates Hello Application object
 *
 * Parameters:
 *  hinst           hInstance of application
 *
 * Return Value:
 *  TRUE if initialization succeeded, FALSE otherwise.
 */
BOOL InitInstance (HINSTANCE hinst)
{
    HRESULT hr;
    TCHAR ach[STR_LEN];

    // Intialize OLE
    hr = OleInitialize(NULL);
    if (FAILED(hr))
       return FALSE;

    // Create an instance of the Hello Application object. Object is
    // created with refcount 0.
    LoadString(hinst, IDS_HelloMessage, ach, sizeof(ach));
    hr = CHello::Create(hinst, ach, &g_phello);
    if (FAILED(hr))
        return FALSE;
    return TRUE;
}

/*
 * ProcessCmdLine
 *
 * Purpose:
 *  Check if command line contains /Automation. If so, the class factory of
the
 *  application object is registered.  If not, the application was started
stand-alone and
 *  so the window is shown. The application object is registered using
RegisterActiveObject.
 *
 * Parameters:
 *  pCmdLine       Command line passed to application
 *  pdwRegisterCF   Returns id returned after class factory registration.
Can be used to
 *                  revoke class factory registration.
 *  pdwRegisterActiveObject   Returns id returned after active object
registration. Can be used to
 *                   revoke active object registration.
 *  nCmdShow         Specifies how window is to be shown if application was
started stand alone.
 *
 * Return Value:
 *    TRUE if OLE initialization succeeded, FALSE otherwise.
 *
 */
BOOL ProcessCmdLine(LPSTR pCmdLine, LPDWORD pdwRegisterCF, LPDWORD
pdwRegisterActiveObject, int nCmdShow)
{
   LPCLASSFACTORY pcf = NULL;
   HRESULT hr;

   *pdwRegisterCF = 0;
   *pdwRegisterActiveObject = 0;

   // Expose class factory for application object if command line contains
the
   // Automation switch
   if (_fstrstr(pCmdLine, "-Automation") != NULL
       || _fstrstr(pCmdLine, "/Automation") != NULL)
   {
       pcf = new CHelloCF;
       if (!pcf)
           goto error;
       pcf->AddRef();
       hr = CoRegisterClassObject(CLSID_Hello, pcf,
                                     CLSCTX_LOCAL_SERVER, REGCLS_SINGLEUSE,
                                     pdwRegisterCF);
       if (hr != NOERROR)
           goto error;
       pcf->Release();
   }
   else g_phello->ShowWindow(nCmdShow);    // Show window if started stand-
alone

   // Register Hello application object in the Running Object Table (ROT).
This
   // allows controllers to connect to a running application object instead
of creating
   // a new instance. Use weak registration so that the ROT releases it's
reference when
   // all external references are released. If strong registration is used,
the ROT will not
   // release it's reference until RevokeActiveObject is called and so will
keep
   // the object alive even after all external references have been
released.
   RegisterActiveObject(g_phello, CLSID_Hello, ACTIVEOBJECT_WEAK,
pdwRegisterActiveObject);
   return TRUE;
```

```
error:
    if (pcf)
        pcf->Release();
    return FALSE;
}

/*
 * Uninitialize
 *
 *  Purpose:
 *   Revoke class factory and active object registration and uninitialize
OLE.
 *
 * Parameters:
 *  dwRegisterCF ID returned after class factory registration.
 *  dwRegisterActiveObject ID returned after active object registration.
 *
 */
void Uninitialize(DWORD dwRegisterCF, DWORD dwRegisterActiveObject)
{
    if (dwRegisterCF != 0)
        CoRevokeClassObject(dwRegisterCF);
    if (dwRegisterActiveObject != 0)
        RevokeActiveObject(dwRegisterActiveObject, NULL);
    OleUninitialize();
}


/*
 * MainWndProc
 *
 * Purpose:
 *  Window procedure for main window. The main window is a dialog.
 *
 */
LRESULT CALLBACK MainWndProc (HWND hwnd, UINT msg, WPARAM wParam, LPARAM
lParam)
{
   switch (msg)
   {
      case WM_COMMAND:
      {
          switch(GET_WM_COMMAND_ID(wParam, lParam))
          {
              case IDC_SAYHELLO:
                  g_phello->SayHello();
                  break;

              case IDC_HELLODISPLAY:
                  switch(GET_WM_COMMAND_CMD(wParam, lParam))
                  {
                      TCHAR ach[100];

                      case EN_CHANGE:
```

```
                      // Update HelloMsg property of Hello object
corresponding to this
                       // window when the contents of the edit control
change.
                       GetDlgItemText(hwnd, IDC_HELLODISPLAY, ach, 100);
                       g_phello->put_HelloMessage(TO_OLE_STRING(ach));
                  }
                  return 0L;
                  default:
                      return 0L;
                  break;
          }
      }
      break;

      case WM_CLOSE:
         // Hide the window to release the refcount added by
CoLockObjectExternal
         // (See CHello::ShowWindow)
         g_phello->ShowWindow(SW_HIDE);
         return DefDlgProc(hwnd, msg, wParam, lParam);
         break;

      case WM_DESTROY:
         PostQuitMessage(0);
         break;

      default:
         return DefDlgProc(hwnd, msg, wParam, lParam);
   }

   return NULL;
}

/*
 * LoadTypeInfo
 *
 *  Purpose:
 *   Gets type information of an object's interface from type library.
 *
 * Parameters:
 *  ppunkStdDispatch    Returns type information.
 *  clsid               Interface id of object in type library.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT LoadTypeInfo(ITypeInfo FAR* FAR* pptinfo, REFCLSID clsid)
{
    HRESULT hr;
    LPTYPELIB ptlib = NULL;
    LPTYPEINFO ptinfo = NULL;
    *pptinfo = NULL;

    // Load Type Library.
    hr = LoadRegTypeLib(LIBID_Hello, 2, 0, 0x0409, &ptlib);
    if (FAILED(hr))
    {
        // if it wasn't registered, try to load it from the path
        // if this succeeds, it will have registered the type library for us
        // for the next time.
        hr = LoadTypeLib(OLESTR("hello.tlb"), &ptlib);
        if(FAILED(hr))
            return hr;
    }

    // Get type information for interface of the object.
    hr = ptlib->GetTypeInfoOfGuid(clsid, &ptinfo);
    if (FAILED(hr))
    {
        ptlib->Release();
        return hr;
    }

    ptlib->Release();
    *pptinfo = ptinfo;
    return NOERROR;
}

#ifdef WIN32

#ifndef UNICODE
char* ConvertToAnsi(OLECHAR FAR* szW)
{
  static char achA[STRCONVERT_MAXLEN];

  WideCharToMultiByte(CP_ACP, 0, szW, -1, achA, STRCONVERT_MAXLEN, NULL,
NULL);
  return achA;
}

OLECHAR* ConvertToUnicode(char FAR* szA)
{
  static OLECHAR achW[STRCONVERT_MAXLEN];

  MultiByteToWideChar(CP_ACP, 0, szA, -1, achW, STRCONVERT_MAXLEN);
  return achW;
}
#endif

#endif
```

LINES

SAMPLE: Lines: OLE Automation Server that implements collections

LINES is an OLE Automation server application that implements collections. LINES allows a collection on lines to be drawn on a pane using OLE Automation. It exposes the following objects.

Application object:

Properties
Application
FullName
Name
Pane Returns the pane in the drawing window.
Parent
Visible
Methods
CreateLine Returns a newly created Line object with no start or end point.
CreatePointReturns a newly created Point object initialized to (0,0).

Pane object: Properties
Lines Returns a collection of all of the Line objects in the drawing.
Points Returns a collection of all of the Point objects in the drawing.
MaxX Returns the maximum visible X coordinate value in twips.
MaxY Returns the maximum visible Y coordinate value in twips.
Methods
Clear Removes all of the Lines and Points from the drawing and refreshes the client area. The result is a blank slate, as if the application had just been launched.
Refresh Clears the drawing area and redraws each of the lines in the Lines collection.

Line object:

Properties
Color An RGB color.
EndPoint A Point object inidcating the end point for the line.
StartPoint A Point object indicating the start point for the line.
Thickness A thickness, represented in twips.

Point object:

Properties
x
y

Lines collection: Standard collection properties and methods. Objects that are added must be of type Line. Objects returned are of type Line.
Add and Remove cause redrawing of the pane.
Adding a line causes two points to be added to the Points collection.
Removing a line may cause one or more points to be removed from the points collection.

Points collection:

Standard collection properties and methods. Objects returned are of type Point.
This collection does not allow addition and removal of members.
The Points collection does not have duplicates. If there are two lines in the drawing which share an end point (x, y), (x, y) only appears in the collection once.

The sample the following features:

· Supports dual interfaces which allows access of automation properties and methods through vtable binding and through IDispatch.
· Provides Rich Error information for vtable-binding controllers. This is does by implementing ISupportErrorInfo and using IErrorInfo.
· Implements 2 collections. · Implements active object registration using RegisterActiveObject and RevokeActiveObject. · Implements correct shut-down behavior. · Includes a .reg file which contains Lines.Application as progID. · When created for automation, is invisible initially.

To compile: -----------

Requires OLE 2.02 or later. Use the external makefile called makefile to compile. In Win16, run the WXSRVER.EXE from \OLE2\BIN before running the makefile. The makefile invokes mktyplib.exe that reads lines.odl and creates the type library, lines.tlb. It then compliles the source files.

To run: -------

Change lines.reg to provide the full path of lines.exe and lines.tlb. Register lines.reg in the registration database by double-clicking it. The VB directory contains VB 3.0 files to control this sample. Load vb.mak into VB 3.0 and run. This will launch and control LINES. VB 3.0 uses late binding (not vtable-binding) to control LINES. A vtable-binding controller for LINES can be built along the lines of the HELLCTRL sample.

Files: ------

LINES.ODL Object description language that describes the objects that LINES exposes. TLB.H Header file generated by mktyplib.exe MAKEFILE Makefile for project.

Other files implement the LINES automation server.
==

MAKEFILE (LINES OLE Sample)

```
####
#makefile - makefile for lines.exe
#
#       Copyright (C) 1994, Microsoft Corporation
#
#Purpose:
#  Builds the OLE 2.0 Automation object, lines.exe.
#
#
#  Usage: NMAKE                  ; build with defaults
#     or: NMAKE option           ; build with the given option(s)
#     or: NMAKE clean            ; erase all compiled files
#
#     option: dev = [win16 | win32]    ; dev=win32 is the default
#             DEBUG=[0 | 1]            ; DEBUG=1 is the default
#             HOST=[DOS | NT | WIN95]  ; HOST=DOS (for win16)
#                                      ; HOST=NT (for win32 on NT)
#                                      ; HOST=WIN95 (for win32 on Win95)
#
#Notes:
#  This makefile assumes that the PATH, INCLUDE and LIB environment
#  variables are setup properly.
#
#############################################################################
##


##########################################################################
#
# Default Settings
#

CPU = i386

!if "$(dev)" == ""
dev = win32
HOST = NT
!endif

!if !("$(dev)" == "win16" || "$(dev)" == "win32")
!error Invalid dev option, choose from [win16 | win32]
!endif

!if "$(dev)" == "win16"
TARGET  = WIN16
!if "$(HOST)" == ""
HOST  = DOS
!endif
!endif

!if "$(dev)" == "win32"
TARGET  = WIN32
```

```
!if "$(HOST)" == ""
HOST  = NT
!endif
!endif

!ifdef NODEBUG
DEBUG = 0
!endif

!if "$(DEBUG)" == "0"
NODEBUG = 1
!endif

!if "$(DEBUG)" == ""
DEBUG = 1
!endif


###########################################################################
#
# WIN16 Settings
#
!if "$(TARGET)" == "WIN16"

CC   = cl
LINK = link
!if "$(HOST)" == "DOS"
WX   = wx /w
!else
WX   =
!endif

RCFLAGS = -dWIN16
CFLAGS = -c -W3 -AM -GA -GEs -DWIN16
LINKFLAGS = /NOD /NOI /BATCH /ONERROR:NOEXE

LIBS = libw.lib mlibcew.lib

!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS) -Od -Zi -D_DEBUG $(CL)
LINKFLAGS = $(LINKFLAGS) /COD
!else
CFLAGS = $(CFLAGS) -Ox $(CL)
LINKFLAGS = $(LINKFLAGS) /FAR /PACKC
!endif
!endif


###########################################################################
#
# WIN32 Settings
#
!if "$(TARGET)" == "WIN32"
```

```
WX =

!include <ntwin32.mak>

CC = $(cc)
CFLAGS = $(cflags) $(cvarsmt) -DINC_OLE2 $(cdebug)

!if "$(HOST)" == "NT"
CFLAGS = $(CFLAGS) -DUNICODE
!endif

!ifndef NODEBUG
CFLAGS = $(CFLAGS) -D_DEBUG
!endif

LINK = $(link)
LINKFLAGS = $(linkdebug) $(guilflags)
RCFLAGS = -DWIN32

!endif

##########################################################################
#
# Build rules
#

.cpp.obj:
    @echo Compiling $<...
    $(CC) $<

.c.obj:

    @echo Compiling $<...
    $(CC) $<


##########################################################################
#
# Application Settings
#

APPS = lines


!if "$(TARGET)" == "WIN16"
LIBS = ole2.lib compobj.lib ole2disp.lib typelib.lib commdlg.lib $(LIBS)
!endif
!if "$(TARGET)" == "WIN32"
LIBS = $(ole2libsmt)
!endif

OBJS = main.obj app.obj appcf.obj pane.obj line.obj point.obj lines.obj
points.obj \
       enumvar.obj errinfo.obj
```

```
###########################################################################
#
# Default Goal
#

goal : setflags $(APPS).exe

setflags :
    set CL=$(CFLAGS)


###########################################################################
#
# Clean (erase) generated files
#
clean :
    if exist *.obj       del *.obj
    if exist $(APPS).exe del $(APPS).exe
    if exist $(APPS).tlb del $(APPS).tlb
    if exist $(APPS).map del $(APPS).map
    if exist $(APPS).res del $(APPS).res
    if exist tlb.h       del tlb.h
    if exist *.log       del *.log
    if exist *.pdb       del *.pdb


###########################################################################
#
# Application Build (WIN16 Specific)
#

!if "$(TARGET)" == "WIN16"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
    link $(LINKFLAGS) @<<
$(OBJS),
$@,,
$(LIBS),
$(APPS).def
<<
    rc -k -t $(APPS).res $@
!endif


###########################################################################
#
# Application Build (WIN32 Specific)
#
!if "$(TARGET)" == "WIN32"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
      $(LINK) @<<
        $(LINKFLAGS)
        -out:$@
        -map:$*.map
        $(OBJS)
        $(APPS).res
        $(LIBS)
<<
!endif


##########################################################################
#
# Application Build (Common)
#

$(APPS).res : $(APPS).rc
    rc $(RCFLAGS) -r -fo$@ $?


##########################################################################
#
# Dependencies
#

tlb.h : lines.odl
     if exist tlb.h  del tlb.h
     if exist lines.tlb  del lines.tlb
     $(WX) mktyplib /D$(TARGET) /h tlb.h /o lines.log /tlb lines.tlb
lines.odl
     type lines.log

main.obj : main.cpp lines.h tlb.h
     $(CC) main.cpp
app.obj : app.cpp lines.h tlb.h
     $(CC) app.cpp
appcf.obj : appcf.cpp lines.h tlb.h
     $(CC) appcf.cpp
pane.obj : pane.cpp lines.h tlb.h
     $(CC) pane.cpp
line.obj : line.cpp lines.h tlb.h
     $(CC) line.cpp
point.obj : point.cpp lines.h tlb.h
     $(CC) point.cpp
lines.obj : lines.cpp lines.h tlb.h
     $(CC) lines.cpp
points.obj : points.cpp lines.h tlb.h
     $(CC) points.cpp
enumvar.obj : enumvar.cpp lines.h tlb.h
     $(CC) enumvar.cpp
errinfo.obj : errinfo.cpp lines.h tlb.h
     $(CC) errinfo.cpp
```

LINES.H (LINES OLE Sample)

```
#ifdef WIN32

#ifdef UNICODE
    #define FROM_OLE_STRING(str) str
    #define TO_OLE_STRING(str) str
#else
    #define FROM_OLE_STRING(str) ConvertToAnsi(str)
    char* ConvertToAnsi(OLECHAR FAR* szW);
    #define TO_OLE_STRING(str) ConvertToUnicode(str)
    OLECHAR* ConvertToUnicode(char FAR* szA);
    // Maximum length of string that can be converted between Ansi & Unicode
    #define STRCONVERT_MAXLEN 300
#endif

#else  // WIN16
  #define APIENTRY far pascal
  #define TCHAR char
  #define TEXT(sz) sz
  #define FROM_OLE_STRING(str) str
  #define TO_OLE_STRING(str) str
  #define LPTSTR LPSTR

  // Windows NT defines the following in windowsx.h
  #define GET_WM_COMMAND_ID(w,l) (w)
  #define GET_WM_COMMAND_CMD(w,l) HIWORD(l)
  #define GET_WM_COMMAND_HWND(w,l) LOWORD(l)
#endif

#include "tlb.h"

#define STRICT

// Menu IDs
#define IDM_DRAWLINE  101
#define IDM_CLEAR     102
#define IDM_EXIT      103

// Dialog ID
#define IDD_DRAWLINE  101
// Icon ID
#define IDI_ICON      102

// Dialog Control IDs
#define IDC_THICKNESS     101
#define IDC_CHOOSECOLOR   102
#define IDC_STARTPOINT_X  103
#define IDC_STARTPOINT_Y  104
#define IDC_ENDPOINT_X    105
#define IDC_ENDPOINT_Y    106
#define IDC_STATIC         -1

// MAX len of string table entries
```

```
#define STR_LEN   200

// String table constants
#define IDS_Name                   1
#define IDS_ErrorLoadingTypeLib    2
#define IDS_ProgID                 3
#define IDS_Unexpected             1001
#define IDS_OutOfMemory            1002
#define IDS_InvalidIndex           1003
#define IDS_CollectionFull         1004
#define IDS_LineFromOtherInstance  1005
#define IDS_CantAddEndPoints       1006
#define IDS_PointFromOtherInstance 1007
#define IDS_NoVisibleXCoordinate   1008
#define IDS_NoVisibleYCoordinate   1009
#define IDS_NoStartPoint           1010
#define IDS_NoEndPoint             1011

// SCODEs for the Lines Application. This is required for vtable-binding
// Automation objects want to return custom HRESULTs. All the OLE-defined
// FACILITY_ITF codes have a code value which lies in the region 0x0000-
0x01FFF.
// While it is legal for the definer to use any code, it is highly
recommended
// that only code values in the range 0x0200-0xFFFF be used, as this will
reduce the
// possiblity of accidental confusion with any OLE-defined errors.
#define LINES_E_FIRST MAKE_SCODE(SEVERITY_ERROR, FACILITY_ITF, 0x0200)

#define LINES_E_UNEXPECTED              (LINES_E_FIRST + 0x0)
// Unexpected error
#define LINES_E_OUTOFMEMORY             (LINES_E_FIRST + 0x1)
// Out of memory
#define LINES_E_INVALIDINDEX            (LINES_E_FIRST + 0x2)
// Invalid index to Points or Lines collections
#define LINES_E_COLLECTIONFULL          (LINES_E_FIRST + 0x3)
// Points or Lines collection is full
#define LINES_E_LINEFROMOTHERINSTANCE   (LINES_E_FIRST + 0x4)
// Line from another instance of this application cannot be added.
#define LINES_E_CANTADDENDPOINTS        (LINES_E_FIRST + 0x5)
// End points of line cannot be added to collection. (Make sure line has two
end points).
#define LINES_E_POINTFROMOTHERINSTANCE  (LINES_E_FIRST + 0x6)
// Point from another instance of this application cannot be added.
#define LINES_E_NOVISIBLEXCOORDINATE    (LINES_E_FIRST + 0x7)
// No visible X coordinate.
#define LINES_E_NOVISIBLEYCOORDINATE    (LINES_E_FIRST + 0x8)
// No visible Y coordinate.
#define LINES_E_NOSTARTPOINT            (LINES_E_FIRST + 0x9)
// Line does not have a start point.
#define LINES_E_NOENDPOINT              (LINES_E_FIRST + 0xA)
// Line does not have an end point.

// Number of SCODEs
#define SCODE_COUNT 11
```

```
// Function prototypes
int APIENTRY WinMain (HINSTANCE, HINSTANCE, LPSTR, int);
BOOL InitApplication (HINSTANCE);
BOOL InitInstance (HINSTANCE);
BOOL ProcessCmdLine(LPSTR pCmdLine, LPDWORD pdwRegisterCF, LPDWORD
pdwRegisterActiveObject, int nCmdShow);
void Uninitialize(DWORD dwRegisterCF, DWORD dwRegisterActiveObject);
HRESULT LoadTypeInfo(ITypeInfo FAR* FAR* pptinfo, REFCLSID clsid);

#ifdef WIN16
extern "C" LRESULT __export CALLBACK MainWndProc (HWND, UINT, WPARAM,
LPARAM);
extern "C" BOOL __export CALLBACK DrawLineDialogFunc(HWND, UINT, WPARAM,
LPARAM);
#else
extern "C" LRESULT CALLBACK MainWndProc (HWND, UINT, WPARAM, LPARAM);
extern "C" BOOL CALLBACK DrawLineDialogFunc(HWND, UINT, WPARAM, LPARAM);
#endif


// Class definitions
class CPane;
class CLine;
class CPoint;
class CLines;
class CPoints;

// ISupportErrorInfo interface implementation
interface CSupportErrorInfo : public ISupportErrorInfo
{
public:
    // IUnknown methods
    STDMETHOD(QueryInterface)(REFIID riid, LPVOID FAR* ppvObj);
    STDMETHOD_(ULONG, AddRef)(void);
    STDMETHOD_(ULONG, Release)(void);

    // ISupportErrorInfo method
    STDMETHOD(InterfaceSupportsErrorInfo)(REFIID riid);

    CSupportErrorInfo(IUnknown FAR* punkObject, REFIID riid);
private:
    LPUNKNOWN m_punkObject;  // IUnknown of Object that implements this
interface
    GUID m_iid;
};

class FAR CApplicationCF : public IClassFactory
{
public:
    // IUnknown methods
    STDMETHOD(QueryInterface)(REFIID riid, LPVOID FAR* ppvObj);
    STDMETHOD_(ULONG, AddRef)(void);
    STDMETHOD_(ULONG, Release)(void);
    // IClassFactory methods
    STDMETHOD(CreateInstance)(IUnknown FAR* punkOuter, REFIID riid,
                              void FAR* FAR* ppv);
    STDMETHOD(LockServer)(BOOL fLock);

    CApplicationCF();

private:
    ULONG m_cRef;                   // Reference count
};


class FAR CApplication : public IApplication
{
public:
    /* IUnknown methods */
    STDMETHOD(QueryInterface)(REFIID riid, LPVOID FAR* ppvObj);
    STDMETHOD_(ULONG, AddRef)();
    STDMETHOD_(ULONG, Release)();

    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(UINT FAR* pctinfo);
    STDMETHOD(GetTypeInfo)(
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo);
    STDMETHOD(GetIDsOfNames)(
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid);
    STDMETHOD(Invoke)(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr);

    /* IApplication automation exposed properties and methods */
    STDMETHOD(get_Application)(IApplication FAR* FAR* ppApplication);
    STDMETHOD(get_FullName)(BSTR FAR* pbstr);
    STDMETHOD(get_Name)(BSTR FAR* pbstr);
    STDMETHOD(get_Parent)(IApplication FAR* FAR* ppApplication);
    STDMETHOD(put_Visible)(VARIANT_BOOL bVisible);
    STDMETHOD(get_Visible)(VARIANT_BOOL FAR* pbVisible);
    STDMETHOD(Quit)();
    STDMETHOD(get_Pane)(IPane FAR* FAR* ppPane);
    STDMETHOD(CreateLine)(ILine FAR* FAR* ppLine);
    STDMETHOD(CreatePoint)(IPoint FAR* FAR* ppPoint);

    /* CApplication methods */
    STDMETHOD_(void, Draw)();
    STDMETHOD_(void, OnSize)(unsigned int nWidth, unsigned int nHeight);
    STDMETHOD_(void, ShowWindow)(int nCmdShow);
    STDMETHOD_(void, CreateAndDrawLine)();
    STDMETHOD_(void, ClearPane)();
    STDMETHOD(RaiseException)(int nID);

    static HRESULT Create(HWND hwnd, CApplication FAR* FAR*
ppApplication); // Creates and intializes Applicaton object
    CApplication();
    ~CApplication();

public:
    HWND m_hwnd;                      // Application window.
    HINSTANCE m_hinst;                // Hinstance of application.
    BSTR m_bstrName;                  // Name of application.
     BSTR m_bstrProgID;                // ProgID of application.
    BOOL m_bUserClosing;              // User is closing the application.

private:
    LPTYPEINFO m_ptinfo;              // Type information of IApplication
interface.
    BSTR m_bstrFullName;              // Full name of application.
    BOOL m_bVisible;                  // Is window visible?
    CPane FAR* m_pPane;               // Pointer to the Pane object.
    EXCEPINFO m_excepinfo;            // Information to raise an exception on
error.
    BOOL m_bRaiseException;           // Properties and methods use this to
signal that an exception is to be raised.
    ULONG m_cRef;                     // Reference count.
    CSupportErrorInfo m_SupportErrorInfo; // ISupportErrorInfo interface
implementation
};

class FAR CPane : public IPane
{
public:
    /* IUnknown methods */
    STDMETHOD(QueryInterface)(REFIID riid, LPVOID FAR* ppvObj);
    STDMETHOD_(ULONG, AddRef)();
    STDMETHOD_(ULONG, Release)();


    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(UINT FAR* pctinfo);
    STDMETHOD(GetTypeInfo)(
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo);
    STDMETHOD(GetIDsOfNames)(
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid);
    STDMETHOD(Invoke)(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr);

    /* IPane automation exposed properties & methods */
    STDMETHOD(get_Lines)(ILines FAR* FAR* ppLines);
    STDMETHOD(get_Points)(IPoints FAR* FAR* ppPoints);
    STDMETHOD(get_MaxX)(int FAR* pnMaxX);
    STDMETHOD(get_MaxY)(int FAR* pnMaxY);
    STDMETHOD(Clear)();
    STDMETHOD(Refresh)();

    /* CPane methods */
    STDMETHOD_(void, Draw)();
    STDMETHOD_(void, OnSize)(unsigned int nWidth, unsigned int nHeight);
    STDMETHOD_(HDC,  GetDC)();
    STDMETHOD_(void, ReleaseDC)(HDC hdc);
    STDMETHOD_(void, InvalidateRect)(LPRECT prc);
    STDMETHOD_(void, Update)(void);
    STDMETHOD_(BOOL, AddPoint)(CPoint FAR* pPoint);
    STDMETHOD_(void, RemovePoint)(CPoint FAR* pPoint);
    STDMETHOD(RaiseException)(int nID);

    static HRESULT Create(HWND hwnd, CPane FAR* FAR* ppPane); // Creates and 
intializes Pane object
    CPane();
    ~CPane();

private:
    LPTYPEINFO m_ptinfo;           // Type information of IPane interface.
    HWND m_hwnd;                   // Main window.
    CLines FAR *m_pLines;          // Lines collection.
    CPoints FAR* m_pPoints;        // Points collection.
    int m_nMaxX;                   // Maximum visible X coordinate in twips.
    int m_nMaxY;                   // Maximum visible Y coordinate in twips.
    int m_nWidth;                  // Width of pane in twips.
    int m_nHeight;                 // Height of pane in twips.
    EXCEPINFO m_excepinfo;         // Information to raise an exception on 
error.
    BOOL m_bRaiseException;        // Properties and methods use this to 
signal that an exception is to be raised.
    ULONG m_cRef;                  // Reference count.
    CSupportErrorInfo m_SupportErrorInfo; // ISupportErrorInfo interface 
implementation
};

class FAR CLine : public ILine
{
public:
    /* IUnknown methods */
    STDMETHOD(QueryInterface)(REFIID riid, LPVOID FAR* ppvObj);
    STDMETHOD_(ULONG, AddRef)();
    STDMETHOD_(ULONG, Release)();

    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(UINT FAR* pctinfo);
    STDMETHOD(GetTypeInfo)(
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo);
    STDMETHOD(GetIDsOfNames)(
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid);
    STDMETHOD(Invoke)(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr);

    /* ILine automation exposed properties & methods */
    STDMETHOD(get_Color)(long FAR* plColorref);
    STDMETHOD(put_Color)(long lColorref);
    STDMETHOD(get_EndPoint)(IPoint FAR* FAR* ppPoint);
    STDMETHOD(putref_EndPoint)(IPoint FAR* pPoint);
    STDMETHOD(get_StartPoint)(IPoint FAR* FAR* ppPoint);
    STDMETHOD(putref_StartPoint)(IPoint FAR* pPoint);
    STDMETHOD(get_Thickness)(int FAR* pnThickness);
    STDMETHOD(put_Thickness)(int nThickness);

    /* CLine methods */
    STDMETHOD_(void, Draw)(HDC hdc);
    STDMETHOD_(void, GetInvalidateRect)(LPRECT prc);
    STDMETHOD_(BOOL, AddEndPointsToPane)(CPane FAR* pPane);
    STDMETHOD_(void, RemoveEndPointsFromPane)(CPane FAR* pPane);
    STDMETHOD(RaiseException)(int nID);

    static HRESULT Create(CLine FAR* FAR* ppLine); // Creates and intializes
Line object
    CLine();
    ~CLine();

private:
    LPTYPEINFO m_ptinfo;           // Type information of ILine interface.
    COLORREF m_colorref;           // RGB color of line.
    CPoint FAR* m_pPointStart;     // Start point of line.
    CPoint FAR* m_pPointEnd;       // End point of line.
    int m_nThickness;              // Line thickness in twips.
    EXCEPINFO m_excepinfo;          // Information to raise an exception on
error.
    BOOL m_bRaiseException;         // Properties and methods use this to
signal that an exception is to be raised.
    ULONG m_cRef;                   // Reference count.
    CSupportErrorInfo m_SupportErrorInfo; // ISupportErrorInfo interface
implementation
};

class FAR CPoint : public IPoint
{
public:
    /* IUnknown methods */
    STDMETHOD(QueryInterface)(REFIID riid, LPVOID FAR* ppvObj);
    STDMETHOD_(ULONG, AddRef)();
    STDMETHOD_(ULONG, Release)();

    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(UINT FAR* pctinfo);
    STDMETHOD(GetTypeInfo)(
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo);
    STDMETHOD(GetIDsOfNames)(
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid);
    STDMETHOD(Invoke)(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr);

    /* IPoint automation exposed properties & methods */
    STDMETHOD(get_x)(int FAR* pnX);
    STDMETHOD(put_x)(int nX);
    STDMETHOD(get_y)(int FAR* pnY);
    STDMETHOD(put_y)(int nY);

    /* CPoint methods */
    STDMETHOD_(ULONG, InternalAddRef)();
    STDMETHOD_(ULONG, InternalRelease)();
    STDMETHOD_(int, get_x)();
    STDMETHOD_(int, get_y)();

    static HRESULT Create(CPoint FAR* FAR* ppPoint); // Creates and
intializes Point object
    CPoint();
    ~CPoint();

private:
    LPTYPEINFO m_ptinfo;           // Type information of IPoint interface.
    int m_nX;                      // X coordinate of point in twips.
    int m_nY;                      // Y coordinate of point in twips.
    ULONG m_cInternalRef;          // Reference count for the use of the
Points collection only.
    ULONG m_cRef;                  // Reference count.
    CSupportErrorInfo m_SupportErrorInfo; // ISupportErrorInfo interface
implementation
};

class FAR CLines : public ILines
{
public:
    /* IUnknown methods */
    STDMETHOD(QueryInterface)(REFIID riid, LPVOID FAR* ppvObj);
    STDMETHOD_(ULONG, AddRef)();
    STDMETHOD_(ULONG, Release)();

    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(UINT FAR* pctinfo);
    STDMETHOD(GetTypeInfo)(
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo);
    STDMETHOD(GetIDsOfNames)(
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid);
    STDMETHOD(Invoke)(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr);

    /* ILines automation exposed properties & methods */
    STDMETHOD(Add)(ILine FAR* pLine);
    STDMETHOD(get_Count)(long FAR* lCount);
    STDMETHOD(get_Item)(long lIndex, ILine FAR* FAR* ppLine);
    STDMETHOD(get__NewEnum)(IUnknown FAR* FAR* ppunkEnum);
    STDMETHOD(Remove)(long lIndex);

    /* CLines methods */
    STDMETHOD_(void, Draw)(HDC hdc);
    STDMETHOD_(void, Clear)();
    STDMETHOD(RaiseException)(int nID);
    static HRESULT Create(ULONG lMaxSize, long lLBound, CPane FAR* pPane,
CLines FAR* FAR* ppLines); // Creates and intializes Lines object
    CLines();
    ~CLines();

private:
    LPTYPEINFO m_ptinfo;           // Type information of ILines interface.
    SAFEARRAY FAR *m_psa;          // Safe array that holds Lines collection
items.
    ULONG m_cElements;             // Number of items in Lines collection.
    ULONG m_cMax;                  // Maximum number of items Lines
collection can hold.
    long m_lLBound;                // Lower bound of index of Lines
collection.
    CPane FAR* m_pPane;            // Pointer to the Pane object that
contains this Lines collection.
    EXCEPINFO m_excepinfo;         // Information to raise an exception on
error.
    BOOL m_bRaiseException;        // Properties and methods use this to
signal that an exception is to be raised.
    ULONG m_cRef;                  // Reference count.
    CSupportErrorInfo m_SupportErrorInfo; // ISupportErrorInfo interface
implementation
};

class FAR CPoints : public IPoints
{
public:
    /* IUnknown methods */
    STDMETHOD(QueryInterface)(REFIID riid, LPVOID FAR* ppvObj);
    STDMETHOD_(ULONG, AddRef)();
    STDMETHOD_(ULONG, Release)();

    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(UINT FAR* pctinfo);
    STDMETHOD(GetTypeInfo)(
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo);
    STDMETHOD(GetIDsOfNames)(
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid);
    STDMETHOD(Invoke)(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr);
    /* IPoints automation exposed properties & methods */
    STDMETHOD(get_Count)(long FAR* plCount);
    STDMETHOD(get_Item)(long lIndex, IPoint FAR* FAR* ppPoint);
    STDMETHOD(get__NewEnum)(IUnknown FAR* FAR* ppunkEnum);

    /* CPoints methods */
    STDMETHOD_(BOOL, Add)(CPoint FAR* pPointNew);
    STDMETHOD_(BOOL, Remove)(CPoint FAR* pPointRemove);
    STDMETHOD_(void, Clear)();
    STDMETHOD_(void, GetMaxXMaxY)(int FAR* pnX, int FAR* pnY, LPRECT prc);
    STDMETHOD(RaiseException)(int nID);

    static HRESULT Create(ULONG lMaxSize, long lLBound, CPane FAR* pPane,
CPoints FAR* FAR* ppPoints); // Creates and intializes Points object
    CPoints();
    ~CPoints();

private:
    LPTYPEINFO m_ptinfo;           // Type information of IPoints interface.
    SAFEARRAY FAR *m_psa;          // Safe array that holds Points
collection items.
    ULONG m_cElements;             // Number of items in Points collection.
    ULONG m_cMax;                  // Maximum number of items Points
collection can hold.
    long m_lLBound;                // Lower bound of index of Points
collection.
    CPane FAR* m_pPane;            // Pointer to the Pane object that
contains this Points collection.
    EXCEPINFO m_excepinfo;         // Information to raise an exception on
error.
    BOOL m_bRaiseException;        // Properties and methods use this to
signal that an exception is to be raised.
    ULONG m_cRef;                  // Reference count.
    CSupportErrorInfo m_SupportErrorInfo; // ISupportErrorInfo interface
implementation
};

class FAR CEnumVariant : public IEnumVARIANT
{
public:
    // IUnknown methods
    STDMETHOD(QueryInterface)(REFIID riid, LPVOID FAR* ppvObj);
    STDMETHOD_(ULONG, AddRef)();
    STDMETHOD_(ULONG, Release)();

    // IEnumVARIANT methods
    STDMETHOD(Next)(ULONG cElements, VARIANT FAR* pvar, ULONG FAR*
pcElementFetched);
    STDMETHOD(Skip)(ULONG cElements);
    STDMETHOD(Reset)();
    STDMETHOD(Clone)(IEnumVARIANT FAR* FAR* ppenum);

    static HRESULT Create(SAFEARRAY FAR*, ULONG, CEnumVariant FAR* FAR*); //
Creates and intializes Enumerator
```

```
    CEnumVariant();
    ~CEnumVariant();

private:
    ULONG m_cRef;           // Reference count
    ULONG m_cElements;      // Number of elements in enumerator.
    long m_lLBound;         // Lower bound of index.
    long m_lCurrent;        // Current index.
    SAFEARRAY FAR* m_psa;   // Safe array that holds elements.
};

// Structure filled by dialog from user input to create a line.
typedef struct _lineinfo
{
    COLORREF colorref;    // RGB color of line.
    POINT ptStart;        // Start point of line.
    POINT ptEnd;          // End point of line.
    int nThickness;       // Thickness of line.
} LINEINFO, FAR* LPLINEINFO;

extern CApplication FAR* g_pApplication;
```

TLB.H (LINES OLE Sample)

```
/* This header file machine-generated by mktyplib.exe */
/* Interface to type library: Lines */

#ifndef _Lines_H_
#define _Lines_H_

DEFINE_GUID(LIBID_Lines,0x3C591B20,0x1F13,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x
10,0x3D,0xE1);

DEFINE_GUID(IID_IPoint,0x3C591B25,0x1F13,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x1
0,0x3D,0xE1);

/* Definition of interface: IPoint */
DECLARE_INTERFACE_(IPoint, IDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* IUnknown methods */
    STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
    STDMETHOD_(ULONG, AddRef)(THIS) PURE;
    STDMETHOD_(ULONG, Release)(THIS) PURE;

    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(THIS_ UINT FAR* pctinfo) PURE;

    STDMETHOD(GetTypeInfo)(
      THIS_
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo) PURE;

    STDMETHOD(GetIDsOfNames)(
      THIS_
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid) PURE;

    STDMETHOD(Invoke)(
      THIS_
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr) PURE;
#endif

    /* IPoint methods */
    STDMETHOD(get_x)(THIS_ int FAR* retval) PURE;
    STDMETHOD(put_x)(THIS_ int Value) PURE;
    STDMETHOD(get_y)(THIS_ int FAR* retval) PURE;
    STDMETHOD(put_y)(THIS_ int Value) PURE;
};

DEFINE_GUID(IID_ILine,0x3C591B24,0x1F13,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x10
,0x3D,0xE1);

/* Definition of interface: ILine */
DECLARE_INTERFACE_(ILine, IDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* IUnknown methods */
    STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
    STDMETHOD_(ULONG, AddRef)(THIS) PURE;
    STDMETHOD_(ULONG, Release)(THIS) PURE;

    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(THIS_ UINT FAR* pctinfo) PURE;

    STDMETHOD(GetTypeInfo)(
      THIS_
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo) PURE;

    STDMETHOD(GetIDsOfNames)(
      THIS_
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid) PURE;

    STDMETHOD(Invoke)(
      THIS_
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr) PURE;
#endif

    /* ILine methods */
    STDMETHOD(get_Color)(THIS_ long FAR* retval) PURE;
    STDMETHOD(put_Color)(THIS_ long rgb) PURE;
    STDMETHOD(get_EndPoint)(THIS_ IPoint FAR* FAR* retval) PURE;
    STDMETHOD(putref_EndPoint)(THIS_ IPoint FAR* Point) PURE;
    STDMETHOD(get_StartPoint)(THIS_ IPoint FAR* FAR* retval) PURE;
    STDMETHOD(putref_StartPoint)(THIS_ IPoint FAR* Point) PURE;
    STDMETHOD(get_Thickness)(THIS_ int FAR* retval) PURE;
    STDMETHOD(put_Thickness)(THIS_ int Value) PURE;
};

DEFINE_GUID(IID_ILines,0x3C591B26,0x1F13,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x1
0,0x3D,0xE1);

/* Definition of interface: ILines */
DECLARE_INTERFACE_(ILines, IDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* IUnknown methods */
    STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
    STDMETHOD_(ULONG, AddRef)(THIS) PURE;
    STDMETHOD_(ULONG, Release)(THIS) PURE;

    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(THIS_ UINT FAR* pctinfo) PURE;

    STDMETHOD(GetTypeInfo)(
      THIS_
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo) PURE;

    STDMETHOD(GetIDsOfNames)(
      THIS_
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,

      DISPID FAR* rgdispid) PURE;

    STDMETHOD(Invoke)(
      THIS_
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr) PURE;
#endif

    /* ILines methods */
    STDMETHOD(Add)(THIS_ ILine FAR* NewLine) PURE;
    STDMETHOD(get_Count)(THIS_ long FAR* retval) PURE;
    STDMETHOD(get_Item)(THIS_ long Index, ILine FAR* FAR* retval) PURE;
    STDMETHOD(get__NewEnum)(THIS_ IUnknown * FAR* retval) PURE;
    STDMETHOD(Remove)(THIS_ long Index) PURE;
};
```

```
DEFINE_GUID(IID_IPoints,0x3C591B27,0x1F13,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x
10,0x3D,0xE1);

/* Definition of interface: IPoints */
DECLARE_INTERFACE_(IPoints, IDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* IUnknown methods */
    STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
    STDMETHOD_(ULONG, AddRef)(THIS) PURE;
    STDMETHOD_(ULONG, Release)(THIS) PURE;

    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(THIS_ UINT FAR* pctinfo) PURE;

    STDMETHOD(GetTypeInfo)(
      THIS_
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo) PURE;

    STDMETHOD(GetIDsOfNames)(
      THIS_
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid) PURE;

    STDMETHOD(Invoke)(
      THIS_
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr) PURE;
#endif

    /* IPoints methods */
    STDMETHOD(get_Count)(THIS_ long FAR* retval) PURE;
    STDMETHOD(get_Item)(THIS_ long Index, IPoint FAR* FAR* retval) PURE;
    STDMETHOD(get__NewEnum)(THIS_ IUnknown * FAR* retval) PURE;
};

DEFINE_GUID(IID_IPane,0x3C591B23,0x1F13,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x10
,0x3D,0xE1);

/* Definition of interface: IPane */
DECLARE_INTERFACE_(IPane, IDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS
```

```
    /* IUnknown methods */
    STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
    STDMETHOD_(ULONG, AddRef)(THIS) PURE;
    STDMETHOD_(ULONG, Release)(THIS) PURE;

    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(THIS_ UINT FAR* pctinfo) PURE;

    STDMETHOD(GetTypeInfo)(
      THIS_
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo) PURE;

    STDMETHOD(GetIDsOfNames)(
      THIS_
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid) PURE;

    STDMETHOD(Invoke)(
      THIS_
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr) PURE;
#endif

    /* IPane methods */
    STDMETHOD(get_Lines)(THIS_ ILines FAR* FAR* retval) PURE;
    STDMETHOD(get_Points)(THIS_ IPoints FAR* FAR* retval) PURE;
    STDMETHOD(get_MaxX)(THIS_ int FAR* retval) PURE;
    STDMETHOD(get_MaxY)(THIS_ int FAR* retval) PURE;
    STDMETHOD(Clear)(THIS) PURE;
    STDMETHOD(Refresh)(THIS) PURE;
};

DEFINE_GUID(IID_IApplication,0x3C591B22,0x1F13,0x101B,0xB8,0x26,0x00,0xDD,0x
01,0x10,0x3D,0xE1);

/* Definition of interface: IApplication */
DECLARE_INTERFACE_(IApplication, IDispatch)
{
#ifndef NO_BASEINTERFACE_FUNCS

    /* IUnknown methods */
    STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
    STDMETHOD_(ULONG, AddRef)(THIS) PURE;
    STDMETHOD_(ULONG, Release)(THIS) PURE;

    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(THIS_ UINT FAR* pctinfo) PURE;

    STDMETHOD(GetTypeInfo)(

      THIS_
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo) PURE;

    STDMETHOD(GetIDsOfNames)(
      THIS_
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid) PURE;

    STDMETHOD(Invoke)(
      THIS_
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr) PURE;
#endif

    /* IApplication methods */
    STDMETHOD(get_Application)(THIS_ IApplication FAR* FAR* retval) PURE;
    STDMETHOD(get_FullName)(THIS_ BSTR FAR* retval) PURE;
    STDMETHOD(get_Name)(THIS_ BSTR FAR* retval) PURE;
    STDMETHOD(get_Parent)(THIS_ IApplication FAR* FAR* retval) PURE;
    STDMETHOD(put_Visible)(THIS_ VARIANT_BOOL VisibleFlag) PURE;
    STDMETHOD(get_Visible)(THIS_ VARIANT_BOOL FAR* retval) PURE;
    STDMETHOD(Quit)(THIS) PURE;
    STDMETHOD(get_Pane)(THIS_ IPane FAR* FAR* retval) PURE;
    STDMETHOD(CreateLine)(THIS_ ILine FAR* FAR* retval) PURE;
    STDMETHOD(CreatePoint)(THIS_ IPoint FAR* FAR* retval) PURE;
};

DEFINE_GUID(CLSID_Lines,0x3C591B21,0x1F13,0x101B,0xB8,0x26,0x00,0xDD,0x01,0x
10,0x3D,0xE1);

class Lines;

#endif
```

APP.CPP (LINES OLE Sample)

```
/**************************************************************************
**
**    OLE Automation Lines Object.
**
**    app.cpp
**
**    CApplication implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "lines.h"

/*
 * CApplication::Create
 *
 * Purpose:
 *  Creates an instance of the Application automation object and initializes
it.
 *
 * Parameters:
 *  hinst            HINSTANCE of application.
 *  ppApplication    Returns Application automation object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CApplication::Create(HINSTANCE hinst, CApplication FAR* FAR* ppApplication )
{
    TCHAR ach[STR_LEN];
    TCHAR achFullName[260];
    HRESULT hr;
    CApplication FAR* pApplication = NULL;
    HWND hwnd;

    *ppApplication = NULL;

    // Create application object.
    pApplication = new CApplication();
    if (pApplication == NULL)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }

    LoadString(hinst, IDS_Name, ach, sizeof(ach));
    hwnd = CreateWindow(TEXT("MainWndClass"), ach,
                       WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       650, 650,
                       NULL, NULL, hinst, NULL);
    if (!hwnd)
    {
       hr = ResultFromScode(E_OUTOFMEMORY);
       goto error;
    }
    pApplication->m_hwnd = hwnd;
    pApplication->m_hinst = hinst;

    // Set name and fullname of application.
    pApplication->m_bstrName = SysAllocString(TO_OLE_STRING(ach));
    if (NULL == pApplication->m_bstrName)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }
    GetModuleFileName(hinst, achFullName, sizeof(achFullName));
    pApplication->m_bstrFullName =
SysAllocString(TO_OLE_STRING(achFullName));
    if (NULL == pApplication->m_bstrFullName)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }
    // ProgID
    LoadString(hinst, IDS_ProgID, ach, sizeof(ach));
    pApplication->m_bstrProgID = SysAllocString(TO_OLE_STRING(ach));
    if (NULL == pApplication->m_bstrProgID)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }

    // Load type information for the application object from type library.
    // If required, notify user on failure.
    hr = LoadTypeInfo(&pApplication->m_ptinfo, IID_IApplication);
    if (FAILED(hr))
    {
        LoadString(hinst, IDS_ErrorLoadingTypeLib, ach, sizeof(ach));
        MessageBox(NULL, ach, FROM_OLE_STRING(pApplication->m_bstrName),
MB_OK);
        goto error;
    }
    // Create Pane
    hr = CPane::Create(hwnd, &pApplication->m_pPane);
    if (FAILED(hr))
       goto error;
    pApplication->m_pPane->AddRef();

    *ppApplication = pApplication;
    return NOERROR;

error:
    if (pApplication == NULL)
        return ResultFromScode(E_OUTOFMEMORY);

    if (pApplication->m_bstrFullName)
        SysFreeString(pApplication->m_bstrFullName);
    if (pApplication->m_bstrName)
        SysFreeString(pApplication->m_bstrName);
    if (pApplication->m_bstrProgID)
        SysFreeString(pApplication->m_bstrProgID);
    if (pApplication->m_ptinfo)
        pApplication->m_ptinfo->Release();
    if (pApplication->m_pPane)
        pApplication->m_pPane->Release();

    // Set to NULL to prevent destructor from attempting to free again
    pApplication->m_bstrFullName = NULL;
    pApplication->m_bstrName = NULL;
     pApplication->m_bstrProgID = NULL;
    pApplication->m_ptinfo = NULL;
    pApplication->m_pPane = NULL;


    delete pApplication;
    return hr;
}

/*
 * CApplication::CApplication
 *
 * Purpose:
 *  Constructor for CApplication object. Initializes members to NULL.
 *
 */
#pragma warning (disable : 4355)
CApplication::CApplication() : m_SupportErrorInfo(this, IID_IApplication)
#pragma warning (default : 4355)
{
    extern ULONG g_cObj;

    m_hwnd = NULL;
    m_bstrFullName = NULL;
    m_bstrName = NULL;
     m_bstrProgID = NULL;
    m_ptinfo = NULL;
    m_pPane = NULL;
    m_cRef = 0;
    m_bVisible = 0;
    m_bUserClosing = FALSE;
}

/*
 * CApplication::~CApplication
 *
 * Purpose:
 *  Destructor for CApplication object. Frees Application message BSTR and
default
 *  IDispatch implementation. Closes the aplication.
 *
 */
CApplication::~CApplication()
{
     extern ULONG g_cObj;

     if (m_bstrFullName) SysFreeString(m_bstrFullName);
     if (m_bstrName) SysFreeString(m_bstrName);
     if (m_bstrProgID) SysFreeString(m_bstrProgID);
     if (m_ptinfo) m_ptinfo->Release();
     if (m_pPane) m_pPane->Release();
     if (!m_bUserClosing && IsWindow(m_hwnd)) DestroyWindow(m_hwnd);
}

/*
 * CApplication::QueryInterface, AddRef, Release
 *
 * Purpose:
 *  Implements IUnknown::QueryInterface, AddRef, Release
 *
 */
STDMETHODIMP
CApplication::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
    *ppv = NULL;

    if (iid == IID_IUnknown || iid == IID_IDispatch || iid ==
IID_IApplication)
        *ppv = this;
    else if (iid == IID_ISupportErrorInfo)
        *ppv = &m_SupportErrorInfo;
    else return ResultFromScode(E_NOINTERFACE);

    AddRef();
    return NOERROR;
}


STDMETHODIMP_(ULONG)
CApplication::AddRef(void)
{
#ifdef _DEBUG
```

```
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, App\r\n"), m_cRef+1);
    OutputDebugString(ach);
#endif

    return ++m_cRef;
}

STDMETHODIMP_(ULONG)
CApplication::Release(void)
{
#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, App\r\n"), m_cRef-1);
    OutputDebugString(ach);
#endif

    if(--m_cRef == 0)
    {
        delete this;
        return 0;
    }
    return m_cRef;
}

/*
 * CApplication::GetTypeInfoCount
 *
 * Purpose:
 *  Implements IDispatch::GetTypeInfoCount.
 *
 */
STDMETHODIMP
CApplication::GetTypeInfoCount(UINT FAR* pctinfo)
{
    *pctinfo = 1;
    return NOERROR;
}

/*
 * CApplication::GetTypeInfo
 *
 * Purpose:
 *  Implements IDispatch::GetTypeInfo.
 *
 */
STDMETHODIMP
CApplication::GetTypeInfo(
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo)
{
    *pptinfo = NULL;

    if(itinfo != 0)
```

```
        return ResultFromScode(DISP_E_BADINDEX);

    m_ptinfo->AddRef();
    *pptinfo = m_ptinfo;

    return NOERROR;
}

/*
 * CApplication::GetIDsOfNames
 *
 * Purpose:
 *  Implements IDispatch::GetIDsOfNames.  The standard implementation,
DispGetIDsOfNames,
 *  is used.
 *
 */
STDMETHODIMP
CApplication::GetIDsOfNames(
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid)
{
    return DispGetIDsOfNames(m_ptinfo, rgszNames, cNames, rgdispid);
}

/*
 * CApplication::Invoke
 *
 * Purpose:
 *  Implements IDispatch::Invoke.  The standard implementation, DispInvoke,
 *  is used. Properties and methods exposed by the application object will
 *  set m_bRaiseException to raise an exception.
 *
 */
STDMETHODIMP
CApplication::Invoke(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr)
{
    HRESULT hr;

    m_bRaiseException = FALSE;
    hr =  DispInvoke(
        this, m_ptinfo,
        dispidMember, wFlags, pdispparams,
         pvarResult, pexcepinfo, puArgErr);
    if (m_bRaiseException)
    {
       if (NULL != pexcepinfo)
           _fmemcpy(pexcepinfo, &m_excepinfo, sizeof(EXCEPINFO));
       return ResultFromScode(DISP_E_EXCEPTION);
    }
    else return hr;
}

/*
 * CApplication::get_Application, get_FullName, get_Name, get_Parent,
get_Visible, put_Visible, Quit
 *
 * Purpose:
 *  Implements the standard Application, FullName, Name, Parent & Visible
properties
 *  and the Quit method.
 *
 */
STDMETHODIMP
CApplication::get_Application(IApplication FAR* FAR* ppApplication)
{
    HRESULT hr;

    hr = QueryInterface(IID_IDispatch, (void FAR* FAR*)ppApplication);
    if (FAILED(hr))
        return RaiseException(IDS_Unexpected);
    return NOERROR;
}

STDMETHODIMP
CApplication::get_FullName(BSTR FAR* pbstr)
{
    *pbstr = SysAllocString(m_bstrFullName);
    return NOERROR;
}

STDMETHODIMP
CApplication::get_Name(BSTR FAR* pbstr)
{
    *pbstr = SysAllocString(m_bstrName);
    return NOERROR;
}

STDMETHODIMP
CApplication::get_Parent(IApplication FAR* FAR* ppApplication)
{
    HRESULT hr;

    hr = QueryInterface(IID_IDispatch, (void FAR* FAR*)ppApplication);
    if (FAILED(hr))
        return RaiseException(IDS_Unexpected);
    return NOERROR;
}
```

```
STDMETHODIMP
CApplication::put_Visible(VARIANT_BOOL bVisible)
{
    ShowWindow(bVisible ? SW_SHOW : SW_HIDE);
    return NOERROR;
}

STDMETHODIMP
CApplication::get_Visible(VARIANT_BOOL FAR* pbVisible)
{
    *pbVisible = m_bVisible;
    return NOERROR;
}

STDMETHODIMP
CApplication::Quit()
{
    // CoDisconnectObject has no effect for an inproc server.  So the
controller
    // will GP fault if it attempts to access the object (including calling
IUnknown::Release())
    // after Quit has been called. For a local server, CoDisconnectObject
will disconnect
    // the object from external connections. So the controller will get an
RPC error if
    // it access the object after calling Quit. The controller will not GP
fault in this case.
    CoDisconnectObject((LPUNKNOWN)this, 0);
    PostMessage(m_hwnd, WM_CLOSE, 0, 0L);
    return NOERROR;
}

/*
 * CApplication::get_Pane
 *
 * Purpose:
 *  Returns pane object.
 *
 */
STDMETHODIMP
CApplication::get_Pane(IPane FAR* FAR* ppPane)
{
    HRESULT hr;


    hr = m_pPane->QueryInterface(IID_IDispatch, (void FAR* FAR*)ppPane);
    if (FAILED(hr))
        return RaiseException(IDS_Unexpected);
    return NOERROR;
}

/*
 * CApplication::CreateLine
 *
 * Purpose:
 *  Returns a newly created line object with no start or end point.
 *
 */
STDMETHODIMP
CApplication::CreateLine(ILine FAR* FAR* ppLine)
{
    CLine FAR* pline = NULL;
    HRESULT hr;

    // Create new item and QI for IDispatch
    hr = CLine::Create(&pline);
    if (FAILED(hr))
        {hr = RaiseException(IDS_OutOfMemory); goto error;}

    hr = pline->QueryInterface(IID_IDispatch, (void FAR* FAR*)ppLine);
    if (FAILED(hr))
        {hr = RaiseException(IDS_Unexpected); goto error;}
    return NOERROR;

error:
    if (pline)
        delete pline;
    return hr;
}

/*
 * CApplication::CreateLine
 *
 * Purpose:
 *  Returns a newly created point object intialized to (0,0).
 *
 */
STDMETHODIMP
CApplication::CreatePoint(IPoint FAR* FAR* ppPoint)
{
    CPoint FAR* ppoint = NULL;
    HRESULT hr;

    // Create new item and QI for IDispatch
    hr = CPoint::Create(&ppoint);
    if (FAILED(hr))
        {hr = RaiseException(IDS_OutOfMemory); goto error;}

    hr = ppoint->QueryInterface(IID_IDispatch, (void FAR* FAR*)ppPoint);
    if (FAILED(hr))
        {hr = RaiseException(IDS_Unexpected); goto error;}
    return NOERROR;

error:
    if (ppoint)
        delete ppoint;
    return hr;
}
```

```
/*
 *
 * The following methods are not exposed through Automation
 *
 */

STDMETHODIMP_(void)
CApplication::Draw()
{
   m_pPane->Draw();
}

/*
 * CApplication::OnSize
 *
 * Purpose:
 *  Called when application window receives WM_SIZE.
 *
 */
STDMETHODIMP_(void)
CApplication::OnSize(unsigned int nWidth, unsigned int nHeight)
{
   m_pPane->OnSize(nWidth, nHeight);
}

STDMETHODIMP_(void)
CApplication::ShowWindow(int nCmdShow)
{
    // Return if curently hidden and asked to hide or currently visible
    // and asked to show.
    if ((!m_bVisible && nCmdShow == SW_HIDE) || (m_bVisible && nCmdShow !=
SW_HIDE))
        return;

    m_bVisible = (nCmdShow == SW_HIDE) ? FALSE : TRUE;

    // The Automation object shutdown behavior is as follows:
    // 1. If the object application is visible, it shuts down only in
response to an
    // explicit user command (File/Exit) or it's programmatic equivalent
(for example
    // the Quit method of the Application object).
    // 2. If the object application is not visible, it goes away when it's
last
    // object is released.
    //
    // CoLockObjectExternal can be used to increment the ref count of the
application object
    // when it is visible. This will implement shutdown behavior 1. When the
application
    // goes invisible, CoLockObjectExternal is used to decrement the ref
count. This will
    // implement shutdown behavior 2.

    if (m_bVisible)
```

```
        CoLockObjectExternal(this, TRUE /*fLock*/, TRUE/*ignored when
fLock==TRUE*/);
    else CoLockObjectExternal(this, FALSE/*fLock*/,
TRUE/*fLastLockReleases*/);
    ::ShowWindow (m_hwnd, nCmdShow);
}

STDMETHODIMP_(void)
CApplication::CreateAndDrawLine()
{
    LINEINFO lineinfo;
    CLine FAR* pLine = NULL;
    CPoint FAR* pPointStart = NULL;
    CPoint FAR* pPointEnd = NULL;
    ILines FAR* pLines = NULL;
    int nRet;
    HRESULT hr;


    nRet = DialogBoxParam(m_hinst, MAKEINTRESOURCE(IDD_DRAWLINE), m_hwnd,
                      (DLGPROC)DrawLineDialogFunc, (LPARAM)
(LPLINEINFO)&lineinfo);
    if (nRet != IDOK)
        return;
    hr = CLine::Create(&pLine);
    if (FAILED(hr))
        goto error;
    hr = CPoint::Create(&pPointStart);
    if (FAILED(hr))
        goto error;
    hr = CPoint::Create(&pPointEnd);
    if (FAILED(hr))
        goto error;

    pPointStart->put_x(lineinfo.ptStart.x);
    pPointStart->put_y(lineinfo.ptStart.y);
    pPointEnd->put_x(lineinfo.ptEnd.x);
    pPointEnd->put_y(lineinfo.ptEnd.y);

    pLine->putref_StartPoint(pPointStart);
    pLine->putref_EndPoint(pPointEnd);
    pLine->put_Thickness(lineinfo.nThickness);
    pLine->put_Color(lineinfo.colorref);

    hr = m_pPane->get_Lines(&pLines);
    if (FAILED(hr))
        goto error;
    hr = pLines->Add(pLine);
    if (FAILED(hr))
        goto error;
    pLines->Release();
    return;

error:
    if (pLine) delete pLine;
    if (pPointStart) delete pPointStart;
    if (pPointEnd) delete pPointEnd;
    if (pLines) pLines->Release();
    MessageBox(m_hwnd, TEXT("Cannot create Line"), m_bstrName, MB_OK);
}

STDMETHODIMP_(void)
CApplication::ClearPane()
{
    m_pPane->Clear();
}

/*
 * CApplication::RaiseException
 *
 * Purpose:
 *  Fills the EXCEPINFO structure and signal IDispatch::Invoke to return
DISP_E_EXCEPTION.
 *  Sets ErrorInfo object for vtable-binding controllers.
 *
 */
STDMETHODIMP
CApplication::RaiseException(int nID)
{
    extern SCODE g_scodes[];
    TCHAR szError[STR_LEN];
    ICreateErrorInfo *pcerrinfo;
    IErrorInfo *perrinfo;
    HRESULT hr;

    _fmemset(&m_excepinfo, 0, sizeof(EXCEPINFO));

    m_excepinfo.wCode = nID;
    if (LoadString(m_hinst, nID, szError, sizeof(szError)))
        m_excepinfo.bstrDescription =
SysAllocString(TO_OLE_STRING(szError));
    m_excepinfo.bstrSource = SysAllocString(m_bstrProgID);

    m_bRaiseException = TRUE;

    // Set ErrInfo object so that vtable binding containers can get
    // rich error information.
    hr = CreateErrorInfo(&pcerrinfo);
    if (SUCCEEDED(hr))
    {
       pcerrinfo->SetGUID(IID_IApplication);
       if (m_excepinfo.bstrSource)
           pcerrinfo->SetSource(m_excepinfo.bstrSource);
       if (m_excepinfo.bstrDescription)
           pcerrinfo->SetDescription(m_excepinfo.bstrDescription);
       hr = pcerrinfo->QueryInterface(IID_IErrorInfo, (LPVOID FAR*)
&perrinfo);
       if (SUCCEEDED(hr))
       {
          SetErrorInfo(0, perrinfo);
```

```
            perrinfo->Release();
         }
         pcerrinfo->Release();
     }

     return ResultFromScode(g_scodes[nID-1001]);
}
```

APPCF.CPP   (LINES OLE Sample)

```
/*************************************************************************
**
**    OLE Simple InProc Automation Object.
**
**    Applicationcf.cpp
**
**    CApplicationCF (class factory) implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/
#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "lines.h"

CApplicationCF::CApplicationCF(void)
{
    m_cRef = 0;
}

/*
 * CApplicationCF::QueryInterface, AddRef, Release
 *
 * Purpose:
 *  Implements IUnknown::QueryInterface, AddRef, Release
 *
 */
STDMETHODIMP
CApplicationCF::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
    *ppv = NULL;

    if (iid == IID_IUnknown || iid == IID_IClassFactory)
        *ppv = this;
    else
        return ResultFromScode(E_NOINTERFACE);

    AddRef();
    return NOERROR;
}


STDMETHODIMP_(ULONG)
```

```
CApplicationCF::AddRef(void)
{
    return ++m_cRef;
}


STDMETHODIMP_(ULONG)
CApplicationCF::Release(void)
{
    if(--m_cRef == 0)
    {
        delete this;
        return 0;
    }
    return m_cRef;
}

/*
 * CApplicationCF::CreateInstance, LockServer
 *
 * Purpose:
 *  Implements IClassFactory::CreateInstance, LockServer
 *
 */
STDMETHODIMP
CApplicationCF::CreateInstance(IUnknown FAR* punkOuter,
                         REFIID riid,
                         void FAR* FAR* ppv)
{
    HRESULT hr;

    *ppv = NULL;

    // This implementation does'nt allow aggregation
    if (punkOuter)
        return ResultFromScode(CLASS_E_NOAGGREGATION);

    // This is REGCLS_SINGLEUSE class factory, so CreateInstance will be
    // called atmost once. An application objects has a REGCLS_SINGLEUSE
class
    // factory. The global application object has already been created when
    // CreateInstance is called. A REGCLS_MULTIPLEUSE class factory's
    // CreateInstance would be called multiple times and would create a new
    // object each time. An MDI application would have a REGCLS_MULTIPLEUSE
    // class factory for it's document objects.
    hr = g_pApplication->QueryInterface(riid, ppv);
    if (FAILED(hr))
    {
        g_pApplication->Quit();
        return hr;
    }
    return NOERROR;
}

STDMETHODIMP
```

```
CApplicationCF::LockServer(BOOL fLock)
{
    CoLockObjectExternal(g_pApplication, fLock, TRUE);
    return NOERROR;
}
```

ENUMVAR.CPP   (LINES OLE Sample)

```
/*************************************************************************
**
**    OLE Automation Collection sample
**
**    enumvar.cpp
**
**    CEnumVariant implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1993 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "lines.h"

/*
 * CEnumVariant::Create
 *
 * Purpose:
 *  Creates an instance of the IEnumVARIANT enumerator object and
initializes it.
 *
 * Parameters:
 *  psa        Safe array containing items to be enumerated.
 *  cElements  Number of items to be enumerated.
 *  ppenumvariant    Returns enumerator object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CEnumVariant::Create(SAFEARRAY FAR* psa, ULONG cElements, CEnumVariant FAR*
FAR* ppenumvariant)
{
    HRESULT hr;
    CEnumVariant FAR* penumvariant = NULL;
    long lLBound;

    *ppenumvariant = NULL;

    penumvariant = new CEnumVariant();
    if (penumvariant == NULL)
        goto error;

    penumvariant->m_cRef = 0;

    // Copy elements into safe array that is used in enumerator
implemenatation and
    // initialize state of enumerator.
    hr = SafeArrayGetLBound(psa, 1, &lLBound);
    if (FAILED(hr))
        goto error;
    penumvariant->m_cElements = cElements;
    penumvariant->m_lLBound = lLBound;
    penumvariant->m_lCurrent = lLBound;
    hr = SafeArrayCopy(psa, &penumvariant->m_psa);
    if (FAILED(hr))
       goto error;

    *ppenumvariant = penumvariant;
    return NOERROR;

error:
    if (penumvariant == NULL)
        return ResultFromScode(E_OUTOFMEMORY);

    if (penumvariant->m_psa)
        SafeArrayDestroy(penumvariant->m_psa);
    penumvariant->m_psa = NULL;
    delete penumvariant;
    return hr;
}

/*
 * CEnumVariant::CEnumVariant
 *
 * Purpose:
 *  Constructor for CEnumVariant object. Initializes members to NULL.
 *
 */
CEnumVariant::CEnumVariant()
{
    m_psa = NULL;
}

/*
 * CEnumVariant::~CEnumVariant
 *
 * Purpose:
 *  Destructor for CEnumVariant object.
 *
 */
CEnumVariant::~CEnumVariant()
{
    if (m_psa) SafeArrayDestroy(m_psa);
}
```

```
/*
 * CEnumVariant::QueryInterface, AddRef, Release
 *
 * Purpose:
 *  Implements IUnknown::QueryInterface, AddRef, Release
 *
 */
STDMETHODIMP
CEnumVariant::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
    *ppv = NULL;

    if (iid == IID_IUnknown || iid == IID_IEnumVARIANT)
        *ppv = this;
    else return ResultFromScode(E_NOINTERFACE);

    AddRef();
    return NOERROR;
}


STDMETHODIMP_(ULONG)
CEnumVariant::AddRef(void)
{

#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, Enum\r\n"), m_cRef+1);
    OutputDebugString(ach);
#endif

    return ++m_cRef;  // AddRef Application Object if enumerator will
outlive application object
}


STDMETHODIMP_(ULONG)
CEnumVariant::Release(void)
{

#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, Enum\r\n"), m_cRef-1);
    OutputDebugString(ach);
#endif

    if(--m_cRef == 0)
    {
        delete this;
        return 0;
    }
    return m_cRef;
}
```

```
/*
 * CEnumVariant::Next
 *
 * Purpose:
 *  Retrieves the next cElements elements. Implements IEnumVARIANT::Next.
 *
 */
STDMETHODIMP
CEnumVariant::Next(ULONG cElements, VARIANT FAR* pvar, ULONG FAR*
pcElementFetched)
{
    HRESULT hr;
    ULONG l;
    long l1;
    ULONG l2;

    if (pcElementFetched != NULL)
        *pcElementFetched = 0;

    for (l=0; l<cElements; l++)
        VariantInit(&pvar[l]);

    // Retrieve the next cElements elements.
    for (l1=m_lCurrent, l2=0; l1<(long)(m_lLBound+m_cElements) &&
l2<cElements; l1++, l2++)
    {
       hr = SafeArrayGetElement(m_psa, &l1, &pvar[l2]);
       if (FAILED(hr))
           goto error;
    }
    // Set count of elements retrieved
    if (pcElementFetched != NULL)
        *pcElementFetched = l2;
    m_lCurrent = l1;

    return  (l2 < cElements) ? ResultFromScode(S_FALSE) : NOERROR;

error:
    for (l=0; l<cElements; l++)
        VariantClear(&pvar[l]);
    return hr;
}

/*
 * CEnumVariant::Skip
 *
 * Purpose:
 *  Skips the next cElements elements. Implements IEnumVARIANT::Skip.
 *
 */
STDMETHODIMP
CEnumVariant::Skip(ULONG cElements)
{
    m_lCurrent += cElements;
    if (m_lCurrent > (long)(m_lLBound+m_cElements))
    {
        m_lCurrent =  m_lLBound+m_cElements;
        return ResultFromScode(S_FALSE);
    }
    else return NOERROR;
}

/*
 * CEnumVariant::Reset
 *
 * Purpose:
 *  Resets the current element in the enumerator to the beginning.
Implements IEnumVARIANT::Reset.
 *
 */
STDMETHODIMP
CEnumVariant::Reset()
{
    m_lCurrent = m_lLBound;
    return NOERROR;
}

/*
 * CEnumVariant::Clone
 *
 * Purpose:
 *  Creates a copy of the current enumeration state. Implements
IEnumVARIANT::Clone.
 *
 */
STDMETHODIMP
CEnumVariant::Clone(IEnumVARIANT FAR* FAR* ppenum)
{
    CEnumVariant FAR* penum = NULL;
    HRESULT hr;

    *ppenum = NULL;

    hr = CEnumVariant::Create(m_psa, m_cElements, &penum);
    if (FAILED(hr))
        goto error;
    penum->AddRef();
    penum->m_lCurrent = m_lCurrent;

    *ppenum = penum;
    return NOERROR;

error:
    if (penum)
        penum->Release();
    return hr;
}
```

ERRINFO.CPP (LINES OLE Sample)

```
/*************************************************************************
**
**    OLE Automation Lines Object.
**
**    errinfo.cpp
**
**    CSupportErrorInfo implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "lines.h"

CSupportErrorInfo::CSupportErrorInfo(IUnknown FAR* punkObject, REFIID riid)
{
    m_punkObject = punkObject;
    m_iid = riid;
}

STDMETHODIMP
CSupportErrorInfo::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
    return m_punkObject->QueryInterface(iid, ppv);
}


STDMETHODIMP_(ULONG)
CSupportErrorInfo::AddRef(void)
{
    return m_punkObject->AddRef();
}

STDMETHODIMP_(ULONG)
CSupportErrorInfo::Release(void)
{
    return m_punkObject->Release();
}

STDMETHODIMP
CSupportErrorInfo::InterfaceSupportsErrorInfo(REFIID riid)
```

```
{
    return (riid == m_iid) ? NOERROR : ResultFromScode(S_FALSE);
}
```

LINE.CPP   (LINES OLE Sample)

```
/*************************************************************************
**
**    OLE Automation Lines Object.
**
**    line.cpp
**
**    CLine implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "lines.h"

/*
 * CLine::Create
 *
 * Purpose:
 *  Creates an instance of the Line automation object and initializes it.
 *
 * Parameters:
 *  ppLine    Returns Line automation object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CLine::Create(CLine FAR* FAR* ppLine )
{
    HRESULT hr;
    CLine FAR* pLine = NULL;

    *ppLine = NULL;

    pLine = new CLine();
    if (pLine == NULL)
        goto error;

    // Load type information for the line from type library.
    hr = LoadTypeInfo(&pLine->m_ptinfo, IID_ILine);
    if (FAILED(hr))
        goto error;

    *ppLine = pLine;
    return NOERROR;

error:
    if (pLine == NULL)
        return ResultFromScode(E_OUTOFMEMORY);

    if (pLine->m_ptinfo)
        pLine->m_ptinfo->Release();

    // Set to NULL to prevent destructor from attempting to free again
    pLine->m_ptinfo = NULL;

    delete pLine;
    return hr;
}

/*
 * CLine::CLine
 *
 * Purpose:
 *  Constructor for CLine object. Initializes members to NULL.
 *
 */
#pragma warning (disable : 4355)
CLine::CLine() : m_SupportErrorInfo(this, IID_ILine)
#pragma warning (default : 4355)
{
    m_ptinfo = NULL;
    m_pPointStart = NULL;
    m_pPointEnd = NULL;
    m_nThickness = 0;
    m_colorref = 0;
    m_cRef = 0;
}

/*
 * CLine::~CLine
 *
 * Purpose:
 *  Destructor for CLine object.
 *
 */
CLine::~CLine()
{
     if (m_ptinfo) m_ptinfo->Release();
     if (m_pPointStart) m_pPointStart->Release();
     if (m_pPointEnd) m_pPointEnd->Release();
}

/*
 * CLine::QueryInterface, AddRef, Release
 *
 * Purpose:
 *  Implements IUnknown::QueryInterface, AddRef, Release
 *
 */
STDMETHODIMP
CLine::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
    *ppv = NULL;

    if (iid == IID_IUnknown || iid == IID_IDispatch || iid == IID_ILine)
        *ppv = this;
    else if (iid == IID_ISupportErrorInfo)
        *ppv = &m_SupportErrorInfo;
    else return ResultFromScode(E_NOINTERFACE);

    AddRef();
    return NOERROR;
}


STDMETHODIMP_(ULONG)
CLine::AddRef(void)
{

#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, Line\r\n"), m_cRef+1);
    OutputDebugString(ach);
#endif

    return ++m_cRef;
}


STDMETHODIMP_(ULONG)
CLine::Release(void)
{
#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, Line\r\n"), m_cRef-1);
    OutputDebugString(ach);
#endif

    if(--m_cRef == 0)
    {
        delete this;
        return 0;
    }
    return m_cRef;
}

/*
 * CLine::GetTypeInfoCount
 *
 * Purpose:
 *  Implements IDispatch::GetTypeInfoCount.
 *
 */
STDMETHODIMP
CLine::GetTypeInfoCount(UINT FAR* pctinfo)
{
    *pctinfo = 1;
    return NOERROR;
}

/*
 * CLine::GetTypeInfo
 *
 * Purpose:
 *  Implements IDispatch::GetTypeInfo.
 *
 */
STDMETHODIMP
CLine::GetTypeInfo(
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo)
{
    *pptinfo = NULL;

    if(itinfo != 0)
        return ResultFromScode(DISP_E_BADINDEX);

    m_ptinfo->AddRef();
    *pptinfo = m_ptinfo;

    return NOERROR;
}

/*
 * CLine::GetIDsOfNames
 *
 * Purpose:
 *  Implements IDispatch::GetIDsOfNames.  The standard implementation,
DispGetIDsOfNames,
 *  is used.
 *
 */
STDMETHODIMP
CLine::GetIDsOfNames(
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid)
{
    return DispGetIDsOfNames(m_ptinfo, rgszNames, cNames, rgdispid);
}
```

```
/*
 * CLine::Invoke
 *
 * Purpose:
 *  Implements IDispatch::Invoke.  The standard implementation, DispInvoke,
 *  is used. Properties and methods exposed by the application object will
 *  set m_bRaiseException to raise an exception.
 *
 */
STDMETHODIMP
CLine::Invoke(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr)
{
    HRESULT hr;

     // VB 3.0/Disptest have a bug in which they pass DISPATCH_PROPERTYPUT
    // instead of DISPATCH_PROPERTYPUTREF for the EndPoint and StartPoint
    // properties. Future versions of VB will correctly pass
DISPATCH_PROPERTYPUTREF.
    // EndPoint has DISPID == 1 and StartPoint has DISPID == 2 - see
lines.odl.
    // The following code works around the VB 3.0/Disptest bug.
    if ((dispidMember == 1 || dispidMember == 2)
        && wFlags == DISPATCH_PROPERTYPUT)
        wFlags = DISPATCH_PROPERTYPUTREF;

    m_bRaiseException = FALSE;
    hr =  DispInvoke(
        this, m_ptinfo,
        dispidMember, wFlags, pdispparams,
        pvarResult, pexcepinfo, puArgErr);
    if (m_bRaiseException)
    {
       if (NULL != pexcepinfo)
           _fmemcpy(pexcepinfo, &m_excepinfo, sizeof(EXCEPINFO));
       return ResultFromScode(DISP_E_EXCEPTION);
    }
    else return hr;
}

/*
 *
 * Properties and methods exposed through automation.
 *
 */
STDMETHODIMP
CLine::get_Color(long FAR* plColorref)
{
    *plColorref =  m_colorref;
    return NOERROR;
}

STDMETHODIMP
CLine::put_Color(long lColorref)
{
    m_colorref = (COLORREF)lColorref;
    return NOERROR;
}

STDMETHODIMP

CLine::get_EndPoint(IPoint FAR* FAR* ppPoint)
{
    HRESULT hr;

    *ppPoint = NULL;

    if (NULL == m_pPointEnd)
        return RaiseException(IDS_NoStartPoint);

    hr = m_pPointEnd->QueryInterface(IID_IDispatch, (void FAR*
FAR*)ppPoint);
    if (FAILED(hr))
        return RaiseException(IDS_Unexpected);
    return NOERROR;
}

STDMETHODIMP
CLine::putref_EndPoint(IPoint FAR* pPointNew)
{
    HRESULT hr;
    CPoint FAR* pPoint;

    // Save the IPoint interface so we can easily access private data
    hr = pPointNew->QueryInterface(IID_IPoint, (void FAR* FAR*)&pPoint);
    if (FAILED(hr))
        return RaiseException(IDS_PointFromOtherInstance);

    // Replace the old point with the new
    if (m_pPointEnd)
       m_pPointEnd->Release();

    m_pPointEnd = pPoint;
    return NOERROR;
}

STDMETHODIMP
CLine::get_StartPoint(IPoint FAR* FAR* ppPoint)
{
    HRESULT hr;

    *ppPoint = NULL;
    if (NULL == m_pPointStart)
        return RaiseException(IDS_NoStartPoint);

    hr = m_pPointStart->QueryInterface(IID_IDispatch, (void FAR*
FAR*)ppPoint);
    if (FAILED(hr))
        return RaiseException(IDS_Unexpected);

    return NOERROR;
}

STDMETHODIMP
CLine::putref_StartPoint(IPoint FAR* pPointNew)
{
    HRESULT hr;
    CPoint FAR* pPoint;

    // Save the IPoint interface so we can easily access private data
    hr = pPointNew->QueryInterface(IID_IPoint, (void FAR* FAR*)&pPoint);
    if (FAILED(hr))
        return RaiseException(IDS_PointFromOtherInstance);

    // Replace the old point with the new
    if (m_pPointStart)
       m_pPointStart->Release();

    m_pPointStart = pPoint;
    return NOERROR;
}

STDMETHODIMP
CLine::get_Thickness(int FAR* pnThickness)
{
    *pnThickness = m_nThickness;
    return NOERROR;
}

STDMETHODIMP
CLine::put_Thickness(int nThickness)
{
    m_nThickness = nThickness;
    return NOERROR;
}

/*
 *
 * The following methods are not exposed through Automation
 *
 */

STDMETHODIMP_(void)
CLine::Draw(HDC hdc)
{
    HPEN hpen, hpenOld;
    if (m_pPointStart == NULL)
        return;
    if (m_pPointEnd == NULL)
        return;

    hpen = CreatePen(PS_SOLID, m_nThickness, m_colorref);
    hpenOld = (HGDIOBJ)SelectObject(hdc, (HGDIOBJ)hpen);

    MoveToEx(hdc, m_pPointStart->get_x(), m_pPointStart->get_y(), NULL);
    LineTo(hdc, m_pPointEnd->get_x(), m_pPointEnd->get_y());

    SelectObject(hdc, (HGDIOBJ)hpenOld);
    DeleteObject((HGDIOBJ)hpen);
    return;
}

/*
 * CLine::GetInvalidateRect
 *
 * Purpose:
 *  Get the rectangle that cicumscribes the line. This rectangle is used to
invalidate the area
 *  of the window where the line is to be draw or erased.
 *
 */
STDMETHODIMP_(void)
CLine::GetInvalidateRect(LPRECT prc)
{
    int nX1, nY1, nX2, nY2;

    nX1 = m_pPointStart->get_x();
    nY1 = m_pPointStart->get_y();

    nX2 = m_pPointEnd->get_x();
    nY2 = m_pPointEnd->get_y();

    if (nX1 < nX2)
    {
        prc->left = nX1;
        prc->right = nX2;
    }
    else

    {
        prc->left = nX2;
        prc->right = nX1;
    }
    if (nY1 < nY2)
    {
        prc->top = nY1;
        prc->bottom = nY2;
    }
    else
    {
        prc->top = nY2;
        prc->bottom = nY1;
    }

    InflateRect(prc, m_nThickness, m_nThickness);
}

STDMETHODIMP_(BOOL)
CLine::AddEndPointsToPane(CPane FAR* pPane)
{
    // Does the line have end points?
    if (NULL == m_pPointStart)
       return FALSE;
    if (NULL == m_pPointEnd)
       return FALSE;

    // Add points to Point collection in the Pane object.
    if (pPane->AddPoint(m_pPointStart))
        return pPane->AddPoint(m_pPointEnd);
    return FALSE;

}

STDMETHODIMP_(void)
CLine::RemoveEndPointsFromPane(CPane FAR* pPane)
{
    // Remove the end points of the line to be removed from the Points
collection in the Pane.
    if (NULL != m_pPointStart)
        pPane->RemovePoint(m_pPointStart);
    if (NULL != m_pPointEnd)
       pPane->RemovePoint(m_pPointEnd);

}

/*
 * CLine::RaiseException
 *
 * Purpose:
 *  Fills the EXCEPINFO structure and signal IDispatch::Invoke to return
DISP_E_EXCEPTION.
 *  Sets ErrorInfo object for vtable-binding controllers.
 *
 */
STDMETHODIMP
CLine::RaiseException(int nID)
{
    extern SCODE g_scodes[];
    TCHAR szError[STR_LEN];
    ICreateErrorInfo *pcerrinfo;
    IErrorInfo *perrinfo;
    HRESULT hr;

    _fmemset(&m_excepinfo, 0, sizeof(EXCEPINFO));

    m_excepinfo.wCode = nID;
    if (LoadString(g_pApplication->m_hinst, nID, szError, sizeof(szError)))
        m_excepinfo.bstrDescription =
SysAllocString(TO_OLE_STRING(szError));
    m_excepinfo.bstrSource = SysAllocString(g_pApplication->m_bstrProgID);

    m_bRaiseException = TRUE;

    // Set ErrInfo object so that vtable binding containers can get
    // rich error information.
    hr = CreateErrorInfo(&pcerrinfo);
    if (SUCCEEDED(hr))
    {
       pcerrinfo->SetGUID(IID_ILine);
       if (m_excepinfo.bstrSource)
           pcerrinfo->SetSource(m_excepinfo.bstrSource);
       if (m_excepinfo.bstrDescription)
           pcerrinfo->SetDescription(m_excepinfo.bstrDescription);
       hr = pcerrinfo->QueryInterface(IID_IErrorInfo, (LPVOID FAR*)
&perrinfo);
       if (SUCCEEDED(hr))
       {
          SetErrorInfo(0, perrinfo);
          perrinfo->Release();
       }
       pcerrinfo->Release();
    }

    return ResultFromScode(g_scodes[nID-1001]);
}
```

LINES.CPP (LINES OLE Sample)

```
/*************************************************************************
**
**    OLE Automation Lines object
**
**    lines.cpp
**
**    CLines collection implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1993 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "lines.h"

/*
 * CLines::Create
 *
 * Purpose:
 *  Creates an instance of a Lines collection object and initializes it.
 *
 * Parameters:
 *  lMaxSize   Maximum number of items that can added to collection.
 *  lLBound    Lower bound of index of collection.
 *  pPane      Pointer to pane that contains this collection. Required
because the pane coordinates
 *             the lines and points collection.
 *  ppLines    Returns Lines collection object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CLines::Create(ULONG lMaxSize, long lLBound, CPane FAR* pPane, CLines FAR*
FAR* ppLines)
{
    HRESULT hr;
    CLines FAR* pLines = NULL;
    SAFEARRAYBOUND sabound[1];

    *ppLines = NULL;
    // Create new collection
    pLines = new CLines();
    if (pLines == NULL)
        goto error;

    pLines->m_cMax = lMaxSize;
    pLines->m_cElements = 0;
    pLines->m_lLBound = lLBound;
    pLines->m_pPane = pPane;

    // Load type information for the Lines collection from type library.
    hr = LoadTypeInfo(&pLines->m_ptinfo, IID_ILines);
    if (FAILED(hr))
        goto error;

    // Create a safe array of variants which is used to implement the
collection.
    sabound[0].cElements = lMaxSize;
    sabound[0].lLbound = lLBound;
    pLines->m_psa = SafeArrayCreate(VT_VARIANT, 1, sabound);
    if (pLines->m_psa == NULL)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }

    *ppLines = pLines;
    return NOERROR;

error:
    if (pLines == NULL)
        return ResultFromScode(E_OUTOFMEMORY);
    if (pLines->m_ptinfo)
        pLines->m_ptinfo->Release();
    if (pLines->m_psa)
        SafeArrayDestroy(pLines->m_psa);

    pLines->m_psa = NULL;
    pLines->m_ptinfo = NULL;

    delete pLines;
    return hr;
}

/*
 * CLines::CLines
 *
 * Purpose:
 *  Constructor for CLines object. Initializes members to NULL.
 *
 */
#pragma warning (disable : 4355)
CLines::CLines() : m_SupportErrorInfo(this, IID_ILines)
#pragma warning (default : 4355)
```

```
{
    m_cRef = 0;
    m_psa = NULL;
    m_ptinfo = NULL;
}

/*
 * CLines::~CLines
 *
 * Purpose:
 *  Destructor for CLines object.
 *
 */
CLines::~CLines()
{
     if (m_ptinfo) m_ptinfo->Release();
     if (m_psa) SafeArrayDestroy(m_psa);
}

/*
 * CLines::QueryInterface, AddRef, Release
 *
 * Purpose:
 *  Implements IUnknown::QueryInterface, AddRef, Release
 *
 */
STDMETHODIMP
CLines::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
    *ppv = NULL;

    if (iid == IID_IUnknown || iid == IID_ILines || iid == IID_IDispatch)
        *ppv = this;
    else if (iid == IID_ISupportErrorInfo)
        *ppv = &m_SupportErrorInfo;
    else return ResultFromScode(E_NOINTERFACE);

    AddRef();
    return NOERROR;
}


STDMETHODIMP_(ULONG)
CLines::AddRef(void)
{

#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, Lines\r\n"), m_cRef+1);
    OutputDebugString(ach);
#endif

    return ++m_cRef;
}
```

```
STDMETHODIMP_(ULONG)
CLines::Release(void)
{

#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, Lines\r\n"), m_cRef-1);
    OutputDebugString(ach);
#endif

    if(--m_cRef == 0)
    {
        delete this;
        return 0;
    }
    return m_cRef;
}

/*
 * CLines::GetTypeInfoCount
 *
 * Purpose:
 *  Implements IDispatch::GetTypeInfoCount.
 *
 */
STDMETHODIMP
CLines::GetTypeInfoCount(UINT FAR* pctinfo)
{
    *pctinfo = 1;
    return NOERROR;
}

/*
 * CLines::GetTypeInfo
 *
 * Purpose:
 *  Implements IDispatch::GetTypeInfo.
 *
 */
STDMETHODIMP
CLines::GetTypeInfo(
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo)
{
    *pptinfo = NULL;

    if(itinfo != 0)
        return ResultFromScode(DISP_E_BADINDEX);

    m_ptinfo->AddRef();
    *pptinfo = m_ptinfo;
    return NOERROR;
}

/*
 * CLines::GetIDsOfNames
 *
 * Purpose:
 *  Implements IDispatch::GetIDsOfNames.  The standard implementation,
DispGetIDsOfNames,
 *  is used.
 *
 */
STDMETHODIMP
CLines::GetIDsOfNames(
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid)
{
    return DispGetIDsOfNames(m_ptinfo, rgszNames, cNames, rgdispid);
}

/*
 * CLines::Invoke
 *
 * Purpose:
 *  Implements IDispatch::Invoke.  The standard implementation, DispInvoke,
 *  is used. Properties and methods exposed by the lines collection object
will
 *  set m_bRaiseException to raise an exception.
 *
 */
STDMETHODIMP
CLines::Invoke(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr)
{
    HRESULT hr;

    m_bRaiseException = FALSE;
    hr =  DispInvoke(
        this, m_ptinfo,
        dispidMember, wFlags, pdispparams,
        pvarResult, pexcepinfo, puArgErr);
    if (m_bRaiseException)
    {
       if (NULL != pexcepinfo)
           _fmemcpy(pexcepinfo, &m_excepinfo, sizeof(EXCEPINFO));
       return ResultFromScode(DISP_E_EXCEPTION);
    }
    else return hr;
}

/*
 *
 * Properties and methods exposed through automation.
 *
 */

/*
 * CLines::Add
 *
 * Purpose:
 *  Adds a line to the lines collection.
 *
 * Parameters:
 *  pLineNew    IDispatch of line to be added to collection.
 *
 */
STDMETHODIMP
CLines::Add(ILine FAR* pLineNew)
{
    HRESULT hr;
    LONG l;
    CLine FAR* pLine = NULL;
    VARIANT v;
    HDC hdc;

    // Is the collection full?
    if (m_cElements == m_cMax)
        return RaiseException(IDS_CollectionFull);

    hr = pLineNew->QueryInterface(IID_ILine, (void FAR* FAR*)&pLine);
    if (FAILED(hr))
        return RaiseException(IDS_LineFromOtherInstance);

    // Add end points of line to the pane's point collection
    if (FALSE == pLine->AddEndPointsToPane(m_pPane))
        {hr = RaiseException(IDS_CantAddEndPoints); goto error;}

    // Add new line to collection
    l = m_lLBound+m_cElements;
    VariantInit(&v);
    V_VT(&v) = VT_DISPATCH;
    V_DISPATCH(&v) = pLineNew;
    hr = SafeArrayPutElement(m_psa, &l, &v);
    if (FAILED(hr))
        {hr = RaiseException(IDS_Unexpected); goto error;}
    m_cElements++;

    // Draw the new line
    hdc = m_pPane->GetDC();
    pLine->Draw(hdc);
    m_pPane->ReleaseDC(hdc);

    pLine->Release();
    return NOERROR;

error:
    if (pLine)
        pLine->Release();
    return hr;
}

/*
 * CLines::get_Count
 *
 * Purpose:
 *  Returns number of items in collection.
 *
 */
STDMETHODIMP
CLines::get_Count(long FAR* lCount)
{
    *lCount = m_cElements;
    return NOERROR;
}

/*
 * CLines::get_Item
 *
 * Purpose:
 *  Retrieves item from collection, given an index.
 *
 * Parameters:
 *   lIndex   Index of item to be retrieved.
 *   ppLine   Returns IDispatch of item retrieved from collection.
 *
 */
STDMETHODIMP
CLines::get_Item(long lIndex, ILine FAR* FAR* ppLine)
{
    HRESULT hr;
    VARIANT v;

    // Check if index is within range
    if (lIndex < m_lLBound || lIndex >= (long)(m_lLBound+m_cElements))
        return RaiseException(IDS_InvalidIndex);

    // Retrieve and return item. Note that SafeArrayGetElement AddRefs, so
an additional AddRef
    // is not required.
    VariantInit(&v);
    hr = SafeArrayGetElement(m_psa, &lIndex, &v);
    if (FAILED(hr))
        return RaiseException(IDS_Unexpected);
    *ppLine = (ILine FAR*) V_DISPATCH(&v);
    return NOERROR;
}

/*
 * CLines::get_NewEnum
 *
 * Purpose:
 *  Returns an enumerator (IEnumVARIANT) for the items curently in the
collection.
 *  The NewEnum property is restricted and so is invisible to users of an
 *  automation controller's scripting language. Automation controllers that
support
 *  a 'For Each' statement to iterate through the elements of a collection
will use
 *  the enumerator returned by NewEnum. The enumerator creates a snapshot of
the
 *  the current state of the collection.
 *
 */
STDMETHODIMP
CLines::get__NewEnum(IUnknown FAR* FAR* ppunkEnum)
{
    CEnumVariant FAR* penum = NULL;;
    HRESULT hr;

    *ppunkEnum = NULL;

    // Create new enumerator for items currently in collection and QI for
IUnknown
    hr = CEnumVariant::Create(m_psa, m_cElements, &penum);
    if (FAILED(hr))
        {hr = RaiseException(IDS_OutOfMemory); goto error;}
    hr = penum->QueryInterface(IID_IUnknown, (VOID FAR* FAR*)ppunkEnum);
    if (FAILED(hr))
        {hr = RaiseException(IDS_Unexpected); goto error;}
    return NOERROR;

error:
    if (penum)
        delete penum;
    return hr;
}


/*
 * CLines::Remove
 *
 * Purpose:
 *  Removes specified item from collection.
 *
 * Parameters:
 *   lIndex   Index of item to be removed.
 *
 */
STDMETHODIMP
```

```
CLines::Remove(long lIndex)
{
    HRESULT hr;
    long l;
    VARIANT HUGEP *pvar;
    CLine FAR* pLine = NULL;
    RECT rc;

    // Check if integer index is within range.
    if (lIndex < m_lLBound || lIndex >= (long)(m_lLBound+m_cElements))
        return RaiseException(IDS_InvalidIndex);

    hr = SafeArrayAccessData(m_psa, (void HUGEP* FAR*)&pvar);
    if (FAILED(hr))
        return RaiseException(IDS_Unexpected);
    V_DISPATCH(&pvar[lIndex-m_lLBound])->QueryInterface(IID_ILine, (void 
FAR* FAR*)&pLine);

    // Ask the pane to invalidate the area where the line is drawn.
    pLine->GetInvalidateRect(&rc);
    m_pPane->InvalidateRect(&rc);

    // Remove end points of line from the pane's point collection.
    pLine->RemoveEndPointsFromPane(m_pPane);
    // Remove Line
    V_DISPATCH(&pvar[lIndex-m_lLBound])->Release();
    // Move up the array elements, after the element to be removed.
    for (l=lIndex-m_lLBound; l<(long)(m_cElements-1); l++)
        pvar[l] = pvar[l+1];
    // Remove last element.
    V_VT(&pvar[l]) = VT_EMPTY;

    pLine->Release();
    hr = SafeArrayUnaccessData(m_psa);
    if (FAILED(hr))
       return RaiseException(IDS_Unexpected);

    m_cElements--;
    m_pPane->Update();     // Ask the pane to repaint invalidated areas 
caused by removal of line.
    return NOERROR;
}

/*
 *
 * The following methods are not exposed through Automation
 *
 */

/*
 * CLines::Draw
 *
 * Purpose:
 *  Draws all lines in collection.
 *
 */
STDMETHODIMP_(void)
CLines::Draw(HDC hdc)
{
    HRESULT hr;
    long l;
    CLine FAR* pLine = NULL;
    VARIANT HUGEP *pvar;

    // Draw each line in the Lines collection
    hr = SafeArrayAccessData(m_psa, (void HUGEP* FAR*)&pvar);
    if (FAILED(hr))
       return;
    for (l=0; l<(long)m_cElements; l++)
    {
        hr = V_DISPATCH(&pvar[l])->QueryInterface(IID_ILine, (void FAR*
FAR*)&pLine);
        if (FAILED(hr))
            continue;
        pLine->Draw(hdc);
        pLine->Release();
    }
    hr = SafeArrayUnaccessData(m_psa);
    if (FAILED(hr))
        return;
    return;
}

/*
 * CLines::Clear
 *
 * Purpose:
 *  Removes all items from collection.
 *
 */
STDMETHODIMP_(void)
CLines::Clear(void)
{
    SafeArrayDestroyData(m_psa);
    SafeArrayAllocData(m_psa);
    m_cElements = 0;
}

/*
 * CLines::RaiseException
 *
 * Purpose:
 *  Fills the EXCEPINFO structure and signal IDispatch::Invoke to return
DISP_E_EXCEPTION.
 *  Sets ErrorInfo object for vtable-binding controllers.
 *
 */
STDMETHODIMP
CLines::RaiseException(int nID)
{
    extern SCODE g_scodes[];
    TCHAR szError[STR_LEN];
    ICreateErrorInfo *pcerrinfo;
    IErrorInfo *perrinfo;
    HRESULT hr;

    _fmemset(&m_excepinfo, 0, sizeof(EXCEPINFO));

    m_excepinfo.wCode = nID;
    if (LoadString(g_pApplication->m_hinst, nID, szError, sizeof(szError)))
        m_excepinfo.bstrDescription =
SysAllocString(TO_OLE_STRING(szError));
    m_excepinfo.bstrSource = SysAllocString(g_pApplication->m_bstrProgID);

    m_bRaiseException = TRUE;

    // Set ErrInfo object so that vtable binding containers can get
    // rich error information.
    hr = CreateErrorInfo(&pcerrinfo);
    if (SUCCEEDED(hr))
    {
       pcerrinfo->SetGUID(IID_ILines);
       if (m_excepinfo.bstrSource)
           pcerrinfo->SetSource(m_excepinfo.bstrSource);
       if (m_excepinfo.bstrDescription)
           pcerrinfo->SetDescription(m_excepinfo.bstrDescription);
       hr = pcerrinfo->QueryInterface(IID_IErrorInfo, (LPVOID FAR*)
&perrinfo);
       if (SUCCEEDED(hr))
       {
          SetErrorInfo(0, perrinfo);
          perrinfo->Release();
       }
       pcerrinfo->Release();
    }

    return ResultFromScode(g_scodes[nID-1001]);
}
```

MAIN.CPP (LINES OLE Sample)

```
/*********************************************************************
**
**    OLE Automation Lines Object.
**
**    main.cpp
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*********************************************************************/
#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
  #include <commdlg.h>
#endif
#include <initguid.h>
#include "lines.h"

// Globals
CApplication FAR* g_pApplication;

SCODE g_scodes[SCODE_COUNT] =                // Array of SCODEs for easy
lookup
{    LINES_E_UNEXPECTED,
     LINES_E_OUTOFMEMORY,
     LINES_E_INVALIDINDEX,
     LINES_E_COLLECTIONFULL,
     LINES_E_LINEFROMOTHERINSTANCE,
     LINES_E_CANTADDENDPOINTS,
     LINES_E_POINTFROMOTHERINSTANCE,
     LINES_E_NOVISIBLEXCOORDINATE,
     LINES_E_NOVISIBLEYCOORDINATE,
     LINES_E_NOSTARTPOINT,
     LINES_E_NOENDPOINT
};

/*
 * WinMain
 *
 * Purpose:
 *  Main entry point of application.
 *
 */
int APIENTRY WinMain (HINSTANCE hinst, HINSTANCE hinstPrev, LPSTR pCmdLine,
int nCmdShow)
{
    MSG msg;
    DWORD dwRegisterCF;
    DWORD dwRegisterActiveObject;

    //  It is recommended that all OLE applications set
    //  their message queue size to 96. This improves the capacity
    //  and performance of OLE's LRPC mechanism.
    int cMsg = 96;                  // Recommend msg queue size for OLE
    while (cMsg && !SetMessageQueue(cMsg))  // take largest size we can get.
        cMsg -= 8;
    if (!cMsg)
        return -1;                  // ERROR: we got no message queue

    if (!hinstPrev)
       if (!InitApplication(hinst)) // Register window class
          return FALSE;

    if (!InitInstance(hinst))   // Initialize OLE and create Lines
application object
        return (FALSE);

    // Determine if /Automation was specified in command line and register
class
    // factory and active object. Show window if application was started
stand alone.
    if (!ProcessCmdLine(pCmdLine, &dwRegisterCF, &dwRegisterActiveObject,
nCmdShow))
    {
       Uninitialize(dwRegisterCF, dwRegisterActiveObject);
       return (FALSE);
    }

    while (GetMessage(&msg, NULL, 0, 0))
    {
       TranslateMessage(&msg);
       DispatchMessage(&msg);
    }

    Uninitialize(dwRegisterCF, dwRegisterActiveObject);
    return (msg.wParam);
}

/*
 * InitApplication
 *
 * Purpose:
 *  Registers window class
 *
 * Parameters:
 *  hinst       hInstance of application
 *
 * Return Value:
 *  TRUE if initialization succeeded, FALSE otherwise.
 */
BOOL InitApplication (HINSTANCE hinst)
```

```
{
   WNDCLASS wc;

   wc.style = CS_DBLCLKS | CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc = MainWndProc;
   wc.cbClsExtra = 0;
   wc.cbWndExtra = 0;
   wc.hInstance = hinst;
   wc.hIcon = LoadIcon(hinst, MAKEINTRESOURCE(IDI_ICON));
   wc.hCursor = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
   wc.lpszMenuName = TEXT("AppMenu");
   wc.lpszClassName = TEXT("MainWndClass");

   return RegisterClass(&wc);
}

/*
 * InitInstance
 *
 * Purpose:
 *  Intializes OLE and creates Lines Application object
 *
 * Parameters:
 *  hinst           hInstance of application
 *
 * Return Value:
 *  TRUE if initialization succeeded, FALSE otherwise.
 */
BOOL InitInstance (HINSTANCE hinst)
{
    HRESULT hr;


    // Intialize OLE
    hr = OleInitialize(NULL);
    if (FAILED(hr))
       return FALSE;

    // Create an instance of the Lines Application object. Object is
    // created with refcount 0.
    hr = CApplication::Create(hinst, &g_pApplication);
    if (FAILED(hr))
        return FALSE;
    return TRUE;
}

/*
 * ProcessCmdLine
 *
 * Purpose:
 *  Check if command line contains /Automation. If so, the class factory of
the
 *  application object is registered.  If not, the application was started
stand-alone and
```

```
 *  so the window is shown. The application object is registered using
RegisterActiveObject.
 *
 * Parameters:
 *  pCmdLine       Command line passed to application
 *  pdwRegisterCF   Returns id returned after class factory registration.
Can be used to
 *                  revoke class factory registration.
 *  pdwRegisterActiveObject   Returns id returned after active object
registration. Can be used to
 *                  revoke active object registration.
 *  nCmdShow        Specifies how window is to be shown if application was
started stand alone.
 *
 * Return Value:
 *    TRUE if OLE initialization succeeded, FALSE otherwise.
 *
 */
BOOL ProcessCmdLine(LPSTR pCmdLine, LPDWORD pdwRegisterCF, LPDWORD
pdwRegisterActiveObject, int nCmdShow)
{
   LPCLASSFACTORY pcf = NULL;
   HRESULT hr;

   *pdwRegisterCF = 0;
   *pdwRegisterActiveObject = 0;

   // Expose class factory for application object if command line contains
the
   // Automation switch
   if (_fstrstr(pCmdLine, "-Automation") != NULL
       || _fstrstr(pCmdLine, "/Automation") != NULL)
   {
       pcf = new CApplicationCF;
       if (!pcf)
           goto error;
       pcf->AddRef();
       hr = CoRegisterClassObject(CLSID_Lines, pcf,
                                  CLSCTX_LOCAL_SERVER, REGCLS_SINGLEUSE,
                                  pdwRegisterCF);
       if (hr != NOERROR)
           goto error;
       pcf->Release();
   }
   else g_pApplication->ShowWindow(nCmdShow);    // Show window if started
stand-alone

   // Register Lines application object in the Running Object Table (ROT).
This
   // allows controllers to connect to a running application object instead
of creating
   // a new instance. Use weak registration so that the ROT releases it's
reference when
   // all external references are released. If strong registration is used,
the ROT will not
```

```
   // release it's reference until RevokeActiveObject is called and so will
keep
   // the object alive even after all external references have been
released.
   RegisterActiveObject(g_pApplication, CLSID_Lines, ACTIVEOBJECT_WEAK,
pdwRegisterActiveObject);
   return TRUE;

error:
    if (pcf)
        pcf->Release();
    return FALSE;
}

/*
 * Uninitialize
 *
 *  Purpose:
 *   Revoke class factory and active object registration and uninitialize
OLE.
 *
 * Parameters:
 *  dwRegisterCF ID returned after class factory registration.
 *  dwRegisterActiveObject ID returned after active object registration.
 *
 */
void Uninitialize(DWORD dwRegisterCF, DWORD dwRegisterActiveObject)
{
    if (dwRegisterCF != 0)
        CoRevokeClassObject(dwRegisterCF);
    if (dwRegisterActiveObject != 0)
        RevokeActiveObject(dwRegisterActiveObject, NULL);
    OleUninitialize();
}

/*
 * MainWndProc
 *
 * Purpose:
 *  Window procedure for main window
 *
 */
LRESULT CALLBACK MainWndProc (HWND hwnd, UINT msg, WPARAM wParam, LPARAM
lParam)
{
   switch (msg)
   {
      case WM_PAINT:
          g_pApplication->Draw();
          break;

      case WM_SIZE:
          g_pApplication->OnSize(LOWORD(lParam), HIWORD(lParam));
          break;
      case WM_COMMAND:
      {
          switch(GET_WM_COMMAND_ID(wParam,lParam))
          {
              case IDM_DRAWLINE:
                  g_pApplication->CreateAndDrawLine();
                  break;

              case IDM_CLEAR:
                  g_pApplication->ClearPane();
                  break;

              case IDM_EXIT:
                  PostMessage(hwnd, WM_CLOSE, 0, 0L);
                  break;
          }
      }
      break;

      case WM_CLOSE:
         // Hide the window to release the refcount added by
CoLockObjectExternal
         // (See CLines::ShowWindow)
         g_pApplication->m_bUserClosing = TRUE;
         g_pApplication->ShowWindow(SW_HIDE);
         DestroyWindow(hwnd);
         return 0L;

      case WM_DESTROY:
         PostQuitMessage(0);
         break;

      default:
         return DefWindowProc(hwnd, msg, wParam, lParam);
   }

   return NULL;
}

/*
 * DrawLineDialogFunc
 *
 * Purpose:
 *  Dialog function of DrawLine dialog. Prompts user and returns information
to draw a line.
 *
 */
BOOL CALLBACK DrawLineDialogFunc(HWND hwndDlg, UINT msg, WPARAM wParam,
LPARAM lParam)
{
   static LPLINEINFO plineinfo;
   BOOL b;
   CHOOSECOLOR cc;
   COLORREF clrCustom[16];
   int n;

   switch (msg)
   {
      case WM_INITDIALOG:
         plineinfo = (LPLINEINFO)lParam;
         // Initialize edit controls. Units are twips.
         SetDlgItemText(hwndDlg, IDC_STARTPOINT_X, TEXT("4500"));
         SetDlgItemText(hwndDlg, IDC_STARTPOINT_Y, TEXT("4500"));
         SetDlgItemText(hwndDlg, IDC_ENDPOINT_X, TEXT("8000"));
         SetDlgItemText(hwndDlg, IDC_ENDPOINT_Y, TEXT("8000"));
         SetDlgItemText(hwndDlg, IDC_THICKNESS, TEXT("100"));
         plineinfo->colorref = 0;
         return TRUE;

      case WM_COMMAND:
         switch(GET_WM_COMMAND_ID(wParam,lParam))
         {
             case IDOK:
                 plineinfo->ptStart.x = GetDlgItemInt(hwndDlg,
IDC_STARTPOINT_X, &b, TRUE);
                 plineinfo->ptStart.y = GetDlgItemInt(hwndDlg,
IDC_STARTPOINT_Y, &b, TRUE);
                 plineinfo->ptEnd.x = GetDlgItemInt(hwndDlg, IDC_ENDPOINT_X,
&b, TRUE);
                 plineinfo->ptEnd.y = GetDlgItemInt(hwndDlg, IDC_ENDPOINT_Y,
&b, TRUE);
                 plineinfo->nThickness = GetDlgItemInt(hwndDlg,
IDC_THICKNESS, &b, TRUE);
                 EndDialog(hwndDlg, IDOK);
                 return TRUE;

             case IDCANCEL:
                 EndDialog(hwndDlg, IDCANCEL);
                 return TRUE;

             case IDC_CHOOSECOLOR:
                 memset(&cc, 0, sizeof(CHOOSECOLOR));
                 cc.lStructSize = sizeof(CHOOSECOLOR);
                 cc.hwndOwner = hwndDlg;
                 cc.rgbResult = RGB(0, 0, 0);
                 cc.Flags = CC_RGBINIT;
                 for (n=0; n<16; n++)
                    clrCustom[n] = RGB(255, 255, 255);
                 cc.lpCustColors = clrCustom;
                 if (ChooseColor(&cc))
                     plineinfo->colorref = cc.rgbResult;
         }
         break;
   }
   return FALSE;
}

/*
 * LoadTypeInfo
 *
 *  Purpose:
 *   Gets type information of an object's interface from type library.
 *
 * Parameters:
 *  ppunkStdDispatch    Returns type information.
 *  clsid               Interface id of object in type library.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT LoadTypeInfo(ITypeInfo FAR* FAR* pptinfo, REFCLSID clsid)
{
    HRESULT hr;
    LPTYPELIB ptlib = NULL;
    LPTYPEINFO ptinfo = NULL;

    *pptinfo = NULL;

    // Load Type Library.
    hr = LoadRegTypeLib(LIBID_Lines, 1, 0, 0x0409, &ptlib);
    if (FAILED(hr))
    {
        // if it wasn't registered, try to load it from the path
        // if this succeeds, it will have registered the type library for us
        // for the next time.
        hr = LoadTypeLib(OLESTR("lines.tlb"), &ptlib);
        if(FAILED(hr))
            return hr;
    }

    // Get type information for interface of the object.
    hr = ptlib->GetTypeInfoOfGuid(clsid, &ptinfo);
    if (FAILED(hr))
    {
        ptlib->Release();
        return hr;
    }


    ptlib->Release();
    *pptinfo = ptinfo;
    return NOERROR;
}

#ifdef WIN32

#ifndef UNICODE
char* ConvertToAnsi(OLECHAR FAR* szW)
{
  static char achA[STRCONVERT_MAXLEN];

  WideCharToMultiByte(CP_ACP, 0, szW, -1, achA, STRCONVERT_MAXLEN, NULL,
NULL);
```

```
  return achA;
}

OLECHAR* ConvertToUnicode(char FAR* szA)
{
  static OLECHAR achW[STRCONVERT_MAXLEN];

  MultiByteToWideChar(CP_ACP, 0, szA, -1, achW, STRCONVERT_MAXLEN);
  return achW;
}
#endif

#endif
```

PANE.CPP   (LINES OLE Sample)

```
/*************************************************************************
**
**    OLE Automation Lines Object.
**
**    pane.cpp
**
**    CPane implementation
**
**     Written by Microsoft Product Support Services, Windows Developer 
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "lines.h"

/*
 * CPane::Create
 *
 * Purpose:
 *  Creates an instance of the Pane automation object and initializes it.
 *
 * Parameters:
 *  hwnd      Handle of the window of the pane.
 *  ppPane    Returns Pane automation object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CPane::Create(HWND hwnd, CPane FAR* FAR* ppPane )
{
    HRESULT hr;
    CPane FAR* pPane = NULL;

    *ppPane = NULL;

    pPane = new CPane();
    if (pPane == NULL)
        goto error;

    pPane->m_hwnd = hwnd;
    // Create an empty Lines collection of size 100
    hr = CLines::Create(100, 1, pPane, &pPane->m_pLines);
    if (FAILED(hr))
       goto error;
    pPane->m_pLines->AddRef();

    // Create an empty Points collection of size 200
    hr = CPoints::Create(200, 1, pPane, &pPane->m_pPoints);
    if (FAILED(hr))
       goto error;
    pPane->m_pPoints->AddRef();

    // Load type information for the pane object from type library.
    hr = LoadTypeInfo(&pPane->m_ptinfo, IID_IPane);
    if (FAILED(hr))
        goto error;

    *ppPane = pPane;
    return NOERROR;

error:
    if (pPane == NULL)
        return ResultFromScode(E_OUTOFMEMORY);

    if (pPane->m_ptinfo)
        pPane->m_ptinfo->Release();
    if (pPane->m_pLines)
        pPane->m_pLines->Release();
    if (pPane->m_pPoints)
        pPane->m_pPoints->Release();

    // Set to NULL to prevent destructor from attempting to free again
    pPane->m_ptinfo = NULL;
    pPane->m_pLines = NULL;
    pPane->m_pPoints = NULL;

    delete pPane;
    return hr;
}

/*
 * CPane::CPane
 *
 * Purpose:
 *  Constructor for CPane object. Initializes members to NULL.
 *
 */
#pragma warning (disable : 4355)
CPane::CPane() : m_SupportErrorInfo(this, IID_IPane)
#pragma warning (default : 4355)
{
    m_cRef = 0;
    m_hwnd = NULL;
    m_ptinfo = NULL;
    m_pLines = NULL;
    m_pPoints = NULL;
    m_nMaxX = -1;
    m_nMaxY = -1;
}

/*
 * CPane::~CPane
 *
 * Purpose:
 *  Destructor for CPane object.
 *
 */
CPane::~CPane()
{
     if (m_ptinfo) m_ptinfo->Release();
     if (m_pLines) m_pLines->Release();
     if (m_pPoints) m_pPoints->Release();
}

/*
 * CPane::QueryInterface, AddRef, Release
 *
 * Purpose:
 *  Implements IUnknown::QueryInterface, AddRef, Release
 *
 */
STDMETHODIMP
CPane::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
    *ppv = NULL;

    if (iid == IID_IUnknown || iid == IID_IDispatch || iid == IID_IPane)
        *ppv = this;
    else if (iid == IID_ISupportErrorInfo)
        *ppv = &m_SupportErrorInfo;
    else return ResultFromScode(E_NOINTERFACE);

    AddRef();
    return NOERROR;
}


STDMETHODIMP_(ULONG)
CPane::AddRef(void)
{
#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, Pane\r\n"), m_cRef+1);
    OutputDebugString(ach);
#endif

    return ++m_cRef;
}
```

```
STDMETHODIMP_(ULONG)
CPane::Release(void)
{
#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, Pane\r\n"), m_cRef-1);
    OutputDebugString(ach);
#endif

    if(--m_cRef == 0)
    {
        delete this;
        return 0;
    }
    return m_cRef;
}

/*
 * CPane::GetTypeInfoCount
 *
 * Purpose:
 *  Implements IDispatch::GetTypeInfoCount.
 *
 */
STDMETHODIMP
CPane::GetTypeInfoCount(UINT FAR* pctinfo)
{
    *pctinfo = 1;
    return NOERROR;
}

/*
 * CPane::GetTypeInfo
 *
 * Purpose:
 *  Implements IDispatch::GetTypeInfo.
 *
 */
STDMETHODIMP
CPane::GetTypeInfo(
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo)
{
    *pptinfo = NULL;

    if(itinfo != 0)
        return ResultFromScode(DISP_E_BADINDEX);

    m_ptinfo->AddRef();
    *pptinfo = m_ptinfo;

    return NOERROR;
}
```

```
/*
 * CPane::GetIDsOfNames
 *
 * Purpose:
 *  Implements IDispatch::GetIDsOfNames.  The standard implementation,
DispGetIDsOfNames,
 *  is used.
 *
 */
STDMETHODIMP
CPane::GetIDsOfNames(
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid)
{
    return DispGetIDsOfNames(m_ptinfo, rgszNames, cNames, rgdispid);
}

/*
 * CPane::Invoke
 *
 * Purpose:
 *  Implements IDispatch::Invoke.  The standard implementation, DispInvoke,
 *  is used. Properties and methods exposed by the pane object will
 *  set m_bRaiseException to raise an exception.
 *
 */
STDMETHODIMP
CPane::Invoke(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr)
{
    HRESULT hr;

    m_bRaiseException = FALSE;
    hr =  DispInvoke(
        this, m_ptinfo,
        dispidMember, wFlags, pdispparams,
        pvarResult, pexcepinfo, puArgErr);
    if (m_bRaiseException)
    {
       if (NULL != pexcepinfo)
           _fmemcpy(pexcepinfo, &m_excepinfo, sizeof(EXCEPINFO));
       return ResultFromScode(DISP_E_EXCEPTION);
    }
    else return hr;
}

/*
 * CPane::get_Lines
 *
 * Purpose:
 *  Returns the collection of all line objects in the drawing.
 *
 */
STDMETHODIMP
CPane::get_Lines(ILines FAR* FAR* ppLines)
{

    HRESULT hr;

    hr = m_pLines->QueryInterface(IID_IDispatch, (void FAR* FAR*)ppLines);
    if (FAILED(hr))
        return RaiseException(IDS_Unexpected);
    return NOERROR;
}

/*
 * CPane::get_Points
 *
 * Purpose:
 *  Returns the collection of all point objects in the drawing.
 *
 */
STDMETHODIMP
CPane::get_Points(IPoints FAR* FAR* ppPoints)
{
    HRESULT hr;

    hr = m_pPoints->QueryInterface(IID_IDispatch, (void FAR* FAR*)ppPoints);
    if (FAILED(hr))
        return RaiseException(IDS_Unexpected);
    return NOERROR;
}

/*
 * CPane::get_MaxX, get_MaxY
 *
 * Purpose:
 *  Returns the maximum visible X & Y coordinate values in twips.
 *
 */
STDMETHODIMP
CPane::get_MaxX(int FAR* pnMaxX)
{
     *pnMaxX = m_nMaxX;
    if (m_nMaxX == -1)
        return RaiseException(IDS_NoVisibleXCoordinate);
    return NOERROR;
}
```

```
STDMETHODIMP
CPane::get_MaxY(int FAR* pnMaxY)
{
    *pnMaxY = m_nMaxY;
    if (m_nMaxY == -1)
        return RaiseException(IDS_NoVisibleYCoordinate);
    return NOERROR;
}

/*
 * CPane::Clear
 *
 * Purpose:
 *  Removes all the Lines and Points from the drawing and refreshes the
client area.
 *  The result is a blank slate as if the application had just been
launched.
 *
 */
STDMETHODIMP
CPane::Clear()
{
    m_pLines->Clear();
    m_pPoints->Clear();
    ::InvalidateRect(m_hwnd, NULL, TRUE);
    UpdateWindow(m_hwnd);
    m_nMaxX = m_nMaxY = -1;
    return NOERROR;
}

/*
 * CPane::Refresh
 *
 * Purpose:
 *  Invalidates the entire drawing window so the drawing is cleared
 *  and each of the lines in the Lines collection is redrawn.
 *
 */
STDMETHODIMP
CPane::Refresh()
{
    ::InvalidateRect(m_hwnd, NULL, TRUE);
    UpdateWindow(m_hwnd);
    return NOERROR;
}

/*
 *
 * The following methods are not exposed through Automation
 *
 */

STDMETHODIMP_(void)
CPane::Draw(void)
{
    PAINTSTRUCT ps;

    BeginPaint(m_hwnd, &ps);
    SetMapMode(ps.hdc, MM_TWIPS);
    SetWindowOrgEx(ps.hdc, 0, m_nHeight, NULL);
    m_pLines->Draw(ps.hdc);
    EndPaint(m_hwnd, &ps);
}

/*
 * CPane::OnSize
 *
 * Purpose:
 *  Called when the pane is resized. Remembers the new size and gets the new
maximum visible
 *  x and y coordinates.
 *
 */
STDMETHODIMP_(void)
CPane::OnSize(unsigned int nWidth, unsigned int nHeight)
{
   HDC hdc;
   POINT pt;
   RECT rc;

   // Translate the height and width of pane from device units to twips.
   hdc = ::GetDC(m_hwnd);
   SetMapMode(hdc, MM_TWIPS);
   pt.x = nWidth; pt.y = nHeight;
   DPtoLP(hdc, &pt, 1);
   m_nWidth = pt.x;
   m_nHeight = -pt.y;
   ::ReleaseDC(m_hwnd, hdc);

   // Get the new max visible x and y coordinates
   SetRect(&rc, 0, 0, m_nWidth, m_nHeight);
   m_pPoints->GetMaxXMaxY(&m_nMaxX, &m_nMaxY, &rc);
}

STDMETHODIMP_(HDC)
CPane::GetDC()
{
    HDC hdc;

    hdc = ::GetDC(m_hwnd);
    SetMapMode(hdc, MM_TWIPS);
    SetWindowOrgEx(hdc, 0, m_nHeight, NULL);
    return hdc;
}

STDMETHODIMP_(void)
CPane::ReleaseDC(HDC hdc)
{
    ::ReleaseDC(m_hwnd, hdc);
```

```
}

STDMETHODIMP_(void)
CPane::InvalidateRect(LPRECT prc)
{
    RECT rc;
    HDC hdc;
    int t;

   // Translate the area to be invalidated from twips to device units
   hdc = ::GetDC(m_hwnd);
   SetMapMode(hdc, MM_TWIPS);
   SetWindowOrgEx(hdc, 0, m_nHeight, NULL);
   rc = *prc;
   LPtoDP(hdc, (LPPOINT)&rc.left, 1);
   LPtoDP(hdc, (LPPOINT)&rc.right, 1);

   // Make sure left is not greater than right and top is not greater than
bottom. Otherwise
   // InvalidateRect will not invalidate.
   if (rc.right < rc.left)
   {
       t = rc.right;
       rc.right = rc.left;
       rc.left = t;
   }
   if (rc.bottom < rc.top)
   {
       t = rc.bottom;
       rc.bottom = rc.top;
       rc.top = t;
   }

   ::InvalidateRect(m_hwnd, &rc, TRUE);
   ::ReleaseDC(m_hwnd, hdc);
}

STDMETHODIMP_(void)
CPane::Update(void)
{
    UpdateWindow(m_hwnd);
}

/*
 * CPane::AddPoint
 *
 * Purpose:
 *  Adds a point to the point collection. Updates max visible coordinates is
required.
 *
 */
STDMETHODIMP_(BOOL)
CPane::AddPoint(CPoint FAR* pPoint)
{
    int nX, nY;
    nX = pPoint->get_x();
    nY = pPoint->get_y();

    if (nX > m_nMaxX && nX <= m_nWidth)
        m_nMaxX = nX;
    if (nY > m_nMaxY && nY <= m_nHeight)
        m_nMaxY = nY;

    return m_pPoints->Add(pPoint);
}

/*
 * CPane::RemovePoint
 *
 * Purpose:
 *  Removes point from point collection. Updates max visible coordinates is
required.
 *
 */
STDMETHODIMP_(void)
CPane::RemovePoint(CPoint FAR* pPoint)
{
    RECT rc;

    // If point was removed from the collection, update max visible X & Y
coordinates
    if (m_pPoints->Remove(pPoint))
    {
        SetRect(&rc, 0, 0, m_nWidth, m_nHeight);
        m_pPoints->GetMaxXMaxY(&m_nMaxX, &m_nMaxY, &rc);
    }
}

/*
 * CPane::RaiseException
 *
 * Purpose:
 *  Fills the EXCEPINFO structure and signal IDispatch::Invoke to return
DISP_E_EXCEPTION.
 *  Sets ErrorInfo object for vtable-binding controllers.
 *
 */
STDMETHODIMP
CPane::RaiseException(int nID)
{
    extern SCODE g_scodes[];
    TCHAR szError[STR_LEN];
    ICreateErrorInfo *pcerrinfo;
    IErrorInfo *perrinfo;
    HRESULT hr;

    _fmemset(&m_excepinfo, 0, sizeof(EXCEPINFO));

    m_excepinfo.wCode = nID;
    if (LoadString(g_pApplication->m_hinst, nID, szError, sizeof(szError)))
        m_excepinfo.bstrDescription =
SysAllocString(TO_OLE_STRING(szError));
    m_excepinfo.bstrSource = SysAllocString(g_pApplication->m_bstrProgID);

    m_bRaiseException = TRUE;

    // Set ErrInfo object so that vtable binding containers can get
    // rich error information.
    hr = CreateErrorInfo(&pcerrinfo);
    if (SUCCEEDED(hr))
    {
       pcerrinfo->SetGUID(IID_IPane);
       if (m_excepinfo.bstrSource)
           pcerrinfo->SetSource(m_excepinfo.bstrSource);
       if (m_excepinfo.bstrDescription)
           pcerrinfo->SetDescription(m_excepinfo.bstrDescription);
       hr = pcerrinfo->QueryInterface(IID_IErrorInfo, (LPVOID FAR*)
&perrinfo);
       if (SUCCEEDED(hr))
       {
          SetErrorInfo(0, perrinfo);
          perrinfo->Release();
       }
       pcerrinfo->Release();
    }

    return ResultFromScode(g_scodes[nID-1001]);
}
```

POINT.CPP   (LINES OLE Sample)

```
/*************************************************************************
**
**    OLE Automation Points Object.
**
**    point.cpp
**
**    CPoint implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1994 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "lines.h"

/*
 * CPoint::Create
 *
 * Purpose:
 *  Creates an instance of the Point automation object and initializes it.
 *
 * Parameters:
 *  ppPoint    Returns Point automation object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CPoint::Create(CPoint FAR* FAR* ppPoint )
{
    HRESULT hr;
    CPoint FAR* pPoint = NULL;

    *ppPoint = NULL;

    pPoint = new CPoint();
    if (pPoint == NULL)
        goto error;

    // Load type information for the point from type library.
    hr = LoadTypeInfo(&pPoint->m_ptinfo, IID_IPoint);
    if (FAILED(hr))
        goto error;

    *ppPoint = pPoint;
    return NOERROR;

error:
    if (pPoint == NULL)
        return ResultFromScode(E_OUTOFMEMORY);

    if (pPoint->m_ptinfo)
        pPoint->m_ptinfo->Release();

    // Set to NULL to prevent destructor from attempting to free again
    pPoint->m_ptinfo = NULL;

    delete pPoint;
    return hr;
}

/*
 * CPoint::CPoint
 *
 * Purpose:
 *  Constructor for CPoint object. Initializes members to NULL.
 *
 */
#pragma warning (disable : 4355)
CPoint::CPoint() : m_SupportErrorInfo(this, IID_IPoint)
#pragma warning (default : 4355)
{
    m_cRef = 0;
    m_cInternalRef = 0;
    m_ptinfo = NULL;
    m_nX = -1;
    m_nY = -1;
}

/*
 * CPoint::~CPoint
 *
 * Purpose:
 *  Destructor for CPoint object. Frees Point message BSTR and default
 *  IDispatch implementation. Closes the aplication.
 *
 */
CPoint::~CPoint()
{
     if (m_ptinfo) m_ptinfo->Release();
}

/*
 * CPoint::QueryInterface, AddRef, Release
 *
 * Purpose:
```

```
 *  Implements IUnknown::QueryInterface, AddRef, Release
 *
 */
STDMETHODIMP
CPoint::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
    *ppv = NULL;

    if (iid == IID_IUnknown || iid == IID_IDispatch || iid == IID_IPoint)
        *ppv = this;
    else if (iid == IID_ISupportErrorInfo)
        *ppv = &m_SupportErrorInfo;
    else return ResultFromScode(E_NOINTERFACE);

    AddRef();
    return NOERROR;
}


STDMETHODIMP_(ULONG)
CPoint::AddRef(void)
{
#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, Point\r\n"), m_cRef+1);
    OutputDebugString(ach);
#endif

    return ++m_cRef;
}

STDMETHODIMP_(ULONG)
CPoint::Release(void)
{
#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, Point\r\n"), m_cRef-1);
    OutputDebugString(ach);
#endif

    if(--m_cRef == 0)
    {
        delete this;
        return 0;
    }
    return m_cRef;
}

/*
 * CPoint::GetTypeInfoCount
 *
 * Purpose:
 *  Implements IDispatch::GetTypeInfoCount.
 *
 */
STDMETHODIMP
CPoint::GetTypeInfoCount(UINT FAR* pctinfo)
{
    *pctinfo = 1;
    return NOERROR;
}

/*
 * CPoint::GetTypeInfo
 *
 * Purpose:
 *  Implements IDispatch::GetTypeInfo.
 *
 */
STDMETHODIMP
CPoint::GetTypeInfo(
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo)
{
    *pptinfo = NULL;

    if(itinfo != 0)
        return ResultFromScode(DISP_E_BADINDEX);

    m_ptinfo->AddRef();
    *pptinfo = m_ptinfo;

    return NOERROR;
}

/*
 * CPoint::GetIDsOfNames
 *
 * Purpose:
 *  Implements IDispatch::GetIDsOfNames.  The standard implementation,
DispGetIDsOfNames,
 *  is used.
 *
 */
STDMETHODIMP
CPoint::GetIDsOfNames(
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid)
{
    return DispGetIDsOfNames(m_ptinfo, rgszNames, cNames, rgdispid);
}

/*
 * CPoint::Invoke
 *
 * Purpose:
 *  Implements IDispatch::Invoke.  The standard implementation, DispInvoke,
 *  is used.
 *
 */
STDMETHODIMP
CPoint::Invoke(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr)
{
    return DispInvoke(
        this, m_ptinfo,
        dispidMember, wFlags, pdispparams,
        pvarResult, pexcepinfo, puArgErr);
}

STDMETHODIMP
CPoint::get_x(int FAR* pnX)
{
    *pnX = m_nX;
    return NOERROR;
}

STDMETHODIMP
CPoint::put_x(int nX)
{
    m_nX = nX;
    return NOERROR;
}

STDMETHODIMP
CPoint::get_y(int FAR* pnY)
{
    *pnY = m_nY;
    return NOERROR;
}

STDMETHODIMP
CPoint::put_y(int nY)
{
    m_nY = nY;
    return NOERROR;
}

/*
 *
 * The following methods are not exposed through Automation
 *
 */
```

```
/*
 * CPoint::InternalAddRef, InternalRelease
 *
 * Purpose:
 *  Implements an internal ref count for the use of the points collection.
 *  The points collection does not have duplicates, instead a reference
count is incremented.
 *  A point is removed from the collection when the reference count drops to
0.
 *
 */
STDMETHODIMP_(ULONG)
CPoint::InternalAddRef(void)
{
#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Internal Ref = %ld, Point\r\n"), m_cInternalRef+1);
    OutputDebugString(ach);
#endif

    return ++m_cInternalRef;
}

STDMETHODIMP_(ULONG)
CPoint::InternalRelease(void)
{
#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Internal Ref = %ld, Point\r\n"), m_cInternalRef-1);
    OutputDebugString(ach);
#endif

    return --m_cInternalRef;
}

STDMETHODIMP_(int)
CPoint::get_x(void)
{
    return m_nX;
}


STDMETHODIMP_(int)
CPoint::get_y(void)
{
    return m_nY;
}
```

POINTS.CPP (LINES OLE Sample)

```
/*************************************************************************
**
**    OLE Automation Points object
**
**    points.cpp
**
**    CPoints collection implementation
**
**     Written by Microsoft Product Support Services, Windows Developer
Support
**    (c) Copyright Microsoft Corp. 1993 All Rights Reserved
**
*************************************************************************/

#include <windows.h>
#include <windowsx.h>
#ifdef WIN16
  #include <ole2.h>
  #include <compobj.h>
  #include <dispatch.h>
  #include <variant.h>
  #include <olenls.h>
#endif
#include "lines.h"

/*
 * CPoints::Create
 *
 * Purpose:
 *  Creates an instance of a collection object and initializes it.
 *
 * Parameters:
 *  lMaxSize   Maximum number of items that can added to collection.
 *  lLBound    Lower bound of index of collection.
 *  pPane      Pointer to pane that contains this collection. Required
because the pane coordinates
 *             the lines and points collection.
 *  ppPoints    Returns collection object.
 *
 * Return Value:
 *  HRESULT
 *
 */
HRESULT
CPoints::Create(ULONG lMaxSize, long lLBound, CPane FAR* pPane, CPoints FAR*
FAR* ppPoints)
{
    HRESULT hr;
    CPoints FAR* pPoints = NULL;
    SAFEARRAYBOUND sabound[1];

    *ppPoints = NULL;
    // Create new collection
    pPoints = new CPoints();
    if (pPoints == NULL)
        goto error;

    pPoints->m_cMax = lMaxSize;
    pPoints->m_cElements = 0;
    pPoints->m_lLBound = lLBound;
    pPoints->m_pPane = pPane;

    // Load type information for the points collection from type library.
    hr = LoadTypeInfo(&pPoints->m_ptinfo, IID_IPoints);
    if (FAILED(hr))
        goto error;

    // Create a safe array which is used to implement the collection.
    sabound[0].cElements = lMaxSize;
    sabound[0].lLbound = lLBound;
    pPoints->m_psa = SafeArrayCreate(VT_VARIANT, 1, sabound);
    if (pPoints->m_psa == NULL)
    {
        hr = ResultFromScode(E_OUTOFMEMORY);
        goto error;
    }

    *ppPoints = pPoints;
    return NOERROR;

error:
    if (pPoints == NULL)
        return ResultFromScode(E_OUTOFMEMORY);
    if (pPoints->m_ptinfo)
        pPoints->m_ptinfo->Release();
    if (pPoints->m_psa)
        SafeArrayDestroy(pPoints->m_psa);

    pPoints->m_psa = NULL;
    pPoints->m_ptinfo = NULL;

    delete pPoints;
    return hr;
}

/*
 * CPoints::CPoints
 *
 * Purpose:
 *  Constructor for CPoints object. Initializes members to NULL.
 *
 */
#pragma warning (disable : 4355)
CPoints::CPoints() : m_SupportErrorInfo(this, IID_IPoints)
#pragma warning (default : 4355)
{
    m_cRef = 0;
    m_psa = NULL;
    m_ptinfo = NULL;
}

/*
 * CPoints::~CPoints
 *
 * Purpose:
 *  Destructor for CPoints object.
 *
 */
CPoints::~CPoints()
{
     if (m_ptinfo) m_ptinfo->Release();
     if (m_psa) SafeArrayDestroy(m_psa);
}

/*
 * CPoints::QueryInterface, AddRef, Release
 *
 * Purpose:
 *  Implements IUnknown::QueryInterface, AddRef, Release
 *
 */
STDMETHODIMP
CPoints::QueryInterface(REFIID iid, void FAR* FAR* ppv)
{
    *ppv = NULL;

    if (iid == IID_IUnknown || iid == IID_IPoints || iid == IID_IDispatch)
        *ppv = this;
    else if (iid == IID_ISupportErrorInfo)
        *ppv = &m_SupportErrorInfo;
    else return ResultFromScode(E_NOINTERFACE);

    AddRef();
    return NOERROR;
}


STDMETHODIMP_(ULONG)
CPoints::AddRef(void)
{

#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, Points\r\n"), m_cRef+1);
    OutputDebugString(ach);
#endif

    return ++m_cRef;
}
```

```
STDMETHODIMP_(ULONG)
CPoints::Release(void)
{

#ifdef _DEBUG
    TCHAR ach[50];
    wsprintf(ach, TEXT("Ref = %ld, Points\r\n"), m_cRef-1);
    OutputDebugString(ach);
#endif

    if(--m_cRef == 0)
    {
        delete this;
        return 0;
    }
    return m_cRef;
}

/*
 * CPoints::GetTypeInfoCount
 *
 * Purpose:
 *  Implements IDispatch::GetTypeInfoCount.
 *
 */
STDMETHODIMP
CPoints::GetTypeInfoCount(UINT FAR* pctinfo)
{
    *pctinfo = 1;
    return NOERROR;
}

/*
 * CPoints::GetTypeInfo
 *
 * Purpose:
 *  Implements IDispatch::GetTypeInfo.
 *
 */
STDMETHODIMP
CPoints::GetTypeInfo(
      UINT itinfo,
      LCID lcid,
      ITypeInfo FAR* FAR* pptinfo)
{
    *pptinfo = NULL;

    if(itinfo != 0)
        return ResultFromScode(DISP_E_BADINDEX);

    m_ptinfo->AddRef();
    *pptinfo = m_ptinfo;

    return NOERROR;
```

```
}

/*
 * CPoints::GetIDsOfNames
 *
 * Purpose:
 *  Implements IDispatch::GetIDsOfNames.  The standard implementation,
DispGetIDsOfNames,
 *  is used.
 *
 */
STDMETHODIMP
CPoints::GetIDsOfNames(
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      UINT cNames,
      LCID lcid,
      DISPID FAR* rgdispid)
{
    return DispGetIDsOfNames(m_ptinfo, rgszNames, cNames, rgdispid);
}

/*
 * CPoints::Invoke
 *
 * Purpose:
 *  Implements IDispatch::Invoke.  The standard implementation, DispInvoke,
 *  is used. Properties and methods exposed by the points object will
 *  set m_bRaiseException to raise an exception.
 *
 */
STDMETHODIMP
CPoints::Invoke(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      WORD wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      UINT FAR* puArgErr)
{
    HRESULT hr;

    m_bRaiseException = FALSE;
    hr =  DispInvoke(
        this, m_ptinfo,
        dispidMember, wFlags, pdispparams,
        pvarResult, pexcepinfo, puArgErr);
    if (m_bRaiseException)
    {
       if (NULL != pexcepinfo)
           _fmemcpy(pexcepinfo, &m_excepinfo, sizeof(EXCEPINFO));
       return ResultFromScode(DISP_E_EXCEPTION);
    }
    else return hr;
}

/*
 * CPoints::get_Count
 *
 * Purpose:
 *  Returns number of items in collection.
 *
 */
STDMETHODIMP
CPoints::get_Count(long FAR* plCount)
{
    *plCount = m_cElements;
    return NOERROR;
}

/*
 * CPoints::get_Item
 *
 * Purpose:
 *  Retrieves item from collection, given an index.
 *
 * Parameters:
 *   lIndex   Index of item to be retrieved.
 *
 * Returns
 *  IDispatch of item retrieved from collection.
 *
 */
STDMETHODIMP
CPoints::get_Item(long lIndex, IPoint FAR* FAR* ppPoint)
{
    HRESULT hr;
    VARIANT v;
    LPDISPATCH pdisp = NULL;

    // Check if integer index is within range
    if (lIndex < m_lLBound || lIndex >= (long)(m_lLBound+m_cElements))
        return RaiseException(IDS_InvalidIndex);

    // Retrieve and return item. Note that SafeArrayGetElement AddRefs so an
additional AddRef
    // is not required.
    VariantInit(&v);
    hr = SafeArrayGetElement(m_psa, &lIndex, &v);
    if (FAILED(hr))
        return RaiseException(IDS_Unexpected);
    *ppPoint = (IPoint FAR*) V_DISPATCH(&v);
    return NOERROR;
}

/*
 * CPoints::get_NewEnum
 *
 * Purpose:
 *  Returns an enumerator (IEnumVARIANT) for the items curently in the
collection.
 *  The NewEnum property is restricted and so is invisible to users of an
 *  automation controller's scripting language. Automation controllers that
support
 *  a 'For Each' statement to iterate through the elements of a collection
will use
 *  the enumerator returned by NewEnum. The enumerator creates a snapshot of
the
 *  the current state of the collection.
 *
 */
STDMETHODIMP
CPoints::get__NewEnum(IUnknown FAR* FAR* ppunkEnum)
{
    CEnumVariant FAR* penum = NULL;
    HRESULT hr;

    // Create new enumerator for items currently in collection and QI for
IUnknown
    hr = CEnumVariant::Create(m_psa, m_cElements, &penum);
    if (FAILED(hr))
        {hr = RaiseException(IDS_OutOfMemory); goto error;}
    hr = penum->QueryInterface(IID_IUnknown, (VOID FAR* FAR*)ppunkEnum);
    if (FAILED(hr))
        {hr = RaiseException(IDS_Unexpected); goto error;}
    return NOERROR;

error:
    if (penum)
        delete penum;
    return hr;
}

/*
 *
 * The following methods are not exposed through Automation
 *
 */

/*
 * CPoints::Add
 *
 * Purpose:
 *  Adds an item to the collection. The points collection does not have
duplicates.
 *
 * Parameters:
 *  pPointAdd    Point to be added to collection.
 *
 */
STDMETHODIMP_(BOOL)
CPoints::Add(CPoint FAR* pPointAdd)
```

```
{
    HRESULT hr;
    LONG l;
    VARIANT v;
    VARIANT HUGEP *pvar;
    int nX, nY;
    BOOL bFound = FALSE;
    LPDISPATCH pdispPointAdd = NULL;
    CPoint FAR *pPoint;
    LPDISPATCH pdispPoint;

    // Is the collection full?
    if (m_cElements == m_cMax)
        goto error;

    // Get coordinates of point to be added to collection
    nX = pPointAdd->get_x();
    nY = pPointAdd->get_y();

    hr = SafeArrayAccessData(m_psa, (void HUGEP* FAR*)&pvar);
    if (FAILED(hr))
        goto error;

    // Check if point to be added is already in the collection.
    for (l=0; l<(long)m_cElements; l++)
    {
        pdispPoint = V_DISPATCH(&pvar[l]);
        hr = pdispPoint->QueryInterface(IID_IPoint, (void FAR*
FAR*)&pPoint);
        if (FAILED(hr))
            continue;
        if (nX == pPoint->get_x() &&  nY == pPoint->get_y())
        {
            // Point is already in the collection. AddRef it.
            // Internal AddRef keeps track of number of times this point was
added to this collection. When
            //    this internal ref count drops to 0, this point can be
removed from this collection.
            pPoint->InternalAddRef();
            pdispPoint->AddRef();


            bFound = TRUE;
            pPoint->Release();
            break;
        }
        pPoint->Release();
    }
    hr = SafeArrayUnaccessData(m_psa);
    if (FAILED(hr))
       goto error;

    // Add point if it was not in the collection.
    if (!bFound)
    {
        l = m_lLBound+m_cElements;

        hr = pPointAdd->QueryInterface(IID_IDispatch, (void FAR*
FAR*)&pdispPointAdd);
        if (FAILED(hr))
            goto error;
        VariantInit(&v);
        V_VT(&v) = VT_DISPATCH;
        V_DISPATCH(&v) = pdispPointAdd;
        hr = SafeArrayPutElement(m_psa, &l, &v);
        if (FAILED(hr))
            goto error;

        // Internal AddRef keeps track of number of times this point was
added to this collection. When
        //    this internal ref count drops to 0, this point can be removed
from this collection.
        pPointAdd->InternalAddRef();
        pdispPointAdd->Release();

        m_cElements++;
    }
    return TRUE;

error:
    if (pdispPointAdd)
        pdispPointAdd->Release();
    return FALSE;
}

/*
 * CPoints::Remove
 *
 * Purpose:
 *  Removes specified item from collection. The points collection does not
have duplicates.
 *
 * Parameters:
 *   pPointRemove    Point to be removed from collection.
 *
 */
STDMETHODIMP_(BOOL)
CPoints::Remove(CPoint FAR* pPointRemove)
{
    HRESULT hr;
    long l, lIndex;
    VARIANT HUGEP *pvar;
    ULONG cRef;
    int nX, nY;
    CPoint FAR* pPoint = NULL;
    BOOL bFound = FALSE;
    BOOL bRet = FALSE;

    // Get coordinates of point to be removed from collection
    nX = pPointRemove->get_x();
    nY = pPointRemove->get_y();

    hr = SafeArrayAccessData(m_psa, (void HUGEP* FAR*)&pvar);
    if (FAILED(hr))
       goto error;


    // Check if point to be removed is in the collection.
    for (l=0; l<(long)m_cElements; l++)
    {
        hr = V_DISPATCH(&pvar[l])->QueryInterface(IID_IPoint, (void FAR*
FAR*)&pPoint);
        if (FAILED(hr))
            continue;

        if (nX == pPoint->get_x() &&  nY == pPoint->get_y())
        {
            // Release point. Note that this collection does not have
duplicates. Duplicate
            // points are handled by increasing the ref count.
            V_DISPATCH(&pvar[l])->Release();
            cRef = pPoint->InternalRelease();
            bFound = TRUE;
            lIndex = l;
            pPoint->Release();
            break;
        }
        pPoint->Release();
    }

    // If the internal ref count of point to be removed has dropped to 0,
move up the array elements
    // after the element to be removed.
    if (bFound && cRef == 0)
    {
        for (l=lIndex; l<(long)(m_cElements-1); l++)
            pvar[l] = pvar[l+1];

        // Remove last element.
        V_VT(&pvar[l]) = VT_EMPTY;
        m_cElements--;
        bRet = TRUE;
    }

    hr = SafeArrayUnaccessData(m_psa);
    if (FAILED(hr))
       goto error;
    return bRet;

error:
    return FALSE;
}

/*
 * CPoints::Clear
 *
 * Purpose:
 *  Removes all items from collection.
 *
 */
STDMETHODIMP_(void)
CPoints::Clear(void)
{
    SafeArrayDestroyData(m_psa);
    SafeArrayAllocData(m_psa);
    m_cElements = 0;
}

/*
 * CPoints::GetMaxXMaxY
 *
 * Purpose:
 *  Get the maximum X and Y coordinates that are in the rectangle, prc.
 *
 */

STDMETHODIMP_(void)
CPoints::GetMaxXMaxY(int FAR *pnX, int FAR *pnY, LPRECT prc)
{
    HRESULT hr;
    long l;
    VARIANT HUGEP *pvar;
    int nMaxX, nMaxY;
    CPoint FAR* pPoint;
    POINT pt;

    hr = SafeArrayAccessData(m_psa, (void HUGEP* FAR*)&pvar);
    if (FAILED(hr))
       return;

    nMaxX = -1; nMaxY = -1;

    // Find the maximume X and Y coordinates that are in the prc rectangle.
    for (l=0; l<(long)m_cElements; l++)
    {
        V_DISPATCH(&pvar[l])->QueryInterface(IID_IPoint, (void FAR*
FAR*)&pPoint);
        pt.x = pPoint->get_x();
        pt.y = pPoint->get_y();

        if (PtInRect(prc, pt))
        {
            if (pt.x > nMaxX)
                nMaxX = pt.x;

            if (pt.y > nMaxY)
                nMaxY = pt.y;
        }
        pPoint->Release();
    }
    *pnX = nMaxX;
    *pnY = nMaxY;

    SafeArrayUnaccessData(m_psa);
    return;
}

/*
 * CPoints::RaiseException
 *
 * Purpose:
 *  Fills the EXCEPINFO structure and signal IDispatch::Invoke to return
DISP_E_EXCEPTION.
 *  Sets ErrorInfo object for vtable-binding controllers.
 *
 */
STDMETHODIMP
CPoints::RaiseException(int nID)
{
    extern SCODE g_scodes[];
    TCHAR szError[STR_LEN];
    ICreateErrorInfo *pcerrinfo;
    IErrorInfo *perrinfo;
    HRESULT hr;

    _fmemset(&m_excepinfo, 0, sizeof(EXCEPINFO));

    m_excepinfo.wCode = nID;
    if (LoadString(g_pApplication->m_hinst, nID, szError, sizeof(szError)))
        m_excepinfo.bstrDescription =
SysAllocString(TO_OLE_STRING(szError));
    m_excepinfo.bstrSource = SysAllocString(g_pApplication->m_bstrProgID);

    m_bRaiseException = TRUE;

    // Set ErrInfo object so that vtable binding containers can get
    // rich error information.
    hr = CreateErrorInfo(&pcerrinfo);
    if (SUCCEEDED(hr))
    {
       pcerrinfo->SetGUID(IID_IPoints);
       if (m_excepinfo.bstrSource)
           pcerrinfo->SetSource(m_excepinfo.bstrSource);
       if (m_excepinfo.bstrDescription)
           pcerrinfo->SetDescription(m_excepinfo.bstrDescription);
       hr = pcerrinfo->QueryInterface(IID_IErrorInfo, (LPVOID FAR*)
&perrinfo);
       if (SUCCEEDED(hr))
       {
          SetErrorInfo(0, perrinfo);
          perrinfo->Release();
       }
       pcerrinfo->Release();
    }
    return ResultFromScode(g_scodes[nID-1001]);
}
```

BTTNCUR

This sample creates a DLL for generating standard Windows button images and cursors. For information on compiling and building the sample, see MAKEFILE   (BTTNCUR Sample).

MAKEFILE (BTTNCUR Sample)

```
#
# Makefile
# Standard Button Images and Cursors DLL Version 1.1, March 1993
#
#
!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

all: bttncur.dll

OLE_FLAGS = -DWIN32S

!ifndef NO_DEBUG
OLE_FLAGS = $(OLE_FLAGS) /D_DEBUG /DDEBUG /D_DEBUGTRACE=0
!endif

.SUFFIXES: .h .c .obj .exe .res .rc

#
#


INCLS    = ..\include\bttncur.h bttncuri.h
OBJS     = bttncur.obj cursors.obj

RCFILES1 = bttncur.rcv res\harrows.cur res\help.cur res\larrows.cur
RCFILES2 = res\magnify.cur res\neswarrs.cur res\nodrop.cur res\nwsearrs.cur
RCFILES3 = res\rarrow.cur res\sarrows.cur res\sizebarh.cur res\sizebarv.cur
RCFILES4 = res\splith.cur res\splitv.cur res\tabletop.cur res\varrows.cur
RCFILES5 = res\stdim72.bmp res\stdim96.bmp res\stdim120.bmp
RCFILES  = $(RCFILES1) $(RCFILES2) $(RCFILES3) $(RCFILES4) $(RCFILES5)


#
# Tool Directives
#
.c.obj:
    $(cc) $(cflags) $(cvars) $(cdebug) $(OLE_FLAGS) $*.c

.rc.res:
    rc -r -DWIN32 -DDEBUG $*.rc

clean:
    -del *.obj
    -del *.dll
    -del *.res
    -del *.lib
    -del *.exp
    -del *.map

bttncur.dll: $(OBJS) bttncur.lib bttncur.def bttncur.res
    $(link) $(linkdebug) $(dlllflags) bttncur.exp $(OBJS) $*.res -out:$@
-map:$*.map $(guilibsdll) -base:0x76000000
```

```
    if not exist ..\bin mkdir ..\bin
    copy bttncur.dll ..\bin

bttncur.lib: $(OBJS) bttncur.def
    $(implib) $(OBJS) -out:bttncur.lib -def:bttncur.def -machine:$(CPU)
    if not exist ..\lib mkdir ..\lib
    copy bttncur.lib ..\lib

##### Dependencies #####
bttncur.obj  : bttncur.c    $(INCLS)
cursors.obj  : cursors.c    $(INCLS)
bttncur.res  : bttncur.rc   $(INCLS) $(RCFILES)
```

BTTNCUR.DEF (BTTNCUR Sample)

```
LIBRARY     BTTNCUR
DESCRIPTION 'Windows Interface Buttons & Cursors 1.1, (c)1992-1993 Microsoft
Corporation'
CODE        EXECUTE READ SHARED
EXPORTS     WEP                         @1 RESIDENTNAME
            UIToolButtonDraw            @2
            UICursorLoad                @3
            UIToolConfigureForDisplay   @4
            UIToolButtonDrawTDD         @5
```

BTTNCUR.RC (BTTNCUR Sample)

```
/*
 * BTTNCUR.RC
 * Buttons & Cursors Version 1.1, March 1993
 *
 * Resource file for the Button Image & Cursors DLL, including cursors and
 * standard bitmaps.
 *
 * Copyright (c)1992-1993 Microsoft Corporation, All Rights Reserved,
 * as applied to redistribution of this source code in source form
 * License is granted to use of compiled code in shipped binaries.
 */

#define STRICT
#include <windows.h>
#include "bttncur.h"
#include "bttncuri.h"

IDB_STANDARDIMAGES72    BITMAP      res\stdim72.bmp
IDB_STANDARDIMAGES96    BITMAP      res\stdim96.bmp
IDB_STANDARDIMAGES120   BITMAP      res\stdim120.bmp

IDC_RIGHTARROW      CURSOR          res\rarrow.cur
IDC_CONTEXTHELP     CURSOR          res\help.cur
IDC_MAGNIFY         CURSOR          res\magnify.cur
IDC_NODROP          CURSOR          res\nodrop.cur
IDC_TABLETOP        CURSOR          res\tabletop.cur
IDC_HSIZEBAR        CURSOR          res\sizebarh.cur
IDC_VSIZEBAR        CURSOR          res\sizebarv.cur
IDC_HSPLITBAR       CURSOR          res\splith.cur
IDC_VSPLITBAR       CURSOR          res\splitv.cur
IDC_SMALLARROWS     CURSOR          res\sarrows.cur
IDC_LARGEARROWS     CURSOR          res\larrows.cur
IDC_HARROWS         CURSOR          res\harrows.cur
IDC_VARROWS         CURSOR          res\varrows.cur
IDC_NESWARROWS      CURSOR          res\neswarrs.cur
IDC_NWSEARROWS      CURSOR          res\nwsearrs.cur


//Include the version information
rcinclude bttncur.rcv
```

BTTNCUR.RCV (BTTNCUR Sample)

```
/*
 * BTTNCUR.RCV
 * Buttons & Cursors Version 1.1, March 1993
 *
 * Version resource file for the Button Images and Cursors DLL.
 *
 * Copyright (c)1992-1993 Microsoft Corporation, All Rights Reserved,
 * as applied to redistribution of this source code in source form
 * License is granted to use of compiled code in shipped binaries.
 */

#ifndef WIN32
#include <ver.h>
#endif

//Default is nodebug
#ifndef DEBUG
#define VER_DEBUG                   0
#else
#define VER_DEBUG                   VS_FF_DEBUG
#endif

VS_VERSION_INFO VERSIONINFO
 FILEVERSION        1,1,0,0
 PRODUCTVERSION     1,1,0,0
 FILEFLAGSMASK      VS_FFI_FILEFLAGSMASK
 FILEFLAGS          VER_DEBUG
 FILEOS             VOS_DOS_WINDOWS16
 FILETYPE           VFT_DLL
 FILESUBTYPE        VFT_UNKNOWN

 BEGIN
   BLOCK "StringFileInfo"
     BEGIN
      BLOCK "040904E4"
       BEGIN
        VALUE "CompanyName",     "Microsoft Corporation\0", "\0"
        VALUE "FileDescription", "Microsoft Windows(R) Standard Button
Images and Cursors", "\0"
        VALUE "FileVersion",     "1.10\0", "\0"
        VALUE "InternalName",    "BTTNCUR.DLL", "\0"
        VALUE "LegalCopyright",  "Copyright \251 1993 Microsoft Corp.", "\0"
        VALUE "OriginalFilename","BTTNCUR.DLL", "\0"
        VALUE "ProductName",     "Microsoft Windows(TM) Standard Button
Images and Cursors", "\0"
        VALUE "ProductVersion",  "1.10\0"
       END
   END

   BLOCK "VarFileInfo"
     BEGIN
      VALUE "Translation", 0x0409, 0x04E4
```

```
    END
END
```

BTTNCUR.C (BTTNCUR Sample)

```
/*
 * BTTNCUR.C
 * Buttons & Cursors Version 1.1, March 1993
 *
 * Public functions to generate different states of toolbar buttons from
 * a single bitmap.  States are normal, pressed, checked, and disabled.
 *
 * Copyright (c)1992-1993 Microsoft Corporation, All Rights Reserved,
 * as applied to redistribution of this source code in source form
 * License is granted to use of compiled code in shipped binaries.
 */

#define STRICT
#include <windows.h>
#include <memory.h>
#include "bttncur.h"
#include "bttncuri.h"



//Display sensitive information
TOOLDISPLAYDATA     tdd;

//Library instance
HINSTANCE           ghInst;
/*
23/1/94 (kerenm)
Add intance count for 32bit dll
*/
int                 cInstances = 0;

//Cache GDI objects to speed drawing.
HDC     hDCGlyphs    = NULL;
HDC     hDCMono      = NULL;
HBRUSH  hBrushDither = NULL;

//Standard images to use in case caller doesn't provide them
HBITMAP rghBmpStandardImages[3];

//Standard button colors.
const COLORREF crStandard[4]={ RGB(0, 0, 0)          //STDCOLOR_BLACK
                      , RGB(128, 128, 128)    //STDCOLOR_DKGRAY
                      , RGB(192, 192, 192)    //STDCOLOR_LTGRAY
                      , RGB(255, 255, 255)};  //STDCOLOR_WHITE


/*
 * Mapping from image identifier to button type (command/attribute).
 * Version 1.00 of this DLL has no attribute images defined, so
 * the code will only support three states for each command
 * button.  Any state is, however, valid for an application
 * defined image.
 */

UINT mpButtonType[TOOLIMAGE_MAX-TOOLIMAGE_MIN+1]=
      {
      BUTTONTYPE_COMMAND, BUTTONTYPE_COMMAND, BUTTONTYPE_COMMAND,
      BUTTONTYPE_COMMAND, BUTTONTYPE_COMMAND, BUTTONTYPE_COMMAND,
      BUTTONTYPE_COMMAND, BUTTONTYPE_COMMAND, BUTTONTYPE_COMMAND
      };




/*
 * LibMain
 *
 * Purpose:
 *  DLL-specific entry point called from LibEntry.  Initializes
 *  global variables and loads standard image bitmaps.
 *
 * Parameters:
 *  hInstance       HANDLE instance of the DLL.
 *  wDataSeg        WORD segment selector of the DLL's data segment.
 *  wHeapSize       WORD byte count of the heap.
 *  lpCmdLine       LPSTR to command line used to start the module.
 *
 * Return Value:
 *  HANDLE          Instance handle of the DLL.
 */

BOOL xxxLibMain(HINSTANCE hInstance)
   {
    int i;

/*
23/1/94 (kerenm)
For WIN32S - should done only once. On NT ghInst is instance dataso it will
always be zero.
*/
#ifdef WIN32
   if (ghInst) {
      return(TRUE);
   }
#endif
   ghInst=hInstance;

    tdd.uDPI     =96;
   tdd.cyBar    =CYBUTTONBAR96;
   tdd.cxButton =TOOLBUTTON_STD96WIDTH;
   tdd.cyButton =TOOLBUTTON_STD96HEIGHT;
   tdd.cxImage  =TOOLBUTTON_STD96IMAGEWIDTH;
   tdd.cyImage  =TOOLBUTTON_STD96IMAGEHEIGHT;
   tdd.uIDImages=IDB_STANDARDIMAGES96;

   for (i=0; i < 3; i++)
      {
```

```
      rghBmpStandardImages[i]=LoadBitmap(hInstance
         , MAKEINTRESOURCE(IDB_STANDARDIMAGESMIN+i));

      if (NULL==rghBmpStandardImages[i])
         return FALSE;
      }


   //Perform global initialization.
   if (ToolButtonInit())
      {
      CursorsCache(hInstance);


      return TRUE;
      }

   return FALSE;

   }

#ifdef WIN32

void FAR PASCAL WEP(int);

BOOL WINAPI DllMain
(
   HINSTANCE hInstance,
   ULONG     Reason,
   PCONTEXT  Context
)
{
   OutputDebugString("bttncur LibMain:  bttncur.dll loaded\r\n");

   UNREFERENCED_PARAMETER(Context);

   if (Reason == DLL_PROCESS_DETACH) {

#ifdef WIN32
     if (!--cInstances) {
        WEP(0);
     }
#else
      WEP(0);
#endif
      return TRUE;
   }
   else if (Reason != DLL_PROCESS_ATTACH)
      return TRUE;

/* 1/23/94   */
#ifdef WIN32
   cInstances++;
#endif
   return xxxLibMain(hInstance);
```

```
}

#else

HANDLE FAR PASCAL LibMain(HANDLE hInstance, WORD wDataSeg
                    , WORD cbHeapSize, LPSTR lpCmdLine)
   {

    //Perform global initialization.
   if (xxxLibMain(hInstance))
      if (0!=cbHeapSize)
         UnlockData(0);

   return hInstance;

   }

#endif


/*
 * WEP
 *
 * Purpose:
 *  Required DLL Exit function.  Does nothing.
 *
 * Parameters:
 *  bSystemExit     BOOL indicating if the system is being shut
 *                  down or the DLL has just been unloaded.
 *
 * Return Value:
 *  void
 *
 */

void FAR PASCAL WEP(int bSystemExit)
   {
   /*
    * **Developers:  Note that WEP is called AFTER Windows does any
    *                automatic task cleanup.  You may see warnings for
    *                that two DCs, a bitmap, and a brush, were not
    *                deleted before task termination.  THIS IS NOT A
    *                PROBLEM WITH THIS CODE AND IT IS NOT A BUG.  This
    *                WEP function is properly called and performs the
    *                cleanup as appropriate.  The fact that Windows is
    *                calling WEP after checking task cleanup is not
    *                something we can control.  Just to prove it, the
    *                OutputDebugStrings in this and ToolButtonFree
    *                show that the code is exercised.
    */

#ifdef DEBUG
   OutputDebugString ("BttnCur WEP Entry\r\n");
#endif
```

```
   CursorsFree();
   ToolButtonFree();
#ifdef DEBUG
   OutputDebugString ("BttnCur WEP Exit\r\n");
#endif
   return;
   }






/*
 * UIToolConfigureForDisplay
 * Public API
 *
 * Purpose:
 *  Initializes the library to scale button images for the display type.
 *  Without calling this function the library defaults to 96 DPI (VGA).
 *  By calling this function an application acknowledges that it must
 *  use the data returned from this function to configure itself for
 *  the display.
 *
 * Parameters:
 *  lpDD            LPTOOLDISPLAYDATA to fill with the display-sensitive
 *                  size values.
 *
 * Return Value:
 *  BOOL            TRUE if the sizes were obtained, FALSE otherwise.
 */

BOOL WINAPI UIToolConfigureForDisplay(LPTOOLDISPLAYDATA lpDD)
   {
   int         cy;
   HDC         hDC;


   if (NULL==lpDD || IsBadWritePtr(lpDD, sizeof(TOOLDISPLAYDATA)))
      return FALSE;

   /*
    * Determine the aspect ratio of the display we're currently
    * running on and calculate the necessary information.
    *
    * By retrieving the logical Y extent of the display driver, you
    * only have limited possibilities:
    *      LOGPIXELSY      Display
    *      ----------------------------------------
    *         48             CGA    (unsupported)
    *         72             EGA
    *         96             VGA
    *        120             8514/a (i.e. HiRes VGA)
    */

   hDC=GetDC(NULL);
```

```
if (NULL==hDC)
   return FALSE;

cy=GetDeviceCaps(hDC, LOGPIXELSY);
ReleaseDC(NULL, hDC);

/*
 * Instead of single comparisons, check ranges instead, so in case
 * we get something funky, we'll act reasonable.
 */
if (72 >=cy)
   {
   lpDD->uDPI     =72;
   lpDD->cyBar    =CYBUTTONBAR72;
   lpDD->cxButton =TOOLBUTTON_STD72WIDTH;
   lpDD->cyButton =TOOLBUTTON_STD72HEIGHT;
   lpDD->cxImage  =TOOLBUTTON_STD72IMAGEWIDTH;
   lpDD->cyImage  =TOOLBUTTON_STD72IMAGEHEIGHT;
   lpDD->uIDImages=IDB_STANDARDIMAGES72;
   }
else
   {
   if (72 < cy && 120 > cy)
      {
      lpDD->uDPI     =96;
      lpDD->cyBar    =CYBUTTONBAR96;
      lpDD->cxButton =TOOLBUTTON_STD96WIDTH;
      lpDD->cyButton =TOOLBUTTON_STD96HEIGHT;
      lpDD->cxImage  =TOOLBUTTON_STD96IMAGEWIDTH;
      lpDD->cyImage  =TOOLBUTTON_STD96IMAGEHEIGHT;
      lpDD->uIDImages=IDB_STANDARDIMAGES96;
      }
   else
      {
      lpDD->uDPI     =120;
      lpDD->cyBar    =CYBUTTONBAR120;
      lpDD->cxButton =TOOLBUTTON_STD120WIDTH;
      lpDD->cyButton =TOOLBUTTON_STD120HEIGHT;
      lpDD->cxImage  =TOOLBUTTON_STD120IMAGEWIDTH;
      lpDD->cyImage  =TOOLBUTTON_STD120IMAGEHEIGHT;
      lpDD->uIDImages=IDB_STANDARDIMAGES120;
      }
   }

return TRUE;
}
```

```
/*
 * ToolButtonInit
 * Internal
 *
 * Purpose:
 *  Initializes GDI objects for drawing images through UIToolButtonDraw.
 *  If the function fails, the function has already performed proper
 *  cleanup.
 *
 * Parameters:
 *  None
 *
 * Return Value:
 *  BOOL            TRUE if initialization succeeded.  FALSE otherwise.
 */

static BOOL ToolButtonInit(void)
   {
   COLORREF        rgbHi;

   //DC for BitBltting the image (the glyph)
   hDCGlyphs=CreateCompatibleDC(NULL);

   //Create a monochrome DC and a brush for doing pattern dithering.
   hDCMono=CreateCompatibleDC(NULL);

   //Windows 3.0 doesn't support COLOR_BTNHIGHLIGHT, so leave it white.
   if (0x0300 < (UINT)GetVersion())
      rgbHi=GetSysColor(COLOR_BTNHIGHLIGHT);
   else
      rgbHi=crStandard[STDCOLOR_WHITE];

   hBrushDither=HBrushDitherCreate(GetSysColor(COLOR_BTNFACE), rgbHi);

   if (NULL==hDCGlyphs || NULL==hDCMono || NULL==hBrushDither)
      {
      //On failure, cleanup whatever might have been allocated.
      ToolButtonFree();
      return FALSE;
      }

   return TRUE;
   }





/*
 * ToolButtonFree
 * Internal
 *
 * Purpose:
 *  Free all GDI allocations made during initialization.  Note that the
 *  DEBUG output included here shows that WEP is called and cleanup actually
 *  occurs.  However, if you watch debug output in DBWIN or on a terminal,
 *  the debugging version of Windows does automatic app cleanup before WEP
 *  is called, leading some to believe that this code is buggy.  The
 *  debug output below shows that we do perform all necessary cleanup.
 *
 * Parameters:
 *  None
 *
 * Return Value:
 *  None
 */

static void ToolButtonFree(void)
   {
   UINT        i;

   if (NULL!=hDCMono)
      DeleteDC(hDCMono);
   hDCMono=NULL;


   if (NULL!=hDCGlyphs)
      DeleteDC(hDCGlyphs);
   hDCGlyphs=NULL;

   if (NULL!=hBrushDither)
      DeleteObject(hBrushDither);
   hBrushDither=NULL;

   for (i=0; i < 3; i++)
      {
      if (NULL!=rghBmpStandardImages[i])
         DeleteObject(rghBmpStandardImages[i]);
      rghBmpStandardImages[i]=NULL;
      }

   return;
   }





/*
 * HBrushDitherCreate
 * Internal
 *
 * Purpose:
 *  Creates and returns a handle to a pattern brush created from
 *  an 8*8 monochrome pattern bitmap.  We use the button face and
 *  highlight colors to indicate the resulting colors of a PatBlt
 *  using this brush.
 *
 * Parameters:
 *  rgbFace         COLORREF of the button face color.
 *  rgbHilight      COLORREF of the button highlight color.
 *
 * Return Value:
 *  HBITMAP         Handle to the dither bitmap.
 */

static HBRUSH HBrushDitherCreate(COLORREF rgbFace, COLORREF rgbHilight)
   {
   struct  //BITMAPINFO with 16 colors
      {
      BITMAPINFOHEADER    bmiHeader;
      RGBQUAD             bmiColors[16];
      } bmi;

   HBRUSH          hBrush=NULL;
   DWORD           patGray[8];
   HDC             hDC;
   HBITMAP         hBmp;
   static COLORREF rgbFaceOld   =0xFFFFFFFF;  //Initially an impossible
color
   static COLORREF rgbHilightOld=0xFFFFFFFF;  //so at first we always create

   /*
    * If the colors haven't changed from last time, just return the
    * existing brush.
    */
   if (rgbFace==rgbFaceOld && rgbHilight==rgbHilightOld)
      return hBrushDither;

   rgbFaceOld=rgbFace;
   rgbHilightOld=rgbHilight;

   /*
    * We're going to create an 8*8 brush for PatBlt using the
    * button face color and button highlight color.  We use this
    * brush to affect the pressed state and the disabled state.
    */
   bmi.bmiHeader.biSize         = sizeof(BITMAPINFOHEADER);
   bmi.bmiHeader.biWidth        = 8;
   bmi.bmiHeader.biHeight       = 8;
   bmi.bmiHeader.biPlanes       = 1;
   bmi.bmiHeader.biBitCount     = 1;
   bmi.bmiHeader.biCompression  = BI_RGB;
   bmi.bmiHeader.biSizeImage    = 0;
   bmi.bmiHeader.biXPelsPerMeter= 0;
   bmi.bmiHeader.biYPelsPerMeter= 0;
   bmi.bmiHeader.biClrUsed      = 0;
   bmi.bmiHeader.biClrImportant = 0;

   bmi.bmiColors[0].rgbBlue     = GetBValue(rgbFace);
   bmi.bmiColors[0].rgbGreen    = GetGValue(rgbFace);
   bmi.bmiColors[0].rgbRed      = GetRValue(rgbFace);
   bmi.bmiColors[0].rgbReserved = 0;

   bmi.bmiColors[1].rgbBlue     = GetBValue(rgbHilight);
   bmi.bmiColors[1].rgbGreen    = GetGValue(rgbHilight);
   bmi.bmiColors[1].rgbRed      = GetRValue(rgbHilight);
   bmi.bmiColors[1].rgbReserved = 0;

   //Create the byte array for CreateDIBitmap.
   patGray[6]=patGray[4]=patGray[2]=patGray[0]=0x5555AAAAL;
   patGray[7]=patGray[5]=patGray[3]=patGray[1]=0xAAAA5555L;

   //Create the bitmap
   hDC=GetDC(NULL);
   hBmp=CreateDIBitmap(hDC, &bmi.bmiHeader, CBM_INIT, patGray
               , (LPBITMAPINFO)&bmi, DIB_RGB_COLORS);
   ReleaseDC(NULL, hDC);

   //Create the brush from the bitmap
   if (NULL!=hBmp)
      {
      hBrush=CreatePatternBrush(hBmp);
      DeleteObject(hBmp);
      }

   /*
    * If we could recreate a brush, clean up and make it the current
    * pattern.  Otherwise the best we can do it return the old one,
    * which will be colored wrong, but at least it works.
    */
   if (NULL!=hBrush)
      {
      if (NULL!=hBrushDither)
         DeleteObject(hBrushDither);

      hBrushDither=hBrush;
      }

   return hBrushDither;
   }





/*
 * UIToolButtonDraw
 * Public API
 *
 * Purpose:
 *  Draws the complete image of a toolbar-style button with a given
 *  image in the center and in a specific state.  The button is drawn
 *  on a specified hDC at a given location, so this function is useful
 *  on standard owner-draw buttons as well as on toolbar controls that
 *  have only one window but show images of multiple buttons.
 *
 * Parameters:
 *  hDC             HDC on which to draw.
 *  x, y            int coordinates at which to draw.
 *  dx, dy          int dimensions of the *button*, not necessarily the
image.
 *  hBmp            HBITMAP from which to draw the image.
 *  bmx, bmy        int dimensions of each bitmap in hBmp.  If hBmp is NULL
 *                  then these are forced to the standard sizes.
 *  iImage          int index to the image to draw in the button
 *  uStateIn        UINT containing the state index for the button and the
 *                  color control bits.
 *
 * Return Value:
 *  BOOL            TRUE if drawing succeeded, FALSE otherwise meaning that
 *                  hDC is NULL or hBmp is NULL and iImage is not a valid
 *                  index for a standard image.
 */

BOOL WINAPI UIToolButtonDraw(HDC hDC, int x, int y, int dx, int dy
   , HBITMAP hBmp, int bmx, int bmy, int iImage, UINT uStateIn)
   {
   return UIToolButtonDrawTDD(hDC, x, y, dx, dy, hBmp, bmx, bmy, iImage
      , uStateIn, &tdd);
   }





/*
 * UIToolButtonDrawTDD
 * Public API
 *
 * Purpose:
 *  Draws the complete image of a toolbar-style button with a given
 *  image in the center and in a specific state.  The button is drawn
 *  on a specified hDC at a given location, so this function is useful
 *  on standard owner-draw buttons as well as on toolbar controls that
 *  have only one window but show images of multiple buttons.
 *
 *  This is the same as UIToolButtonDraw but adds the pTDD configuration
 *  structure.  UIToolButtonDraw calls us with that pointing to the
 *  default 96dpi structure.
 *
 * Parameters:
 *  hDC             HDC on which to draw.
 *  x, y            int coordinates at which to draw.
 *  dx, dy          int dimensions of the *button*, not necessarily the
image.
 *  hBmp            HBITMAP from which to draw the image.
 *  bmx, bmy        int dimensions of each bitmap in hBmp.  If hBmp is NULL
 *                  then these are forced to the standard sizes.
 *  iImage          int index to the image to draw in the button
 *  uStateIn        UINT containing the state index for the button and the
 *                  color control bits.
 *  pTDD            LPTOOLDISPLAYDATA containing display configuration.
 *                  Can be NULL if hBmp is non-NULL.
 *
 * Return Value:
 *  BOOL            TRUE if drawing succeeded, FALSE otherwise meaning that
 *                  hDC is NULL or hBmp is NULL and iImage is not a valid
 *                  index for a standard image.
 */

BOOL WINAPI UIToolButtonDrawTDD(HDC hDC, int x, int y, int dx, int dy
   , HBITMAP hBmp, int bmx, int bmy, int iImage, UINT uStateIn
   , LPTOOLDISPLAYDATA pTDD)
   {
   static COLORREF crSys[5];  //Avoid stack arrays in DLLs: use static
   UINT            uState=(UINT)LOBYTE((WORD)uStateIn);
   UINT            uColors=(UINT)HIBYTE((WORD)uStateIn & PRESERVE_ALL);
   int             xOffsetGlyph, yOffsetGlyph;
   int             i, iSaveDC;
   HDC             hMemDC;
   HGDIOBJ         hObj;
   HBRUSH          hBR;
   HBITMAP         hBmpT;
   HBITMAP         hBmpMono;
   HBITMAP         hBmpMonoOrg;
   HBITMAP         hBmpSave=NULL;

   if (NULL==hDC)
      return FALSE;

   /*
    * If we're given no image bitmap, then use the standard and validate the
    * image index.  We also enforce the standard bitmap size and the size of
    * the button (as requested by User Interface designers).
    */
   if (NULL==hBmp && !(uState & BUTTONGROUP_BLANK))
      {
      hBmp=rghBmpStandardImages[pTDD->uIDImages-IDB_STANDARDIMAGESMIN];

      bmx=pTDD->cxImage;            //Force bitmap dimensions
      bmy=pTDD->cyImage;

      dx=pTDD->cxButton;            //Force button dimensions
      dy=pTDD->cyButton;

      if (iImage > TOOLIMAGE_MAX)
         return FALSE;

      /*
       * If we are using a standard command button, verify that the state
       * does not contain the LIGHTFACE group which only applies to
       * attribute buttons.
       */
      if (BUTTONTYPE_COMMAND==mpButtonType[iImage]
         && (uState & BUTTONGROUP_LIGHTFACE))
         return FALSE;
      }
   //Create a dithered bitmap.
   hBmpMono=CreateBitmap(dx-2, dy-2, 1, 1, NULL);

   if (NULL==hBmpMono)
      return FALSE;

   hBmpMonoOrg=(HBITMAP)SelectObject(hDCMono,  hBmpMono);


   //Save the DC state before we munge on it.
   iSaveDC=SaveDC(hDC);

   /*
    * Draw a button sans image.  This also fills crSys with the system
    * colors for us which has space for five colors.  We don't use the
    * fifth, the frame color, in this function.
    */
   DrawBlankButton(hDC, x, y, dx, dy, (BOOL)(uState & BUTTONGROUP_DOWN),
crSys);

   //Shift coordinates to account for the button's border
   x++;
   y++;
   dx-=2;
   dy-=2;

   /*
    * Determine the offset necessary to center the image but also reflect
    * the pushed-in state, which means just adding 1 to the up state.
    */
   i=(uState & BUTTONGROUP_DOWN) ? 1 : 0;
   xOffsetGlyph=((dx-bmx) >> 1)+i;
   yOffsetGlyph=((dy-bmy) >> 1)+i;


   //Select the given image bitmap into the glyph DC before calling
MaskCreate
   if (NULL!=hBmp)
      hBmpSave=(HBITMAP)SelectObject(hDCGlyphs, hBmp);


   /*
    * Draw the face on the button.  If we have an up or [mouse]down
    * button then we can just draw it as-is.  For indeterminate,
    * disabled, or down disabled we have to gray the image and possibly
    * add a white shadow to it (disabled/down disabled).
    *
    * Also note that for the intermediate state we first draw the normal
    * up state, then proceed to add disabling looking highlights.
    */

   //Up, mouse down, down, indeterminate
   if ((uState & BUTTONGROUP_ACTIVE) && !(uState & BUTTONGROUP_BLANK))
      {
      BOOL            fColorsSame=TRUE;
```

```
/*
 * In here we pay close attention to the system colors.  Where
 * the source image is black, we paint COLOR_BTNTEXT.  Where
 * light gray, we paint COLOR_BTNFACE.  Where dark gray we paint
 * COLOR_BTNSHADOW, and where white we paint COLOR_BTNHILIGHT.
 *
 * The uColors variable contains flags to prevent color
 * conversion.  To do a little optimization, we just do a
 * single BitBlt if we're preserving all colors or if no colors
 * are different than the standards, which is by far the most
 * common case.  Otherwise, cycle through the four colors we can
 * convert and do a BitBlt that converts it to the system color.
 */

//See what colors are different.
for (i=STDCOLOR_BLACK; i<=STDCOLOR_WHITE; i++)
   fColorsSame &= (crSys[i]==crStandard[i]);

if (PRESERVE_ALL==uColors || fColorsSame)
   {
   BitBlt(hDC, x+xOffsetGlyph, y+yOffsetGlyph, bmx, bmy
         , hDCGlyphs, iImage*bmx, 0, SRCCOPY);
   }
else
   {
   /*
    * Cycle through hard-coded colors and create a mask that has all
    * regions of that color in white and all other regions black.
    * Then we select a pattern brush of the color to convert to:
    * if we aren't converting the color then we use a brush of
    * the standard hard-coded color, otherwise we use the actual
    * system color.  The ROP_DSPDxax means that anything that's
    * 1's in the mask get the pattern, anything that's 0 is unchanged
    * in the destination.
    *
    * To prevent too many Blts to the screen, we use an intermediate
    * bitmap and DC.
    */

   hMemDC=CreateCompatibleDC(hDC);

   //Make sure conversion of monochrome to color stays B&W
   SetTextColor(hMemDC, 0L);                     //0's in mono -> 0
   SetBkColor(hMemDC, (COLORREF)0x00FFFFFF);     //1's in mono -> 1

   hBmpT=CreateCompatibleBitmap(hDC, bmx, bmy);
   SelectObject(hMemDC, hBmpT);

   //Copy the unmodified bitmap to the temporary bitmap
   BitBlt(hMemDC, 0, 0, bmx, bmy, hDCGlyphs, iImage*bmx, 0, SRCCOPY);

   for (i=STDCOLOR_BLACK; i<=STDCOLOR_WHITE; i++)
      {
      //Convert pixels of the color to convert to 1's in the mask
```

```
            SetBkColor(hDCGlyphs, crStandard[i]);
            BitBlt(hDCMono, 0, 0, bmx, bmy, hDCGlyphs, iImage*bmx, 0,
SRCCOPY);

            //Preserve or modify the color depending on the flag.
            hBR=CreateSolidBrush((uColors & (1 << i))
                             ? crStandard[i] : crSys[i]);

            if (NULL!=hBR)
               {
               hObj=SelectObject(hMemDC, hBR);

               if (NULL!=hObj)
                  {
                  BitBlt(hMemDC, 0, 0, dx-1, dy-1, hDCMono, 0, 0,
ROP_DSPDxax);
                  SelectObject(hMemDC, hObj);
                  }

               DeleteObject(hBR);
               }
            }

         //Now put the final version on the display and clean up
         BitBlt(hDC, x+xOffsetGlyph, y+yOffsetGlyph, dx-1, dy-1
               , hMemDC, 0, 0, SRCCOPY);

         DeleteDC(hMemDC);
         DeleteObject(hBmpT);

         }
      }


   //Disabled and indeterminate states (unless we're blank)
   if ((uState & BUTTONGROUP_DISABLED ||
ATTRIBUTEBUTTON_INDETERMINATE==uState)
      && !(uState & BUTTONGROUP_BLANK))
      {
      //Grayed state (up or down, no difference)
      MaskCreate(iImage, dx, dy, bmx, bmy, xOffsetGlyph, yOffsetGlyph, 0);

      //Make sure conversion of monochrome to color stays B&W
      SetTextColor(hDC, 0L);                     //0's in mono -> 0
      SetBkColor(hDC, (COLORREF)0x00FFFFFF);     //1's in mono -> 1

      //If we're disabled, up or down, draw the highlighted shadow.
      if (uState & BUTTONGROUP_DISABLED)
         {
         hBR=CreateSolidBrush(crSys[SYSCOLOR_HILIGHT]);

         if (NULL!=hBR)
            {
            hObj=SelectObject(hDC, hBR);
            if (NULL!=hObj)
               {
               //Draw hilight color where we have 0's in the mask
               BitBlt(hDC, x+1, y+1, dx-2, dy-2, hDCMono, 0, 0,
ROP_PSDPxax);
               SelectObject(hDC, hObj);
               }
            DeleteObject(hBR);
            }
         }

      //Draw the gray image.
      hBR=CreateSolidBrush(crSys[SYSCOLOR_SHADOW]);

      if (NULL!=hBR)
         {
         hObj=SelectObject(hDC, hBR);

         if (NULL!=hObj)
            {
            //Draw the shadow color where we have 0's in the mask
            BitBlt(hDC, x, y, dx-2, dy-2, hDCMono, 0, 0, ROP_PSDPxax);
            SelectObject(hDC, hObj);
            }

         DeleteObject(hBR);
         }
      }

   //If the button is selected do the dither brush avoiding the glyph
   if (uState & BUTTONGROUP_LIGHTFACE)
      {
      HBRUSH      hBRDither;

      /*
       * Get the dither brush.  This function will recreate it if
       * necessary or return the global one if the colors already match.
       */
      hBRDither=HBrushDitherCreate(crSys[SYSCOLOR_FACE],
crSys[SYSCOLOR_HILIGHT]);
      hObj=SelectObject(hDC, hBRDither);

      if (NULL!=hObj)
         {
         /*
          * The mask we create now determines where the dithering
          * ends up.  In the down disabled state, we have to preserve
          * the highlighted shadow, so the mask we create must have
          * two masks of the original glyph, one of them offset by
          * one pixel in both x & y.  For the indeterminate state,
          * we have to mask all highlighted areas.  The state passed
          * to MaskCreate matters here (we've used zero before).
          */
         MaskCreate(iImage, dx, dy, bmx, bmy
                  , xOffsetGlyph-1, yOffsetGlyph-1, uState);
         //Convert monochrome masks to B&W color bitmap in the BitBlt.
         SetTextColor(hDC, 0L);
         SetBkColor(hDC, (COLORREF)0x00FFFFFF);

         /*
          * Only draw the dither brush where the mask is 1's.  For
          * the indeterminate state we have to not overdraw the
          * shadow highlight so we use dx-3, dy-3 instead of dx-1
          * and dy-1.  We do this whether or not we're blank.
          */
         i=(ATTRIBUTEBUTTON_INDETERMINATE==uState
            || BLANKBUTTON_INDETERMINATE==uState) ? 3 : 1;

         BitBlt(hDC, x+1, y+1, dx-i, dy-i, hDCMono, 0, 0, ROP_DSPDxax);
         SelectObject(hDC, hObj);
         }

      //DO NOT delete hBRDither!  It's a reference to a shared global.
      }

   //Cleanup hDCGlyphs:  Must do AFTER calling MaskCreate
   if (NULL!=hBmpSave)
      SelectObject(hDCGlyphs, hBmpSave);

   SelectObject(hDCMono,   hBmpMonoOrg);
   DeleteObject(hBmpMono);

   //Restore everything in the DC.
   RestoreDC(hDC, iSaveDC);
   return TRUE;
   }




/*
 * DrawBlankButton
 *
 * Purpose:
 *  Draws a button with no face using the current system colors in either
 *  an up or down state.
 *
 * Parameters:
 *  hDC             HDC on which to draw
 *  x, y            int coordinates where we start drawing
 *  dx,dy           int size of the button
 *  fDown           BOOL indicating the up or down state of the button
 *  pcr             COLORREF FAR * to five colors in which we store text,
 *                  shadow, face, highlight, and frame colors.  This is
 *                  a matter of convenience for the caller, since we have
 *                  to load these colors anyway we might as well send them
 *                  back.
 *
 * Return Value:
 *  None
 */

static void DrawBlankButton(HDC hDC, int x, int y, int dx, int dy
   , BOOL fDown, COLORREF FAR *pcr)
   {
   //Get the current system colors for buttons.
   pcr[0]=GetSysColor(COLOR_BTNTEXT);
   pcr[1]=GetSysColor(COLOR_BTNSHADOW);
   pcr[2]=GetSysColor(COLOR_BTNFACE);

   //Windows 3.0 doesn't support COLOR_BTNHIGHLIGHT, so leave it white.
   if (0x0300 < (UINT)GetVersion())
      pcr[3]=GetSysColor(COLOR_BTNHIGHLIGHT);
   else
      pcr[3]=crStandard[STDCOLOR_WHITE];

   pcr[4]=GetSysColor(COLOR_WINDOWFRAME);

   //Draw the border around the button.
   PatB(hDC, x+1,    y,      dx-2, 1,    pcr[4]);
   PatB(hDC, x+1,    y+dy-1, dx-2, 1,    pcr[4]);
   PatB(hDC, x,      y+1,    1,    dy-2, pcr[4]);
   PatB(hDC, x+dx-1, y+1,    1,    dy-2, pcr[4]);

   //Shift coordinates to account for the border we just drew
   x++;
   y++;
   dx-=2;
   dy-=2;

   //Paint the interior grey as a default.
   PatB(hDC, x, y, dx, dy, pcr[2]);

   /*
    * Draw shadows and highlights.  The DOWN grouping that contains
    * down, mouse down, and down disabled are drawn depressed.  Up,
    * indeterminate, and disabled are drawn up.
    */

   if (fDown)
      {
      PatB(hDC, x, y, 1,  dy, pcr[1]);
      PatB(hDC, x, y, dx, 1,  pcr[1]);
      }
   else
      {
      //Normal button look.
      PatB(hDC, x, y, 1,    dy-1, pcr[3]);
      PatB(hDC, x, y, dx-1, 1,    pcr[3]);

      PatB(hDC, x+dx-1, y,      1,  dy, pcr[1]);
      PatB(hDC, x,      y+dy-1, dx, 1,  pcr[1]);
      PatB(hDC, x+1+dx-3, y+1,    1,    dy-2, pcr[1]);
      PatB(hDC, x+1,      y+dy-2, dx-2, 1,    pcr[1]);
      }

   return;
   }





/*
 * PatB
 * Internal
 *
 * Purpose:
 *  A more convenient PatBlt operation for drawing button borders and
 *  highlights.
 *
 * Parameters:
 *  hDC             HDC on which to paint.
 *  x, y            int coordinates at which to paint.
 *  dx, dy          int dimensions of rectangle to paint.
 *  rgb             COLORREF to use as the background color.
 *
 * Return Value:
 *  None
 */

static void PatB(HDC hDC, int x, int y, int dx, int dy, COLORREF rgb)
   {
   RECT        rc;

   SetBkColor(hDC, rgb);
   SetRect(&rc, x, y, x+dx, y+dy);
   ExtTextOut(hDC, 0, 0, ETO_OPAQUE, &rc, NULL, 0, NULL);
   }




/*
 * MaskCreate
 * Internal
 *
 * Purpose:
 *  Creates a monochrome mask bitmap of the given image at the given offset
 *  in the global hDCMono.  Anywhere in the image that you have the light
 *  gray (STDCOLOR_LTGRAY) or the white highlight (STDCOLOR_WHITE) you get
 *  get 1's.  All other pixels are 0's
 *
 * Parameters:
 *  iImage          UINT index of the image for which to create a mask.
 *  dx, dy          int dimensions of the button.
 *  bmx, bmy        int dimensions of the bitmap to use.
 *  xOffset         int offset for x inside hDCMono where we paint.
 *  yOffset         int offset for y inside hDCMono where we paint.
 *  uState          UINT state of the image.  Special cases are made
 *                  for ATTRIBUTEBUTTON_DOWNDISABLED and
 *                  ATTRIBUTEBUTTON_INDETERMINATE.  In any case where you
 *                  do not want a special case, pass zero here, regardless
 *                  of the true button state.
 *
 * Return Value:
 *  None
 */

static void MaskCreate(UINT iImage, int dx, int dy, int bmx, int bmy
   ,int xOffset, int yOffset, UINT uState)
   {
   //Initalize whole area with zeros
   PatBlt(hDCMono, 0, 0, dx, dy, WHITENESS);

   if (uState & BUTTONGROUP_BLANK)
      return;

   //Convert face colored pixels to 1's. all others to black.
   SetBkColor(hDCGlyphs, crStandard[STDCOLOR_LTGRAY]);
   BitBlt(hDCMono, xOffset, yOffset, bmx, bmy, hDCGlyphs, iImage*bmx, 0,
SRCCOPY);

   //In the indeterminate state, don't turn highlight's to 1's. Leave black.
   if (ATTRIBUTEBUTTON_INDETERMINATE!=uState)
      {
      //Convert highlight colored pixels to 1's and OR them with the
previous.
      SetBkColor(hDCGlyphs, crStandard[STDCOLOR_WHITE]);
      BitBlt(hDCMono, xOffset, yOffset, bmx, bmy, hDCGlyphs, iImage*bmx, 0,
SRCPAINT);
      }

   /*
    * For the down disabled state, AND this same mask with itself at an
    * offset of 1, which accounts for the highlight shadow.
    */
   if (ATTRIBUTEBUTTON_DOWNDISABLED==uState)
      BitBlt(hDCMono, 1, 1, dx-1, dy-1, hDCMono,  0, 0, SRCAND);

   return;
   }
```

BTTNCURI.H   (BTTNCUR Sample)

```
/*
 * BTTNCURI.H
 *
 * Private include file for the Button Images and Cursors DLL.
 *
 * Copyright (c)1992-1993 Microsoft Corporation, All Right Reserved,
 * as applied to redistribution of this source code in source form
 * License is granted to use of compiled code in shipped binaries.
 */

#ifdef __cplusplus
extern "C"
     {
#endif

//Function prototypes.

//BTTNCUR.C
//HANDLE FAR  PASCAL LibMain(HANDLE, WORD, WORD, LPSTR);
//void   FAR  PASCAL WEP(int);
static BOOL        ToolButtonInit(void);
static void        ToolButtonFree(void);
static HBRUSH      HBrushDitherCreate(COLORREF, COLORREF);
static void        DrawBlankButton(HDC, int, int, int, int, BOOL, COLORREF
FAR *);
static void        PatB(HDC, int, int, int, int, COLORREF);
static void        MaskCreate(UINT, int, int, int, int, int, int, UINT);


//CURSORS.C
void               CursorsCache(HINSTANCE);
void               CursorsFree(void);



/*
 * Wierd Wild Wooly Waster (raster) Ops for special bltting.  See the
 * Windows SDK reference on Raster Operation Codes for explanation of
 * these.  The DSPDxax and PSDPxax is a reverse-polish notation for
 * operations where D==Destination, S==Source, P==Patterm, a==AND,
 * x==XOR.  Both of these codes are actually described in Programming
 * Windows by Charles Petzold, Second Edition, pages 622-624.
 */
#define ROP_DSPDxax  0x00E20746
#define ROP_PSDPxax  0x00B8074A


/*
 * Color indices into an array of standard hard-coded black, white, and
 * gray colors.
 */
#define STDCOLOR_BLACK      0
#define STDCOLOR_DKGRAY     1
#define STDCOLOR_LTGRAY     2
#define STDCOLOR_WHITE      3

/*
 * Color indices into an array of system colors, matching those in
 * the hard-coded array for the colors they replace.
 */

#define SYSCOLOR_TEXT       0
#define SYSCOLOR_SHADOW     1
#define SYSCOLOR_FACE       2
#define SYSCOLOR_HILIGHT    3


/*
 * Button types, used internally to distinguish command buttons from
 * attribute buttons to enforce three-state or six-state possibilities.
 * Command buttons can only have three states (up, mouse down, disabled)
 * while attribute buttons add (down, down disabled, and indeterminate).
 */

#define BUTTONTYPE_COMMAND      0
#define BUTTONTYPE_ATTRIBUTE    1


#ifdef __cplusplus
     }
#endif
```

CHANGES.TXT (BTTNCUR Sample)

Changes in BTTNCUR.DLL Source Code and Documentation from Version 1.00b to Version 1.1. See also below for 1.00 to 1.00b changes.

BTTNCUR.DLL version 1.1 is now fully compatible and tested under Windows 3.0. Previously the demo application crashed under Windows 3.0 on calls to the non-existent MoveToEx (Win32 compatible). Version 1.1 only uses MoveToEx for builds where the symbol WINVER < 0x030a.

The images have been updated slightly and are now provided for 72dpi (ega), 96dpi (vga), and 120dpi (super vga)

Version 1.1 completely handles system color changes unlike version 1.00x. This new version will dynamically convert black, light gray, dark gray, and white pixels in the image bitmap into system colors set for the button text, button shadow, button face, and button highlight, respectively. If you have blue, red, yellow, and green button colors, BTTNCUR.DLL will now work perfectly with all of them.

BTTNCUR.DLL Version 1.1 also supports color images better by allowing you to control which colors in the image bitmap are converted to system colors. By default, any black, gray, or white colors are converted into system button colors as decribed in the last paragraph. BTTNCUR.H defines new PRESERVE_* flags for each of the four colors that are liable to be converted. By specifing one or more flags you prevent BTTNCUR from changing that color to a system color. For example, if you want to preserve all black pixels in your image, specify PRESERVE_BLACK when calling UIToolButtonDraw.

Applications should obtain configuration data for the current display through UIToolConfigureForDisplay. With this data the application can configure itself for the correct toolbar size and button sizes and load the appropriate application supplied bitmaps.

Applications using UIToolConfigureForDisplay should now use UIToolButtonDrawTDD instead of UIToolButtonDraw, passing one extra parameter, a pointer to the TOOLDISPLAYDATA. Applications that still call UIToolButtonDraw will always use 96dpi.

NOTE: BTTNCUR.WRI has not been updated to reflect the new APIs.

```
------------------------
BTTNCUR.H changes
    Added PRESERVE_ flags to allow application to control color conversions
    from black, dark gray, light gray, and white into the
    file compatible with C++.

    Added prototype for UIToolConfigureForDisplay, TOOLDISPLAYDATA
structure,
    and definitions for button and image sizes on 72dpi, 96dpi, and 120dpi.

------------------------
```

BTTNCUR.RCV version changes

FILEVERSION and PRODUCTVERSION changed from 1,0,0,2 to 1,0,1,0

VALUE "FileVersion" and VALUE "ProductVersion" changed from "1.00b\0","\0" to "1.1\0","\0"

BTTNCURI.H changes

Defined STDCOLOR_* values as indices into an array in BTTNCUR.C that holds hard-coded default button color that never change regardless of the system colors. Also defined SYSCOLOR_* flags that matched STDCOLOR_* flags for uniform array indices.

Removed NEAR, FAR, and PASCAL from any function that didn't need it so we can port to Windows NT cleanly.

BTTNCUR.C source code changes. There are significant modifications.

Overall:

Updated header comment

Removed NEAR, FAR, and PASCAL from any function that didn't need it so we can port to Windows NT cleanly.

#define STRICT at top of file

Globals:

Eliminated the COLORREFs prefixed with RGB. Only a few are needed statically and were moved to HBrushDitherCreate.

Also added an array of standard colors used in the standard images:

```
static const
COLORREF crStandard[4]={ RGB(0, 0, 0)          //STDCOLOR_BLACK
                       , RGB(128, 128, 128)   //STDCOLOR_DKGRAY
                       , RGB(192, 192, 192)   //STDCOLOR_LTGRAY
                       , RGB(255, 255, 255)}; //STDCOLOR_WHITE
```

Added an array of standard images instead of just 96dpi versions.

UIToolConfigureForDisplay:

Added function to return the resolution of the display and size information about button and image sizes.

ToolButtonInit():

Call to CreatePatternBrush moved into HBrushDitherCreate. Conditionally sets the highlight color for HDitherBrushCreate depending on Windows 3.x or Windows 3.0 (3.0 did not support COLOR_BTNHIGHLIGHT).

ToolButtonFree():

Removed some old debug output no longer useful.

HDitherBitmapCreate()

Renamed to HBrushDitherCreate.
Moved CreatePatterBrush code from ToolButtonInit into this function.

To support changing system colors, this function maintains static variables for the face and highlight colors that we use to create the brush. If the function is called and the current colors in the global hBrushDither are different than the system colors, we recreate the brush and update the global hBrushDither, deleting the old brush. Otherwise we just return hBrushDither.

Note that if we fail to create the new brush we just maintain the old. We'll paint something, albeit not in the right colors, but something nontheless.

UIToolButtonDraw():

Calls UIToolButtonDrawTDD with default display configuration.

UIToolButtonDrawTDD():

This is the function that was overhauled the most, specifically to handle variable colors.

First, we added several local variables of which two are important. crSys is an array of system colors for the text, shadow, face, highlight, and frame, declared as static to keep references to it off DS instead of SS; if it's in SS things will crash. The second important variable is uColor, which receives the color preservation flags passed in the hibyte of the uState parameter to UIToolButtonDraw.

All the code to draw a blank button outline was moved into a separate function DrawBlankButton. Since this procedure needs all the system colors to do it's thing, I've set it up to take an array of five COLORREFs (exactly crSys) in which it stores those color (it also uses it as its color variables). This way we only have to call GetSysColor once for each system color.

Anything dealing with the dithered brush is moved to the BUTTONGROUP_LIGHTFACE case where we just get the current brush by calling HBrushDitherCreate, passing it the current face and highlight colors. Remember that is these colors match those used in the currently held global hBrushDither, this function just returns that global, so it's quite fast. We have to be very careful not to delete this brush where we're done with it since it is global.

The largest amount of new code is under the line:

```
if ((uState & BUTTONGROUP_ACTIVE) && !(uState & BUTTONGROUP_BLANK))
```

This has changed from a single BitBlt call to a much more complex operation to handle specific color conversions and preservations. A little optimization was done to detect when the system colors for buttons match the defaults, that is, black, dark gray, light gray, and white for text, shadow, face, and highlight. If these colors all match, or if the caller requested preservation of all colors, the we just do the single BitBlt of old.

Otherwise we loop through each of the black/white/gray colors that need possible conversion. For each one we create a mask that contains 1's where the converting color exists in the image and 0's everywhere else. For each color then we BitBlt the mask, a brush matching the system color we're converting to, and the destination bitmap (which we initialize with the unmodified image itself) using ROP_DSPDxax. This leaves any color but the one under conversion alone and replaces the converted color with the system color.

If the caller set a specific flag to preserve one or more specific colors, then we replace the standard color with the standard color, resulting in a no-op.

Finally, to reduce flicker for this four Blt operation we create and build the final image in a temporary bitmap, making it 6 total Blts to handle the color changes. But since we optimized for the 99% case where the system colors are the standard colors, this isn't too much of a consideration.

color conversions.

DrawBlankButton():

New internal function. Moved code from UIToolButtonDraw here.

CURSORS.C

Updated header comment
#define STRICT at top of file

Removed PASCAL on both internal functions.

CursorsFree

Eliminated all the code inside this function as it was unnecessary.

UICursorLoad

Eliminated code to revalidate a cursor in the array. Unnecessary.

BCDEMO.C

Tested for running under Windows 3.0 and avoided MoveToEx calls,

```
    using MoveTo instead.  Calling MoveToEx in a Windows 3.1 app
    marked as 3.0 compatible under 3.0 causes an unknown GP fault.

    Uses UIToolButtonConfigureForDisplay and UIToolButtonDrawTDD.


------------------------
BTTNCUR.WRI
    Updated to document PRESERVE_ flags and new capabilities of the library
    with the exception of display configuration (no time).
    Corrected two typos.



----------------------------------------------------------------------------
--

Changes in BTTNCUR.DLL Source Code and Documentation from Version 1.00
to Version 1.00b

------------------------
BTTNCUR.H changes
    Added #ifdef __cplusplus to include extern "C" making the
    file compatible with C++.


------------------------
BTTNCURI.H changes
    Added #ifdef __cplusplus to include extern "C" making the
    file compatible with C++.

    Removed code contained between a #ifdef NEVER that is unused.


------------------------
BTTNCUR.RCV version changes
    FILEVERSION and PRODUCTVERSION changed from 1,0,0,0 to 1,0,0,2

    VALUE "FileVersion" and VALUE "ProductVersion" changed from
        "1.00\0","\0" to "1.00b\0","\0"


------------------------
BTTNCUR.C source code changes.

    Added a global for the frame color
    static COLORREF rgbFrame   =RGB(0, 0, 0);

WEP():
    Added comment about resource cleanup messages in DEBUG mode.


ToolButtonInit():
    Added the line below just before the assignment of hDCGlyphs:
        rgbFrame=GetSysColor(COLOR_WINDOWFRAME);
```

This insures that the frame color is properly shown on plasma displays.

ToolButtonFree():

Added the following lines just before return:

```
        if (NULL!=hBmpStandardImages)
            DeleteObject(hBmpStandardImages);
        hBmpStandardImages=NULL;
```

UIToolButtonDraw():

The image centering is one too high. The line

```
        yOffsetGlyph=((dy-bmy) >> 1)-1;
```

now reads:

```
        yOffsetGlyph=(dy-bmy) >> 1;
```

The declaration HBITMAP hBmp; now read HBITMAP hBmp=NULL;

The line hBmpT=SelectObject(hDCGlyphs, hBmp); is now two:

```
        if (NULL!=hBmp)
            SelectObject(hDCGlyphs, hBmp);
```

The line SelectObject(hDCGlyphs, hBmpT); is now two:

```
        if (NULL!=hBmpT)
            SelectObject(hDCGlyphs, hBmpT);
```

BTTNCUR.BMP

Fixed the images to be 16*15 (the standard) instead of 16*16 as they originally were. Changed the label "16x16" to "16x15".

Copied an image of the disabled state of Context-Sensitive Help to this bitmap as it was previously missing.

BTTNCUR.WRI

Corrected a few typos. Added comment about application termination and erroneous debug warnings.

CURSORS.C   (BTTNCUR Sample)

```
/*
 * CURSORS.C
 * Buttons & Cursors Version 1.1, March 1993
 *
 * Public functions to retrieve new cursors from the BTTNCUR DLL based
 * on ordinal to prevent applications from necessarily calling LoadCursor
 * directly on the DLL.
 *
 * Copyright (c)1992-1993 Microsoft Corporation, All Rights Reserved,
 * as applied to redistribution of this source code in source form
 * License is granted to use of compiled code in shipped binaries.
 */

#define STRICT
#include <windows.h>
#include "bttncur.h"
#include "bttncuri.h"


/*
 * The +1 is because MAX is the highest allowable number and MIN is not
 * necessarily zero.
 */
HCURSOR rgHCursors[IDC_NEWUICURSORMAX-IDC_NEWUICURSORMIN+1];



/*
 * CursorsCache
 * Internal
 *
 * Purpose:
 *  Loads all the cursors available through NewUICursorLoad into
 *  a global array.  This way we can clean up all the cursors without
 *  placing the burden on the application.
 *
 * Parameters:
 *  hInst           HANDLE of the DLL instance.
 *
 * Return Value:
 *  None.  If any of the LoadCursor calls fail, then the corresponding
 *  array entry is NULL and NewUICursorLoad will fail.  Better to fail
 *  an app getting a cursor than failing to load the app just for that
 *  reason; and app can attempt to load the cursor on startup if it's
 *  that important, and fail itself.
 */

void CursorsCache(HINSTANCE hInst)
     {
     UINT            i;

     for (i=IDC_NEWUICURSORMIN; i<=IDC_NEWUICURSORMAX; i++)
```

```
           rgHCursors[i-IDC_NEWUICURSORMIN]=LoadCursor(hInst,
MAKEINTRESOURCE(i));

     return;
     }




/*
 * CursorsFree
 * Internal
 *
 * Purpose:
 *  Frees all the cursors previously loaded through CursorsCache.
 *
 * Parameters:
 *  None
 *
 * Return Value:
 *  None
 */

void CursorsFree(void)
     {
     /*
      * Note that since cursors are discardable resources and should
      * not be used with DestroyCursor, there's nothing to do here.
      * We still provide this API for compatibility and to maintain
      * symmetry.
      */
     return;
     }





/*
 * UICursorLoad
 * Public API
 *
 * Purpose:
 *  Loads and returns a handle to one of the new standard UI cursors
 *  contained in UITOOLS.DLL.  The application must not call DestroyCursor
 *  on this cursor as it is managed by the DLL.
 *
 * Parameters:
 *  iCursor         UINT index to the cursor to load which must be one
 *                  of the following values:
 *
 *                      IDC_RIGHTARROW    Right pointing standard arrow
 *                      IDC_CONTEXTHELP   Arrow with a ? (context help)
 *                      IDC_MAGNIFY       Magnifying glass for zooming
 *                      IDC_NODROP        Circle with a slash
 *                      IDC_TABLETOP      Small arrow pointing down
 *
 *                      IDC_SMALLARROWS   Thin four-headed arrow
 *                      IDC_LARGEARROWS   Wide four-headed arrow
 *                      IDC_HARROWS       Horizontal two-headed arrow
 *                      IDC_VARROWS       Vertical two-headed arrow
 *                      IDC_NESWARROWS    Two-headed arrow pointing NE<->SW
 *                      IDC_NWSEHARROWS   Two-headed arrow pointing NW<->SE
 *
 *                      IDC_HSIZEBAR      Horizontal two-headed arrow with
 *                                        a single vertical bar down the
 *                                        middle
 *
 *                      IDC_VSIZEBAR      Vertical two-headed arrow with a
 *                                        single horizontal bar down the
 *                                        middle
 *
 *                      IDC_HSPLITBAR     Horizontal two-headed arrow with
 *                                        split double vertical bars down
the
 *                                        middle
 *
 *                      IDC_VSPLITBAR     Vertical two-headed arrow with
split
 *                                        double horizontal bars down the
 *                                        middle
 *
 * Return Value:
 *  HCURSOR         Handle to the loaded cursor if successful, NULL
 *                  if iCursor is out of range or the function could not
 *                  load the cursor.
 */

HCURSOR WINAPI UICursorLoad(UINT iCursor)
     {
     HCURSOR     hCur=NULL;

     if ((iCursor >= IDC_NEWUICURSORMIN) && (iCursor <= IDC_NEWUICURSORMAX))
           hCur=rgHCursors[iCursor-IDC_NEWUICURSORMIN];

     return hCur;
     }
```

COMMON

This sample builds the SDKUTIL.LIB, which keeps track of how many times an object is re-entered and keeps the object alive until the first call finally is returned. For information on compiling and building the sample, see MAKEFILE   (COMMON Sample).

MAKEFILE (COMMON Sample)

```
#+--------------------------------------------------------------------------
-
#
#  Microsoft Windows
#  Copyright (C) Microsoft Corporation, 1994.
#
#  File:        makefile
#
#---------------------------------------------------------------------------
-

!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

#
#       builds the sdkutil.LIB library
#

OLEFLAGS =

all: sdkutil.lib

clean:
    -del *.lib
    -del *.obj

stablize.obj: stablize.cxx
    $(cc) $(cvars) $(cflags) $(UFLAGS) $(cdebug) $(OLEFLAGS) stablize.cxx

altassrt.obj: altassrt.cxx
    $(cc) $(cvars) $(cflags) $(UFLAGS) $(cdebug) $(OLEFLAGS) altassrt.cxx

sdkutil.lib: altassrt.obj stablize.obj
    lib -out:sdkutil.lib altassrt.obj stablize.obj
    if not exist ..\lib mkdir ..\lib
    copy sdkutil.lib ..\lib
```

ALTASSRT.CXX   (COMMON Sample)

```
//
+---------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       altassrt.cxx
//
//  Contents:   Assertion routines for the SDK
//
//  Functions:  NTSDKAssert
//              PopUpError
//
//  History:    8-19-94   stevebl   Created
//
//----------------------------------------------------------------------------
--

#include <stdarg.h>
#include <stdio.h>

// ensure that this module always compiles with DEBUG settings.
#undef  NDEBUG

#include "assert.h"

int
PopUpError(
    char const *szMsg,
    int iLine,
    char const *szFile);

extern "C"
{

# ifdef WIN32
#  undef FAR
#  undef NEAR
# else
#  define MessageBoxA MessageBox
# endif

# include <windows.h>
}

//
+---------------------------------------------------------------------------
//
//  Function:   PopUpAssert
//
//  Synopsis:   Display assertion information
//
```

```
//  Effects:    Called when an assertion is hit.
//
//  History:    8-19-94   stevebl   Created
//
//----------------------------------------------------------------------------
--

void _CRTAPI1
PopUpAssert(
    void * szFile,
    int iLine,
    void * szMessage)
{
    int id = PopUpError((char const *)szMessage,iLine, (char const *)
szFile);

    if (id == IDCANCEL)
    {
#ifdef WIN32
        DebugBreak();
#else
        _asm int 3;
#endif
    }
}

//+---------------------------------------------------------------
// Function:    PopUpError
//
// Synopsis:    Displays a dialog box using provided text,
//              and presents the user with the option to
//              continue or cancel.
//
// Arguments:
//      szMsg --        The string to display in main body of dialog
//      iLine --        Line number of file in error
//      szFile --       Filename of file in error
//
// Returns:
//      IDCANCEL --     User selected the CANCEL button
//      IDOK     --     User selected the OK button
//----------------------------------------------------------------

int
PopUpError(
    char const *szMsg,
    int iLine,
    char const *szFile)
{

    int id;
    static char szAssertCaption[128];
    static char szModuleName[128];

    DWORD tid = GetCurrentThreadId();
    DWORD pid = GetCurrentProcessId();
    char * pszModuleName;

    if (GetModuleFileNameA(NULL, szModuleName, 128))
    {
        pszModuleName = strrchr(szModuleName, '\\');
        if (!pszModuleName)
        {
            pszModuleName = szModuleName;
        }
        else
        {
            pszModuleName++;
        }
    }
    else
    {
        pszModuleName = "Unknown";
    }

    sprintf(szAssertCaption,"Process: %s File: %s line %u, thread id %d.%d",
        pszModuleName, szFile, iLine, pid, tid);


    DWORD dwMessageFlags = MB_SETFOREGROUND | MB_TASKMODAL |
                           MB_ICONEXCLAMATION | MB_OKCANCEL;

    id = MessageBoxA(NULL,
                     (char *) szMsg,
                     (LPSTR) szAssertCaption,
                     dwMessageFlags);

    // If id == 0, then an error occurred.  There are two possibilities
    // that can cause the error:  Access Denied, which means that this
    // process does not have access to the default desktop, and everything
    // else (usually out of memory).  Oh well.

    return id;
}
```

STABLIZE.CXX   (COMMON Sample)

```
//
+-------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       stablize.cxx
//
//  Contents:   Stabilization Classes used to stabilize objects during
//              re-entrant calls.
//
//  Classes:    CSafeRefCount
//              CStabilize
//
//  History:    8-26-94   stevebl   Modified from code written by AlexGo
//
//------------------------------------------------------------------------
--

#include <windows.h>
#include <ole2.h>
#include <stablize.h>
#include <assert.h>

//+------------------------------------------------------------------------
//
//  Member:     CSafeRefCount::CSafeRefCount
//
//  Synopsis:   constructor for the safe ref count class
//
//  History:    dd-mmm-yy Author    Comment
//              28-Jul-94 alexgo    author
//              8-26-94   stevebl   modified for use by SDK samples
//
//-------------------------------------------------------------------------

CSafeRefCount::CSafeRefCount()
{
        m_cRefs = 0;
        m_cNest = 0;
        m_fInDelete = FALSE;
}

//+------------------------------------------------------------------------
//
//  Member:     CSafeRefCount::~CSafeRefCount (virtual)
//
//  History:    dd-mmm-yy Author    Comment
//              28-Jul-94 alexgo    author
//              8-26-94   stevebl   modified for use by SDK samples
//
//-------------------------------------------------------------------------
```

```
CSafeRefCount::~CSafeRefCount()
{
        assert(m_cRefs == 0 && m_cNest == 0 && m_fInDelete == TRUE);
}

//+-------------------------------------------------------------------------
//
//  Member:     CSafeRefCount::SafeAddRef
//
//  Synopsis:   increments the reference count on the object
//
//  Returns:    ULONG -- the reference count after the increment
//
//  History:    dd-mmm-yy Author    Comment
//              28-Jul-94 alexgo    author
//              8-26-94   stevebl   modified for use by SDK samples
//
//--------------------------------------------------------------------------

ULONG CSafeRefCount::SafeAddRef()
{
        m_cRefs++;
        return m_cRefs;
}

//+-------------------------------------------------------------------------
//
//  Member:     CSafeRefCount::SafeRelease
//
//  Synopsis:   decrements the reference count on the object
//
//  Effects:    May delete the object!
//
//  Returns:    ULONG -- the reference count after decrement
//
//  Algorithm:  decrements the reference count.  If the reference count
//              is zero AND the nest count is zero AND we are not currently
//              trying to delete our object, then it is safe to delete.
//
//  History:    dd-mmm-yy Author    Comment
//              28-Jul-94 alexgo    author
//              8-26-94   stevebl   modified for use by SDK samples
//
//--------------------------------------------------------------------------

ULONG CSafeRefCount::SafeRelease()
{
        ULONG   cRefs;

        if( m_cRefs > 0 )
        {
                cRefs = --m_cRefs;

                if( m_cRefs == 0 && m_cNest == 0 && m_fInDelete == FALSE )
                {
                        m_fInDelete = TRUE;
                        delete this;
                }
        }
        else
        {
                // somebody is releasing a non-addrefed pointer!!
                assert(NULL == "Release called on a non-addref'ed pointer!
\n");
                cRefs = 0;
        }
        return cRefs;
}

//+---------------------------------------------------------------------------
//
//  Member:     CSafeRefCount::IncrementNestCount
//
//  Synopsis:   increments the nesting count of the object
//
//  Arguments:  none
//
//  Returns:    ULONG; the nesting count after increment
//
//  History:    dd-mmm-yy Author    Comment
//              28-Jul-94 alexgo    author
//              8-26-94   stevebl   modified for use by SDK samples
//
//  Notes:      The nesting count is the count of how many times an
//              an object has been re-entered.  For example, suppose
//              somebody calls pFoo->Bar1(), which makes some calls that
//              eventually call pFoo->Bar2();.  On entrace to Bar2, the
//              nest count of the object should be 2 (since the invocation
//              of Bar1 is still on the stack above us).
//
//              It is important to keep track of the nest count so we do
//              not accidentally delete ourselves during a nested
invocation.
//              If we did, then when the stack unwinds to the original
//              top level call, it could try to access a non-existent member
//              variable and crash.
//
//----------------------------------------------------------------------------

ULONG CSafeRefCount::IncrementNestCount()
{
        m_cNest++;
        return m_cNest;
}

//+---------------------------------------------------------------------------
//
//  Member:     CSafeRefCount::DecrementNestCount
//
```

```
//  Synopsis:   decrements the nesting count and deletes the object
//              (if necessary)
//
//  Effects:    may delete 'this' object!
//
//  Arguments:  none
//
//  Returns:    ULONG, the nesting count after decrement
//
//  Algorithm:  decrements the nesting count.  If the nesting count is zero
//              AND the reference count is zero AND we are not currently
//              trying to delete ourselves, then delete 'this' object
//
//  History:    dd-mmm-yy Author    Comment
//              28-Jul-94 alexgo    author
//              8-26-94   stevebl   modified for use by SDK samples
//
//  Notes:
//
//--------------------------------------------------------------------------

ULONG CSafeRefCount::DecrementNestCount()
{
        ULONG   cNest;

        if( m_cNest > 0 )
        {
                cNest = --m_cNest;

                if( m_cRefs == 0 && m_cNest == 0 && m_fInDelete == FALSE )
                {
                        m_fInDelete = TRUE;
                        delete this;
                }
        }
        else
        {
                // somebody forget to increment the nest count!!
                assert(NULL == "Unbalanced nest count!!");
                cNest = 0;
        }
        return cNest;
}
```

DEFO2V

This sample creates the DLL used by the OLE 2 Interface Viewer. For information on compiling and building the sample, see MAKEFILE   (DEFO2V Sample).

MAKEFILE (DEFO2V Sample)

```
#+---------------------------------------------------------------------------
-
#
#  Microsoft Windows
#  Copyright (C) Microsoft Corporation, 1994.
#
#  File:        makefile
#
#----------------------------------------------------------------------------
-

!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

#
#       Makefile for OLE Sample DEFO2V32.DLL
#
#       builds DEFO2V32.DLL: viewer DLL used by the OLE2VIEW tool.
#

OLEFLAGS = -I ..\..\idl -I ..\..\winhlprs
LINK = $(link)
LINKFLAGS = $(linkdebug)
RCFLAGS = -DWIN32

OBJS = iadvsi_a.obj     \
       iadvsink.obj     \
       idatao_a.obj     \
       idataobj.obj     \
       idisp_a.obj      \
       idisp.obj        \
       tofile_a.obj     \
       tofile.obj       \
       util_a.obj       \
       util.obj

LIBS = $(olelibsdll)
cflags = $(cflags) -DUNICODE -D_UNICODE
all: defo2v32.dll

clean:
    -del *.obj
    -del *.exp
    -del *.res
    -del *.dll
    -del *.lib

iadvsi_a.obj: iadvsi_a.cpp      \
              iadvsink.cpp      \
              precompa.h        \
              precomp.h         \
              defo2v.h          \
              resource.h        \
              util.h            \
              idataobj.h        \
              iadvsink.h
    $(cc) $(cvarsdll) $(cflags) $(cdebug) $(OLEFLAGS) iadvsi_a.cpp

iadvsink.obj: iadvsink.cpp      \
              precomp.h         \
              defo2v.h          \
              resource.h        \
              util.h            \
              idataobj.h        \
              iadvsink.h
    $(cc) $(cvarsdll) $(cflags) $(cdebug) $(OLEFLAGS) iadvsink.cpp

idatao_a.obj: idatao_a.cpp      \
              idataobj.cpp      \
              precompa.h        \
              precomp.h         \
              defo2v.h          \
              resource.h        \
              util.h            \
              idataobj.h        \
              iadvsink.h
    $(cc) $(cvarsdll) $(cflags) $(cdebug) $(OLEFLAGS) idatao_a.cpp

idataobj.obj: idataobj.cpp       \
              precomp.h          \
              defo2v.h           \
              resource.h         \
              util.h             \
              idataobj.h         \
              iadvsink.h
    $(cc) $(cvarsdll) $(cflags) $(cdebug) $(OLEFLAGS) idataobj.cpp

idisp_a.obj: idisp_a.cpp        \
             idisp.cpp          \
             precompa.h         \
             precomp.h          \
             defo2v.h           \
             resource.h         \
             util.h             \
             idisp.h            \
             tofile.h
    $(cc) $(cvarsdll) $(cflags) $(cdebug) $(OLEFLAGS) idisp_a.cpp

idisp.obj:   idisp.cpp          \
             precomp.h          \
             defo2v.h           \
             resource.h         \
             util.h             \
             idisp.h            \
             tofile.h
    $(cc) $(cvarsdll) $(cflags) $(cdebug) $(OLEFLAGS) idisp.cpp

tofile_a.obj: tofile_a.cpp      \
              tofile.cpp        \
              precompa.h        \
              precomp.h         \
              defo2v.h          \
              resource.h        \
              util.h            \
              idisp.h           \
              tofile.h
    $(cc) $(cvarsdll) $(cflags) $(cdebug) $(OLEFLAGS) tofile_a.cpp

tofile.obj:   tofile.cpp        \
              precomp.h         \
              defo2v.h          \
              resource.h        \
              util.h            \
              idisp.h           \
              tofile.h
    $(cc) $(cvarsdll) $(cflags) $(cdebug) $(OLEFLAGS) tofile.cpp

util_a.obj: util_a.cpp  \
            util.cpp    \
            precompa.h  \
            precomp.h   \
            defo2v.h    \
            resource.h  \
            util.h
    $(cc) $(cvarsdll) $(cflags) $(cdebug) $(OLEFLAGS) util_a.cpp

util.obj:   util.cpp    \
            precomp.h   \
            defo2v.h    \
            resource.h  \
            util.h
    $(cc) $(cvarsdll) $(cflags) $(cdebug) $(OLEFLAGS) util.cpp

defo2v.res: defo2v.rc defo2v.h
    rc $(RCFLAGS) -r -fo$@ defo2v.rc

defo2v32.dll: $(OBJS) defo2v.res
    $(LINK) @<<
        $(LINKFLAGS)
        -dll
        -out:$@
        -def:defo2v32.def
        -noentry
        -MERGE:.CRT=.data
        -MERGE:_PAGE=PAGE
        -MERGE:_TEXT=.text
        -SECTION:INIT,d
        -OPT:REF
        -INCREMENTAL:NO
        -IGNORE:4037,4065
        -PDB:NONE
        -NODEFAULTLIB
        -subsystem:native
```

```
          $(OBJS)
          defo2v.res
          $(LIBS)
<<
```

DEFO2V.DEF (DEFO2V Sample)

```
LIBRARY   DEFO2V
EXETYPE   WINDOWS
CODE      PRELOAD MOVEABLE DISCARDABLE
DATA      PRELOAD MOVEABLE SINGLE
HEAPSIZE  4096

SEGMENTS
            WEP_SEG    PRELOAD FIXED

EXPORTS
            WEP PRIVATE
            fnEnumReallyEx          @5
            fnIDispDlg              @10
            fnIDataObjectDlg        @11
            DisplayIDataObject      @100
            DisplayIDispatch        @101
            DisplayITypeLib         @102
            DisplayITypeInfo        @103
```

DEFO2V.H (DEFO2V Sample)

```
// defo2v.h
//
// Header file for OLE 2.0 Object Viewer auxillary DLL (DEFO2V.O2V).
//
// Copyright (c) 1993 Microsoft Corporation, All Rights Reserved.
//
// Charlie Kindel, Program Manager
// Microsoft Vertical Developer Relations
// August 6, 1993
//
// InterNet   :  ckindel@microsoft.com
// CompuServe :  >INTERNET:ckindel@microsoft.com
//
// Revisions:
//  August 6, 1993  cek     First implementation.
//  August 7, 1993  cek     Rewrote without using MFC 2.0.
//
// An O2V module is a DLL that exports the following function
// at:
//
//      HRESULT WINAPI DisplayInterface
//      (
//          HWND        hwndParent, // Ole2View's main frame window handle
//          LPUNKNOWN   lpunk,      // Pointer to interface
//          LPIID       lpiid,      // Pointer to IDD for this interface
//          LPSTR       lpszName    // Name of this interface (i.e.
IUnknown)
//      ) ;
//
// Ole2View finds O2V modules by lookin in the [Interface Viewers] section
in the
// OLE2VIEW.INI file:
//
//  [Interface Viewers]
//  <iid>=filename.o2v,<fn name>
//
// Examples:
//
//  [Interface Viewers]
//  {0000011B-0000-0000-C000-000000000046}=contain.o2v,DisplayIOleContainer
//
// By default, the following entries are assumed (entires are not actually
// in the INI file):
//
//  [Interface Viewers]
//  {0000010E-0000-0000-C000-000000000046}=defo2v.o2v,DisplayIDataObject
//  {00020400-0000-0000-C000-000000000046}=defo2v.o2v,DisplayIDispatch
//
// These 'default' implementations can be overriden by placing entries in
// the ini file.
//
```

```
#ifndef _DEFO2V_H_
#define _DEFO2V_H_

#if WIN32!=300
//#include <oleguid.h>
//#include <coguid.h>
#endif

//#include <dispatch.h>

#include "resource.h"
#include "util.h"

#ifndef WIN32
extern HINSTANCE g_hInst ;
#else
#define g_hInst GetModuleHandle(_T("DEFO2V32.DLL"))
#endif

// Ownerdraw listbox stuff
#define I_NORMAL        0x0000
#define I_LABEL         0x0001
#define I_COLUMNHEAD    0x0002
#define I_COLUMNENTRY   0x0003

typedef struct FAR tagITEMDATA
{
    UINT            nLevel ;
    UINT            uiType ;
    int             cColumns ;
    LPCOLUMNSTRUCT  rgCol ;
    LPVOID          lpData ;

} ITEMDATA, *PITEMDATA, FAR*LPITEMDATA ;

#endif // _DEFO2V_H_
```

DEFO2V.RC (DEFO2V Sample)

```
//
// DEFO2V.RC2 - resources App Studio does not edit directly
//

#ifdef APSTUDIO_INVOKED
    #error this file is not editable by App Studio
#endif //APSTUDIO_INVOKED

/////////////////////////////////////////////////////////////////////////////
/
// Version stamp for this .EXE

#ifndef WIN32
#include "ver.h"
#else
#include "winver.h"
#endif

VS_VERSION_INFO     VERSIONINFO
  FILEVERSION       1,20,20,0
  PRODUCTVERSION    1,20,20,0
  FILEFLAGSMASK     VS_FFI_FILEFLAGSMASK
#ifdef _DEBUG
  FILEFLAGS         VS_FF_DEBUG|VS_FF_PRIVATEBUILD|VS_FF_PRERELEASE
#else
  FILEFLAGS         0 // final version
#endif
  FILEOS            VOS_DOS_WINDOWS16
  FILETYPE          VFT_DLL
  FILESUBTYPE       0   // not used
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904E4" // Lang=US English, CharSet=Windows Multilingual
        BEGIN
            VALUE "CompanyName",     "Microsoft Corporation\0"
            VALUE "FileDescription", "Default Ole2View iface Viewers\0"
            VALUE "FileVersion",     "1.20.000\0"
            VALUE "InternalName",    "DEFO2V.DLL\0"
            VALUE "LegalCopyright",  "Copyright \251 1993 Microsoft Corp.
All Rights Reserved.\0"
            VALUE "LegalTrademarks", "By Charlie Kindel\0"
            VALUE "OriginalFilename","DEFO2V.DLL\0"
            VALUE "ProductName",     "OLE 2.0 Object Viewer\0"
            VALUE "ProductVersion",  "1.20\0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x409, 1252
            // English language (0x409) and the Windows ANSI codepage (1252)
    END
```

```
END

////////////////////////////////////////////////////////////////////////////
/
// Add additional manually edited resources here...
////////////////////////////////////////////////////////////////////////////
/
```

DEFO2V.CPP   (DEFO2V Sample)

```
// defo2v.cpp
//
// Implementation file for DEFO2V.O2V (LibMain, WEP).
//
// Copyright (c) 1993 Microsoft Corporation, All Rights Reserved.
//
// Charlie Kindel, Program Manager
// Microsoft Vertical Developer Relations
// August 7, 1993
//
// InterNet   :  ckindel@microsoft.com
// CompuServe :  >INTERNET:ckindel@microsoft.com
//
// Revisions:
//  August 6, 1993  cek     First implementation.
//  August 7, 1993  cek     Rewrote without using MFC 2.0.
//

#include "precomp.h"

#include <initguid.h>

#include "defo2v.h"

HINSTANCE   g_hInst = NULL ;

extern "C"
HANDLE FAR PASCAL LibMain(HINSTANCE hInst, WORD wDataSeg, WORD cbHeapSize,
LPSTR lpCmdLine)
{

    if (0!=cbHeapSize)
        UnlockData(0);

    g_hInst=hInst;
    return hInst;
}

#pragma code_seg("WEP_SEG")
extern "C"
void FAR PASCAL WEP(int bSystemExit)
{
    return;
}
```

DEFO2V32.DEF (DEFO2V Sample)

```
LIBRARY   DEFO2V32

EXPORTS

; Unicode versions
    fnEnumReallyEx       @100
    fnIDispDlg           @101
    fnIDataObjectDlg     @102
    DisplayIDataObject   @103
    DisplayIDispatch     @104
    DisplayITypeLib      @105
    DisplayITypeInfo     @106

; ANSI versions
    fnEnumReallyExA      @200
    fnIDispDlgA          @201
    fnIDataObjectDlgA    @202
    DisplayIDataObjectA  @203
    DisplayIDispatchA    @204
    DisplayITypeLibA     @205
    DisplayITypeInfoA    @206
```

IADVSINK.H (DEFO2V Sample)

```
#ifndef _IADVSINK_H_
#define _IADVSINK_H_

#if defined(WIN32) && !defined(_UNICODE)
    #define CImpIAdviseSink CImpIAdviseSinkA
    #define CIDataObjectDlg CIDataObjectDlgA
#endif

class FAR CImpIAdviseSink : public IAdviseSink
{
protected:
    ULONG               m_cRef;      //Interface reference count.
    CIDataObjectDlg FAR* m_pIDOD ;       //Back pointer to the application

public:
    CImpIAdviseSink(CIDataObjectDlg FAR*);
    ~CImpIAdviseSink(void);

    STDMETHODIMP QueryInterface(REFIID, LPVOID FAR *);
    STDMETHODIMP_(ULONG) AddRef(void);
    STDMETHODIMP_(ULONG) Release(void);

    //We only implement OnDataChange for now.
    STDMETHODIMP_(void)  OnDataChange(LPFORMATETC, LPSTGMEDIUM);
    STDMETHODIMP_(void)  OnViewChange(DWORD, LONG);
    STDMETHODIMP_(void)  OnRename(LPMONIKER);
    STDMETHODIMP_(void)  OnSave(void);
    STDMETHODIMP_(void)  OnClose(void);
};

#endif //_IADVSINK_H_
```

IADVSINK.CPP   (DEFO2V Sample)

```
/*
 * IADVSINK.CPP
 *
 * Implementation of the IAdviseSink interface for the Data User, Chapter 6
 *
 * Copyright (c)1993 Microsoft Corporation, All Rights Reserved
 *
 * Kraig Brockschmidt, Software Design Engineer
 * Microsoft Systems Developer Relations
 *
 * Internet  :  kraigb@microsoft.com
 * Compuserve:  >INTERNET:kraigb@microsoft.com
 */


#include "precomp.h"
#include "defo2v.h"
#include "idataobj.h"
#include "iadvsink.h"

#if defined(WIN32) && !defined(_UNICODE)
    #pragma message("Building ANSI version of " __FILE__)
#endif

/*
 * CImpIAdviseSink::CImpIAdviseSink
 * CImpIAdviseSink::~CImpIAdviseSink
 *
 * Parameters (Constructor):
 *  pAV             LPAPPVARS to the application
 *
 */

CImpIAdviseSink::CImpIAdviseSink( CIDataObjectDlg FAR* lpIDOD )
    {
    m_cRef=0;
    m_pIDOD=lpIDOD;
    return;
    }

CImpIAdviseSink::~CImpIAdviseSink(void)
    {
    m_pIDOD->m_pSink = NULL; // make sure the container knows I've been
deleted. SteveBl
    return;
    }




/*
 * CImpIAdviseSink::QueryInterface
 * CImpIAdviseSink::AddRef
 * CImpIAdviseSink::Release
 *
 * Purpose:
 *  IUnknown members for CImpIAdviseSink object.
 */

STDMETHODIMP CImpIAdviseSink::QueryInterface(REFIID riid, LPVOID FAR *ppv)
    {
    *ppv=NULL;

    //Any interface on this object is the object pointer.
    if (IsEqualIID(riid, IID_IUnknown) || IsEqualIID(riid, IID_IAdviseSink))
        *ppv=(LPVOID)this;

    /*
     * If we actually assign an interface to ppv we need to AddRef it
     * since we're returning a new pointer.
     */
    if (NULL!=*ppv)
        {
        ((LPUNKNOWN)*ppv)->AddRef();
        return NOERROR;
        }

    return ResultFromScode(E_NOINTERFACE);
    }


STDMETHODIMP_(ULONG) CImpIAdviseSink::AddRef(void)
    {
    return ++m_cRef;
    }


STDMETHODIMP_(ULONG) CImpIAdviseSink::Release(void)
    {
    ULONG   cRefT;

    cRefT=--m_cRef;

    if (0==cRefT)
        delete this;

    return cRefT;
    }




/*
 * IAdviseSink::OnDataChange
 *
 * Purpose:
 *  Notifes the advise sink that data changed in a data object.  On
 *  this message you may request a new data rendering and update your
 *  displays as necessary.  Any data sent to this function is owned
 *  by the caller, not by this advise sink!
 *
 *  All Advise Sink methods should be considered asynchronous and therefore
 *  we should attempt no synchronous calls from within them to an EXE
 *  object.  If we do, we'll get RPC_E_CALLREJECTED as shown below.
 *
 * Parameters:
 *  pFE             LPFORMATETC describing format that changed
 *  pSTM            LPSTGMEDIUM providing the medium in which the data
 *                  is provided.
 *
 * Return Value:
 *  None
 */

STDMETHODIMP_(void) CImpIAdviseSink::OnDataChange(LPFORMATETC pFE,
LPSTGMEDIUM pSTM)
{
    BOOL        fUsable=TRUE;
    UINT        cf;

    //See if we're interested
    cf=pFE->cfFormat;

    if (!(DVASPECT_CONTENT & pFE->dwAspect))
        return;

/*
    //Check media types
    switch (cf)
    {
    #ifdef UNICODE
        case CF_UNICODETEXT:
    #else
        case CF_TEXT:
    #endif
            fUsable=(BOOL)(TYMED_HGLOBAL & pFE->tymed);
        break;
        case CF_BITMAP:
            fUsable=(BOOL)(TYMED_GDI & pFE->tymed);
        break;

        case CF_METAFILEPICT:
            fUsable=(BOOL)(TYMED_MFPICT & pFE->tymed);
        break;

        default:
        break;
    }

    if (!fUsable)
        return;
*/
```

```
    if (NULL==m_pIDOD->m_lpDO)
        return;

    // If we are doing a 'warm link' then the data does not come with the
    // OnDataChange.   Post a message back to ourselves so that we can
    // call GetData when it's convienient
    //
    if (m_pIDOD->m_advf & ADVF_NODATA)
    {
        if (m_pIDOD->m_fDoOnGetDataPosted == FALSE)
        {
            m_pIDOD->m_fDoOnGetDataPosted = TRUE ;
            PostMessage( m_pIDOD->m_hDlg, WM_COMMAND,  IDC_DOGETDATA, 0L ) ;
        }
        return ;
    }

    // for Hot links the data comes with the OnDataChange
    m_pIDOD->GotData( pFE, pSTM ) ;
    return ;
}

/*
 * IAdviseSink::OnViewChange
 *
 * Purpose:
 *  Notifes the advise sink that presentation data changed in the data
 *  object to which we're connected providing the right time to update
 *  displays using such presentations.
 *
 * Parameters:
 *  dwAspect        DWORD indicating which aspect has changed.
 *  lindex          LONG indicating the piece that changed.
 *
 * Return Value:
 *  None
 */

STDMETHODIMP_(void) CImpIAdviseSink::OnViewChange(DWORD dwAspect, LONG
lindex)
    {
    return;
    }





/*
 * IAdviseSink::OnRename
 *
 * Purpose:
 *  Informs the advise sink that an IOleObject has been renamed, primarily
 *  when its linked.
 *
 * Parameters:
 *  pmk             LPMONIKER providing the new name of the object
 *
 * Return Value:
 *  None
 */

STDMETHODIMP_(void) CImpIAdviseSink::OnRename(LPMONIKER pmk)
    {
    return;
    }






/*
 * IAdviseSink::OnSave
 *
 * Purpose:
 *  Informs the advise sink that the OLE object has been saved
 *  persistently.  The primary purpose of this is for containers that
 *  want to make optimizations for objects that are not in a saved
 *  state, so on this you have to disable such optimizations.
 *
 * Parameters:
 *  None
 *
 * Return Value:
 *  None
 */

STDMETHODIMP_(void) CImpIAdviseSink::OnSave(void)
    {
    return;
    }






/*
 * IAdviseSink::OnClose
 *
 * Purpose:
 *  Informs the advise sink that the OLE object has closed and is
 *  no longer bound in any way.  On this you typically change state
 *  variables and redraw shading, etc.
 *
 * Parameters:
 *  None
 *
 * Return Value:
```

```
 *  None
 */

STDMETHODIMP_(void) CImpIAdviseSink::OnClose(void)
    {
    return;
    }
```

IADVSI_A.CPP (DEFO2V Sample)

```
// iadvsi_a.cpp
//
// Creates ANSI versions of all modules.  Any exported function get's "A"
// appended.
//
//

#ifdef UNICODE
#undef UNICODE
#endif
#ifdef _UNICODE
#undef _UNICODE
#endif

#include "precompa.h"
#include "iadvsink.cpp"
```

IDATAOBJ.H   (DEFO2V Sample)

```
// idataobj.h
//
// Header file for the IDataObject interface viewer.
//
// Copyright (c) 1993 Microsoft Corporation, All Rights Reserved.
//
// Charlie Kindel, Program Manager
// Microsoft Vertical Developer Relations
// August 6, 1993
//
// InterNet   :  ckindel@microsoft.com
// CompuServe :  >INTERNET:ckindel@microsoft.com
//
// Revisions:
//  August 6, 1993  cek     First implementation.
//

#ifndef _IDATAOBJ_H_
#define _IDATAOBJ_H_

#if defined(WIN32) && !defined(_UNICODE)
    #define CImpIAdviseSink CImpIAdviseSinkA
    #define fnIDataObjectDlg fnIDataObjectDlgA
    #define fnEditSubclass fnEditSubclassA
    #define CIDataObjectDlg CIDataObjectDlgA
    #define DisplayIDataObject DisplayIDataObjectA
#endif

STDAPI DisplayIDataObject( HWND hwndParent, LPDATAOBJECT lpDO, LPIID lpiid,
LPTSTR lpszName ) ;

class CImpIAdviseSink ;

#define CCHOUTPUTMAX        8192
#define CLINESMAX           500
#define WM_OUTPUTBUFFERHASDATA      (WM_USER+1000)

////////////////////////////////////////////////////////////////////////////
/
// CIDataObjectDlg dialog
//
extern "C"
BOOL CALLBACK fnIDataObjectDlg( HWND hDlg, UINT uiMsg, WPARAM wParam, LPARAM
lParam ) ;

class FAR CIDataObjectDlg
{
    friend BOOL CALLBACK fnIDataObjectDlg( HWND hDlg, UINT uiMsg, WPARAM
wParam, LPARAM lParam ) ;
    friend class CImpIAdviseSink ;

public:
    CIDataObjectDlg( HWND hwnd, LPDATAOBJECT lpDO, LPIID lpiid, LPTSTR
lpszName ) ;
    ~CIDataObjectDlg() ;

    int DoModal( void ) ;

    LPDATAOBJECT    m_lpDO ;
    LPIID           m_lpiid ;
    LPTSTR          m_lpszName ;

    FORMATETC       m_fetc ;
    DWORD           m_advf ;
    BOOL            m_fDoOnGetDataPosted ;

    HWND        m_hWndParent ;
    HWND        m_hDlg ;

    HWND        m_btnDoGetData ;
    HWND        m_btnSetupAdvise ;
    HWND        m_lbGetData ;
    HWND        m_edtAdvise ;
    TCHAR       m_szOutput[CCHOUTPUTMAX] ;
    METAFILEPICT m_MetaFile ;

    UINT        m_cchOutput ;
    UINT        m_cLinesOutput ;

    HWND        m_chkUpdateDisplay ;
    BOOL        m_fUpdateDisplay ;
    HWND        m_chkPrimeFirst ;

    HWND        m_lbFmtEtc ;

    HWND        m_chkDump ;

    DWORD       m_dwTime ;
    DWORD       m_cOnDataChanges ;

    CImpIAdviseSink FAR* m_pSink ;
    DWORD               m_dwConn ;

// Implementation
protected:
    BOOL DoIDataObject( UINT nLevel, LPDATAOBJECT pI ) ;
    BOOL DoIEnumFormatEtc( UINT nLevel, LPENUMFORMATETC pI ) ;

    int AddItem( UINT nLevel, LPTSTR sz, LPVOID lpData, UINT uiType, int
cColumns, LPCOLUMNSTRUCT  rgCol ) ;
    int AddItem( UINT nLevel, LPTSTR sz, LPVOID lpData, UINT uiType ) ;
    int AddItem( UINT nLevel, LPTSTR sz, LPVOID lpData ) ;

    BOOL OnInitDialog();

    void OnDoGetData() ;
    HRESULT GotData( LPFORMATETC lpfetc, LPSTGMEDIUM lpstm ) ;
    void OnSetupAdvise() ;
    void OnKillAdvise() ;

    void OnSize(UINT nType, int cx, int cy);
    void OnDrawItem(int nIDCtl, LPDRAWITEMSTRUCT lpDrawItemStruct);
    void OnMeasureItem(int nIDCtl, LPMEASUREITEMSTRUCT lpMeasureItemStruct);
    void OnDblClkFormatEtc();
    void OnSelChangeFormatEtc();
    void OnDestroy();
    BOOL  WriteToOutput( LPTSTR lpsz ) ;
    void OnOutputBufferHasData() ;
};


#endif // _IDATAOBJ_H_
```

IDATAOBJ.CPP   (DEFO2V Sample)

```
// idataobj.cpp
//
// Implementation file for the IDataObject interface viewer.
//
// Copyright (c) 1993 Microsoft Corporation, All Rights Reserved.
//
// Revisions:
//  August 7, 1993  cek     First implementation.
//  February 19, 1994 cek   Removed XRT stuff.  Fixed minor bugs.
//  April 9, 1994 cek       Pulled from XRTView and genericized again.
//
#include "precomp.h"
#include "defo2v.h"
#include "idataobj.h"
#include "iadvsink.h"

#if defined(WIN32) && !defined(_UNICODE)
    #pragma message("Building ANSI version of " __FILE__)
#endif

static HFONT       g_hFont ;
static HFONT       g_hFontBold ;
static UINT        g_cyFont ;

static WNDPROC     g_pfnEdit = NULL ;

extern "C"
LRESULT EXPORT CALLBACK fnEditSubclass( HWND hwnd, UINT uiMsg, WPARAM wP,
LPARAM lP ) ;

LPCTSTR MyGetClipboardFormatName( UINT cf ) ;

// DisplayIDataObject
//
// This is the "DisplayInterface" or "Interface Viewer" function for the
// IDataObject interface.   This implementation simply pops up a dialog
// box that has an ownerdraw listbox in it.
//
// It is expected that this function will be called from the Ole2View
// tool.   See the Ole2View helpfile in the OLE 2.01 and MSVC++ 1.5 kits
// for details on Ole2View interface viewers.
//
// The listbox is filled with the FORMATETCs that are enumerated through
// the IEnumFORMATETC interface that is retrieved from
IDataObject::EnumFormatEtc
//
extern "C"
HRESULT CALLBACK DisplayIDataObject( HWND hwndParent, LPDATAOBJECT lpDO,
LPIID lpiid, LPTSTR lpszName )
{
    HRESULT hr = NULL ;
    CIDataObjectDlg FAR* pdlg ;
    pdlg = new FAR CIDataObjectDlg( hwndParent, lpDO, lpiid, lpszName ) ;

    if (pdlg)
    {
        pdlg->DoModal() ;
        delete pdlg ;
    }

    return hr ;
}

/////////////////////////////////////////////////////////////////////////////
/
// CIDataObjectDlg dialog
//
CIDataObjectDlg::CIDataObjectDlg( HWND hwnd, LPDATAOBJECT lpDO, LPIID lpiid,
LPTSTR lpszName )
{
    m_lbFmtEtc = NULL ;
    m_edtAdvise = NULL ;
    m_MetaFile.hMF = NULL ;

    m_hWndParent = hwnd ;
    m_lpDO = lpDO ;
    m_lpiid = lpiid ;
    m_lpszName = lpszName ;

    m_pSink = NULL ;
    m_dwConn = 0 ;
    m_advf = ADVF_PRIMEFIRST ;

#ifdef _UNICODE
    m_fetc.cfFormat = CF_UNICODETEXT ;
#else
    m_fetc.cfFormat = CF_TEXT ;
#endif
    m_fetc.dwAspect = DVASPECT_CONTENT ;
    m_fetc.ptd = NULL ;
    m_fetc.tymed = TYMED_HGLOBAL ;
    m_fetc.lindex = - 1 ;

    m_fDoOnGetDataPosted = FALSE ;

    m_cchOutput = 0 ;
    m_cLinesOutput = 0 ;
}

CIDataObjectDlg::~CIDataObjectDlg()
{
    if (m_MetaFile.hMF != NULL)
        DeleteMetaFile( m_MetaFile.hMF ) ;
}

int CIDataObjectDlg::DoModal( void )
```

```
{
    return DialogBoxParam( g_hInst, MAKEINTRESOURCE( IDD_IDATAOBJDLG ),
m_hWndParent, (DLGPROC)fnIDataObjectDlg, (LONG)this ) ;
}

extern "C"
BOOL EXPORT CALLBACK fnIDataObjectDlg( HWND hDlg, UINT uiMsg, WPARAM wParam,
LPARAM lParam )
{
    static BOOL fInErrSpace = FALSE ;

    CIDataObjectDlg FAR* pIDOD =(CIDataObjectDlg FAR*)GetWindowLong( hDlg,
DWL_USER ) ;

    switch (uiMsg)
    {
        case WM_INITDIALOG:
            pIDOD=(CIDataObjectDlg FAR*)lParam ;
            if (pIDOD==NULL)
            {
                EndDialog( hDlg, 0 ) ;
                return TRUE ;
            }
            SetWindowLong( hDlg, DWL_USER, (LONG)pIDOD ) ;
            pIDOD->m_hDlg = hDlg ;
            return pIDOD->OnInitDialog() ;
        break ;

        case WM_DESTROY:
            if (pIDOD)
                pIDOD->OnDestroy() ;
        break ;

        case WM_SIZE:
            if (pIDOD)
                pIDOD->OnSize( (UINT)wParam, LOWORD( lParam ),
HIWORD( lParam ) ) ;
        break ;

        case WM_COMMAND:
        {
            #ifdef WIN32
            WORD wNotifyCode = HIWORD(wParam);
            WORD wID = LOWORD(wParam);
            HWND hwndCtl = (HWND) lParam;
            #else
            WORD wNotifyCode = HIWORD(lParam) ;
            WORD wID = wParam ;
            HWND hwndCtl = (HWND)LOWORD(lParam) ;
            #endif

            switch (wID)
            {
                case IDCANCEL:
                    EndDialog( hDlg, IDCANCEL ) ;
                break ;

                case IDC_DOGETDATA:
                    pIDOD->OnDoGetData() ;
                break ;

                case IDC_SETUPADVISE:
                    if (pIDOD->m_dwConn != 0)
                        pIDOD->OnKillAdvise() ;
                    else
                        pIDOD->OnSetupAdvise() ;
                break ;

                case IDC_CLEAROUTPUT:
                    if (pIDOD)
                    {
                        pIDOD->m_cchOutput = 0 ;
                        pIDOD->m_cLinesOutput = 0 ;
                        if (pIDOD->m_MetaFile.hMF != NULL)
                        {
                            DeleteMetaFile( pIDOD->m_MetaFile.hMF ) ;
                            pIDOD->m_MetaFile.hMF = NULL ;
                        }
                        SetWindowText( pIDOD->m_edtAdvise, _T("") ) ;
                        InvalidateRect( pIDOD->m_edtAdvise, NULL, TRUE ) ;
                        UpdateWindow( pIDOD->m_edtAdvise ) ;
                    }
                break ;

                case IDC_FORMATETC:
                    if (wNotifyCode == LBN_SELCHANGE)
                        pIDOD->OnSelChangeFormatEtc() ;
                break ;

                case IDC_UPDATEDISPLAY:
                    pIDOD->m_fUpdateDisplay = Button_GetCheck( pIDOD-
>m_chkUpdateDisplay ) ;
                break ;

                case IDC_PRIMEFIRST:
                    if (Button_GetCheck( pIDOD->m_chkPrimeFirst ))
                        pIDOD->m_advf = ADVF_PRIMEFIRST ;
                    else
                        pIDOD->m_advf = ADVF_NODATA ;
                break ;

                case IDC_ADVISEDATA:
                    if (wNotifyCode == EN_ERRSPACE)
                    {
                        #ifdef _DEBUG
                        OutputDebugString(_T("Output Edit Control reports
EN_ERRSPACE\r\n")) ;
                        #endif
                    }
                break ;
```

```
            }
        }
        break ;

        case WM_DRAWITEM:
            if (pIDOD)
                pIDOD->OnDrawItem( wParam, (LPDRAWITEMSTRUCT)lParam ) ;
        break ;

        case WM_MEASUREITEM:
            if (pIDOD)
                pIDOD->OnMeasureItem( wParam,
(LPMEASUREITEMSTRUCT)lParam ) ;
        break ;

        case WM_OUTPUTBUFFERHASDATA:
            if (pIDOD)
                pIDOD->OnOutputBufferHasData() ;
        break ;

        default:
           return FALSE ;
    }
    return TRUE ;
}


///////////////////////////////////////////////////////////////////////////
/
// CIDataObjectDlg message handlers

BOOL CIDataObjectDlg::OnInitDialog()
{
    m_btnDoGetData = GetDlgItem( m_hDlg, IDC_DOGETDATA ) ;
    m_btnSetupAdvise = GetDlgItem( m_hDlg, IDC_SETUPADVISE ) ;

    m_lbGetData = GetDlgItem( m_hDlg, IDC_GETDATA ) ;
    m_lbFmtEtc = GetDlgItem( m_hDlg, IDC_FORMATETC ) ;

    m_edtAdvise = GetDlgItem( m_hDlg, IDC_ADVISEDATA ) ;

    // Sublcass the edit so we can override it's WM_PAINT and
    // draw a metafile if we want to...
    if (g_pfnEdit == NULL)
        g_pfnEdit = SubclassWindow( m_edtAdvise, fnEditSubclass) ;

    m_chkUpdateDisplay = GetDlgItem( m_hDlg, IDC_UPDATEDISPLAY );
    m_fUpdateDisplay = TRUE ;
    Button_SetCheck( m_chkUpdateDisplay, m_fUpdateDisplay ) ;

    m_chkPrimeFirst = GetDlgItem( m_hDlg, IDC_PRIMEFIRST );
    Button_SetCheck( m_chkPrimeFirst, m_advf == ADVF_PRIMEFIRST ) ;

    //m_chkDump = GetDlgItem( m_hDlg, IDC_DUMPTOFILE ) ;
    //EnableWindow( m_chkDump, FALSE ) ;
    TEXTMETRIC  tm ;
    HDC hdc = GetDC(NULL);
    g_hFont = ReallyCreateFont( hdc, _T( "MS Sans Serif" ), _T( "Regular" ),
8, 0 ) ;
    g_hFontBold = ReallyCreateFont( hdc, _T( "MS Sans Serif" ),
_T( "Bold" ), 8, 0 ) ;
    g_hFont = (HFONT)SelectObject( hdc, g_hFont ) ;
    GetTextMetrics( hdc, &tm ) ;
    g_hFont = (HFONT)SelectObject( hdc, g_hFont ) ;
    ReleaseDC( NULL, hdc ) ;
    g_cyFont = tm.tmHeight + tm.tmExternalLeading ;

    SetWindowFont( m_lbFmtEtc, g_hFont, TRUE ) ;
    SetWindowFont( m_lbGetData, g_hFont, TRUE ) ;
    SetWindowFont( m_edtAdvise, g_hFont, TRUE ) ;

    if (IsEqualCLSID( IID_IEnumFORMATETC, *m_lpiid ))
        DoIEnumFormatEtc( 0, (LPENUMFORMATETC)m_lpDO) ;
    else if (IsEqualCLSID( IID_IDataObject, *m_lpiid ))
    {
        DoIDataObject( 0, (LPDATAOBJECT)m_lpDO ) ;
    }
    else
    {
        MessageBox( m_hDlg, _T( "Wrong IID" ), _T( "Error" ), MB_OK ) ;
    }

    DlgCenter( m_hDlg, m_hWndParent, FALSE ) ;
    RECT rc ;
    GetWindowRect( m_hDlg, &rc ) ;
    SetWindowPos( m_hDlg, NULL, rc.left, rc.top, rc.right - rc.left + 1,
                  rc.bottom - rc.top +1, SWP_NOMOVE|SWP_NOZORDER |
SWP_NOACTIVATE ) ;

    SetWindowText( m_hDlg, m_lpszName ) ;

    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CIDataObjectDlg::OnDestroy()
{
    OnKillAdvise() ;

    if (g_pfnEdit != NULL && m_edtAdvise && IsWindow(m_edtAdvise))
    {
        SubclassWindow(m_edtAdvise, g_pfnEdit) ;
        g_pfnEdit = NULL ;
    }

    int c = ListBox_GetCount( m_lbFmtEtc ) ;
    for (int i = 0 ; i < c ; i++)
    {
        LPITEMDATA lpID = (LPITEMDATA)ListBox_GetItemData( m_lbFmtEtc, i ) ;
        if (lpID)
        {
            if (lpID->uiType == I_COLUMNHEAD)
                delete lpID->rgCol ;

            if (lpID->lpData != NULL)
                delete (LPFORMATETC)lpID->lpData ;
            delete lpID ;
        }
    }

    if (g_hFont)
        DeleteObject( g_hFont );

    if (g_hFontBold)
        DeleteObject (g_hFontBold) ;
}

void CIDataObjectDlg::OnSize(UINT nType, int cx, int cy)
{
    if (m_edtAdvise && !IsWindow( m_edtAdvise ) )
        return ;

    RECT    rc ;
    RECT    rcWnd ;
    GetClientRect( m_hDlg, &rcWnd ) ;
    GetWindowRect( m_edtAdvise, &rc ) ;
    MapWindowPoints( NULL, m_hDlg, (POINT FAR*)&rc, 2 ) ;
    SetWindowPos( m_edtAdvise, NULL, -1, rc.top, cx+2, cy - rc.top + 1,
                SWP_NOZORDER | SWP_NOACTIVATE ) ;
}


void CIDataObjectDlg::OnDrawItem(int nIDCtl, LPDRAWITEMSTRUCT lpDIS )
{
    if (lpDIS->itemID == LB_ERR)
        return ;

    LPITEMDATA      lpID ;
    COLORREF        rgbBack ;
    RECT            rcFocus ;
    BOOL            fSelected ;
    int             x, y, cy ;
    TCHAR           szItem[128] ;

    lpID = (LPITEMDATA)lpDIS->itemData ;
    rcFocus = lpDIS->rcItem ;
    ListBox_GetText( m_lbFmtEtc, lpDIS->itemID, szItem ) ;

    if (fSelected = (lpDIS->itemState & ODS_SELECTED) ? TRUE : FALSE)
    {
        SetTextColor( lpDIS->hDC, GetSysColor( COLOR_HIGHLIGHTTEXT ) ) ;
        SetBkColor( lpDIS->hDC, rgbBack = GetSysColor( COLOR_HIGHLIGHT ) ) ;
    }
    else
    {
        SetTextColor( lpDIS->hDC, GetSysColor( COLOR_WINDOWTEXT ) ) ;
        SetBkColor( lpDIS->hDC, rgbBack = GetSysColor( COLOR_WINDOW ) ) ;
    }

    // if we are loosing focus, remove the focus rect before
    // drawing.
    //
    if ((lpDIS->itemAction) & (ODA_FOCUS))
        if (!((lpDIS->itemState) & (ODS_FOCUS)))
            DrawFocusRect( lpDIS->hDC, &rcFocus ) ;

    y = lpDIS->rcItem.top ;
    x = lpDIS->rcItem.left ;

    int cxChar = GetTextMetricsCorrectly( lpDIS->hDC, NULL ) ;

    if (lpID && (lpID->uiType == I_COLUMNHEAD || lpID->uiType == I_LABEL))
        g_hFontBold = (HFONT)SelectObject( lpDIS->hDC, g_hFontBold ) ;

    cy = (rcFocus.bottom - rcFocus.top - g_cyFont) / 2 ;

    ExtTextOut( lpDIS->hDC, x+2, y + cy, ETO_OPAQUE, &lpDIS->rcItem, NULL,
0, NULL ) ;
    if (lpID)
        ColumnTextOut( lpDIS->hDC, x + 2 + ((cxChar*3) * lpID->nLevel), y +
cy, szItem, lpID->cColumns, lpID->rgCol ) ;

    if (lpID && lpID->uiType == I_COLUMNHEAD )
    {
        COLORREF    rgb ;
        RECT        rc = rcFocus ;
        rgb = SetBkColor( lpDIS->hDC, GetTextColor( lpDIS->hDC ) ) ;
        rc.top = rc.bottom - 1 ;
        rc.left = x + 2 + ((cxChar*3) * lpID->nLevel) ;
        ExtTextOut( lpDIS->hDC, 0, 0, ETO_OPAQUE, &rc, NULL, 0, NULL ) ;
        SetBkColor( lpDIS->hDC, rgb ) ;
    }

    if (lpID && (lpID->uiType == I_COLUMNHEAD || lpID->uiType == I_LABEL))
        g_hFontBold = (HFONT)SelectObject( lpDIS->hDC, g_hFontBold ) ;

    // if we are gaining focus draw the focus rect after drawing
    // the text.
    if ((lpDIS->itemAction) & (ODA_FOCUS))
        if ((lpDIS->itemState) & (ODS_FOCUS))
             DrawFocusRect( lpDIS->hDC, &rcFocus ) ;

    if (fSelected)
    {
        SetTextColor( lpDIS->hDC, GetSysColor( COLOR_WINDOWTEXT ) ) ;
        SetBkColor( lpDIS->hDC, GetSysColor( COLOR_WINDOW ) ) ;
    }

}
```

```
void CIDataObjectDlg::OnMeasureItem(int nIDCtl, LPMEASUREITEMSTRUCT
lpMeasureItemStruct)
{
    lpMeasureItemStruct->itemHeight = g_cyFont ;
}

void CIDataObjectDlg::OnDblClkFormatEtc()
{
}

void CIDataObjectDlg::OnSelChangeFormatEtc()
{

    if (m_MetaFile.hMF != NULL)
    {
        DeleteMetaFile( m_MetaFile.hMF ) ;
        m_MetaFile.hMF = NULL ;
    }
    InvalidateRect( m_edtAdvise, NULL, TRUE ) ;
    SetWindowText( m_edtAdvise, _T("") ) ;

    int i = ListBox_GetCurSel( m_lbFmtEtc ) ;
    if (i != LB_ERR)
    {
        LPITEMDATA lpid = (LPITEMDATA)ListBox_GetItemData( m_lbFmtEtc, i ) ;
        if (lpid && lpid->lpData)
        {
            m_fetc = *(LPFORMATETC)lpid->lpData ;
            return ;
        }
    }

#ifdef UNICODE
    m_fetc.cfFormat = CF_UNICODETEXT ;
#else
    m_fetc.cfFormat = CF_TEXT ;
#endif
    m_fetc.dwAspect = DVASPECT_CONTENT ;
    m_fetc.ptd = NULL ;
    m_fetc.tymed = TYMED_HGLOBAL ;
    m_fetc.lindex = - 1 ;
}


void CIDataObjectDlg::OnDoGetData()
{
    HRESULT     hr ;
    STGMEDIUM   stm ;
    TCHAR       szBuf[256] ;

    if (m_lpDO == NULL)
    {
        ListBox_AddString( m_lbGetData, _T( "IDataObject viewer internal
error: m_lpDO == NULL" ) ) ;
        return ;
    }

    wsprintf( szBuf, _T( "Calling GetData for %s" ),
(LPCTSTR)MyGetClipboardFormatName( m_fetc.cfFormat ) ) ;
    ListBox_AddString( m_lbGetData, szBuf ) ;

    hr = m_lpDO->QueryGetData( &m_fetc ) ;
    if (FAILED( hr ))
    {
        wsprintf( szBuf, _T( "QueryGetData() failed:  %s" ),
                    (LPTSTR)HRtoString( hr ) ) ;
        ListBox_AddString( m_lbGetData, szBuf ) ;
        return ;
    }

    stm.tymed = m_fetc.tymed ;
    hr = m_lpDO->GetData( &m_fetc, &stm ) ;
    if (FAILED( hr ))
    {
        wsprintf( szBuf, _T( "GetData() failed:  %s" ),
                    (LPTSTR)HRtoString( hr ) ) ;
        ListBox_AddString( m_lbGetData, szBuf ) ;
        return ;
    }

    if (FAILED( hr = GotData( &m_fetc, &stm ) ))
    {
        wsprintf( szBuf, _T( "IDataDlg::GotData() failed:  %s" ),
                    (LPTSTR)HRtoString( hr ) ) ;
        ListBox_AddString( m_lbGetData, szBuf ) ;
        return ;
    }

    // If we're warm linked, then don't let the queue overfill.  Allow only
one
    // message at a time.  This is similar to the way Excel handles it's DDE
Links.
    //
    m_fDoOnGetDataPosted = FALSE ;
}


HRESULT CIDataObjectDlg::GotData( LPFORMATETC lpfetc, LPSTGMEDIUM lpstm )
{
    HRESULT hr = NOERROR ;
    BOOL    fHandled = FALSE ;

#ifdef WIN32
    if ((lpfetc->cfFormat == CF_TEXT  || lpfetc->cfFormat == CF_UNICODETEXT)
            && lpstm->tymed & TYMED_HGLOBAL)
#else
    if (lpfetc->cfFormat == CF_TEXT && lpstm->tymed & TYMED_HGLOBAL)
#endif
    {
        LPTSTR lpsz = (LPTSTR)GlobalLock( lpstm->hGlobal ) ;
        if (lpsz == NULL)
        {
            ListBox_AddString( m_lbGetData, _T( "hGlobal returned by GetData
is NULL!" ) ) ;
            ReleaseStgMedium( lpstm );
            return ResultFromScode( E_FAIL ) ;
        }

        if (m_fUpdateDisplay == TRUE)
            WriteToOutput( lpsz ) ;

        GlobalUnlock( lpstm->hGlobal ) ;

        fHandled = TRUE ;
    }

    if (lpfetc->cfFormat == CF_METAFILEPICT && lpstm->tymed & TYMED_MFPICT)
    {
        LPMETAFILEPICT pMF = (LPMETAFILEPICT)GlobalLock(lpstm->hGlobal) ;
        if (pMF)
        {
            if (m_MetaFile.hMF)
            {
                DeleteMetaFile( m_MetaFile.hMF ) ;
                m_MetaFile.hMF = NULL ;
            }

            if (m_fUpdateDisplay)
            {
                m_MetaFile = *pMF ;
                m_MetaFile.hMF = CopyMetaFile( pMF->hMF, NULL ) ;
                InvalidateRect( m_edtAdvise, NULL, FALSE ) ;
            }
        }
        fHandled = TRUE ;
    }

    m_cOnDataChanges++ ;

    ReleaseStgMedium( lpstm );

    if (fHandled == FALSE)
    {
        ListBox_AddString( m_lbGetData, _T( "The IDataObject viewer does not
know how to decode this FORMATETC" ) ) ;
    }

    return  hr;
}

void CIDataObjectDlg::OnSetupAdvise()
{
    TCHAR szBuf[128] ;
    OnKillAdvise() ;
    m_pSink = new CImpIAdviseSink( this ) ;

    if (m_pSink == NULL)
    {
        ListBox_AddString( m_lbGetData, _T( "CImpIAdviseSink constructor
failed" ) ) ;
        return ;
    }

    SetWindowText( m_btnSetupAdvise, _T( "&Kill Advise" ) ) ;

    wsprintf( szBuf, _T( "Advise started with a cfFormat of %s" ),
                (LPCTSTR)MyGetClipboardFormatName( m_fetc.cfFormat ) ) ;
    ListBox_AddString( m_lbGetData, szBuf ) ;

    m_dwTime = GetTickCount() ;
    m_cOnDataChanges = 0 ;

    HRESULT hr = m_lpDO->DAdvise( &m_fetc, m_advf, m_pSink, &m_dwConn ) ;
    if (FAILED( hr ))
    {
        wsprintf( szBuf, _T( "DAdvise() failed:  %s" ),
(LPTSTR)HRtoString( hr ) ) ;
        ListBox_AddString( m_lbGetData, szBuf ) ;
        SetWindowText( m_btnSetupAdvise, _T( "&DAdvise" ) ) ;
        return ;
    }

}

void CIDataObjectDlg::OnKillAdvise()
{
    if (m_dwConn != 0)
    {
        TCHAR szBuf[128] ;
        DWORD dwTime = max( 1000, GetTickCount() - m_dwTime ) ;    // avoid
div by zero

        m_lpDO->DUnadvise( m_dwConn ) ;
        m_dwConn = 0 ;
        wsprintf( szBuf, _T( "  Advise Killed after %lu milliseconds, and
%lu OnDataChanges;" ), dwTime, m_cOnDataChanges ) ;
        ListBox_AddString( m_lbGetData, szBuf ) ;
        wsprintf( szBuf, _T( "  That's %lu.%lu OnDataChange calls per
second." ),
                        m_cOnDataChanges / (dwTime / 1000L),
                        m_cOnDataChanges % (dwTime / 1000L) ) ;
        ListBox_AddString( m_lbGetData, szBuf ) ;
    }

    if (m_pSink != NULL)
    {
        m_pSink->Release() ;
        m_pSink = NULL ;
    }
    SetWindowText( m_btnSetupAdvise, _T( "&DAdvise" ) ) ;
}

// DoIDataObject
//
// This is where we start doing IDataObject dirty work.
// This function calls pIDataObject->EnumFormatEtc twice, once
// each for DATADIR_GET and DATADIR_SET, to get a pIEnumFORMATETC.
//
// We then call DoIEnumFormatEtc() with this pointer to actually
// perform the enumaration.
//
BOOL CIDataObjectDlg::DoIDataObject( UINT nLevel, LPDATAOBJECT
pIDataObject )
{
    LPENUMFORMATETC pEFE ;
    HRESULT     hr1 ;

    hr1 = pIDataObject->EnumFormatEtc( DATADIR_GET, &pEFE ) ;
    if (SUCCEEDED( hr1 ))
    {
        DoIEnumFormatEtc( nLevel, pEFE ) ;
        pEFE->Release() ;
    }
    else
        AddItem( nLevel, _T( "<<DATA_DIRGET failed>>" ), NULL, I_LABEL ) ;

    if (SUCCEEDED(hr1))
    {
        ListBox_SetCurSel( m_lbFmtEtc, 0 ) ;
        OnSelChangeFormatEtc() ;
    }
    return TRUE ;
}

// This member takes a pointer to an IEnumFORMATETC and enumerates
// the FORMATETC structures avaialbe, inserting them into our
// listbox.
//
BOOL CIDataObjectDlg::DoIEnumFormatEtc( UINT nLevel, LPENUMFORMATETC
pIFormatEtc )
{
    LPCOLUMNSTRUCT  rgCol = new COLUMNSTRUCT[5] ;
    UINT    cxChar ;
    HDC     hdc  = GetDC( NULL) ;

    // We use the "ColumnTextOut" code in util.c to create nice
    // columns in our list box.
    //
    g_hFontBold = (HFONT)SelectObject( hdc, g_hFontBold ) ;
    cxChar = GetTextMetricsCorrectly( hdc, NULL ) ;
    rgCol[0].nLeft = 0 ;
    rgCol[0].nRight = cxChar * lstrlen( _T( "CF_METAFILEPICT_" ) ) ;
    rgCol[0].uiFlags = DT_LEFT ;
    rgCol[1].nLeft = rgCol[0].nRight + cxChar ;
    rgCol[1].nRight = rgCol[1].nLeft + cxChar *
lstrlen( _T( "0000:0000_" ) ) ;
    rgCol[1].uiFlags = DT_LEFT ;

    rgCol[2].nLeft = rgCol[1].nRight + cxChar ;
    rgCol[2].nRight = rgCol[2].nLeft + cxChar *
lstrlen( _T( "THUMBNAIL_" ) ) ;
    rgCol[2].uiFlags = DT_LEFT ;

    rgCol[3].nLeft = rgCol[2].nRight + cxChar ;
    rgCol[3].nRight = rgCol[3].nLeft + cxChar *
lstrlen( _T( "0000:0000_" ) ) ;
    rgCol[3].uiFlags = DT_LEFT ;

    rgCol[4].nLeft = rgCol[3].nRight + cxChar ;
    rgCol[4].nRight = rgCol[4].nLeft + cxChar * lstrlen( _T( "_HGLOBAL_" ) )
;
    rgCol[4].uiFlags = DT_LEFT ;

    g_hFontBold = (HFONT)SelectObject( hdc, g_hFontBold ) ;
    ReleaseDC( NULL, hdc ) ;

    AddItem( nLevel, _T( "cfFormat\tptd\tdwAspect\tlindex\ttymed" ), NULL,
I_COLUMNHEAD, 5, rgCol ) ;
    LPFORMATETC     pfetc ;
    FORMATETC       fetc ;
    TCHAR           sz[512] ;
    ULONG           ulFE ; // number returned

    while (pIFormatEtc->Next( 1, &fetc, &ulFE ) == S_OK)
    {
        // now insert into list
        pfetc = new FORMATETC ;
        *pfetc = fetc ;
        lstrcpy( sz, MyGetClipboardFormatName( (UINT)fetc.cfFormat ) ) ;
        lstrcat( sz, _T("\t") ) ;

        TCHAR szTemp[32] ;
        wsprintf( szTemp, _T( "%04lX:%04lX\t" ),
(DWORD)HIWORD( (DWORD)fetc.ptd ), (DWORD)LOWORD( (DWORD)fetc.ptd ) ) ;
        lstrcat( sz, szTemp ) ;

        switch( fetc.dwAspect )
        {
            case DVASPECT_CONTENT: lstrcat( sz, _T( "CONTENT\t" ) ) ;
break ;
            case DVASPECT_THUMBNAIL: lstrcat( sz, _T( "THUMBNAIL\t" ) ) ;
break ;
            case DVASPECT_ICON: lstrcat( sz, _T( "ICON\t" ) ) ; break ;
            case DVASPECT_DOCPRINT: lstrcat( sz, _T( "DOCPRINT\t" ) ) ;
break ;
            default:
                wsprintf( szTemp, _T( "%08lX\t" ), (DWORD)fetc.dwAspect ) ;
                lstrcat( sz, szTemp ) ;
            break ;
        }

        wsprintf( szTemp, _T( "%04lX:%04lX\t" ),
(DWORD)HIWORD( fetc.lindex ), (DWORD)LOWORD( fetc.lindex ) ) ;
        lstrcat( sz, szTemp ) ;

        switch( fetc.tymed )
        {
            case TYMED_HGLOBAL: lstrcat( sz, _T( "HGLOBAL\t" ) ) ; break ;
            case TYMED_FILE: lstrcat( sz, _T( "FILE\t" ) ) ; break ;
            case TYMED_ISTREAM: lstrcat( sz, _T( "ISTREAM\t" ) ) ; break ;
            case TYMED_ISTORAGE: lstrcat( sz, _T( "ISTORAGE\t" ) ) ; break ;
            case TYMED_GDI: lstrcat( sz, _T( "GDI\t" ) ) ; break ;
            case TYMED_MFPICT: lstrcat( sz, _T( "MFPICT\t" ) ) ; break ;
            case TYMED_NULL: lstrcat( sz, _T( "NULL\t" ) ) ; break ;
            default:
                wsprintf( szTemp, _T( "%08lX\t" ), (DWORD)fetc.tymed ) ;
                lstrcat( sz, szTemp ) ;
            break ;
        }

        AddItem( nLevel, sz,
                 pfetc, I_COLUMNENTRY, 5, rgCol ) ;
    }

    return TRUE ;
}

LPCTSTR MyGetClipboardFormatName( UINT cf )
{
    static TCHAR sz[256] ;
    switch( cf )
    {

#ifdef WIN32
        case CF_UNICODETEXT: lstrcpy( sz, _T( "CF_UNICODETEXT" ) ) ; break ;
        case CF_ENHMETAFILE: lstrcpy( sz, _T( "CF_ENHMETAFILE" ) ) ; break ;
#endif
        case CF_TEXT: lstrcpy( sz, _T( "CF_TEXT" ) ) ; break ;
        case CF_BITMAP: lstrcpy( sz, _T( "CF_BITMAP" ) ) ; break ;
        case CF_METAFILEPICT: lstrcpy( sz, _T( "CF_METAFILEPICT" ) ) ; break
;
        case CF_SYLK: lstrcpy( sz, _T( "CF_SYLK" ) ) ; break ;
        case CF_DIF: lstrcpy( sz, _T( "CF_DIF" ) ) ; break ;
        case CF_TIFF: lstrcpy( sz, _T( "CF_TIFF" ) ) ; break ;
        case CF_OEMTEXT: lstrcpy( sz, _T( "CF_OEMTEXT" ) ) ; break ;
        case CF_DIB: lstrcpy( sz, _T( "CF_DIB" ) ) ; break ;
        case CF_PALETTE: lstrcpy( sz, _T( "CF_PALETTE" ) ) ; break ;
        case CF_PENDATA: lstrcpy( sz, _T( "CF_PENDATA" ) ) ; break ;
        case CF_RIFF: lstrcpy( sz, _T( "CF_RIFF" ) ) ; break ;
        case CF_WAVE: lstrcpy( sz, _T( "CF_WAVE" ) ) ; break ;
        case 0: lstrcpy( sz, _T( "\"Wildcard Advise\"" ) ) ; break ;
        default:
            if (!GetClipboardFormatName( (UINT)cf, sz, 254 ))
                wsprintf( sz, _T( "%#08lX" ), (DWORD)cf ) ;
        break ;
    }

    return sz ;
}


UINT DBStrCpy(LPTSTR pszDst, LPCTSTR pszSrc, int far* pcLines) ;

BOOL  CIDataObjectDlg::WriteToOutput( LPTSTR lpsz )
{
    UINT cch;
    int cLines;

    cch = DBStrCpy(NULL, lpsz, &cLines);

    if (cch > (CCHOUTPUTMAX - m_cchOutput))
    {
        MessageBeep(0);
        return FALSE;
    }

    DBStrCpy(&m_szOutput[m_cchOutput], lpsz, NULL);

    m_cchOutput += cch;
    m_cLinesOutput += cLines;

    // If buffer was originally empty, wake up the main loop.
    //
    if (m_cchOutput == cch)
        PostMessage( m_hDlg, WM_OUTPUTBUFFERHASDATA, 0, 0L);

    return TRUE;
}

// This function performs a string copy (or string length if pszDst is NULL)
// It also ensures the destination string is properly formatted for the
// edit controls: line terminators must be CR followed by LF, in that
orderc.
// If pcLines != NULL, returns number of lines in the string in *pcLines
//
// (pulled from the dbwin sample)
//
UINT DBStrCpy(LPTSTR pszDst, LPCTSTR pszSrc, int far* pcLines)
{
    TCHAR ch;
    UINT cch = 0;
    int cLines = 0;

    while (ch = *pszSrc++)
    {
        // If we find a CR or LF, then store a CR/LF in
        // the output string.
        //
        if (ch == 0x0d || ch == 0x0a)
        {
            cch += 2;
            cLines++;
            if (pszDst)
            {
                *pszDst++ = 0x0d;
                *pszDst++ = 0x0a;
            }

            // Skip any other CRs or LFs we find.
            //
            while ((ch = *pszSrc) && (ch == 0x0d || ch == 0x0a))
                ++pszSrc;
        }
        else
        {
            cch++;
            if (pszDst)
                *pszDst++ = ch;
        }
    }
    if (pszDst)
        *pszDst = 0;

    if (pcLines)
        *pcLines = cLines;

    return cch;
}

// Whenever something happens that causes output, it calls the WriteToOutput
function
// which puts the data into a buffer (m_szOutput) and posts a message.  This
is
// the message handler.
//
void CIDataObjectDlg::OnOutputBufferHasData()
{
    if (m_szOutput != NULL && m_cchOutput != 0)
    {
        int cLines = Edit_GetLineCount(m_edtAdvise) + m_cLinesOutput ;

        if (cLines > CLINESMAX)
        {
            SetWindowRedraw( m_edtAdvise, FALSE ) ;
            // If the total # of lines is greater than our arbitrary max
lines
            // remove some from the top.
            //
            Edit_SetSel( m_edtAdvise, 0, Edit_LineIndex( m_edtAdvise, cLines
- CLINESMAX));
            Edit_ReplaceSel( m_edtAdvise, _T(""));
        }
        // Put current output buffer at the end of the edit control
        //
        Edit_SetSel( m_edtAdvise, (UINT)-1, (UINT)-1 )  ;   // select the
end
        Edit_ReplaceSel( m_edtAdvise, m_szOutput ) ;
        Edit_SetSel( m_edtAdvise, (UINT)-1, (UINT)-1 )  ;   // select the
end

        if (cLines > CLINESMAX)
            SetWindowRedraw( m_edtAdvise, TRUE ) ;

        // Reset the output buffer
        m_cLinesOutput = 0 ;
        m_cchOutput = 0 ;
    }
}


// AddItem does a ListBox_AddString() plus a ListBox_SetItemData().  The
// item data that is set tells our owner-draw code what 'level' this
// item is at (indentation) and whether it is a label (bold), column
// head (column info is stored), or column entry (no column info is
// stored, but previous column head is used in WM_DRAWITEM).
//
int CIDataObjectDlg::AddItem( UINT nLevel, LPTSTR sz, LPVOID lpData, UINT
uiType, int cColumns, LPCOLUMNSTRUCT  rgCol )
{
    int i ;
    LPITEMDATA      lpID ;

    i = ListBox_AddString( m_lbFmtEtc,  sz ) ;
    lpID = new ITEMDATA ;
    lpID->uiType = uiType ;
    lpID->cColumns = cColumns ;
    lpID->rgCol = rgCol ;
    lpID->nLevel = nLevel ;
    lpID->lpData = lpData ;
    ListBox_SetItemData( m_lbFmtEtc, i, (DWORD)lpID ) ;
    return i ;
}

int CIDataObjectDlg::AddItem( UINT nLevel, LPTSTR sz, LPVOID lpData, UINT
uiType )
{
    return AddItem( nLevel, sz, lpData, uiType, 0, NULL ) ;
}

int CIDataObjectDlg::AddItem( UINT nLevel, LPTSTR sz, LPVOID lpData )
{
    return AddItem( nLevel, sz, lpData, I_NORMAL, 0, NULL ) ;
}

extern "C"
```

```
LRESULT EXPORT CALLBACK fnEditSubclass( HWND hwnd, UINT uiMsg, WPARAM wP,
LPARAM lP )
{
    if (uiMsg == WM_PAINT)
    {
        HWND hwndParent = GetParent(hwnd) ;
        CIDataObjectDlg FAR* pIDOD =(CIDataObjectDlg
FAR*)GetWindowLong( hwndParent, DWL_USER ) ;
        if (pIDOD && pIDOD->m_MetaFile.hMF != NULL)
        {
            PAINTSTRUCT ps ;
            BeginPaint( hwnd, &ps ) ;

/*
            if (pIDOD->m_MetaFile.mm == MM_ISOTROPIC || pIDOD->m_MetaFile.mm
== MM_ANISOTROPIC)
            {
                if (pIDOD->m_MetaFile.xExt != 0 && pIDOD->m_MetaFile.yExt !=
0)
                {

                }
            }
            else
            {
                SetMapMode( ps.hdc, MM_ANISOTROPIC ) ;
                SetWindowOrg( ps.hdc, 0, 0 ) ;
                SetWindowExt( ps.hdc, pIDOD->m_MetaFile.xExt, pIDOD-
>m_MetaFile.yExt ) ;
                SetViewportExt( ps.hdc, pIDOD->m_MetaFile.xExt, pIDOD-
>m_MetaFile.yExt ) ;
            }
*/
            PlayMetaFile( ps.hdc, pIDOD->m_MetaFile.hMF ) ;

            EndPaint( hwnd, &ps ) ;
            return 0L ;
        }
    }

    return CallWindowProc( g_pfnEdit, hwnd, uiMsg, wP, lP ) ;
}
```

IDATAO_A.CPP (DEFO2V Sample)

```
// ansiwrap.cpp
//
// Creates ANSI versions of all modules.  Any exported function get's "A"
// appended.
//
//
#ifdef UNICODE
#undef UNICODE
#endif
#ifdef _UNICODE
#undef _UNICODE
#endif

#include "precompa.h"
#include "idataobj.cpp"
```

IDISP.H (DEFO2V Sample)

```
// idataobj.h
//
// Header file for the IDispatch interface viewer.
//
// Copyright (c) 1993 Microsoft Corporation, All Rights Reserved.
//
// Charlie Kindel, Program Manager
// Microsoft Vertical Developer Relations
// August 7, 1993
//
// InterNet   :  ckindel@microsoft.com
// CompuServe :  >INTERNET:ckindel@microsoft.com
//
// Revisions:
//  August 7, 1993  cek     First implementation.
//
#ifndef _IDISP_H_
#define _IDISP_H_

#if defined(WIN32) && !defined(_UNICODE)
    #define DisplayIDispatch DisplayIDispatchA
    #define DisplayITypeLib DisplayITypeLibA
    #define DisplayITypeInfo DisplayITypeInfoA
    #define TYPEKINDtoString TYPEKINDtoStringA
    #define fnIDispDlg  fnIDispDlgA
    #define IDispDlg    IDispDlgA
#endif

LPTSTR TYPEKINDtoString( TYPEKIND tk );

////////////////////////////////////////////////////////////////////////////
/
// IDispDlg dialog

extern "C"
BOOL WINAPI fnIDispDlg( HWND hDlg, UINT uiMsg, WPARAM wParam, LPARAM
lParam ) ;

class FAR IDispDlg
{
    friend BOOL WINAPI fnIDispDlg( HWND hDlg, UINT uiMsg, WPARAM wParam,
LPARAM lParam ) ;

// Construction
public:
    IDispDlg( HWND hwnd, LPUNKNOWN lpUnk, LPIID lpiid, LPTSTR lpszName ) ;
    ~IDispDlg() ;
    int DoModal( void ) ;

    // Input parameters (caller allocated)
    LPIID       m_piid ;
    LPTSTR       m_pszName ;
    LPDISPATCH  m_pDispatch ;
    LPTYPEINFO  m_pTypeInfo ;
    LPTYPELIB   m_pTypeLib ;

    // Dialog management
    HWND        m_hWndParent ;
    HWND        m_hDlg ;

    HWND        m_hwndTypeInfoCount;
    HWND        m_hwndTypeInfo;
    HWND        m_hwndTypeAttr;
    HWND        m_hwndFunctionsLbl;
    HWND        m_hwndFunctions;
    HWND        m_hwndVariablesLbl;
    HWND        m_hwndVariables;
    HWND        m_hwndFuncProto ;
    HWND        m_hwndFuncProtoLbl ;
    HWND        m_hwndInfo;
    HWND        m_hwndInfoLbl;
    HWND        m_hwndTypeInfoInfo ;
    HWND        m_btnToFile ;

    BOOL        m_fImplType;
// Implementation
protected:
    void DoGetInfo( ) ;

    BOOL DoTypeAttr( LPTYPEINFO  pTI ) ;
    void DoGetFunctions( LPTYPEINFO lpTI, WORD cFuncs ) ;
    void DoGetVars( LPTYPEINFO lpTI, WORD cVars ) ;
    void DoGetImplTypes( LPTYPEINFO lpTI, WORD cImplTypes );

    BOOL OnInitDialog();
    void OnSize(UINT nType, int cx, int cy);
    void OnDrawItem(int nIDCtl, LPDRAWITEMSTRUCT lpDrawItemStruct);
    void OnMeasureItem(int nIDCtl, LPMEASUREITEMSTRUCT lpMeasureItemStruct);
    void OnDestroy();
    void OnSelChangeTypeInfo();
    void OnSelChangeFunctions();
    void OnSelChangeVariables();
    void OnToFile() ;
    BOOL GetImplFlag();
    void SetImplFlag( BOOL fVal );
};

#endif // _IDISP_H_
```

IDISP.CPP   (DEFO2V Sample)

```
// idisp.cpp
//
// Implementation file for the IDispatch interface viewer.
//
// Copyright (c) 1993 Microsoft Corporation, All Rights Reserved.
//
// Charlie Kindel, Program Manager
// Microsoft Vertical Developer Relations
// August 7, 1993
//
// Internet   :  ckindel@microsoft.com
// CompuServe :  >INTERNET:ckindel@microsoft.com
//
// Revisions:
//  August 7, 1993  cek     First implementation.
//
//
// This module contains the Interface Viewer functions for the
// IDispatch, ITypeInfo, and ITypeLib interfaces.
// This implementation displays all of the
// type information that can be retrieved from an arbitrary IDispatch,
// ITypeInfo, or ITypeLib pointer.
//
// NOTE:  This module exports the following interface viewers:
//
//      DisplayIDispatch()
//      DisplayITypeInfo()
//      DisplayITypeLib()
//
// Each of these functions boils down to the same dialog and code.
//
// Note that attempting to 'look at' an arbitrary ITypeInfo pointer
// which was received from QueryInterface() is a somewhat bogus thing
// to do.   We provide DisplayITypeInfo here only for completeness.
//
//
// ANOTHER NOTE:
// Since this viewer can take an ITypeLib, it makes a great Type Library
// viewer.  That is why it is used by Ole2View for the File.View Type Libary
// command...
//

#include "precomp.h"
#include "defo2v.h"
#include "idisp.h"
#include "tofile.h"

#if defined(WIN32) && !defined(_UNICODE)
    #pragma message("Building ANSI version of " __FILE__)
#endif

static HFONT        g_hFont ;
static UINT         g_cyFont ;

// These functions take a hwnd of a listbox, an IDispatch related structure,
// and string and dump the contents of the structure to the listbox.
// All structure members are prefixed by szLabel in the listbox.
//
void DumpTYPEDESC( HWND hwnd, TYPEDESC FAR* lptdesc, LPTSTR szLabel ) ;
void DumpARRAYDESC( HWND hwnd, ARRAYDESC FAR* lpadesc, LPTSTR szLabel ) ;
void DumpIDLDESC( HWND hwnd, LPIDLDESC lp, LPTSTR szLabel ) ;
void DumpELEMDESC( HWND hwnd, LPELEMDESC lp, LPTSTR szLabel ) ;
void DumpFUNCDESC( HWND hwnd, LPFUNCDESC lpfndesc, LPTSTR szLabel ) ;
void DumpVARDESC( HWND hwnd, LPVARDESC lpvardesc, LPTSTR szLabel ) ;

static TCHAR * g_rgszTYPEKIND[] =
{
    _T("enum"),         /* TKIND_ENUM */
    _T("Struct"),       /* TKIND_RECORD */
    _T("Module"),       /* TKIND_MODULE */
    _T("Interface"),    /* TKIND_INTERFACE */
    _T("Dispinterface"),    /* TKIND_DISPATCH */
    _T("CoClass"),      /* TKIND_COCLASS */
    _T("Alias"),        /* TKIND_ALIAS */

    /* NOTE: the following aren't supported in typeinfo yet */
    _T("Union"),        /* TKIND_UNION */
    _T("Encap Union"),   /* TKIND_ENCUNION */
    _T("Class"),        // TKIND_Class
    _T("CoType"),       // TKIND_COOTYPE
#if WIN32==300
    _T("unused"),
    _T("PropSet"),
    _T("EnventSet"),
    _T("CmdSet"),
#endif
    _T("<error!>"),     // dummy entries just in case the ole spec changes
    _T("<error!>"),     // and new enum's are added!
    _T("<error!>"),
    _T("<error!>"),
    _T("<error!>"),
};

LPTSTR TYPEKINDtoString( TYPEKIND tk )
{
    return (LPTSTR)g_rgszTYPEKIND[tk] ;
}

static TCHAR * g_rgszIDLFLAGS[] =
{
    _T(""),         //IDLFLAG_NONE = 0,
    _T("IN"),       //IDLFLAG_FIN = 1,
    _T("OUT"),      //IDLFLAG_FOUT = 2,
    _T("IN OUT"),   //IDLFLAF_FIN | IDLFLAG_FOUT
    _T("<error!>"),     // dummy entries just in case the ole spec changes
    _T("<error!>"),     // and new enum's are added!
    _T("<error!>"),
    _T("<error!>"),
    _T("<error!>"),
};

static TCHAR * g_rgszFUNCKIND[] =
{
    _T("VIRUTAL"),
    _T("PUREVIRTUAL"),
    _T("NONVIRTUAL"),
    _T("STATIC"),
    _T("DISPATCH"),
    _T("<error!>"),     // dummy entries just in case the ole spec changes
    _T("<error!>"),     // and new enum's are added!
    _T("<error!>"),
    _T("<error!>"),
    _T("<error!>"),
} ;

static TCHAR * g_rgszCALLCONV[] =
{
    _T("0"),
    _T("CDECL"),        // CC_CDECL, = 1,
    _T("PASCAL"),       // CC_MSCPASCAL and CC_PASCAL
    _T("MACPASCAL"),     // CC_MACPASCAL,
    _T("STDCALL"),       // CC_STDCALL,
    _T("THISCALL"),      // CC_THISCALL,
    _T("MAX"),           // end of enum marker
    _T("<error!>"),     // dummy entries just in case the ole spec changes
    _T("<error!>"),     // and new enum's are added!
    _T("<error!>"),
    _T("<error!>"),
    _T("<error!>"),
};

static TCHAR * g_rgszVARKIND[] =
{
    _T("PERINSTANCE"),  //VAR_PERINSTANCE,
    _T("STATIC"),       //VAR_STATIC
    _T("CONST"),        //VAR_CONST
    _T("DISPATCH"),      //VAR_DISPATCH
    _T("<error!>"),     // dummy entries just in case the ole spec changes
    _T("<error!>"),     // and new enum's are added!
    _T("<error!>"),
    _T("<error!>"),
    _T("<error!>"),
} ;

//ASSERTDATA
// DisplayIDispatch
//
// This
STDAPI DisplayIDispatch( HWND hwndParent, LPDISPATCH lpDisp, LPIID lpiid,
LPTSTR lpszName )
{
    HRESULT hr = NULL ;
```

```
//    Assert( hwndParent ) ;
//    Assert( lpDisp ) ;
//    Assert( lpiid ) ;

    IDispDlg    dispdlg( hwndParent, (LPUNKNOWN)lpDisp, lpiid, lpszName ) ;

    dispdlg.DoModal() ;

    return hr ;
}


// DisplayITypeInfo
//
// This
STDAPI DisplayITypeInfo( HWND hwndParent, LPTYPEINFO lpTypeInfo, LPIID
lpiid, LPTSTR lpszName )
{
    HRESULT hr = NULL ;

//    Assert( hwndParent ) ;
//    Assert( lpTypeInfo ) ;
//    Assert( lpiid ) ;

    IDispDlg    dispdlg( hwndParent, (LPUNKNOWN)lpTypeInfo,lpiid, lpszName )
;
    dispdlg.DoModal() ;

    return hr ;
}

STDAPI DisplayITypeLib( HWND hwndParent, LPTYPELIB lpTypeLib, LPIID lpiid,
LPTSTR lpszName )
{
    HRESULT hr = NULL ;
    int     rc;

//    Assert( hwndParent ) ;
//    Assert( lpTypeLib ) ;
//    Assert( lpiid ) ;

    IDispDlg    dispdlg( hwndParent, (LPUNKNOWN)lpTypeLib, lpiid, lpszName )
;
    rc = dispdlg.DoModal() ;

    return hr ;
}


/////////////////////////////////////////////////////////////////////////////
/
// IDispDlg dialog
//
// The constructor is where we begin to descern between the three possible
// interfaces that could be passed in.
//
IDispDlg::IDispDlg(HWND hwnd, LPUNKNOWN lpUnk, LPIID lpiid, LPTSTR
lpszName )
{
    m_hWndParent = hwnd ;
    m_piid = lpiid ;
    m_pszName = lpszName ;

    m_pDispatch = NULL ;
    m_pTypeInfo = NULL ;
    m_pTypeLib = NULL ;
    m_fImplType = FALSE;

    if (IsEqualIID( *m_piid, IID_IDispatch ))
    {
        m_pDispatch = (LPDISPATCH)lpUnk ;
        m_pDispatch->AddRef() ;
    }
#ifdef OB32BUG
    else if (IsEqualIID( *m_piid, IID_IMyTypeInfo ))
#else
    else if (IsEqualIID( *m_piid, IID_ITypeInfo ))
#endif

    {
        m_pTypeInfo = (LPTYPEINFO)lpUnk ;
        m_pTypeInfo->AddRef() ;
    }
#ifdef OB32BUG
    else if (IsEqualIID( *m_piid, IID_IMyTypeLib ))
#else
    else if (IsEqualIID( *m_piid, IID_ITypeLib ))
#endif

    {
        m_pTypeLib = (LPTYPELIB)lpUnk ;
        m_pTypeLib->AddRef() ;
    }
    else
    {
//        AssertSz(0, _T("Some Thing is wrong\n") ) ;
    }
}

IDispDlg::~IDispDlg( )
{
}

int IDispDlg::DoModal( void )
{
    return DialogBoxParam( g_hInst, MAKEINTRESOURCE( IDD_IDISPDLG ),
m_hWndParent, (DLGPROC)fnIDispDlg, (LONG)this ) ;
}
```

```
extern "C"
BOOL EXPORT WINAPI fnIDispDlg( HWND hDlg, UINT uiMsg, WPARAM wParam, LPARAM
lParam )
{
    IDispDlg FAR* pIDD =(IDispDlg FAR*)GetWindowLong( hDlg, DWL_USER ) ;

    switch (uiMsg)
    {
        case WM_INITDIALOG:
            pIDD=(IDispDlg FAR*)lParam ;
            if (pIDD==NULL)
            {
                EndDialog( hDlg, 0 ) ;
                return TRUE ;
            }
            pIDD->m_hDlg = hDlg ;
            SetWindowLong( hDlg, DWL_USER, (LONG)pIDD ) ;
            pIDD->OnInitDialog() ;
            PostMessage( hDlg, WM_USER+1, 0, 0 ) ;
        break ;

        // It's nice to let the dialog appear before we
        // do a lengthy operation.
        case WM_USER+1:
        {
            HCURSOR hcur = SetCursor( LoadCursor( NULL, IDC_WAIT ) ) ;
            pIDD->DoGetInfo() ;
            SetCursor( hcur ) ;
        }
        break ;

        case WM_DESTROY:
            if (pIDD)
                pIDD->OnDestroy() ;
        break ;

        case WM_SIZE:
            if (pIDD)
                pIDD->OnSize( (UINT)wParam, LOWORD( lParam ), HIWORD( lParam
) ) ;
        break ;

        case WM_COMMAND:
        {
#ifdef WIN32
            WORD wNotifyCode = HIWORD(wParam);
            WORD wID = LOWORD(wParam);
            HWND hwndCtl = (HWND) lParam;
#else
            WORD wNotifyCode = HIWORD(lParam) ;
            WORD wID = wParam ;
            HWND hwndCtl = (HWND)LOWORD(lParam) ;
#endif

            switch (wID)
            {
                case IDCANCEL:
                    EndDialog( hDlg, IDCANCEL ) ;
                break ;

                case IDC_TOFILE:
                    pIDD->OnToFile() ;
                break ;

                case IDC_TYPEINFO:
                    if (wNotifyCode == LBN_SELCHANGE)
                        pIDD->OnSelChangeTypeInfo() ;
                break ;

                case IDC_TYPEATTR:
                break ;

                case IDC_FUNCTIONS:
                    switch(wNotifyCode)
                    {
                        case LBN_SELCHANGE:
                            pIDD->OnSelChangeFunctions() ;
                        break;
                    }
                break ;

                case IDC_VARIABLES:
                    switch(wNotifyCode)
                    {
                        case LBN_SELCHANGE:
                            pIDD->OnSelChangeVariables() ;
                        break;
                    }
                break ;
            }
        }
        break ;

        default:
           return FALSE ;
    }
    return TRUE ;
}

void IDispDlg::SetImplFlag( BOOL fVal )
{
    m_fImplType= fVal;
}

BOOL IDispDlg::GetImplFlag( )
{
    return m_fImplType;
}

BOOL IDispDlg::OnInitDialog()
```

```
{
    // Setup member vars, so we have easy access to
    // dialog items.
    //
    m_hwndTypeInfoCount = GetDlgItem( m_hDlg, IDC_TYPEINFOCOUNT ) ;
    m_hwndTypeInfo = GetDlgItem( m_hDlg, IDC_TYPEINFO ) ;
    m_hwndTypeAttr = GetDlgItem( m_hDlg, IDC_TYPEATTR ) ;
    m_hwndFunctionsLbl = GetDlgItem( m_hDlg, IDC_FUNCTIONS_LBL ) ;
    m_hwndFunctions = GetDlgItem( m_hDlg, IDC_FUNCTIONS ) ;
    m_hwndVariablesLbl = GetDlgItem( m_hDlg, IDC_VARIABLES_LBL ) ;
    m_hwndVariables = GetDlgItem( m_hDlg, IDC_VARIABLES ) ;
    m_hwndFuncProtoLbl = GetDlgItem( m_hDlg, IDC_FUNCPROTO_LBL ) ;
    m_hwndFuncProto = GetDlgItem( m_hDlg, IDC_FUNCPROTO ) ;
    m_hwndInfo = GetDlgItem( m_hDlg, IDC_FUNCVARDESC ) ;
    m_hwndInfoLbl = GetDlgItem( m_hDlg, IDC_FUNCVARDESC_LBL ) ;
    m_hwndTypeInfoInfo = GetDlgItem( m_hDlg, IDC_TYPEINFOINFO ) ;
    m_btnToFile = GetDlgItem( m_hDlg, IDC_TOFILE ) ;

    // Create fonts we may use..
    //
    TEXTMETRIC  tm ;
    HDC hdc = GetDC(NULL);
    g_hFont = ReallyCreateFont( hdc, _T("MS Sans Serif"), _T("Regular"), 8,
0 ) ;
    g_hFont = (HFONT) SelectObject( hdc, g_hFont ) ;
    GetTextMetrics( hdc, &tm ) ;
    g_hFont = (HFONT) SelectObject( hdc, g_hFont ) ;
    ReleaseDC( NULL, hdc ) ;
    g_cyFont = tm.tmHeight + tm.tmExternalLeading ;

    SetWindowFont( m_hwndTypeInfoCount, g_hFont, TRUE ) ;
    SetWindowFont( m_hwndVariables, g_hFont, TRUE ) ;
    SetWindowFont( m_hwndTypeAttr, g_hFont, TRUE ) ;
    SetWindowFont( m_hwndInfo, g_hFont, TRUE ) ;
    SetWindowFont( m_hwndTypeInfoInfo, g_hFont, TRUE ) ;
    SetWindowFont( m_hwndFunctions, g_hFont, TRUE ) ;
    SetWindowFont( m_hwndFuncProto, g_hFont, TRUE ) ;

    DlgCenter( m_hDlg, m_hWndParent, FALSE ) ;
    RECT rc ;
    GetWindowRect( m_hDlg, &rc ) ;
    SetWindowPos( m_hDlg, NULL, rc.left, rc.top, rc.right - rc.left + 1,
                  rc.bottom - rc.top +1, SWP_NOMOVE|SWP_NOZORDER |
SWP_NOACTIVATE ) ;

    SetWindowText( m_hDlg, m_pszName ) ;

    return TRUE;  // return TRUE  unless you set the focus to a control
}

void IDispDlg::OnDestroy()
{
    if (g_hFont)
        DeleteObject( g_hFont );
    if (m_pDispatch != NULL)
    {
        m_pDispatch->Release() ;
        m_pDispatch = NULL ;
    }

    if (m_pTypeInfo != NULL)
    {
        m_pTypeInfo->Release() ;
        m_pTypeInfo = NULL ;
    }

    if (m_pTypeLib != NULL)
    {
        m_pTypeLib->Release() ;
        m_pTypeLib = NULL ;
    }
}


void IDispDlg::OnSize(UINT nType, int cx, int cy)
{
//    if (m_hwndTypeAttr && !IsWindow( m_hwndTypeAttr ) )
        return ;

    RECT    rc ;
    RECT    rcWnd ;
    GetClientRect( m_hDlg, &rcWnd ) ;
    GetWindowRect( m_hwndTypeAttr, &rc ) ;
    MapWindowPoints( NULL, m_hDlg, (POINT FAR*)&rc, 2 ) ;
    rc.left = rcWnd.left + 5 ;
    SetWindowPos( m_hwndTypeAttr, NULL, rcWnd.left + 5, rc.top, cx - 10,
rc.bottom - rc.top,
                SWP_NOZORDER | SWP_NOACTIVATE ) ;
}

void IDispDlg::OnToFile()
{
    if (m_pTypeInfo == NULL && m_pTypeLib == NULL)
    {
        MessageBeep( 0 ) ;
        return ;
    }

    if (m_pTypeLib)
        DoTypeLibToFile( m_hDlg, m_pTypeLib ) ;
    else
        DoTypeInfoToFile( m_hDlg, m_pTypeInfo ) ;
}

// This starts our gathering of information.
// Here, we figure out which interface we were called for, get as
// much info as we can from it, including the other two
// interfaces if possible.   We fill the typeinfo listbox
// with all type infos found.
```

```
//
void IDispDlg::DoGetInfo()
{

    TCHAR        szTemp[128] ;
    UINT        uiTypeInfoCount = 0;
    HRESULT hr ;

    ComboBox_ResetContent( m_hwndTypeInfo ) ;

    if (m_pDispatch != NULL)
    {
        // If we were called for IDispatch we need to get at least the
typelib.
        //
        // This whole thing is done somewhat roundabout:
        //
        // 1) Call GetTypeInfo to get the typeinfo for the programmability
interface.
        // 2) Try to get the "containing TypeLib" for the typeinfo.
        // 3a) If we get m_pTypeLib back then this object was implemented
using a Type Library
        //     and will probably have more than one typeinfo.
        // 3b) If the call to GetContainingTypeLib fails then this object
does not
        //     use typelibs and thus probably has only one typeinfo.
        // 4) In either case we eventually get a count of typeinfos and we
        //    use either IDispatch::GetTypeInfo or ITypeLib::GetTypeInfo to
get the
        //    type information for the object.
        //
        hr = m_pDispatch->GetTypeInfoCount( &uiTypeInfoCount ) ;
        if (FAILED( hr ))
        {
            wsprintf( szTemp, _T("IDispatch::GetTypeInfoCount( ) FAILED:
%s"), (LPTSTR)HRtoString( hr ) ) ;
            MessageBox( m_hDlg, szTemp, _T("Could not get type
information"), MB_OK ) ;
            return ;
        }

        if (uiTypeInfoCount <= 0)
        {
            wsprintf( szTemp, _T("This object reports zero TypeInfos.  There
is no programmability information available.") ) ;
            MessageBox( m_hDlg, szTemp, _T("Could not get type
information"), MB_OK ) ;
            return ;
        }

        hr = m_pDispatch->GetTypeInfo( 0, LOCALE_SYSTEM_DEFAULT,
&m_pTypeInfo ) ;
        if (FAILED( hr ))
        {
            wsprintf( szTemp, _T("IDispatch::GetTypeInfo( 0 ) FAILED: %s"),
(LPTSTR)HRtoString( hr ) ) ;
            MessageBox( m_hDlg, szTemp, _T("Could not get type
information"), MB_OK ) ;
            return ;
        }

        UINT    ui ;
        hr = m_pTypeInfo->GetContainingTypeLib( &m_pTypeLib, &ui ) ;
        if (SUCCEEDED( hr ))
        {
            // We no longer need the typeinfo ptr
            //
            m_pTypeInfo->Release() ;
            m_pTypeInfo = NULL ;
            uiTypeInfoCount = m_pTypeLib->GetTypeInfoCount( ) ;
            SetWindowText( m_hwndTypeInfoInfo, _T("This object uses a Type
Library.") ) ;
        }
        else
        {
            // We no longer need the typeinfo ptr.
            //
            m_pTypeInfo->Release() ;
            m_pTypeInfo = NULL ;

            // Bogus.  The app does not support TypeLib (i.e. WinWord 6.0).
Therefore we must
            // use IDispatch::GetTypeInfoCount...
            //
            hr = m_pDispatch->GetTypeInfoCount( &uiTypeInfoCount ) ;
            if (FAILED( hr ))
            {
                wsprintf( szTemp, _T("IDispatch::GetTypeInfoCount FAILED:
%s"), (LPTSTR)HRtoString( hr ) ) ;
                MessageBox(  m_hDlg, szTemp, _T("Could not get count of
typeinfos"), MB_OK ) ;
                return ;
            }

            SetWindowText( m_hwndTypeInfoInfo, _T("This object does not use
a Type Library.") ) ;
        }
    }
    else if (m_pTypeLib != NULL)
    {
        // If we were called with a typelib we are in great shape
        //
        uiTypeInfoCount = m_pTypeLib->GetTypeInfoCount() ;
        SetWindowText( m_hwndTypeInfoInfo, _T("Browsing a TypeLibrary." )) ;
    }
    else if (m_pTypeInfo != NULL)
    {
        // We were called with at TypeInfo.  This is the specialist of
cases.
```

```
        UINT    ui ;
        hr = m_pTypeInfo->GetContainingTypeLib( &m_pTypeLib, &ui ) ;
        if (SUCCEEDED( hr ))
        {
            // We no longer need the typeinfo ptr
            //
            m_pTypeInfo->Release() ;
            m_pTypeInfo = NULL ;
            uiTypeInfoCount = m_pTypeLib->GetTypeInfoCount( ) ;
            SetWindowText( m_hwndTypeInfoInfo, _T("This object uses a Type
Library.") ) ;
        }
        else
        {
            // If we get here then we assume that there is only one
            // typeinfo and we're looking at it.
            //
            uiTypeInfoCount = 1 ;
            SetWindowText( m_hwndTypeInfoInfo, _T("Browsing a TypeInfo") ) ;
        }
    }

    wsprintf( szTemp, _T("%u"), (UINT)uiTypeInfoCount ) ;
    SetWindowText( m_hwndTypeInfoCount, szTemp ) ;

    // Fill combo box with all typeinfos
    //
    //
    for (UINT n = 1 ; n <= uiTypeInfoCount ; n++)
    {
        if (m_pTypeLib != NULL || m_pDispatch != NULL)
        {
            if (m_pTypeInfo != NULL)
            {
                m_pTypeInfo->Release() ;
                m_pTypeInfo = NULL ;
            }

            if (m_pTypeLib != NULL)
            {
                hr = m_pTypeLib->GetTypeInfo( n-1, &m_pTypeInfo ) ;
            }
            else if (m_pDispatch != NULL)
                hr = m_pDispatch->GetTypeInfo( n-1, LOCALE_SYSTEM_DEFAULT,
&m_pTypeInfo ) ;
        }
        else if (m_pTypeInfo != NULL)
            hr = ResultFromScode( S_OK ) ;
        else
            hr = ResultFromScode( E_FAIL ) ;

        if (SUCCEEDED(hr))
        {
            // Note:  Early OLE2 samples (IDispatch Calculator #2)
            // would crash on a GetDocumentation( MEMBERID_NIL... ) call
            // it is commented out for now until we are sure that
            // it is working in OLE 2.01.
            //
#if 1
            BSTR    bstrName ;
            hr = m_pTypeInfo->GetDocumentation( MEMBERID_NIL, &bstrName,
NULL, NULL, NULL ) ;
            if (SUCCEEDED(hr))
            {

                UINT ui ;
#if defined(WIN32) && !defined (OLE2ANSI)
                wsprintf( szTemp, _T("%ws (%u)"), bstrName ? bstrName :
OLESTR("(null name)"), n-1 ) ;
#else
                wsprintf( szTemp, _T("%s (%u)"), bstrName ? bstrName :
_T("(null name)"), n-1 ) ;
#endif
                ui = ComboBox_AddString( m_hwndTypeInfo, szTemp ) ;
                ComboBox_SetItemData( m_hwndTypeInfo, ui, (DWORD)n-1 ) ;
                SysFreeString( bstrName ) ;
            }
            else
#else
            BSTR            rgbstrNames[2] ;

            UINT            cNames ;
            hr = m_pTypeInfo->GetNames( MEMBERID_NIL, rgbstrNames, 1,
&cNames );
            if (SUCCEEDED( hr ))
            {
                UINT ui ;
#if defined(WIN32) && !defined(OLE2ANSI)
                wsprintf( szTemp, _T("%ws (%u)"), (LPOLESTR)rgbstrNames[0],
n-1 ) ;
#else
                wsprintf( szTemp, _T("%s (%u)"), (LPTSTR)rgbstrNames[0], n-1
) ;
#endif
                ui = ComboBox_AddString( m_hwndTypeInfo, szTemp ) ;
                ComboBox_SetItemData( m_hwndTypeInfo, ui, (DWORD)n ) ;
                SysFreeString( rgbstrNames[0] ) ;
            }
            else
#endif
            {
                UINT ui ;
                wsprintf( szTemp, _T("%u (no name)"), n-1 ) ;
                ui = ComboBox_AddString( m_hwndTypeInfo, szTemp ) ;
                ComboBox_SetItemData( m_hwndTypeInfo, ui, (DWORD)n-1 ) ;
            }
        }
        else
        {
            wsprintf(szTemp, _T("%s::GetTypeInfo(#%u) FAILED: %s"),
                        (LPTSTR)(m_pTypeLib == NULL ? _T("IDispatch") :
                            _T("ITypeLib") ), n-1,
(LPTSTR)HRtoString( hr ) ) ;
            MessageBox( m_hDlg, szTemp, _T("Error"), MB_OK ) ;
        }

    }

    ComboBox_SetCurSel( m_hwndTypeInfo, 0 ) ;

    // kick start
    OnSelChangeTypeInfo() ;

    return ;
}

// When the selection changes in the combo, we first make sure
// our current TypeInfo pointer is Released.
// Then get the pTypeInfo for the newly selected item.
// Use that pointer to populate the typeinfo listbox (actually
// m_pTypeAttr).
//
void IDispDlg::OnSelChangeTypeInfo()
{
    HRESULT     hr ;
    UINT        uiTinfo ;
    int         uiItem = ComboBox_GetCurSel( m_hwndTypeInfo );
    TCHAR        szTemp[128] ;

    if (uiItem == CB_ERR)
        return ;

    ListBox_ResetContent( m_hwndTypeAttr ) ;
    ListBox_ResetContent( m_hwndFunctions ) ;
    ListBox_ResetContent( m_hwndVariables ) ;
    ListBox_ResetContent( m_hwndInfo ) ;
    SetWindowText( m_hwndFuncProto, _T("") ) ;
    SetWindowText( m_hwndInfoLbl, _T("F&UNCDESC/VARDESC:") ) ;

    hr = ResultFromScode( E_FAIL ) ;
    if ((m_pDispatch != NULL || m_pTypeLib != NULL) && m_pTypeInfo != NULL)
    {
        m_pTypeInfo->Release() ;
        m_pTypeInfo = NULL ;
        // Get memid from item
        uiTinfo = (UINT)ComboBox_GetItemData( m_hwndTypeInfo, uiItem ) ;

        // Get typeinfo pointer
        if (m_pTypeLib != NULL)
            hr = m_pTypeLib->GetTypeInfo( uiTinfo, &m_pTypeInfo ) ;
        else
            hr = m_pDispatch->GetTypeInfo( uiTinfo, LOCALE_SYSTEM_DEFAULT,
&m_pTypeInfo ) ;
    }
    else if (m_pTypeInfo != NULL)
```

```
        hr = ResultFromScode( S_OK ) ;

    if (SUCCEEDED(hr))
    {

        DWORD   dwHelp ;
        BSTR    bstrName, bstrDoc, bstrHelpFile ;
        hr = m_pTypeInfo->GetDocumentation( MEMBERID_NIL, &bstrName,
&bstrDoc,  &dwHelp, &bstrHelpFile ) ;
        if (SUCCEEDED(hr))
        {
#if defined(WIN32) && !defined(OLE2ANSI)
            wsprintf( szTemp, _T("Name = %ws"), (LPOLESTR)(bstrName ==
NULL ? OLESTR("n/a") : bstrName) ) ;
#else
            wsprintf( szTemp, _T("Name = %s"), (LPTSTR)(bstrName == NULL ?
_T("n/a") : bstrName) ) ;
#endif
            ListBox_AddString( m_hwndTypeAttr, szTemp ) ;


#if defined(WIN32) && !defined(OLE2ANSI)
            wsprintf( szTemp, _T("DocString = %ws"), (bstrDoc == NULL ?
OLESTR("n/a") : bstrDoc) ) ;
#else
            wsprintf( szTemp, _T("DocString = %s"), (bstrDoc == NULL ?
_T("n/a") : bstrDoc) ) ;
#endif
            ListBox_AddString( m_hwndTypeAttr, szTemp ) ;

            wsprintf( szTemp, _T("Help Context ID = %#08lX"),
(DWORD)dwHelp ) ;
            ListBox_AddString( m_hwndTypeAttr, szTemp ) ;
#if defined(WIN32) && !defined(OLE2ANSI)
            wsprintf( szTemp, _T("Help File = %ws"), (LPOLESTR)(bstrHelpFile
== NULL ? OLESTR("n/a") : bstrHelpFile) ) ;
#else
            wsprintf( szTemp, _T("Help File = %s"), (LPTSTR)(bstrHelpFile ==
NULL ? _T("n/a") : bstrHelpFile) ) ;
#endif
            ListBox_AddString( m_hwndTypeAttr, szTemp ) ;
            SysFreeString( bstrName ) ;
            SysFreeString( bstrDoc) ;
            SysFreeString( bstrHelpFile ) ;
        }
        else
        {
            wsprintf( szTemp, _T("GetDocumentation FAILED: %s"), HRtoString(
hr ) ) ;
            ListBox_AddString( m_hwndTypeAttr, szTemp ) ;
        }

        // Get the containing typelib for this typeinfo.
        //
        UINT ui ;
        LPTYPELIB   pTL ;
        hr = m_pTypeInfo->GetContainingTypeLib( &pTL, &ui ) ;
        if (SUCCEEDED( hr ))
        {
            wsprintf( szTemp, _T("GetContainingTypeLib returned valid
pTypelib (%#08lX, %u)"),
                        (LONG)pTL, ui ) ;
            pTL->Release() ;
        }
        else
            wsprintf( szTemp, _T("GetContainingTypeLib FAILED:  %s"),
HRtoString( hr ) ) ;
        ListBox_AddString( m_hwndTypeAttr, szTemp ) ;

        // Call DoTypeAttr to get the TypeAttr info and to
        // fill the functions and variables listboxes.
        //
        DoTypeAttr( m_pTypeInfo ) ;

        // NEED: What other information can we put in this listbox?
        // what else is available, given ITypeInfo  pointer?
        //
        //
    }
    else
    {
        wsprintf( szTemp, _T("GetTypeInfo(#%u) FAILED: %s"), uiTinfo,
HRtoString( hr ) ) ;
        ListBox_AddString( m_hwndTypeAttr, szTemp ) ;
    }
}

BOOL IDispDlg::DoTypeAttr( LPTYPEINFO  pTI )
{
    HRESULT hr ;
    TCHAR    szTemp[64] ;
    LPTYPEATTR  pTA = NULL ;

    hr = pTI->GetTypeAttr( &pTA ) ;
    if (FAILED( hr ))
    {
        wsprintf( szTemp, _T("GetTypeAttr FAILED: %s"), HRtoString( hr ) ) ;
        ListBox_AddString( m_hwndTypeAttr, szTemp ) ;
        return FALSE ;
    }


    if (pTA->typekind > TKIND_MAX)
        wsprintf( szTemp, _T("typekind = <invalid!> (%u)"), pTA-
>typekind ) ;
    else
        wsprintf( szTemp, _T("typekind = %s"), (LPSTR)g_rgszTYPEKIND[pTA-
>typekind] ) ;

    ListBox_AddString( m_hwndTypeAttr, szTemp ) ;
    wsprintf( szTemp, _T("Version =  %u.%03d"), pTA->wMajorVerNum, pTA-
>wMajorVerNum ) ;
    ListBox_AddString( m_hwndTypeAttr, szTemp ) ;
    wsprintf( szTemp, _T("lcid = %u"),  pTA->lcid ) ;
    ListBox_AddString( m_hwndTypeAttr, szTemp ) ;
    wsprintf( szTemp, _T("cFuncs = %d"), pTA->cFuncs ) ;
    ListBox_AddString( m_hwndTypeAttr, szTemp ) ;
    wsprintf( szTemp, _T("cVars = %u"), pTA->cVars ) ;
    ListBox_AddString( m_hwndTypeAttr, szTemp ) ;
    wsprintf( szTemp, _T("cImplTypes = %u"), pTA->cImplTypes ) ;
    ListBox_AddString( m_hwndTypeAttr, szTemp ) ;

    // DumpTYPEDESC will completely decode the tdescAlias for us...
    //
    DumpTYPEDESC( m_hwndTypeAttr, &pTA->tdescAlias, _T("tdescAlias." )) ;

    wsprintf(szTemp, _T("guid = {%08lX-%04X-%04X-%02X%02X-
%02X%02X%02X%02X%02X%02X}"),
        pTA->guid.Data1, pTA->guid.Data2, pTA->guid.Data3,
        pTA->guid.Data4[0], pTA->guid.Data4[1],
        pTA->guid.Data4[2], pTA->guid.Data4[3],
        pTA->guid.Data4[4], pTA->guid.Data4[5],
        pTA->guid.Data4[6], pTA->guid.Data4[7]);

    ListBox_AddString( m_hwndTypeAttr, szTemp ) ;

    wsprintf(szTemp, _T("wTypeFlags = %04X"), pTA->wTypeFlags ) ;
    ListBox_AddString( m_hwndTypeAttr, szTemp ) ;

    wsprintf(szTemp, _T("dwReserved = %#08lX"), pTA->dwReserved ) ;
    ListBox_AddString( m_hwndTypeAttr, szTemp ) ;

    wsprintf(szTemp, _T("cbAlignment = %u"), pTA->cbAlignment ) ;
    ListBox_AddString( m_hwndTypeAttr, szTemp ) ;

    wsprintf(szTemp, _T("cbSizeInstance = %u"), pTA->cbSizeInstance ) ;
    ListBox_AddString( m_hwndTypeAttr, szTemp ) ;

    wsprintf(szTemp, _T("cbSizeVft = %u"), pTA->cbSizeVft ) ;
    ListBox_AddString( m_hwndTypeAttr, szTemp ) ;

    DumpIDLDESC( m_hwndTypeAttr, &pTA->idldescType, _T("idldescType.") ) ;

    wsprintf(szTemp, _T("memidConstructor = %#08lX"), (DWORD)pTA-
>memidConstructor ) ;
    ListBox_AddString( m_hwndTypeAttr, szTemp ) ;

    wsprintf(szTemp, _T("memidDestructor = %#08lX"), (DWORD)pTA-
>memidDestructor ) ;
    ListBox_AddString( m_hwndTypeAttr, szTemp ) ;

    // Get functions
    DoGetFunctions( pTI, pTA->cFuncs ) ;

    // Get variables
    DoGetVars( pTI, pTA->cVars ) ;
#if WIN32==300
    // GetImpl Types
    if ( pTA->typekind == TKIND_COTYPE)
    {
        DoGetImplTypes( pTI, pTA->cImplTypes );
    }
#endif

    if (pTI != NULL)
        pTI->ReleaseTypeAttr( pTA ) ;
    return TRUE ;
}



// Fill the function listbox with a list of all functions.
// The entries in the listbox show just the name of the function and
// when the selection changes, the static box below the list is
// updated to show the return value and parameters (OnFunctionsSelChange()).
//
void IDispDlg::DoGetFunctions( LPTYPEINFO lpTI, WORD cFuncs )
{
    TCHAR    szBuf[128] ;
    HRESULT hr ;
    TCHAR            szTemp[80] ;
    BSTR           rgbstrNames[2] ;

    UINT            cNames ;
    LPFUNCDESC      pFuncDesc ;
    WORD iIndex ;

    ListBox_ResetContent( m_hwndFunctions ) ;

    for ( iIndex = 0 ; iIndex < cFuncs ; iIndex++)
    {
        hr = lpTI->GetFuncDesc( iIndex, &pFuncDesc ) ;
        if (FAILED(hr))
        {
            wsprintf( szBuf, _T("GetFuncDesc FAILED for function #%u (%s)"),
iIndex, (LPTSTR)HRtoString( hr ) ) ;
            ListBox_AddString( m_hwndFunctions, szBuf ) ;
            continue ;
        }

        // We could get the function name from either GetDocumentation or
from
        // GetNames...we do it from GetNames...
        // (I had some problems getting GetDocumentation() to work with some
        // of the automation examples lying around)
        //

        // We AddString each function name into the list box, then
        // SetItemData 'i' so that we know the memberid of the function
        // for OnFunctionSelChange()
        //
        hr = lpTI->GetNames( pFuncDesc->memid, rgbstrNames, 1, &cNames );
        if (SUCCEEDED( hr ))
        {
            // rgbstrNames[0] is the name of the function
#if defined(WIN32) && !defined(UNICODE)
            char * szName = NULL;
            if (rgbstrNames[0])
            {
                szName = new char [wcslen(rgbstrNames[0]) + 1];
                if (szName)
                {
                    wcstombs(szName, rgbstrNames[0], wcslen(rgbstrNames[0])
+ 1);
                }
            }
            ListBox_SetItemData( m_hwndFunctions,
                             ListBox_AddString( m_hwndFunctions, szName ),
                             (LONG)iIndex ) ;
            delete [] szName;
#else
            ListBox_SetItemData( m_hwndFunctions,
                             ListBox_AddString( m_hwndFunctions,
rgbstrNames[0] ),
                             (LONG)iIndex ) ;
#endif
            SysFreeString( rgbstrNames[0] ) ;
        }
        else
        {
            wsprintf( szTemp, _T("GetNames (%lu) FAILED: %s"), pFuncDesc-
>memid, HRtoString( hr ) ) ;
            ListBox_AddString( m_hwndFunctions, szTemp ) ;
        }

        lpTI->ReleaseFuncDesc( pFuncDesc ) ;
    }

    ListBox_SetCurSel( m_hwndFunctions, 0 ) ;
    OnSelChangeFunctions() ;
}

// Set the text of m_hwndFuncProto so that it looks like this:
//
//    RetType FuncName ( Type Param1, Type Param2, ... )
//
// where RetType is a string associated with a VARTYPE (VTtoString())
// FuncName is bstrName.
void IDispDlg::OnSelChangeFunctions()
{
    int     iIndex = ListBox_GetCurSel( m_hwndFunctions ) ;
    #define MAX_NAMES   100
    BSTR    rgbstrNames[MAX_NAMES] ;
    UINT    cNames ;
    if (iIndex == LB_ERR || m_pTypeInfo == NULL)
    {
        SetWindowText( m_hwndFuncProto, _T("") ) ;
        return ;
    }

    TCHAR            szRet[128] ;
    TCHAR            szFuncName[64] ;
    lstrcpy( szFuncName, _T("<???>") ) ;

    LPTSTR           szfn = NULL ;
    LPFUNCDESC      pFuncDesc = NULL;
    HRESULT         hr ;
    UINT            cParams ;

    // the index of the FUNCDESC is stored as itemdata
    iIndex = (int)ListBox_GetItemData( m_hwndFunctions, iIndex ) ;

    // Get the FUNCDESC for this function.
    // Save off cParams and get the return value type
    //
    hr = m_pTypeInfo->GetFuncDesc( iIndex, &pFuncDesc ) ;
    if (SUCCEEDED(hr))
    {
        cParams = pFuncDesc->cParams ;
        lstrcpy( szRet, TYPEDESCtoString( &pFuncDesc->elemdescFunc.tdesc ) )
;
        lstrcat( szRet, _T(" ") ) ;
        // Get the names of the function and it's parameters into
rgbstrNames.
        // cNames gets the number of parameters + 1.
        //
        // Note:  T3 (2004 - 8/5/93) is really weird here.  It sometimes
        // returns cNames - 1 == pFuncDesc->cParams, sometimes it's < and
        // sometimes is >.   There is no sense to it.
        //
        hr = m_pTypeInfo->GetNames( pFuncDesc->memid, rgbstrNames,
MAX_NAMES, &cNames );
        if (SUCCEEDED( hr ))
        {
            // Allocate a string buffer that should be able to hold
            // all of our parameter types and names.
            //
            if ((szfn = (LPTSTR) GlobalAllocPtr( GPTR, max(cNames,cParams) *
(64) )) == NULL )
            {
                MessageBox( m_hDlg, _T("GlobalAlloc Failed!"), _T("Yikes!"),
MB_OK ) ;
                return ;
            }

            // put the return value first
            lstrcpy( szfn, szRet ) ;

            // rgbstrNames[0] is the name of the function
            if (cNames > 0)
            {
#if defined(WIN32) && !defined(UNICODE)
                wcstombs(szFuncName, rgbstrNames[0], wcslen(rgbstrNames[0])
+ 1);
#else
                lstrcpy( szFuncName, rgbstrNames[0] ) ;
#endif
                lstrcat( szfn, szFuncName) ;
                lstrcat( szfn, _T("(") ) ;
                SysFreeString( rgbstrNames[0] ) ;
            }

            // For each parameter get the type and name.
            // The "max(cNames-1,cParams)" should handle the case
            // where a function has optional parameters (i.e. cParamsOpt is
            // non-zero).
            //
            for ( UINT n = 0 ; n < max(cNames-1,cParams) ; n++ )
            {
                if (n == 0)
                    lstrcat( szfn, _T("\r\n    ") ) ;

                if (n < cParams)    // safety check
                {
                    lstrcat( szfn, g_rgszIDLFLAGS[pFuncDesc-
>lprgelemdescParam[n].idldesc.wIDLFlags] ) ;
                    lstrcat( szfn, _T(" ") ) ;
                    lstrcat( szfn, TYPEDESCtoString( &pFuncDesc-
>lprgelemdescParam[n].tdesc ) ) ;
                }

                if (n+1 < cNames)   // saftey check
                {
                    if (n < cParams)
                        lstrcat( szfn, _T(" ") ) ;
#if defined(WIN32) && !defined(UNICODE)
                    wcstombs(&szfn[strlen(szfn) + 1], rgbstrNames[n+1],
wcslen(rgbstrNames[n+1]) + 1);
#else
                    lstrcat( szfn, rgbstrNames[n+1] ) ;
#endif

                    SysFreeString( rgbstrNames[n+1] ) ;
                }
                if (n + 1 == max(cNames-1,cParams))
                    lstrcat( szfn, _T("\r\n    ") );
                else
                    lstrcat( szfn, _T(",\r\n    ") ) ;
            }
            lstrcat( szfn , _T(")")  );
        }
        else
        {
            lstrcat( szRet, _T("GetNames FAILED: ") ) ;
            lstrcat( szRet, HRtoString( hr ) ) ;
        }

        if (szfn)
        {
            SetWindowText( m_hwndFuncProto, szfn ) ;
            GlobalFreePtr( szfn );
        }
        else
            SetWindowText( m_hwndFuncProto, szRet ) ;
    }
    else
    {
        wsprintf( szRet, _T("GetFuncDesc FAILED: %s "), HRtoString( hr ) ) ;
        SetWindowText( m_hwndFuncProto, szRet ) ;
        pFuncDesc = NULL ;
    }

    if (pFuncDesc != NULL)
    {
        TCHAR szTemp[256] ;
        wsprintf( szTemp, _T("FUNC&DESC for %s:"), (LPSTR)szFuncName ) ;
        SetWindowText( m_hwndInfoLbl, szTemp ) ;
        ListBox_ResetContent( m_hwndInfo ) ;

        DWORD    dwHelp ;
        BSTR    bstrName, bstrDoc, bstrHelpFile ;

        hr = m_pTypeInfo->GetDocumentation( pFuncDesc->memid, &bstrName,
&bstrDoc,  &dwHelp, &bstrHelpFile ) ;
        if (SUCCEEDED(hr))
        {
#if defined(WIN32) && !defined(OLE2ANSI)
            wsprintf( szTemp, _T("Name = %ws"), (LPOLESTR)(bstrName ==
NULL ? OLESTR("n/a") : bstrName) ) ;
#else
            wsprintf( szTemp, _T("Name = %s"), (LPTSTR)(bstrName == NULL ?
_T("n/a") : bstrName) ) ;
#endif
            ListBox_AddString( m_hwndInfo, szTemp ) ;
//            ListBox_AddString( m_hwndInfo, bstrName ) ;    stevebl - this
lookes like it was a bug


#if defined(WIN32) && !defined(OLE2ANSI)
            wsprintf( szTemp, _T("DocString = %ws"), (bstrDoc == NULL ?
OLESTR("n/a") : bstrDoc) ) ;
#else
            wsprintf( szTemp, _T("DocString = %s"), (bstrDoc == NULL ?
_T("n/a") : bstrDoc) ) ;
#endif
            ListBox_AddString( m_hwndInfo, szTemp ) ;

            wsprintf( szTemp, _T("Help Context ID = %#08lX"),
(DWORD)dwHelp ) ;
```

```
            ListBox_AddString( m_hwndInfo, szTemp ) ;
#if defined(WIN32) && !defined(OLE2ANSI)
            wsprintf( szTemp, _T("Help File = %ws"), (LPOLESTR)(bstrHelpFile
== NULL ? OLESTR("n/a") : bstrHelpFile) ) ;
#else
            wsprintf( szTemp, _T("Help File = %s"), (LPTSTR)(bstrHelpFile ==
NULL ? _T("n/a") : bstrHelpFile) ) ;
#endif
            ListBox_AddString( m_hwndInfo, szTemp ) ;
            SysFreeString( bstrName ) ;
            SysFreeString( bstrDoc) ;
            SysFreeString( bstrHelpFile ) ;
        }
        else
        {
            wsprintf( szTemp, _T("GetDocumentation FAILED: %s"), HRtoString(
hr ) ) ;
            ListBox_AddString( m_hwndInfo, szTemp ) ;
        }

        DumpFUNCDESC( m_hwndInfo, pFuncDesc, _T("") ) ;
        m_pTypeInfo->ReleaseFuncDesc( pFuncDesc ) ;
    }
}

void IDispDlg::DoGetImplTypes( LPTYPEINFO lpTI, WORD cImplTypes )
{
    HRESULT     hr;
#ifdef WIN32
    ULONG       iIndex;
#else
    UINT        iIndex ;
#endif
    HREFTYPE    hRefType;
    ITypeInfo   *lpitihRef;
    BSTR       bstrName;

    ListBox_ResetContent( m_hwndVariables);
    SetImplFlag( TRUE );
    for ( iIndex  = 0 ; iIndex < cImplTypes ; iIndex++ )
    {
        hr = lpTI->GetRefTypeOfImplType( iIndex, &hRefType );
        if ( SUCCEEDED(hr))
        {
           hr = lpTI->GetRefTypeInfo( hRefType, &lpitihRef);
           if (SUCCEEDED(hr))
           {
                hr = lpitihRef->GetDocumentation( MEMBERID_NIL,
                                                  &bstrName,
                                                  NULL,
                                                  NULL,
                                                  NULL);
                if (SUCCEEDED(hr))
                {
#if defined(WIN32) && !defined(UNICODE)
```

```
                    char * szName = NULL;
                    if (bstrName)
                    {
                        szName = new char [wcslen(bstrName) + 1];
                        if (szName)
                        {
                            wcstombs(szName, bstrName, wcslen(bstrName) +
1);
                        }
                    }
                    ListBox_AddString( m_hwndVariables, szName ) ;
                    delete [] szName;
#else
                    ListBox_AddString( m_hwndVariables, bstrName ) ;
#endif

                }
                SysFreeString( bstrName );

            }
            lpitihRef->Release();
        }
    }
}


void IDispDlg::DoGetVars( LPTYPEINFO lpTI, WORD cVars )
{
    TCHAR    szBuf[128] ;
    WORD    iIndex ;
    TCHAR    szTemp[80] ;
    HRESULT hr ;
    BSTR    rgbstrNames[2] ;
    UINT    cNames ;

    ListBox_ResetContent( m_hwndVariables) ;

    for (iIndex = 0 ; iIndex < cVars ; iIndex++)
    {
        LPVARDESC      pVarDesc ;
        hr = lpTI->GetVarDesc( iIndex, &pVarDesc ) ;
        if (FAILED(hr))
        {
            wsprintf( szBuf, _T("GetVarDesc FAILED for variable #%u (%s)"),
iIndex, HRtoString( hr ) ) ;
            ListBox_AddString( m_hwndVariables, szBuf ) ;
            continue ;
        }
        else
        {
            // memid\tname
            hr = lpTI->GetNames( pVarDesc->memid, rgbstrNames, 1, &cNames );
            if (SUCCEEDED( hr ))
            {
#if defined(WIN32) && !defined(OLE2ANSI)
```

```
                wsprintf( szBuf, _T("%ws (%s"), rgbstrNames[0],
                            TYPEDESCtoString( &pVarDesc->elemdescVar.tdesc )
) ;
#else
                wsprintf( szBuf, _T("%s (%s"), rgbstrNames[0],
                            TYPEDESCtoString( &pVarDesc->elemdescVar.tdesc )
) ;
#endif
                lstrcat( szBuf, _T(")") ) ;
                SysFreeString( rgbstrNames[0] ) ;

                if (pVarDesc->varkind == VAR_CONST)
                {
                    // NEED:  This just prints out the type.  We can get the
value
                    // too.  Need to write at VARIANTtoString() function.
                    //
                    VARIANT varValue ;
                    VariantInit( &varValue ) ;
                    hr = VariantChangeType( &varValue, pVarDesc->lpvarValue,
0, VT_BSTR ) ;
                    if (FAILED(hr))
                        wsprintf( szTemp, _T(" = %s"), HRtoString( hr ) ) ;
                    else
                    {
                        wsprintf( szTemp, _T(" = %s"), varValue.bstrVal ) ;
                        SysFreeString( varValue.bstrVal ) ;
                    }

                    lstrcat( szBuf, szTemp ) ;
                }

                // rgbstrNames[0] is the name of the function
                ListBox_SetItemData( m_hwndVariables,
                                 ListBox_AddString( m_hwndVariables,
szBuf ),
                                 (LONG)iIndex ) ;
            }
            else
            {
                wsprintf( szTemp, _T("GetNames (%lu) FAILED: %s"), pVarDesc-
>memid, HRtoString( hr ) ) ;
                ListBox_AddString( m_hwndVariables, szTemp ) ;
            }

            lpTI->ReleaseVarDesc( pVarDesc ) ;
        }
    }

    ListBox_SetCurSel( m_hwndVariables, 0 ) ;
    OnSelChangeVariables() ;
}


void IDispDlg::OnSelChangeVariables()
```

```
{
    // we are overloading the m_hwndVariables to have
    // contain impltype info, which don't have an list
    // box item data.

    if (GetImplFlag())
    {
        return;
    }
    int         iIndex = ListBox_GetCurSel( m_hwndVariables ) ;
    LPVARDESC   pVarDesc = NULL;
    HRESULT     hr ;
    BSTR        rgbstrNames[2] ;

    UINT        cNames ;
    TCHAR        szTemp[256] ;

    if (iIndex == LB_ERR || m_pTypeInfo == NULL)
        return ;

    ListBox_ResetContent( m_hwndInfo ) ;

    // the index of the VARDESC is stored as itemdata
    iIndex = (int)ListBox_GetItemData( m_hwndVariables, iIndex ) ;

    // Get the VARDESC for this Vartion.
    // Save off cParams and get the return value type
    //
    hr = m_pTypeInfo->GetVarDesc( iIndex, &pVarDesc ) ;
    if (SUCCEEDED(hr))
    {
        hr = m_pTypeInfo->GetNames( pVarDesc->memid, rgbstrNames, 1, &cNames
);
        if (SUCCEEDED(hr))
        {
#if defined(WIN32) && !defined(OLE2ANSI)
            wsprintf( szTemp, _T("Var&DESC for %ws:"), rgbstrNames[0] ) ;
#else
            wsprintf( szTemp, _T("Var&DESC for %s:"), rgbstrNames[0] ) ;
#endif
            SysFreeString( rgbstrNames[0] ) ;

        }
        else
            lstrcpy( szTemp, _T("Var&DESC:") ) ;
        SetWindowText( m_hwndInfoLbl, szTemp ) ;

        DWORD   dwHelp ;
        BSTR    bstrName, bstrDoc, bstrHelpFile ;
        hr = m_pTypeInfo->GetDocumentation( pVarDesc->memid, &bstrName,
&bstrDoc,  &dwHelp, &bstrHelpFile ) ;
        if (SUCCEEDED(hr))
        {
#if defined(WIN32) && !defined(OLE2ANSI)
```

```
            wsprintf( szTemp, _T("Name = %ws"), (LPOLESTR)(bstrName ==
NULL ? OLESTR("n/a") : bstrName) ) ;
#else
            wsprintf( szTemp, _T("Name = %s"), (LPTSTR)(bstrName == NULL ?
_T("n/a") : bstrName) ) ;
#endif
            ListBox_AddString( m_hwndInfo, szTemp ) ;
//            ListBox_AddString( m_hwndInfo, bstrName ) ;    stevebl - this
lookes like it was a bug


#if defined(WIN32) && !defined(OLE2ANSI)
            wsprintf( szTemp, _T("DocString = %ws"), (bstrDoc == NULL ?
OLESTR("n/a") : bstrDoc) ) ;
#else
            wsprintf( szTemp, _T("DocString = %s"), (bstrDoc == NULL ?
_T("n/a") : bstrDoc) ) ;
#endif
            ListBox_AddString( m_hwndInfo, szTemp ) ;

            wsprintf( szTemp, _T("Help Context ID = %#08lX"),
(DWORD)dwHelp ) ;
            ListBox_AddString( m_hwndInfo, szTemp ) ;
#if defined(WIN32) && !defined(OLE2ANSI)
            wsprintf( szTemp, _T("Help File = %ws"), (LPOLESTR)(bstrHelpFile
== NULL ? OLESTR("n/a") : bstrHelpFile) ) ;
#else
            wsprintf( szTemp, _T("Help File = %s"), (LPTSTR)(bstrHelpFile ==
NULL ? _T("n/a") : bstrHelpFile) ) ;
#endif
            ListBox_AddString( m_hwndInfo, szTemp ) ;

            SysFreeString( bstrName ) ;
            SysFreeString( bstrDoc) ;
            SysFreeString( bstrHelpFile ) ;

        }
        else
        {
            wsprintf( szTemp, _T("GetDocumentation FAILED: %s"), HRtoString(
hr ) ) ;
            ListBox_AddString( m_hwndInfo, szTemp ) ;
        }

        DumpVARDESC( m_hwndInfo, pVarDesc, _T("") ) ;
        m_pTypeInfo->ReleaseVarDesc( pVarDesc ) ;

    }
    else
    {
        wsprintf( szTemp, _T("GetVarDesc FAILED: %s "),
(LPTSTR)HRtoString( hr ) ) ;
        ListBox_AddString( m_hwndInfo, szTemp ) ;
    }
}
```

```
void DumpTYPEDESC( HWND hwnd, TYPEDESC FAR* lptdesc, LPTSTR szLabel )
{
    TCHAR szTemp[128] ;
    if (lptdesc == NULL)
    {
        wsprintf( szTemp, _T("%slptdesc is NULL (this is an error)"),
(LPTSTR)szLabel ) ;
        ListBox_AddString( hwnd, szTemp ) ;
        return ;
    }

    // Dump a TYPEDESC
    wsprintf( szTemp, _T("%svt = %s"), (LPTSTR)szLabel,
(LPTSTR)VTtoString( lptdesc->vt ) ) ;
    ListBox_AddString( hwnd, szTemp ) ;

    if ((lptdesc->vt & 0x0FFF) == VT_PTR)
    {
        // lptdesc points to a TYPEDESC that specifies the thing pointed to
        wsprintf( szTemp, _T("%slptdesc->"), (LPTSTR)szLabel ) ;
        DumpTYPEDESC( hwnd, lptdesc->lptdesc, szTemp ) ;
        return ;
    }

    if ((lptdesc->vt & 0x0FFF) == VT_USERDEFINED)
    {
        wsprintf( szTemp, _T("%shreftype = %#08lX"), (LPTSTR)szLabel,
(DWORD)lptdesc->hreftype ) ;
        ListBox_AddString( hwnd, szTemp ) ;
        return ;
    }

    if ((lptdesc->vt & 0x0FFF) == VT_CARRAY)
    {
        wsprintf( szTemp, _T("%slpadesc->"), (LPTSTR)szLabel ) ;
        DumpARRAYDESC( hwnd, lptdesc->lpadesc, szTemp ) ;
        return ;
    }

#ifdef _VT_FUNCPTR_DEFINED
    if ((lptdesc->vt & 0x0FFF) == VT_FUNCPTR)
    {
        wsprintf( szTemp, "%slpfdesc->", szLabel ) ;
        DumpFUNCDESC( hwnd, lptdesc->lpadesc, szTemp ) ;
        return ;
    }
#endif
}


void DumpARRAYDESC( HWND hwnd, ARRAYDESC FAR* lpadesc, LPTSTR szLabel )
{
    TCHAR szTemp[128] ;
    if (lpadesc == NULL)
    {
        wsprintf( szTemp, _T("%slpadesc is NULL (this is an error)"),
szLabel ) ;
        ListBox_AddString( hwnd, szTemp ) ;
        return ;
    }

    wsprintf( szTemp, _T("%stdescElem."), szLabel ) ;
    DumpTYPEDESC( hwnd, &lpadesc->tdescElem, szTemp ) ;

    wsprintf( szTemp, _T("%scDims = %u"),  szLabel, lpadesc->cDims ) ;
    ListBox_AddString( hwnd, szTemp ) ;
}

void DumpIDLDESC( HWND hwnd, LPIDLDESC lpidldesc, LPTSTR szLabel )
{
    TCHAR szTemp[128] ;

    wsprintf(szTemp, _T("%swIDLFlags = %s"), szLabel,
            g_rgszIDLFLAGS[lpidldesc->wIDLFlags] ) ;
    ListBox_AddString( hwnd, szTemp ) ;

#ifndef WIN32
    if (lpidldesc->bstrIDLInfo != NULL)
        wsprintf( szTemp, "%sbstrIDLInfo = %s", szLabel,
                    lpidldesc->bstrIDLInfo ) ;
    else
        wsprintf( szTemp, "%sbstrIDLInfo = NULL", szLabel ) ;
#else
    wsprintf( szTemp, _T("%sdwReserved = %#08lX"), szLabel,
              lpidldesc->dwReserved ) ;
#endif
    ListBox_AddString( hwnd, szTemp ) ;
}

void DumpELEMDESC( HWND hwnd, LPELEMDESC lpelemdesc, LPTSTR szLabel )
{
    TCHAR szTemp[128] ;

    wsprintf( szTemp, _T("%stdesc."), szLabel ) ;
    DumpTYPEDESC( hwnd, &lpelemdesc->tdesc, szTemp ) ;

    wsprintf( szTemp, _T("%sidldesc."), szLabel ) ;
    DumpIDLDESC( hwnd, &lpelemdesc->idldesc, szTemp ) ;
}

void DumpFUNCDESC( HWND hwnd, LPFUNCDESC lpfndesc, LPTSTR szLabel )
{
    TCHAR szTemp[128] ;

    // MEMBERID  memid ;
    wsprintf( szTemp, _T("%smemid = %#08lX"), szLabel, (DWORD)lpfndesc-
>memid ) ;
    ListBox_AddString( hwnd, szTemp ) ;
    // FUNCKIND  funckind ;
    wsprintf( szTemp, _T("%sfunckind = %s"), szLabel,
g_rgszFUNCKIND[lpfndesc->funckind] ) ;
    ListBox_AddString( hwnd, szTemp ) ;

    // INVOKEKIND   invkind ;
    if (lpfndesc->invkind == INVOKE_FUNC)
        wsprintf( szTemp, _T("%sinvkind = FUNC"), szLabel ) ;
    else if (lpfndesc->invkind == INVOKE_PROPERTYGET)
        wsprintf( szTemp, _T("%sinvkind = PROPERTYGET"), szLabel ) ;
    else if (lpfndesc->invkind == INVOKE_PROPERTYPUT)
        wsprintf( szTemp, _T("%sinvkind = PROPERTYPUT"), szLabel ) ;
    else if (lpfndesc->invkind == INVOKE_PROPERTYPUTREF)
        wsprintf( szTemp, _T("%sinvkind = PROPERTYPUTREF"), szLabel ) ;
    ListBox_AddString( hwnd, szTemp ) ;

    // CALLCONV callconv;
    wsprintf( szTemp, _T("%scallconv = %s"), szLabel,
g_rgszCALLCONV[lpfndesc->callconv] ) ;
    ListBox_AddString( hwnd, szTemp ) ;

    // SHORT cParams;
    wsprintf( szTemp, _T("%scParams = %d"), szLabel, lpfndesc->cParams ) ;
    ListBox_AddString( hwnd, szTemp ) ;

    // SHORT cParamsOpt;
    wsprintf( szTemp, _T("%scParamsOpt = %d"), szLabel, lpfndesc->cParamsOpt
) ;
    ListBox_AddString( hwnd, szTemp ) ;

    // SHORT oVft;
    wsprintf( szTemp, _T("%soVft = %d"), (LPTSTR)szLabel, lpfndesc->oVft ) ;
    ListBox_AddString( hwnd, szTemp ) ;

    // WORD wFuncFlags;
    wsprintf( szTemp, _T("%swFuncFlags = %u"), szLabel, lpfndesc->wFuncFlags
) ;
    ListBox_AddString( hwnd, szTemp ) ;

    // ELEMDESC elemdescFunc;
    wsprintf( szTemp, _T("%selemdescFunc."), szLabel ) ;
    DumpELEMDESC( hwnd, &lpfndesc->elemdescFunc, szTemp ) ;

    // ELEMDESC FAR* lprgelemdescParam;  /* array of parameter types */
    for (SHORT s = 0 ; s < lpfndesc->cParams ; s++)
    {
        wsprintf( szTemp, _T("%slprgelemdescParam[%u]."), szLabel, s ) ;
        DumpELEMDESC( hwnd, &lpfndesc->lprgelemdescParam[s], szTemp ) ;
    }

    // SHORT cScodes;
    wsprintf( szTemp, _T("%scScodes = %d"), szLabel, lpfndesc->cScodes ) ;
    ListBox_AddString( hwnd, szTemp ) ;
```

```
#if 1
// BUGBUG:  Run Ole2View, invoke IDispatch Calc #2, double click on it's
// IDispatch, close the viewer.   Invoke Word Basic, double click on it's
// IDispatch.  All calls to GetFuncDesc return a FUNCDESC where cScodes is
// some bogus # like 0x0778.   Note that if we invoke Word Basic right
// after Ole2View starts up this problem does not occur.

    // SCODE FAR* lprgscode;
    if (lpfndesc->cScodes > 0)
    {
        //AssertSz( lpfndesc->lprgscode, "lpfndesc->sCodes > 0 when it
shouldn't be" ) ;
        for (s = 0 ; s < lpfndesc->cScodes ; s++)
        {
            if (lpfndesc->lprgscode == NULL)
                wsprintf( szTemp, _T("%slprgscode = NULL (this is an
error)"), szLabel ) ;
            else
                wsprintf( szTemp, _T("%slprgscode[%u] = %s"), szLabel, s,
                        HRtoString( ResultFromScode( lpfndesc-
>lprgscode[s] ) ) ) ;
            ListBox_AddString( hwnd, szTemp ) ;
        }
    }

    //AssertSz( !(lpfndesc->cScodes < -1), "lpfndesc->sCodes < -1 when it
shouldn't be" ) ;

#endif
}

void DumpVARDESC( HWND hwnd, LPVARDESC lpvardesc, LPTSTR szLabel )
{
    TCHAR szTemp[128] ;

    // MEMBERID memid;
    wsprintf( szTemp, _T("%smemid = %#08lX"), szLabel, (DWORD)lpvardesc-
>memid ) ;
    ListBox_AddString( hwnd, szTemp ) ;

    // WORD wVarFlags;
    wsprintf( szTemp, _T("%swVarFlags = %s"), szLabel,
              lpvardesc->wVarFlags == VARFLAG_FREADONLY ? _T("READONLY") :
_T("NOT READONLY") ) ;
    ListBox_AddString( hwnd, szTemp ) ;

    // VARKIND varkind;
    wsprintf( szTemp, _T("%svarkind = %s"), szLabel,
g_rgszVARKIND[lpvardesc->varkind] ) ;
    ListBox_AddString( hwnd, szTemp ) ;

    // ELEMDESC elemdescVar;
    wsprintf( szTemp, _T("%selemdescVar."), szLabel ) ;
    DumpELEMDESC( hwnd, &lpvardesc->elemdescVar, szTemp ) ;
    // union { ULONG oInst;  VARIANT FAR* lpvarValue;  }
    if (lpvardesc->varkind == VAR_PERINSTANCE)
    {
        wsprintf( szTemp, _T("%soInst = %lu"), szLabel, lpvardesc->oInst ) ;
    }
    else if (lpvardesc->varkind == VAR_CONST)
    {
        // NEED:  This just prints out the type.  We can get the value
        // too.  Need to write at VARIANTtoString() function.
        //
        //
        VARIANT varValue ;
        VariantInit( &varValue ) ;
        HRESULT hr = VariantChangeType( &varValue, lpvardesc->lpvarValue, 0,
VT_BSTR ) ;
        if (FAILED(hr))
            wsprintf( szTemp, _T("%slpvarValue = %s = %s"), szLabel,
                        VTtoString( lpvardesc->lpvarValue->vt ), HRtoString(
hr ) ) ;
        else
        {
            wsprintf( szTemp, _T("%slpvarValue = %s = %s"),  szLabel,
                    VTtoString( lpvardesc->lpvarValue->vt ),
varValue.bstrVal ) ;
            SysFreeString( varValue.bstrVal ) ;
        }

        ListBox_AddString( hwnd, szTemp ) ;
    }
}

// end of idisp.cpp
```

IDISP_A.CPP   (DEFO2V Sample)

```
// ansiwrap.cpp
//
// Creates ANSI versions of all modules.  Any exported function get's "A"
// appended.
//
//

#ifdef UNICODE
#undef UNICODE
#endif
#ifdef _UNICODE
#undef _UNICODE
#endif

#include "precompa.h"
#include "idisp.cpp"
```

PRECOMP.H   (DEFO2V Sample)

```
#ifndef _PRECOMP_H_
#define _PRECOMP_H_

#define STRICT

// Disable "unrefereced formal parameter"
#pragma warning( disable : 4100 )

// Disable "Note: Creating new precompiled header"
#pragma warning( disable : 4699 )

// Disable "4706: assignment within conditional expression"
#pragma warning( disable : 4706 )

// Disable "4201: nonstandard...nameless struct/union"
#pragma warning( disable : 4201 )

// DIsable "C4514: unreferenced inline function has been removed
#pragma warning( disable : 4514 )

#ifdef WIN32
#define EXPORT
#else
#define EXPORT __export
#endif

#ifdef _UNICODE
#ifndef UNICODE
#define UNICODE         // UNICODE is used by Windows headers
#endif
#endif

#ifdef WIN32
    #ifndef WIN32_LEAN_AND_MEAN
    #define WIN32_LEAN_AND_MEAN
    #endif
    #ifndef INC_OLE2
    #define INC_OLE2
    #endif
#else
    #include <memory.h>
#endif



#include <windows.h>
#include <windowsx.h>
#include <ole2.h>

#ifdef WIN32
    #include <tchar.h>
#endif
```

```
#ifdef _UNICODE
#else
#ifndef WIN32
 #define LPTSTR      LPSTR
 #define LPCTSTR     LPCSTR
 #define _T(a_)      a_
 #define _T(a_)      a_
#endif
#endif

#ifndef WIN32
#include <ole2.h>
#include <olenls.h>
#include <ver.h>
#include <compobj.h>
#include <variant.h>
#include <dispatch.h>
#else
#include <oleauto.h>
#endif

#endif // _PRECOMP_H_
```

PRECOMPA.H (DEFO2V Sample)

```
// OLEANSI Precompiled header
//

#if defined(_UNICODE) || defined(UNICODE)
#error UNICODE and _UNICODE must not be defined for ANSIWRAP.CPP
#endif

#include "precomp.h"
```

RESOURCE.H (DEFO2V Sample)

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by DEFO2V.RC
//
#define IDC_IFACE_PROPS                 102
#define IDC_FORMATETC                   102
#define IDD_IDATAOBJDLG                 103
#define IDC_GETDATA                     103
#define IDD_IDISPDLG                    104
#define IDC_TYPEINFO                    1000
#define IDC_TYPEATTR                    1007
#define IDC_VARIABLES                   1008
#define IDC_FUNCTIONS                   1009
#define IDC_INFO                        1010
#define IDC_FUNCTIONS_LBL               1011
#define IDC_VARIABLES_LBL               1012
#define IDC_TYPEINFOCOUNT               1013
#define IDC_FUNCPROTO_LBL               1014
#define IDC_FUNCPROTO                   1015
#define IDC_FUNCVARDESC                 1016
#define IDC_FUNCVARDESC_LBL             1018
#define IDC_TYPEINFOINFO                1019
#define IDC_TOFILE                      1020
#define IDC_DOGETDATA                   1021
#define IDC_SETUPADVISE                 1022
#define IDC_ADVISEDATA                  1023
#define IDC_UPDATEDISPLAY               1024
#define IDC_PRIMEFIRST                  1025
#define IDC_CLEAROUTPUT                 1027

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        101
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1027
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

TOFILE.H (DEFO2V Sample)

```
#ifndef _TOFILE_H_
#define _TOFILE_H_

#if defined(WIN32) && !defined(_UNICODE)
    #define DoTypeInfoToFile    DoTypeInfoToFileA
    #define DoTypeLibToFile     DoTypeLibToFileA
#endif

BOOL DoTypeInfoToFile( HWND hwnd, LPTYPEINFO lpTypeInfo )  ;
BOOL DoTypeLibToFile( HWND hwnd, LPTYPELIB lpTypeLib ) ;

#endif // _TOFILE_H_
```

TOFILE.CPP   (DEFO2V Sample)

```
// tofile.cpp
//
// Dumps a TypeLib or a TypeInfo to a file
//

#include "precomp.h"
#include "defo2v.h"
#include "idisp.h"
#include "tofile.h"
#include <stdlib.h>
#include <commdlg.h>
#include <math.h>

#if defined(WIN32) && !defined(_UNICODE)
    #pragma message("Building ANSI version of " __FILE__)
#endif

static BOOL StringToFile( HFILE hf, LPCTSTR psz )
{
    int cc = lstrlen(psz) ;
#ifndef WIN32
    return _lwrite( hf, psz, cc ) != HFILE_ERROR ;
#else
    DWORD   dw ;

    #ifdef _UNICODE
    char*   sz = new char[cc+1] ;
    cc = WideCharToMultiByte( CP_ACP, 0, psz, cc, sz, cc, NULL, NULL ) ;
    BOOL f = WriteFile((HANDLE)hf, sz, cc, &dw, NULL ) ;
    delete [] sz ;
    return f ;
    #else
    return WriteFile((HANDLE)hf, psz, cc, &dw, NULL ) ;
    #endif
#endif
}

#ifdef WIN32
#define CLOSEFILE(h)      CloseHandle((HANDLE)h)
#else
#define CLOSEFILE(h)      _lclose(h)
#endif


static OLECHAR szNotAvail[] = OLESTR("(not available)") ;

#define BSTR_PRT(bstr) (LPOLESTR)((bstr != NULL && SysStringLen(bstr)) ?
(LPOLESTR)bstr : (LPOLESTR)szNotAvail)

static BOOL TypeInfoToFile( HFILE  hfile, LPTYPEINFO lpTypeInfo ) ;
static LPTSTR BuildParam( LPTSTR sz, LPELEMDESC lpelemdesc, LPTSTR lpszName,
BOOL fOptional ) ;

static HFILE GetFileToSave( HWND hwnd, LPTSTR szTitle )
{
    OPENFILENAME ofn;
    TCHAR szFile[256], szFileTitle[256];
    UINT  i, cbString;
    TCHAR  chReplace;    /* string separator for szFilter */
    HFILE hf = HFILE_ERROR ;
    static TCHAR  szFilter[] = _T("Text Files (*.txt)|*.txt|") ;

    cbString = lstrlen( szFilter ) ;
    chReplace = szFilter[cbString - 1]; /* retrieve wildcard */
    for (i = 0; szFilter[i] != _T('\0'); i++)
    {
        if (szFilter[i] == chReplace)
           szFilter[i] = _T('\0');
    }

    /* Set all structure members to zero. */
    _fmemset(&ofn, 0, sizeof(OPENFILENAME));

    /* Initialize the OPENFILENAME members. */
    szFile[0] = '\0';

    ofn.lStructSize = sizeof(OPENFILENAME);
    ofn.hwndOwner = hwnd;
    ofn.lpstrFilter = szFilter;
    ofn.lpstrFile= szFile;
    ofn.nMaxFile = sizeof(szFile);
    ofn.lpstrFileTitle = szFileTitle;
    ofn.lpstrTitle = szTitle ;

    ofn.nMaxFileTitle = sizeof(szFileTitle);
    ofn.lpstrInitialDir = NULL ;
    ofn.Flags =     OFN_SHOWHELP |
                    OFN_OVERWRITEPROMPT |
                    OFN_HIDEREADONLY ;

    if (GetSaveFileName(&ofn))
    {
        #ifndef WIN32
        OFSTRUCT of ;
        hf = OpenFile( ofn.lpstrFile, &of, OF_CREATE ) ;
        #else
        hf = (HFILE)CreateFile( ofn.lpstrFile, GENERIC_WRITE, 0, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL ) ;
        #endif
    }
    return hf ;

}

BOOL DoTypeInfoToFile( HWND hwnd, LPTYPEINFO lpTypeInfo )
```

```
{
    HFILE hf = GetFileToSave( hwnd, _T("Save TypeInfo to File") ) ;
    if (hf == HFILE_ERROR)
        return FALSE ;

    HCURSOR hcur = SetCursor(LoadCursor( NULL, IDC_WAIT )) ;
    TypeInfoToFile( hf, lpTypeInfo ) ;
    SetCursor( hcur ) ;
    CLOSEFILE( hf ) ;

    return TRUE ;
}

BOOL DoTypeLibToFile( HWND hwnd, LPTYPELIB lpTypeLib )
{
    HRESULT     hr ;
    LPTYPEINFO  lpTypeInfo ;
    HFILE       hf ;

    hf = GetFileToSave( hwnd, _T("Save TypeLib to File") ) ;
    #ifdef WIN32
    if (hf == 0)
        return FALSE ;
    #else
    if (hf == HFILE_ERROR)
        return FALSE ;
    #endif

    HCURSOR hcur = SetCursor(LoadCursor( NULL, IDC_WAIT )) ;

    TLIBATTR FAR* pTlibAttr ;
    lpTypeLib->GetLibAttr(&pTlibAttr ) ;

    BSTR           bstrName = NULL ;
    BSTR           bstrDoc = NULL ;
    BSTR           bstrHelp = NULL ;
    DWORD           dwHelpID ;
    hr = lpTypeLib->GetDocumentation( MEMBERID_NIL, &bstrName, &bstrDoc,
&dwHelpID, &bstrHelp ) ;
    if (SUCCEEDED(hr))
    {

        TCHAR           szBuf[256] ;

        wsprintf( szBuf,
_T("'===============================================================\r\n") )
;
        StringToFile( hf, szBuf ) ;
#if defined (WIN32) && !defined (OLE2ANSI)
        wsprintf( szBuf,    _T("' Type Library: %ws, Library Version %d.
%03d\r\n"),
                            BSTR_PRT(bstrName), pTlibAttr->wMajorVerNum,
pTlibAttr->wMinorVerNum ) ;
#else
```

```
        wsprintf( szBuf,    _T("' Type Library: %s, Library Version %d.
%03d\r\n"),
                            BSTR_PRT(bstrName), pTlibAttr->wMajorVerNum,
pTlibAttr->wMinorVerNum ) ;
#endif
        StringToFile( hf, szBuf ) ;

        wsprintf( szBuf,    _T("' GUID: {%08lX-%04X-%04X-%02X%02X-
%02X%02X%02X%02X%02X%02X}\r\n"),
                            pTlibAttr->guid.Data1, pTlibAttr->guid.Data2,
pTlibAttr->guid.Data3,
                            pTlibAttr->guid.Data4[0], pTlibAttr-
>guid.Data4[1],
                            pTlibAttr->guid.Data4[2], pTlibAttr-
>guid.Data4[3],
                            pTlibAttr->guid.Data4[4], pTlibAttr-
>guid.Data4[5],
                            pTlibAttr->guid.Data4[6], pTlibAttr-
>guid.Data4[7]) ;
        StringToFile( hf, szBuf ) ;

        wsprintf( szBuf,    _T("' LCID: %#08X\r\n"), (DWORD)pTlibAttr-
>lcid ) ;
        StringToFile( hf, szBuf ) ;

#if defined (WIN32) && !defined (OLE2ANSI)
        wsprintf( szBuf,    _T("' Documentation: %ws\r\n"),
BSTR_PRT(bstrDoc) ) ;
#else
        wsprintf( szBuf,    _T("' Documentation: %s\r\n"), BSTR_PRT(bstrDoc)
) ;
#endif
        StringToFile( hf, szBuf ) ;

#if defined (WIN32) && !defined (OLE2ANSI)
        wsprintf( szBuf,    _T("' Help: %ws (Help ID: %#08lX)\r\n"),
BSTR_PRT(bstrHelp), (DWORD)dwHelpID ) ;
#else
        wsprintf( szBuf,    _T("' Help: %s (Help ID: %#08lX)\r\n"),
BSTR_PRT(bstrHelp), (DWORD)dwHelpID ) ;
#endif
        StringToFile( hf, szBuf ) ;

        wsprintf( szBuf,
_T("'=================================================================\r\n\r\n
") ) ;
        StringToFile( hf, szBuf ) ;

        SysFreeString( bstrName ) ;
        SysFreeString( bstrDoc ) ;
        SysFreeString( bstrHelp ) ;

        lpTypeLib->ReleaseTLibAttr( pTlibAttr ) ;
    }
    else
    {
        lpTypeLib->ReleaseTLibAttr( pTlibAttr ) ;
        TCHAR szBuf[128] ;
        wsprintf( szBuf, _T("GetDocumentation for TypeInfo FAILED: %s"),
HRtoString( hr ) ) ;
        StringToFile( hf, szBuf ) ;
        return FALSE ;
    }

    UINT uiTypeInfoCount = lpTypeLib->GetTypeInfoCount() ;
    for (UINT n = 0 ; n < uiTypeInfoCount ; n++)
    {


        hr = lpTypeLib->GetTypeInfo( n, &lpTypeInfo ) ;
        if (SUCCEEDED(hr))
        {
            TypeInfoToFile( hf, lpTypeInfo ) ;
            lpTypeInfo->Release() ;
        }
        else
        {
            TCHAR szBuf[128] ;
            wsprintf( szBuf, _T("ERROR: Could not get TypeInfo %d: %s\r\n"),
n, (LPTSTR)HRtoString(hr) ) ;
            StringToFile( hf, szBuf) ;
        }
    }

    SetCursor( hcur ) ;
    CLOSEFILE( hf ) ;
    return TRUE ;
}

// Given a pointer to a TypeInfo and a handle to an open file, write
// all functions, variables, and constants.   The file is written in
// a form such that it looks like a VBA "include" file.
//
// A header is written first:
//
// '
// ' Type Information for the <wordbasic> TypeInfo
// ' Documentation: <docstring>
// ' Help for this TypeInfo can be found in <helpfile>
// ' The help ID for this TypeInfo is <helpid>
//
// Functions appear in the following form:
//
// '
// ' <funcname>
// ' Documentation: <docstring>
// ' Help for this function can be found in <helpfile>
// ' The help ID for this function is <helpid>
// Declare Function <funcname>(<arglist>) As <returntype>
//
```

```
// Subroutines (functions without return types) appear as:
// '
// ' <funcname>
// ' Documentation: <docstring>
// ' Help for this subroutine can be found in <helpfile>
// ' The help ID for this subroutine is <helpid>
// Declare Sub <funcname>(<arglist>)
//
// Properties/Variables appear as:
//
// '
// ' <varname>
// ' Documentation: <docstring>
// ' Help for this property can be found in <helpfile>
// ' The help ID for this property is <helpid>
// Dim <varname> As <vartype>
//
// Constants appear as:
//
// '
// ' <varname>
// ' Documentation: <docstring>
// ' Help for this property can be found in <helpfile>
// ' The help ID for this property is <helpid>
// Const <varname> As <vartype> = <value>
//
static BOOL TypeInfoToFile( HFILE  hfile, LPTYPEINFO lpTypeInfo )
{
    TCHAR szBuf[512] ;

//    Assert( hfile != HFILE_ERROR ) ;
//    Assert( lpTypeInfo ) ;

    #define MAX_NAMES   128

    HRESULT         hr ;
    BSTR            bstrName = NULL ;
    BSTR            bstrDoc = NULL ;
    BSTR            bstrHelp = NULL ;
    BSTR            rgbstrNames[MAX_NAMES] ;
    DWORD           dwHelpID ;
    int             cNames ;
    WORD            cFuncs, cVars ;
    LPFUNCDESC      pFuncDesc ;
    WORD            iIndex ;

    LPTYPEATTR  pTA = NULL ;

    hr = lpTypeInfo->GetTypeAttr( &pTA ) ;
    if (FAILED( hr ))
    {
        wsprintf( szBuf, _T("GetTypeAttr FAILED: %s"),
(LPTSTR)HRtoString( hr ) ) ;
        StringToFile( hfile, szBuf ) ;
        return FALSE ;
    }

    hr = lpTypeInfo->GetDocumentation( MEMBERID_NIL, &bstrName, &bstrDoc,
&dwHelpID, &bstrHelp ) ;
    if (SUCCEEDED(hr))
    {
        wsprintf( szBuf,
_T("'===============================================================\r\n") )
;
        StringToFile( hfile, szBuf ) ;
#if defined (WIN32) && !defined (OLE2ANSI)
        wsprintf( szBuf,    _T("' Type Info: %ws, TypeInfo Version %d.
%03d\r\n"),
                            BSTR_PRT(bstrName), pTA->wMajorVerNum, pTA-
>wMinorVerNum ) ;
#else
        wsprintf( szBuf,    _T("' Type Info: %s, TypeInfo Version %d.
%03d\r\n"),
                            BSTR_PRT(bstrName), pTA->wMajorVerNum, pTA-
>wMinorVerNum ) ;
#endif
        StringToFile( hfile, szBuf ) ;

        wsprintf( szBuf,    _T("' GUID: {%08lX-%04X-%04X-%02X%02X-
%02X%02X%02X%02X%02X%02X}\r\n"),
                            pTA->guid.Data1, pTA->guid.Data2, pTA-
>guid.Data3,
                            pTA->guid.Data4[0], pTA->guid.Data4[1],
                            pTA->guid.Data4[2], pTA->guid.Data4[3],
                            pTA->guid.Data4[4], pTA->guid.Data4[5],
                            pTA->guid.Data4[6], pTA->guid.Data4[7]);
        StringToFile( hfile, szBuf ) ;

        wsprintf( szBuf,    _T("' LCID: %#08X\r\n"), (DWORD)pTA->lcid ) ;
        StringToFile( hfile, szBuf ) ;

        wsprintf( szBuf,    _T("' TypeKind: %s\r\n"),
(LPTSTR)TYPEKINDtoString(pTA->typekind) ) ;
        StringToFile( hfile, szBuf ) ;

        if (bstrDoc && SysStringLen(bstrDoc))
        {
#if defined (WIN32) && !defined (OLE2ANSI)
            wsprintf( szBuf,    _T("' Documentation: %ws\r\n"),
(LPOLESTR)bstrDoc ) ;
#else
            wsprintf( szBuf,    _T("' Documentation: %s\r\n"),
(LPTSTR)bstrDoc ) ;
#endif
            StringToFile( hfile, szBuf ) ;
        }

        if (bstrHelp && SysStringLen(bstrHelp))
        {
#if defined (WIN32) && !defined (OLE2ANSI)
```

```
            wsprintf( szBuf,    _T("' Help: %ws (Help ID: %#08lX)\r\n"),
(LPOLESTR)bstrHelp,  (DWORD)dwHelpID ) ;
#else
            wsprintf( szBuf,    _T("' Help: %s (Help ID: %#08lX)\r\n"),
(LPTSTR)bstrHelp,  (DWORD)dwHelpID ) ;
#endif
            StringToFile( hfile, szBuf ) ;
        }

        wsprintf( szBuf,
_T("'---------------------------------------------------------------
\r\n\r\n") ) ;
        StringToFile( hfile, szBuf ) ;

        SysFreeString( bstrName ) ;
        SysFreeString( bstrDoc ) ;
        SysFreeString( bstrHelp ) ;
    }
    else
    {
        wsprintf( szBuf, _T("GetDocumentation for TypeInfo FAILED: %s"),
(LPTSTR)HRtoString( hr ) ) ;
        StringToFile( hfile, szBuf ) ;
        return FALSE ;
    }

    cVars = pTA->cVars ;
    cFuncs = pTA->cFuncs ;
    lpTypeInfo->ReleaseTypeAttr( pTA ) ;

    int      cParams ;
    int      cParamsOpt ;
    TCHAR    szFunc[128] ;   // Function/Sub name
    LPTSTR   szParams ;
    LPTSTR   lpsz ;

    //
    // Enumerate through all FUNCDESCS
    for ( iIndex = 0 ; iIndex < cFuncs ; iIndex++)
    {
        hr = lpTypeInfo->GetFuncDesc( iIndex, &pFuncDesc ) ;
        if (FAILED(hr))
        {
            wsprintf( szBuf, _T("GetFuncDesc FAILED for function #%u (%s)
\r\n"), iIndex, (LPTSTR)HRtoString( hr ) ) ;
            StringToFile( hfile, szBuf ) ;
            continue ;
        }

        hr = lpTypeInfo->GetDocumentation( pFuncDesc->memid, &bstrName,
&bstrDoc, &dwHelpID, &bstrHelp ) ;
        if (SUCCEEDED(hr))
        {
#if defined (WIN32) && !defined (OLE2ANSI)
```

```
            wsprintf( szBuf, _T("' Function: %ws\r\n"), BSTR_PRT(bstrName) )
;
#else
            wsprintf( szBuf, _T("' Function: %s\r\n"),
BSTR_PRT(bstrName) ) ;
#endif
            StringToFile( hfile, szBuf ) ;

            if (bstrDoc && SysStringLen(bstrDoc))
            {
#if defined (WIN32) && !defined (OLE2ANSI)
                wsprintf( szBuf, _T("' Documentation: %ws\r\n"),
(LPOLESTR)bstrDoc ) ;
#else
                wsprintf( szBuf, _T("' Documentation: %s\r\n"),
(LPTSTR)bstrDoc ) ;
#endif
                StringToFile( hfile, szBuf ) ;
            }

            if (bstrHelp && SysStringLen(bstrHelp))
            {
#if defined (WIN32) && !defined (OLE2ANSI)
                wsprintf( szBuf, _T("' Help: %ws (Help ID: %#08lX)\r\n"),
(LPOLESTR)bstrHelp, (DWORD)dwHelpID ) ;
#else
                wsprintf( szBuf, _T("' Help: %s (Help ID: %#08lX)\r\n"),
(LPTSTR)bstrHelp, (DWORD)dwHelpID ) ;
#endif
                StringToFile( hfile, szBuf ) ;
            }

            lstrcpy( szBuf, _T("' \r\n") ) ;
            StringToFile( hfile, szBuf ) ;

            SysFreeString( bstrName ) ;
            SysFreeString( bstrDoc ) ;
            SysFreeString( bstrHelp ) ;

        }

        cParams = pFuncDesc->cParams ;
        cParamsOpt = abs(pFuncDesc->cParamsOpt) ;

        // Get the names of the function and it's parameters into
rgbstrNames.
        // cNames gets the number of parameters + 1.
        //
        hr = lpTypeInfo->GetNames( pFuncDesc->memid, rgbstrNames, MAX_NAMES,
(UINT FAR*)&cNames );
        if (SUCCEEDED( hr ))
        {
            // rgbstrNames[0] is the name of the function
            if (cNames > 0)
            {
```

```
#if defined(WIN32) && !defined(UNICODE)
                wcstombs(szFunc, rgbstrNames[0], wcslen(rgbstrNames[0]) +
1);
#else
                lstrcpy( szFunc, rgbstrNames[0] ) ;
#endif
                SysFreeString( rgbstrNames[0] ) ;

            }

            // Allocate a string buffer that should be able to hold
            // all of our parameter types and names.
            //
            if ((szParams = (LPTSTR) GlobalAllocPtr( GPTR,
max(cNames,cParams) * (64) )) == NULL )
            {
                MessageBox( NULL, _T("GlobalAlloc Failed!"), _T("Yikes!"),
MB_OK ) ;
                wsprintf( szBuf, _T("GlobalAlloc failed!\r\n") ) ;
                StringToFile( hfile, szBuf ) ;
                return FALSE ;
            }

            // For each parameter get the type and name.
            // The "max(cNames-1,cParams)" should handle the case
            // where a function has optional parameters (i.e. cParamsOpt is
            // non-zero).
            //
            lstrcpy( szParams, _T("(") ) ;

            // For each required parameter
            //
            for ( int n = 0 ; n < cParams - cParamsOpt ; n++ )
            {
                if (n+1 < cNames)
                {
#if defined(WIN32) && !defined(UNICODE)
                    lpsz = new char [wcslen(rgbstrNames[n+1]) + 1];
                    wcstombs(lpsz, rgbstrNames[n+1],
wcslen(rgbstrNames[n+1]) + 1);
#else
                    lpsz = rgbstrNames[n+1] ;
#endif
                }
                else
                    lpsz = NULL ;
                BuildParam( szParams+lstrlen(szParams),
                            &pFuncDesc->lprgelemdescParam[n], lpsz , FALSE )
;
#if defined(WIN32) && !defined(UNICODE)
                delete [] lpsz;
#endif
                if (n+1 < cNames)
                    SysFreeString( rgbstrNames[n+1] ) ;
                // If it's the last one then no comma
                if (n + 1 != max(cNames-1,cParams))
                    lstrcat( szParams, _T(", ") ) ;
            }

            // For each optional parameter
            //
            for (n = cParams - cParamsOpt ; n < cParams ; n++)
            {
                if (n+1 < cNames)
                {
#if defined(WIN32) && !defined(UNICODE)
                    lpsz = new char [wcslen(rgbstrNames[n+1]) + 1];
                    wcstombs(lpsz, rgbstrNames[n+1],
wcslen(rgbstrNames[n+1]) + 1);
#else
                    lpsz = rgbstrNames[n+1] ;
#endif
                }
                else
                    lpsz = NULL ;
                BuildParam( szParams+lstrlen(szParams),
                            &pFuncDesc->lprgelemdescParam[n], lpsz , FALSE )
;
#if defined(WIN32) && !defined(UNICODE)
                delete [] lpsz;
#endif
                if (n+1 < cNames)
                    SysFreeString( rgbstrNames[n+1] ) ;


                // If it's the last one then no comma
                if (n + 1 != max(cNames-1,cParams))
                    lstrcat( szParams,  _T(", ") ) ;
            }

            lstrcat( szParams , _T(")")  );

            // Is it a function or sub?
            //
            if (pFuncDesc->elemdescFunc.tdesc.vt == VT_EMPTY ||
                pFuncDesc->elemdescFunc.tdesc.vt == VT_NULL ||
                pFuncDesc->elemdescFunc.tdesc.vt == VT_VOID)
            {
                // Declare Sub <subname>(<arglist>)
                wsprintf( szBuf, _T("Declare Sub %s "), (LPTSTR)szFunc ) ;
                StringToFile( hfile, szBuf) ;
                StringToFile( hfile, szParams) ;
                StringToFile( hfile, _T("\r\n\r\n") ) ;
            }
            else
            {
                // Declare Function <funcname>(<arglist>) As <returntype>
                wsprintf( szBuf, _T("Declare Function %s "),
(LPTSTR)szFunc ) ;
                StringToFile( hfile, szBuf) ;
                StringToFile( hfile, szParams) ;
                wsprintf( szBuf, _T(" As %s\r\n\r\n"),
(LPTSTR)VTtoString2( pFuncDesc->elemdescFunc.tdesc.vt ) ) ;
                StringToFile( hfile, szBuf) ;
            }

            GlobalFreePtr( szParams ) ;
        }
        else
        {
            wsprintf( szBuf, _T("GetNames (%lu) FAILED: %s"), pFuncDesc-
>memid, (LPTSTR)HRtoString( hr ) ) ;
            StringToFile( hfile, szBuf ) ;
        }
        lpTypeInfo->ReleaseFuncDesc( pFuncDesc ) ;
    }

    // '
    // ' <varname>
    // ' Documentation: <docstring>
    // ' Help for this property can be found in <helpfile>
    // ' The help ID for this property is <helpid>
    // Dim <varname> As <vartype>
    // '
    // ' <varname>
    // ' Documentation: <docstring>
    // ' Help for this property can be found in <helpfile>
    // ' The help ID for this property is <helpid>
    // Const <varname> As <vartype> = <value>
    LPVARDESC pVarDesc ;
    for (iIndex = 0 ; iIndex < cVars ; iIndex++)
    {
        hr = lpTypeInfo->GetVarDesc( iIndex, &pVarDesc ) ;
        if (FAILED(hr))
        {
            wsprintf( szBuf, _T("GetVarDesc FAILED for variable #%u (%s)
\r\n"), iIndex, (LPTSTR)HRtoString( hr ) ) ;
            StringToFile( hfile, szBuf ) ;
            continue ;
        }
        else
        {
            hr = lpTypeInfo->GetDocumentation( pVarDesc->memid, &bstrName,
&bstrDoc, &dwHelpID, &bstrHelp ) ;
            if (SUCCEEDED(hr))
            {
#if defined (WIN32) && !defined (OLE2ANSI)
                wsprintf( szBuf,     _T("' Variable/Constant: %ws\r\n"),
BSTR_PRT(bstrName) ) ;
#else
                wsprintf( szBuf,     _T("' Variable/Constant: %s\r\n"),
BSTR_PRT(bstrName) ) ;
```

```
#endif
                StringToFile( hfile, szBuf ) ;

                if (bstrDoc && SysStringLen(bstrDoc))
                {
#if defined (WIN32) && !defined (OLE2ANSI)
                    wsprintf( szBuf, _T("' Documentation: %ws\r\n"),
(LPOLESTR)bstrDoc ) ;
#else
                    wsprintf( szBuf, _T("' Documentation: %s\r\n"),
(LPTSTR)bstrDoc ) ;
#endif
                    StringToFile( hfile, szBuf ) ;
                }

                if (bstrHelp && SysStringLen(bstrHelp))
                {
#if defined (WIN32) && !defined (OLE2ANSI)
                    wsprintf( szBuf, _T("' Help: %ws (Help ID: %#08lX)
\r\n"), (LPOLESTR)bstrHelp, (DWORD)dwHelpID ) ;
#else
                    wsprintf( szBuf, _T("' Help: %s (Help ID: %#08lX)\r\n"),
(LPTSTR)bstrHelp, (DWORD)dwHelpID ) ;
#endif
                    StringToFile( hfile, szBuf ) ;
                }
                lstrcpy( szBuf, _T("' \r\n") ) ;
                StringToFile( hfile, szBuf ) ;
                SysFreeString( bstrName ) ;
                SysFreeString( bstrDoc ) ;
                SysFreeString( bstrHelp ) ;

            }
            hr = lpTypeInfo->GetNames( pVarDesc->memid, rgbstrNames, 1,
(UINT FAR*)&cNames );
            if (SUCCEEDED( hr ))
            {

                if (pVarDesc->varkind == VAR_CONST)
                {
                    // Const constname [As type] = expression
#if defined (WIN32) && !defined (OLE2ANSI)
                    wsprintf( szBuf, _T("Const %ws As %s = "),
                                        (LPOLESTR)rgbstrNames[0],
                                        (LPTSTR)TYPEDESCtoString( &pVarDesc-
>elemdescVar.tdesc )
                                     ) ;
#else
                    wsprintf( szBuf, _T("Const %s As %s = "),
                                        (LPTSTR)rgbstrNames[0],
                                        (LPTSTR)TYPEDESCtoString( &pVarDesc-
>elemdescVar.tdesc )
                                     ) ;
#endif
                    VARIANT varValue ;
                    VariantInit( &varValue ) ;
                    hr = VariantChangeType( &varValue, pVarDesc->lpvarValue,
0, VT_BSTR ) ;
                    if (FAILED(hr))
                        lstrcat( szBuf, (LPTSTR)HRtoString( hr ) ) ;
                    else
                    {
#if defined (WIN32) && !defined (UNICODE)
                        wcstombs(&(szBuf[strlen(szBuf)+1]),
varValue.bstrVal, wcslen(varValue.bstrVal)+1);
#else
                        lstrcat( szBuf, varValue.bstrVal ) ;
#endif
                        SysFreeString( varValue.bstrVal ) ;

                    }
                }
                else
                {
                    // Dim varname[([subscripts])][As type]
#if defined (WIN32) && !defined (OLE2ANSI)
                    wsprintf( szBuf, _T("Dim %ws "), (LPOLESTR)
(rgbstrNames[0] ? rgbstrNames[0]: OLESTR("(name less)")) ) ;
#else
                    wsprintf( szBuf, _T("Dim %s "), (LPTSTR)
(rgbstrNames[0] ? rgbstrNames[0]: _T("(name less)")) ) ;
#endif
                    if ((pVarDesc->elemdescVar.tdesc.vt & 0x0FFF) == VT_PTR
&&
                        (pVarDesc->elemdescVar.tdesc.lptdesc->vt & 0x0FFF)
== VT_SAFEARRAY)
                    {
                        lstrcat( szBuf, _T("() As ") ) ;
                        lstrcat( szBuf, VTtoString2( pVarDesc-
>elemdescVar.tdesc.vt ) ) ;
                    }
                    else
                    {
                        lstrcat( szBuf, _T("As ") ) ;
                        lstrcat( szBuf, TYPEDESCtoString( &pVarDesc-
>elemdescVar.tdesc ) ) ;
                    }
                }
                SysFreeString( rgbstrNames[0] ) ;

                lstrcat( szBuf, _T("\r\n\r\n") ) ;
                StringToFile( hfile, szBuf ) ;
            }
            else
            {
                wsprintf( szBuf, _T("GetNames (%lu) FAILED: %s\r\n"),
pVarDesc->memid, (LPTSTR)HRtoString( hr ) ) ;
                StringToFile( hfile, szBuf ) ;
            }
            lpTypeInfo->ReleaseVarDesc( pVarDesc ) ;
        }
    }

    return TRUE ;
}

static LPTSTR BuildParam( LPTSTR sz, LPELEMDESC lpelemdesc, LPTSTR lpszName,
BOOL fOptional )
{
    // [Optional][ByVal|ByRef][ParamArray] varname [()] As type

    if ((lpelemdesc->tdesc.vt & 0x0FFF) == VT_PTR &&
        (lpelemdesc->tdesc.lptdesc->vt & 0x0FFF) == VT_SAFEARRAY)
    {
        lstrcat( sz, _T("ParamArray ") ) ;
    }
    else
    {
        if (fOptional)
            lstrcpy( sz, _T("Optional ") ) ;

        if ((lpelemdesc->tdesc.vt & 0x0FFF) == VT_PTR)
            lstrcat( sz, _T("ByRef ") ) ;
        else
            lstrcat( sz, _T("ByVal ") ) ;
    }

    if (lpszName != NULL)
        lstrcat( sz, lpszName ) ;

    if (lpszName != NULL)
        lstrcat( sz, _T(" As ") ) ;
    else
        lstrcat( sz, _T(" ") ) ;

    if ((lpelemdesc->tdesc.vt & 0x0FFF) == VT_PTR)
        lstrcat( sz, _T("Variant") ) ;
    else
    {
        lstrcat( sz, TYPEDESCtoString( &lpelemdesc->tdesc ) ) ;
    }
    return sz ;
}
```

TOFILE_A.CPP   (DEFO2V Sample)

```
// ansiwrap.cpp
//
// Creates ANSI versions of all modules.  Any exported function get's "A"
// appended.
//
//

#ifdef UNICODE
#undef UNICODE
#endif
#ifdef _UNICODE
#undef _UNICODE
#endif

#include "precompa.h"
#include "tofile.cpp"
```

UTIL.H (DEFO2V Sample)

```
// util.h
//
// Header file for utiltiy functions used by DEFO2V
//
// Copyright (c) 1993 Microsoft Corporation, All Rights Reserved.
//
// Charlie Kindel, Program Manager
// Microsoft Vertical Developer Relations
// August 6, 1993
//
// InterNet   :  ckindel@microsoft.com
// CompuServe :  >INTERNET:ckindel@microsoft.com
//
// Revisions:
//  August 6, 1993  cek     First implementation.
//
#ifndef _UTIL_H_
#define _UTIL_H_

#if defined(WIN32) && !defined(_UNICODE)
    #define ReallyCreateFont ReallyCreateFontA
    #define GetTextMetricsCorrectly GetTextMetricsCorrectlyA
    #define ColumnTextOut ColumnTextOutA
    #define HRtoString HRtoStringA
    #define VTtoString VTtoStringA
    #define VTtoString2 VTtoString2A
    #define TYPEDESCtoString TYPEDESCtoStringA
    #define fnEnumReallyEx fnEnumReallyExA
    #define DlgCenter  DlgCenterA
#endif

/// Utilities
#define RCF_NORMAL      0x0000
#define RCF_ITALIC      0x0001
#define RCF_UNDERLINE   0x0002
#define RCF_STRIKEOUT   0x0004
#define RCF_BOLD        0x0008
#define RCF_NODEFAULT   0x0010
#define RCF_TRUETYPEONLY 0x0011

HFONT WINAPI ReallyCreateFont( HDC hDC, LPTSTR lpszFace, LPTSTR lpszStyle,
UINT nPointSize, UINT uiFlags ) ;
void WINAPI DlgCenter( HWND hwndCenter, HWND hwndWithin, BOOL fClient ) ;
int WINAPI GetTextMetricsCorrectly( HDC hDC, LPTEXTMETRIC lpTM ) ;

typedef struct FAR tagCOLUMNSTRUCT
{
   int   nLeft ;       // starting x position of the column
   int   nRight ;      // ending x position of the column
   UINT  uiFlags ;      // format flags

} COLUMNSTRUCT, *PCOLUMNSTRUCT, FAR *LPCOLUMNSTRUCT ;
```

```
void WINAPI ColumnTextOut( HDC hdc, int nX, int nY, LPTSTR lpszIN, int
cColumns, LPCOLUMNSTRUCT rgColumns ) ;
LPTSTR HRtoString( HRESULT hr ) ;
LPTSTR VTtoString( VARTYPE vt ) ;
LPTSTR VTtoString2( VARTYPE vt ) ;
LPTSTR TYPEDESCtoString( TYPEDESC FAR* lptdesc ) ;




#endif
```

UTIL.CPP   (DEFO2V Sample)

```
// utixl.cpp
//
// Implementation file for utility functions.
//
// Copyright (c) 1993 Microsoft Corporation, All Rights Reserved.
//
// Charlie Kindel, Program Manager
// Microsoft Vertical Developer Relations
// August 7, 1993
//
// InterNet   :  ckindel@microsoft.com
// CompuServe :  >INTERNET:ckindel@microsoft.com
//
// Revisions:
//  August 7, 1993  cek     First implementation.
//
#include "precomp.h"
#include "defo2v.h"
#include "util.h"

#if defined(WIN32) && !defined(_UNICODE)
    #pragma message("Building ANSI version of " __FILE__)
#endif

/////
// Utility functions
//

static TCHAR szBold[]        = _T("Bold") ;
static TCHAR szItalic[]      = _T("Italic") ;
static TCHAR szBoldItalic[]  = _T("Bold Italic") ;
static TCHAR szRegular[]     = _T("Regular") ;

extern "C"
int EXPORT WINAPI
    fnEnumReallyEx( LPENUMLOGFONT lpLogFont,
                LPNEWTEXTMETRIC lpTextMetrics,
                int nFontType,
                LPENUMLOGFONT lpELF ) ;

 /****************************************************************
 *  HFONT WINAPI
 *    ReallyCreateFontEx( HDC hDC, LPTSTR lpszFace,
 *                        LPTSTR lpszStyle, int nPointSize, UINT uiFlags )
 *
 *  Description:
 *
 *    Creates a font based on facename, pointsize, and style.
 *
 *    Uses 3.1 API's to correctly select TT fonts.
 *
 *    HDC   hDC          the target DC (optional)
 *
 *    LPTSTR lpszFace    pointer the facename (required)
 *
 *    LPTSTR lpszStyle   pointer to the style (like "Demibold Italic")
 *                      (optional).
 *
 *    int nPointSize  size in points (required)
 *
 *    UINT  uiFlags      Flags, same as for ReallyCreateFont() except
 *                      RCF_BOLD and RCF_ITALIC are ignored if lpszStyle
 *                      is not NULL.
 *
 *  Comments:
 *
 ****************************************************************/

HFONT WINAPI
   ReallyCreateFont( HDC hDC, LPTSTR lpszFace, LPTSTR lpszStyle, UINT
nPointSize, UINT uiFlags )
{
   ENUMLOGFONT  elf ;
   FONTENUMPROC lpfn ;
   HDC          hdcCur ;

   lpfn = (FONTENUMPROC)fnEnumReallyEx ;

   // if the DC wasn't specified then use the display...
   //
   if (!hDC)
   {
      if (!(hdcCur = GetDC( NULL )))
         return FALSE ;
   }
   else
      hdcCur = hDC ;

   if (hdcCur)
   {
      /*
       * EnumFontFamilies takes the family name as the second param.
       * For example the family name might be "Lucida Sans" and the
       * style might be "Demibold Roman".
       *
       * The last parameter is app. defined.  In our case we pass a
       * structure that has a LOGFONT and a TCHAR array as it's members.
       * We put the style in the char array, and when this function
       * returns the LOGFONT has the logfont for the font we want to create.
       */
      if (lpszStyle)
         lstrcpy( (LPTSTR) elf.elfStyle, lpszStyle ) ;
      else
        switch( uiFlags & ~(RCF_NODEFAULT | RCF_STRIKEOUT | RCF_UNDERLINE) )
        {
            case RCF_BOLD:
                lstrcpy( (LPTSTR) elf.elfStyle, szBold ) ;
            break ;

            case RCF_ITALIC:
                lstrcpy( (LPTSTR) elf.elfStyle, szItalic ) ;
            break ;

            case RCF_BOLD | RCF_ITALIC:
                lstrcpy( (LPTSTR) elf.elfStyle, szBold ) ;
                lstrcat( (LPTSTR) elf.elfStyle, _T(" ") ) ;
                lstrcat( (LPTSTR) elf.elfStyle, szItalic ) ;
            break ;

            default:
                lstrcpy( (LPTSTR) elf.elfStyle, szRegular ) ;
                lpszStyle = (LPTSTR) elf.elfStyle ;
            break ;
        }

      if (lpszFace)
#ifndef WIN32
         EnumFontFamilies( hdcCur, lpszFace, lpfn, (LONG)(LPVOID) &elf ) ;
#else
         EnumFontFamilies( hdcCur, lpszFace, lpfn, (LPARAM)(LPTSTR) &elf ) ;
#endif

      if (!lpszFace || lstrcmpi( elf.elfLogFont.lfFaceName, lpszFace))
      {
         if ((uiFlags & RCF_NODEFAULT) == RCF_NODEFAULT)
         {
            if (hdcCur != hDC)
               ReleaseDC( NULL, hdcCur ) ;

            return NULL ;
         }
         else
            GetObject( GetStockObject( ANSI_VAR_FONT ),
                       sizeof( LOGFONT ), (LPTSTR)&elf.elfLogFont ) ;
      }

      // See pages 4-48, 4-49, of Reference Vol. 1 for explaination
      // of why we set lfWidth to 0
      //
      elf.elfLogFont.lfWidth = 0 ;

      // The equation for calculating point size based on font
      // height (a point is 1/72 of an inch) is:
      //
      //
      //    pt = (height * 72) / number of pixels in the Y direction
      //
      // Using GetTextMetrics() you should calculate height as:
      //
      //    height = tm.tmHeight - tm.tmInternalLeading
      //
      // This is because Windows' interpretation of a font height
      // is different from everyone else's.
      //
      //
      elf.elfLogFont.lfHeight = -MulDiv( nPointSize,
                              GetDeviceCaps( hdcCur, LOGPIXELSY),
                              72 )  ;

      elf.elfLogFont.lfStrikeOut = (BYTE)((uiFlags & RCF_STRIKEOUT) ==
RCF_STRIKEOUT) ;
      elf.elfLogFont.lfUnderline = (BYTE)((uiFlags & RCF_UNDERLINE) ==
RCF_UNDERLINE) ;

      if (!lpszStyle)
      {
         elf.elfLogFont.lfWeight    = (uiFlags & RCF_BOLD) ? FW_BOLD :
FW_NORMAL ;
         elf.elfLogFont.lfItalic    = (BYTE)((uiFlags & RCF_ITALIC) ==
RCF_ITALIC) ;
      }

   }

   if (hdcCur != hDC)
      ReleaseDC( NULL, hdcCur ) ;

   return CreateFontIndirect( &elf.elfLogFont ) ;

} /* ReallyCreateFontEx()  */

/*****************************************************************
 *  int CALLBACK
 *    fnEnumReallyEx( LPLOGFONT lpLogFont,
 *                   LPTEXTMETRIC lpTextMetrics,
 *                   int nFontType, LPLOGFONT lpLF )
 *
 *  Description:
 *
 *    Call back for EnumFonts and ReallySelectFontEx().
 *
 *    DO NOT FORGET TO EXPORT!!!!!!!!!!!!!!!!!!!!!!!!!!!!
 *
 *  Comments:
 *
 *****************************************************************/
extern "C"
int EXPORT WINAPI
fnEnumReallyEx( LPENUMLOGFONT lpELFin,
                LPNEWTEXTMETRIC lpTextMetrics,
                int nFontType,
                LPENUMLOGFONT lpELF )
{
   /* If we are not true type, then we will only be called once,
    * in in this case we need to compare the weight and italic
    * members of the TM struct to some strings.
    */
```

```
if (!(nFontType & TRUETYPE_FONTTYPE))
{
   lpELF->elfLogFont = lpELFin->elfLogFont ;

   /* Parse szStyle.  It may be something like "Bold Italic" or
    * "Demibold Italic".  Let's be somewhat smart about this.
    *
    * If "Demibold Italic" is passed in then he wants bold and
    * italic.
    *
    * We want to search szStyle for the word "bold".  Got a better
    * search strategy, go ahead and put it here (but tell me about it!).
    *
    * WARNING!!!
    *   The style string is language dependent.
    */
   if (lpELF->elfStyle)
   {
      /*
       * Assume "Regular"
       */
      lpELF->elfLogFont.lfWeight = FW_NORMAL ;
      lpELF->elfLogFont.lfItalic = FALSE ;

      if (!lstrcmpi( (LPTSTR) lpELF->elfStyle, szBold ))
      {
         lpELF->elfLogFont.lfWeight = FW_BOLD ;
         return 0 ;
      }

      if (!lstrcmpi( (LPTSTR) lpELF->elfStyle, szItalic))
      {
         lpELF->elfLogFont.lfItalic = TRUE ;
         return 0 ;
      }

      if (!lstrcmpi( (LPTSTR) lpELF->elfStyle, szBoldItalic ))
      {
         lpELF->elfLogFont.lfWeight = FW_BOLD ;
         lpELF->elfLogFont.lfItalic = TRUE ;
         return 0 ;
      }
   }

   return 0 ;  /* stop the enumeration */

}

/* We now know we have a TT font.  For each style in the
 * family passed in, we will get here.
 *
 * If we get an exact match copy it into lpELF and return.
 * Otherwise get as close as possible.
 */
if (0==lstrcmpi( (LPTSTR) lpELF->elfStyle, (LPTSTR)lpELFin->elfStyle ))
```

```
   {
      *lpELF = *lpELFin;
      return 0;
   }

   lpELF->elfLogFont = lpELFin->elfLogFont ;

   return 1 ;

}/* fnEnumReallyEx() */


/****************************************************************
 *  void WINAPI
 *  DlgCenter( HWND hwndCenter, HWND hwndWithin, BOOL fClient )
 *
 *  Description:
 *
 *    Centers a window within another window.
 *
 *  Type/Parameter
 *          Description
 *
 *    HWND  hwndCenter
 *          Window to center.  This does not have to be a child of
 *          hwndWithin.
 *
 *    HWND  hwndWithin
 *          Window to center the above window within.  Can be NULL.
 *
 *    BOOL  fClient
 *          If TRUE the window is centered within the available client
 *          area.  Otherwise it's centered within the entire window area.
 *
 *  Comments:
 *
 ****************************************************************/
void WINAPI
DlgCenter( HWND hwndCenter, HWND hwndWithin, BOOL fClient )
{
   RECT rcWithin ;
   RECT rcCenter ;
   int   x, y ;
   int   dxCenter, dyCenter ;
   int   dxScreen, dyScreen ;

   dxScreen = GetSystemMetrics( SM_CXSCREEN ) ;
   dyScreen = GetSystemMetrics( SM_CYSCREEN ) ;

   if (!IsWindow(hwndCenter))
      return ;

   if (hwndWithin && !IsWindow(hwndWithin))
      return ;
```

```
if (hwndWithin == NULL)
{
   rcWithin.left = rcWithin.top = 0 ;
   rcWithin.right = dxScreen ;
   rcWithin.bottom = dyScreen ;
}
else
{
   if (fClient)
   {
      /*
       * First get screen coords of rectangle we're going to be
       * centered within.
       */
      GetClientRect( hwndWithin, (LPRECT)&rcWithin ) ;
      ClientToScreen( hwndWithin, (LPPOINT)&rcWithin.left ) ;
      ClientToScreen( hwndWithin, (LPPOINT)&rcWithin.right ) ;
   }
   else
      GetWindowRect( hwndWithin, (LPRECT)&rcWithin ) ;
}

GetWindowRect( hwndCenter, (LPRECT)&rcCenter ) ;
dxCenter = rcCenter.right - rcCenter.left ;
dyCenter = rcCenter.bottom - rcCenter.top ;

/*
 * Now we have both the within and center rects in screen coords.
 *
 * SetWindowPos behaves differently for Non child windows
 * than for child windows.  For popups it's coordinates
 * are relative to the upper left corner of the screen.  For
 * children it's coords are relative to the upper left corner
 * of the client area of the parent.
 */
x = ((rcWithin.right - rcWithin.left) - dxCenter) / 2 ;
y = ((rcWithin.bottom - rcWithin.top) - dyCenter) / 2 ;

if (hwndWithin == GetParent( hwndCenter ) &&
    !(GetWindowLong( hwndCenter, GWL_STYLE ) & WS_CHILD ))
{
   x += rcWithin.left ;
   y += rcWithin.top ;

   if (x + dxCenter > dxScreen )
      x = dxScreen - dxCenter ;

   if (y + dyCenter > dyScreen )
      y = dyScreen - dyCenter ;

   SetWindowPos( hwndCenter, NULL,
                 max(x,0),
                 max(y,0),
                 0, 0, SWP_NOACTIVATE | SWP_NOSIZE | SWP_NOZORDER ) ;
      return ;
   }

   SetWindowPos( hwndCenter, NULL,
                 max(x,0),
                 max(y,0),
                 0, 0, SWP_NOACTIVATE | SWP_NOSIZE | SWP_NOZORDER ) ;

   return ;

}/* DlgCenter() */

// column text out

/*****************************************************************
 *  int WINAPI
 *    GetTextMetricsCorrectly( HDC hDC, LPTEXTMETRIC lpTextMetrics )
 *
 *  Description:
 *
 *    This function gets the textmetrics of the font currently
 *    selected into the hDC.  It returns the average char width as
 *    the return value.
 *
 *    This function computes the average character width correctly
 *    by using GetTextExtent() on the string "abc...xzyABC...XYZ"
 *    which works out much better for proportional fonts.
 *
 *    Note that this function returns the same TEXTMETRIC
 *    values that GetTextMetrics() does, it simply has a different
 *    return value.
 *
 *  Comments:
 *
 ****************************************************************/
int WINAPI
   GetTextMetricsCorrectly( HDC hDC, LPTEXTMETRIC lpTM )
{
   int    nAveWidth ;
   TCHAR   rgchAlphabet[52] ;
   int  i ;
   SIZE size ;

   // Get the TM of the current font.  Note that GetTextMetrics
   // gets the incorrect AveCharWidth value for proportional fonts.
   // This is the whole poshort in this exercise.
   //
   if (lpTM)
    GetTextMetrics(hDC, lpTM);

   // If it's not a variable pitch font GetTextMetrics was correct
   // so just return.
   //
   if (lpTM && !(lpTM->tmPitchAndFamily & FIXED_PITCH))
      return lpTM->tmAveCharWidth ;
   else
   {
      for ( i = 0 ; i <= 25 ; i++)
         rgchAlphabet[i] = (TCHAR)(i+(int)'a') ;

      for ( i = 0 ; i<=25 ; i++)
         rgchAlphabet[i+25] = (TCHAR)(i+(int)'A') ;

      GetTextExtentPoint( hDC, (LPTSTR)rgchAlphabet, 52, &size ) ;
      nAveWidth = size.cx / 26 ;
      nAveWidth = (nAveWidth + 1) / 2 ;
   }

   // Return the calculated average char width
   //
   return nAveWidth ;

} /* GetTextMetricsCorrectly()  */

/****************************************************************
 *
 *  Description:
 *
 *    The ColumntTextOut function writes a character string at
 *    the specified location, using tabs to identify column breaks.  Each
 *    column is aligned according to the array of COLUMNSTRUCTs.
 *
 *    A COLUMNSTRUCT looks like this:
 *
 *    {
 *       int   nLeft ;       // starting x position of the column
 *       int   nRight ;      // ending x position of the column
 *       UINT  uiFlags ;      // format flags
 *    }
 *
 *    If a column has another column to the left of it, it's nLeft member
 *    is ignored.
 *
 *    uiFlags can be any of the following:
 *
 *          #define DT_LEFT      0x0000
 *          #define DT_CENTER    0x0001
 *          #define DT_RIGHT     0x0002
 *
 *  Comments:
 *
 ****************************************************************/
void WINAPI ColumnTextOut( HDC hdc, int nX, int nY, LPTSTR lpszIN,
                            int cColumns, LPCOLUMNSTRUCT rgColumns )
{

#define ETOFLAGS ETO_CLIPPED

    COLUMNSTRUCT   CS ;               // local copy for speed
    RECT           rc ;               // current cell rect
```

```
int            cIteration = 0 ;  // what column
LPTSTR          lpNext ;          // next string (current is lpsz)
int            cch ;             // count of chars in current string
SIZE           size ;            // extent of current string
#ifdef WIN32
LONG          dx ;              // column width
#else
int             dx ;
#endif

rc.left = nX ;
rc.top = nY ;
rc.right = 0 ;

if (rgColumns == NULL || cColumns <= 1)
{
    int Tab = 15 ;
    TabbedTextOut( hdc, nX, nY, lpszIN, lstrlen(lpszIN), 1, &Tab, nX ) ;
    return ;
}

// For each sub string (bracketed by a tab)
//
LPTSTR lpsz = lpszIN ;
while (*lpsz)
{
    if (cIteration >= cColumns)
        return ;

    for (cch = 0, lpNext = lpsz ;
         *lpNext != '\t' && *lpNext ;
         lpNext++, cch++)
        ;

    CS = rgColumns[cIteration] ;

    // If it's the leftmost column use
    //
    if (cIteration == 0)
    {
        rc.left = nX + CS.nLeft ;
        rc.right = nX + CS.nRight ;
    }
    else
        rc.right = nX + CS.nRight ;

    GetTextExtentPoint( hdc, lpsz, cch, &size ) ;
    rc.bottom = rc.top + size.cy ;

    // If the width of the column is 0 do nothing
    //
    if ((dx = (rc.right - rc.left)) > 0)
    {
        switch(CS.uiFlags)
        {
                case DT_CENTER:
                    ExtTextOut( hdc, rc.left + ((dx - size.cx) / (int)2),
                    rc.top, ETOFLAGS, &rc, lpsz, cch, NULL ) ;
                break ;

                case DT_RIGHT:
                    // If it's right justified then make the left border 0
                    //
                    //rc.left = nX + rgColumns[0].nLeft ;
                    ExtTextOut( hdc, rc.left + (dx - size.cx),
                    rc.top, ETOFLAGS, &rc, lpsz, cch, NULL ) ;
                break ;

                case DT_LEFT:
                default:
                    ExtTextOut( hdc, rc.left, rc.top, ETOFLAGS, &rc, lpsz,
cch, NULL ) ;
                break ;
            }
        }
        rc.left = rc.right ;
        cIteration++ ;
        lpsz = lpNext + 1 ;
    }

} // ColumnTextOut()


LPTSTR HRtoString( HRESULT hr )
{
    static  TCHAR sz[256] ;
    LPTSTR  szErrName ;
    SCODE   sc ;
#define CASE_SCODE(sc)  case sc: szErrName = (LPTSTR)_T(#sc) ; break ;

    sc = GetScode( hr ) ;

    switch (sc) {
        /* SCODE's defined in SCODE.H */
        CASE_SCODE(S_OK)
 // same value as S_OK      CASE_SCODE(S_TRUE)
        CASE_SCODE(S_FALSE)
        CASE_SCODE(E_UNEXPECTED)
        CASE_SCODE(E_NOTIMPL)
        CASE_SCODE(E_OUTOFMEMORY)
        CASE_SCODE(E_INVALIDARG)
        CASE_SCODE(E_NOINTERFACE)
        CASE_SCODE(E_POINTER)
        CASE_SCODE(E_HANDLE)
        CASE_SCODE(E_ABORT)
        CASE_SCODE(E_FAIL)
        CASE_SCODE(E_ACCESSDENIED)

        /* SCODE's defined in DVOBJ.H */
        CASE_SCODE(DATA_E_FORMATETC)
```

```
// same as DATA_E_FORMATETC     CASE_SCODE(DV_E_FORMATETC)
        CASE_SCODE(DATA_S_SAMEFORMATETC)
        CASE_SCODE(VIEW_E_DRAW)
//  same as VIEW_E_DRAW         CASE_SCODE(E_DRAW)
        CASE_SCODE(VIEW_S_ALREADY_FROZEN)
        CASE_SCODE(CACHE_E_NOCACHE_UPDATED)
        CASE_SCODE(CACHE_S_FORMATETC_NOTSUPPORTED)
        CASE_SCODE(CACHE_S_SAMECACHE)
        CASE_SCODE(CACHE_S_SOMECACHES_NOTUPDATED)

        /* SCODE's defined in OLE2.H */
        CASE_SCODE(OLE_E_OLEVERB)
        CASE_SCODE(OLE_E_ADVF)
        CASE_SCODE(OLE_E_ENUM_NOMORE)
        CASE_SCODE(OLE_E_ADVISENOTSUPPORTED)
        CASE_SCODE(OLE_E_NOCONNECTION)
        CASE_SCODE(OLE_E_NOTRUNNING)
        CASE_SCODE(OLE_E_NOCACHE)
        CASE_SCODE(OLE_E_BLANK)
        CASE_SCODE(OLE_E_CLASSDIFF)
        CASE_SCODE(OLE_E_CANT_GETMONIKER)
        CASE_SCODE(OLE_E_CANT_BINDTOSOURCE)
        CASE_SCODE(OLE_E_STATIC)
        CASE_SCODE(OLE_E_PROMPTSAVECANCELLED)
        CASE_SCODE(OLE_E_INVALIDRECT)
        CASE_SCODE(OLE_E_WRONGCOMPOBJ)
        CASE_SCODE(OLE_E_INVALIDHWND)
        CASE_SCODE(DV_E_DVTARGETDEVICE)
        CASE_SCODE(DV_E_STGMEDIUM)
        CASE_SCODE(DV_E_STATDATA)
        CASE_SCODE(DV_E_LINDEX)
        CASE_SCODE(DV_E_TYMED)
        CASE_SCODE(DV_E_CLIPFORMAT)
        CASE_SCODE(DV_E_DVASPECT)
        CASE_SCODE(DV_E_DVTARGETDEVICE_SIZE)
        CASE_SCODE(DV_E_NOIVIEWOBJECT)
        CASE_SCODE(OLE_S_USEREG)
        CASE_SCODE(OLE_S_STATIC)
        CASE_SCODE(OLE_S_MAC_CLIPFORMAT)
        CASE_SCODE(CONVERT10_E_OLESTREAM_GET)
        CASE_SCODE(CONVERT10_E_OLESTREAM_PUT)
        CASE_SCODE(CONVERT10_E_OLESTREAM_FMT)
        CASE_SCODE(CONVERT10_E_OLESTREAM_BITMAP_TO_DIB)
        CASE_SCODE(CONVERT10_E_STG_FMT)
        CASE_SCODE(CONVERT10_E_STG_NO_STD_STREAM)
        CASE_SCODE(CONVERT10_E_STG_DIB_TO_BITMAP)
        CASE_SCODE(CONVERT10_S_NO_PRESENTATION)
        CASE_SCODE(CLIPBRD_E_CANT_OPEN)
        CASE_SCODE(CLIPBRD_E_CANT_EMPTY)
        CASE_SCODE(CLIPBRD_E_CANT_SET)
        CASE_SCODE(CLIPBRD_E_BAD_DATA)
        CASE_SCODE(CLIPBRD_E_CANT_CLOSE)
        CASE_SCODE(DRAGDROP_E_NOTREGISTERED)
        CASE_SCODE(DRAGDROP_E_ALREADYREGISTERED)
        CASE_SCODE(DRAGDROP_E_INVALIDHWND)
        CASE_SCODE(DRAGDROP_S_DROP)
        CASE_SCODE(DRAGDROP_S_CANCEL)
        CASE_SCODE(DRAGDROP_S_USEDEFAULTCURSORS)
        CASE_SCODE(OLEOBJ_E_NOVERBS)
        CASE_SCODE(OLEOBJ_S_INVALIDVERB)
        CASE_SCODE(OLEOBJ_S_CANNOT_DOVERB_NOW)
        CASE_SCODE(OLEOBJ_S_INVALIDHWND)
        CASE_SCODE(INPLACE_E_NOTUNDOABLE)
        CASE_SCODE(INPLACE_E_NOTOOLSPACE)
        CASE_SCODE(INPLACE_S_TRUNCATED)

        /* SCODE's defined in STORAGE.H */
        CASE_SCODE(STG_E_INVALIDFUNCTION)
        CASE_SCODE(STG_E_FILENOTFOUND)
        CASE_SCODE(STG_E_PATHNOTFOUND)
        CASE_SCODE(STG_E_TOOMANYOPENFILES)
        CASE_SCODE(STG_E_ACCESSDENIED)
        CASE_SCODE(STG_E_INVALIDHANDLE)
        CASE_SCODE(STG_E_INSUFFICIENTMEMORY)
        CASE_SCODE(STG_E_INVALIDPOINTER)
        CASE_SCODE(STG_E_NOMOREFILES)
        CASE_SCODE(STG_E_DISKISWRITEPROTECTED)
        CASE_SCODE(STG_E_SEEKERROR)
        CASE_SCODE(STG_E_WRITEFAULT)
        CASE_SCODE(STG_E_READFAULT)
        CASE_SCODE(STG_E_LOCKVIOLATION)
        CASE_SCODE(STG_E_FILEALREADYEXISTS)
        CASE_SCODE(STG_E_INVALIDPARAMETER)
        CASE_SCODE(STG_E_MEDIUMFULL)
        CASE_SCODE(STG_E_ABNORMALAPIEXIT)
        CASE_SCODE(STG_E_INVALIDHEADER)
        CASE_SCODE(STG_E_INVALIDNAME)
        CASE_SCODE(STG_E_UNKNOWN)
        CASE_SCODE(STG_E_UNIMPLEMENTEDFUNCTION)
        CASE_SCODE(STG_E_INVALIDFLAG)
        CASE_SCODE(STG_E_INUSE)
        CASE_SCODE(STG_E_NOTCURRENT)
        CASE_SCODE(STG_E_REVERTED)
        CASE_SCODE(STG_E_CANTSAVE)
        CASE_SCODE(STG_E_OLDFORMAT)
        CASE_SCODE(STG_E_OLDDLL)
        CASE_SCODE(STG_E_SHAREREQUIRED)
        CASE_SCODE(STG_S_CONVERTED)
//        CASE_SCODE(STG_S_BUFFEROVERFLOW)
//        CASE_SCODE(STG_S_TRYOVERWRITE)

        /* SCODE's defined in COMPOBJ.H */
        CASE_SCODE(CO_E_NOTINITIALIZED)
        CASE_SCODE(CO_E_ALREADYINITIALIZED)
        CASE_SCODE(CO_E_CANTDETERMINECLASS)
        CASE_SCODE(CO_E_CLASSSTRING)
        CASE_SCODE(CO_E_IIDSTRING)
        CASE_SCODE(CO_E_APPNOTFOUND)
        CASE_SCODE(CO_E_APPSINGLEUSE)
        CASE_SCODE(CO_E_ERRORINAPP)
        CASE_SCODE(CO_E_DLLNOTFOUND)
        CASE_SCODE(CO_E_ERRORINDLL)
        CASE_SCODE(CO_E_WRONGOSFORAPP)
        CASE_SCODE(CO_E_OBJNOTREG)
        CASE_SCODE(CO_E_OBJISREG)
        CASE_SCODE(CO_E_OBJNOTCONNECTED)
        CASE_SCODE(CO_E_APPDIDNTREG)
        CASE_SCODE(CLASS_E_NOAGGREGATION)
        CASE_SCODE(REGDB_E_READREGDB)
        CASE_SCODE(REGDB_E_WRITEREGDB)
        CASE_SCODE(REGDB_E_KEYMISSING)
        CASE_SCODE(REGDB_E_INVALIDVALUE)
        CASE_SCODE(REGDB_E_CLASSNOTREG)
        CASE_SCODE(REGDB_E_IIDNOTREG)
#if WIN32!=300
//        CASE_SCODE(RPC_E_FIRST)
        CASE_SCODE(RPC_E_CALL_REJECTED)
        CASE_SCODE(RPC_E_CALL_CANCELED)
        CASE_SCODE(RPC_E_CANTPOST_INSENDCALL)
        CASE_SCODE(RPC_E_CANTCALLOUT_INASYNCCALL)
        CASE_SCODE(RPC_E_CANTCALLOUT_INEXTERNALCALL)
        CASE_SCODE(RPC_E_CONNECTION_TERMINATED)
        CASE_SCODE(RPC_E_SERVER_DIED)
        CASE_SCODE(RPC_E_CLIENT_DIED)
        CASE_SCODE(RPC_E_INVALID_DATAPACKET)
        CASE_SCODE(RPC_E_CANTTRANSMIT_CALL)
        CASE_SCODE(RPC_E_CLIENT_CANTMARSHAL_DATA)
        CASE_SCODE(RPC_E_CLIENT_CANTUNMARSHAL_DATA)
        CASE_SCODE(RPC_E_SERVER_CANTMARSHAL_DATA)
        CASE_SCODE(RPC_E_SERVER_CANTUNMARSHAL_DATA)
        CASE_SCODE(RPC_E_INVALID_DATA)
        CASE_SCODE(RPC_E_INVALID_PARAMETER)
        CASE_SCODE(RPC_E_UNEXPECTED)
#endif
        /* SCODE's defined in MONIKER.H */
        CASE_SCODE(MK_E_CONNECTMANUALLY)
        CASE_SCODE(MK_E_EXCEEDEDDEADLINE)
        CASE_SCODE(MK_E_NEEDGENERIC)
        CASE_SCODE(MK_E_UNAVAILABLE)
        CASE_SCODE(MK_E_SYNTAX)
        CASE_SCODE(MK_E_NOOBJECT)
        CASE_SCODE(MK_E_INVALIDEXTENSION)
        CASE_SCODE(MK_E_INTERMEDIATEINTERFACENOTSUPPORTED)
        CASE_SCODE(MK_E_NOTBINDABLE)
        CASE_SCODE(MK_E_NOTBOUND)
        CASE_SCODE(MK_E_CANTOPENFILE)
        CASE_SCODE(MK_E_MUSTBOTHERUSER)
        CASE_SCODE(MK_E_NOINVERSE)
        CASE_SCODE(MK_E_NOSTORAGE)
        CASE_SCODE(MK_S_REDUCED_TO_SELF)
        CASE_SCODE(MK_S_ME)
        CASE_SCODE(MK_S_HIM)
        CASE_SCODE(MK_S_US)
        CASE_SCODE(MK_S_MONIKERALREADYREGISTERED)
```

```
/* SCODE's defined in DISPATCH.H */
        CASE_SCODE(DISP_E_UNKNOWNINTERFACE)
        CASE_SCODE(DISP_E_MEMBERNOTFOUND)
        CASE_SCODE(DISP_E_PARAMNOTFOUND)
        CASE_SCODE(DISP_E_TYPEMISMATCH)
        CASE_SCODE(DISP_E_UNKNOWNNAME)
        CASE_SCODE(DISP_E_NONAMEDARGS)
        CASE_SCODE(DISP_E_BADVARTYPE)
        CASE_SCODE(DISP_E_EXCEPTION)
        CASE_SCODE(DISP_E_OVERFLOW)
        CASE_SCODE(DISP_E_BADINDEX)
        CASE_SCODE(DISP_E_UNKNOWNLCID)
        CASE_SCODE(DISP_E_ARRAYISLOCKED)
        CASE_SCODE(DISP_E_BADPARAMCOUNT)
#if WIN32!=300
        CASE_SCODE(DISP_E_PARAMNOTOPTIONAL)
        CASE_SCODE(DISP_E_BADCALLEE)
        CASE_SCODE(DISP_E_NOTACOLLECTION)
        CASE_SCODE(TYPE_E_BADMODULEKIND)

#endif
        CASE_SCODE(TYPE_E_IOERROR)
//        CASE_SCODE(TYPE_E_COMPILEERROR)
        CASE_SCODE(TYPE_E_CANTCREATETMPFILE)
//        CASE_SCODE(TYPE_E_ILLEGALINDEX)
//        CASE_SCODE(TYPE_E_IDNOTFOUND)
        CASE_SCODE(TYPE_E_BUFFERTOOSMALL)
//        CASE_SCODE(TYPE_E_READONLY)
        CASE_SCODE(TYPE_E_INVDATAREAD)
        CASE_SCODE(TYPE_E_UNSUPFORMAT)
//        CASE_SCODE(TYPE_E_ALREADYCONTAINSNAME)
//        CASE_SCODE(TYPE_E_NOMATCHINGARITY)
        CASE_SCODE(TYPE_E_REGISTRYACCESS)
        CASE_SCODE(TYPE_E_LIBNOTREGISTERED)
//        CASE_SCODE(TYPE_E_DUPLICATEDEFINITION)
//        CASE_SCODE(TYPE_E_DESTNOTKNOWN)
        CASE_SCODE(TYPE_E_UNDEFINEDTYPE)
        CASE_SCODE(TYPE_E_QUALIFIEDNAMEDISALLOWED)
        CASE_SCODE(TYPE_E_INVALIDSTATE)
        CASE_SCODE(TYPE_E_WRONGTYPEKIND)
        CASE_SCODE(TYPE_E_ELEMENTNOTFOUND)
        CASE_SCODE(TYPE_E_AMBIGUOUSNAME)
//        CASE_SCODE(TYPE_E_INVOKEFUNCTIONMISMATCH)
        CASE_SCODE(TYPE_E_DLLFUNCTIONNOTFOUND)
//        CASE_SCODE(TYPE_E_WRONGPLATFORM)
//        CASE_SCODE(TYPE_E_ALREADYBEINGLAIDOUT)
        CASE_SCODE(TYPE_E_CANTLOADLIBRARY)

        default:
            szErrName = _T("UNKNOWN SCODE") ;
    }

    wsprintf( sz, _T("%s (0x%lx)"), (LPTSTR)szErrName, sc);

    return (LPTSTR)sz ;
```

```
}

static TCHAR * g_rgszVT[] =
{
    _T("Void"),             //VT_EMPTY           = 0,   /* [V]   [P]
nothing                     */
    _T("Null"),             //VT_NULL            = 1,   /* [V]        SQL
style Null              */
    _T("Integer"),          //VT_I2              = 2,   /* [V][T][P]  2 byte
signed int           */
    _T("Long"),             //VT_I4              = 3,   /* [V][T][P]  4 byte
signed int           */
    _T("Single"),           //VT_R4              = 4,   /* [V][T][P]  4 byte
real                 */
    _T("Double"),           //VT_R8              = 5,   /* [V][T][P]  8 byte
real                 */
    _T("Currency"),         //VT_CY              = 6,   /* [V][T][P]
currency                    */
    _T("Date"),             //VT_DATE            = 7,   /* [V][T][P]  date
*/
    _T("String"),           //VT_BSTR            = 8,   /* [V][T][P]  binary
string               */
    _T("Object"),           //VT_DISPATCH        = 9,   /* [V][T]
IDispatch FAR*              */
    _T("SCODE"),            //VT_ERROR           = 10,  /* [V][T]     SCODE
*/
    _T("Boolean"),          //VT_BOOL            = 11,  /* [V][T][P]
True=-1, False=0            */
    _T("Variant"),          //VT_VARIANT         = 12,  /* [V][T][P]
VARIANT FAR*                */
    _T("pIUnknown"),        //VT_UNKNOWN         = 13,  /* [V][T]
IUnknown FAR*               */
    _T("Unicode"),          //VT_WBSTR           = 14,  /* [V][T]     wide
binary string          */
    _T(""),                 //                   = 15,
    _T("BYTE"),             //VT_I1              = 16,  /*    [T]     signed
char                 */
    _T("char"),             //VT_UI1             = 17,  /*    [T]
unsigned char               */
    _T("USHORT"),           //VT_UI2             = 18,  /*    [T]
unsigned short              */
    _T("ULONG"),            //VT_UI4             = 19,  /*    [T]
unsigned short              */
    _T("int64"),            //VT_I8              = 20,  /*    [T][P]  signed
64-bit int           */
    _T("uint64"),           //VT_UI8             = 21,  /*    [T]
unsigned 64-bit int         */
    _T("int"),              //VT_INT             = 22,  /*    [T]     signed
machine int          */
    _T("UINT"),             //VT_UINT            = 23,  /*    [T]
unsigned machine int        */
    _T("VOID"),             //VT_VOID            = 24,  /*    [T]     C
style void                */
    _T("HRESULT"),          //VT_HRESULT         = 25,  /*    [T]
*/
```

```
    _T("PTR"),              //VT_PTR             = 26,  /*    [T]
pointer type                */
    _T("SAFEARRAY"),        //VT_SAFEARRAY       = 27,  /*    [T]     (use
VT_ARRAY in VARIANT)   */
    _T("CARRAY"),           //VT_CARRAY          = 28,  /*    [T]     C
style array               */
    _T("USERDEFINED"),      //VT_USERDEFINED     = 29,  /*    [T]     user
defined type          */
    _T("LPSTR"),            //VT_LPSTR           = 30,  /*    [T][P]  null
terminated string      */
    _T("LPWSTR"),           //VT_LPWSTR          = 31,  /*    [T][P]  wide
null terminated string */
    _T(""),                 //                   = 32,
    _T(""),                 //                   = 33,
    _T(""),                 //                   = 34,
    _T(""),                 //                   = 35,
    _T(""),                 //                   = 36,
    _T(""),                 //                   = 37,
    _T(""),                 //                   = 38,
    _T(""),                 //                   = 39,
    _T(""),                 //                   = 40,
    _T(""),                 //                   = 41,
    _T(""),                 //                   = 42,
    _T(""),                 //                   = 43,
    _T(""),                 //                   = 44,
    _T(""),                 //                   = 45,
    _T(""),                 //                   = 46,
    _T(""),                 //                   = 47,
    _T(""),                 //                   = 48,
    _T(""),                 //                   = 49,
    _T(""),                 //                   = 50,
    _T(""),                 //                   = 51,
    _T(""),                 //                   = 52,
    _T(""),                 //                   = 53,
    _T(""),                 //                   = 54,
    _T(""),                 //                   = 55,
    _T(""),                 //                   = 56,
    _T(""),                 //                   = 57,
    _T(""),                 //                   = 58,
    _T(""),                 //                   = 59,
    _T(""),                 //                   = 60,
    _T(""),                 //                   = 61,
    _T(""),                 //                   = 62,
    _T(""),                 //                   = 63,
    _T("FILETIME"),         //VT_FILETIME        = 64,  /*       [P]
FILETIME                    */
    _T("BLOB"),             //VT_BLOB            = 65,  /*       [P]  Length
prefixed bytes        */
    _T("STREAM"),           //VT_STREAM          = 66,  /*       [P]  Name
of the stream follows  */
    _T("STORAGE"),          //VT_STORAGE         = 67,  /*       [P]  Name
of the storage follows */
    _T("STREAMED_OBJECT"),  //VT_STREAMED_OBJECT = 68,  /*       [P]  Stream
contains an object   */
```

```
    _T("STORED_OBJECT"),     //VT_STORED_OBJECT   = 69,  /*       [P]
Storage contains an object  */
    _T("BLOB_OBJECT"),       //VT_BLOB_OBJECT     = 70,  /*       [P]  Blob
contains an object     */
    _T("CF"),                //VT_CF              = 71,  /*       [P]
Clipboard format         */
    _T("CLSID"),             //VT_CLSID           = 72   /*       [P]  A
Class ID                  */
};

LPTSTR VTtoString( VARTYPE vt )
{
    static TCHAR szBuf[64];

    if (vt <= VT_CLSID)
        return (LPTSTR)g_rgszVT[vt] ;

    if (vt & VT_VECTOR)
    {
        vt &= ~VT_VECTOR ;
        if (vt <= VT_CLSID)
            wsprintf( szBuf, _T("VECTOR of %s"), (LPTSTR)g_rgszVT[vt] ) ;
        else
            wsprintf( szBuf, _T("<Unknown %08lX>"), vt & VT_VECTOR ) ;
        return (LPTSTR)szBuf ;
    }

    if (vt & VT_ARRAY)
    {
        vt &= ~VT_ARRAY ;
        if (vt <= VT_CLSID)
            wsprintf( szBuf, _T("Array of %s"), (LPTSTR)g_rgszVT[vt] ) ;
        else
            wsprintf( szBuf, _T("<Unknown %08lX>"), vt & VT_ARRAY ) ;
        return (LPTSTR)szBuf ;
    }

    if (vt & VT_BYREF)
    {
        vt &= ~VT_BYREF ;
        if (vt <= VT_CLSID)
            wsprintf( szBuf, _T("%s BYREF "), (LPTSTR)g_rgszVT[vt] ) ;
        else
            wsprintf( szBuf, _T("<Unknown %08lX>"), vt & VT_BYREF ) ;
        return (LPTSTR)szBuf ;
    }

    if (vt & VT_RESERVED)
    {
        vt &= ~VT_RESERVED ;
        if (vt <= VT_CLSID)
            wsprintf( szBuf, _T("RESERVED (%s)"), (LPTSTR)g_rgszVT[vt] ) ;
        else
            wsprintf( szBuf, _T("<Unknown %08lX>"), vt & VT_RESERVED ) ;
        return (LPTSTR)szBuf ;
    }

    wsprintf( szBuf, _T("<Unknown %08lX>"), vt ) ;
    return (LPTSTR)szBuf ;
}

LPTSTR VTtoString2( VARTYPE vt )
{
    static TCHAR szBuf[64];

    vt &= ~0xF000 ;

    if (vt <= VT_CLSID)
        return (LPTSTR)g_rgszVT[vt] ;

    wsprintf( szBuf, _T("<Unknown %08lX>"), vt ) ;
    return (LPTSTR)szBuf ;
}

LPTSTR TYPEDESCtoString( TYPEDESC FAR* lptdesc )
{
    static TCHAR sz[64] ;

    lstrcpy( sz, VTtoString( lptdesc->vt ) ) ;
    if ((lptdesc->vt & 0x0FFF) == VT_PTR)
    {
        // lptdesc points to a TYPEDESC that specifies the thing pointed to
        lstrcat( sz, _T(" to ") ) ;
        lstrcat( sz, VTtoString( lptdesc->lptdesc->vt ) ) ;
        if ((lptdesc->lptdesc->vt & 0x0FFF) == VT_CARRAY)
        {
            lstrcat( sz,  _T(" of ") ) ;
            lstrcat( sz,  VTtoString( lptdesc->lptdesc->lpadesc-
>tdescElem.vt ) ) ;
            TCHAR szNum[16] ;
            wsprintf( szNum, _T("[%u]"), lptdesc->lptdesc->lpadesc-
>cDims ) ;
            lstrcat( sz, szNum ) ;
        }
    }

    if ((lptdesc->vt & 0x0FFF) == VT_CARRAY)
    {
        lstrcat( sz,  _T(" of ") ) ;
        lstrcat( sz,  VTtoString( lptdesc->lpadesc->tdescElem.vt ) ) ;
        TCHAR szNum[16] ;
        wsprintf( szNum, _T("[%u]"), lptdesc->lpadesc->cDims ) ;
        lstrcat( sz, szNum ) ;
    }
    return sz ;
}



int _lwriteT( int handle, TCHAR *taBuf, unsigned int count )
```

```
{

#ifdef _UNICODE
    char   achBuf[256];
    wcstombs( achBuf, taBuf, 256 );
    return (_lwrite( handle,  achBuf, count ));
#else
    return (_lwrite( handle, taBuf, count ));
#endif

}

HFILE WINAPI OpenFileT( LPTSTR tszFileName, LPOFSTRUCT lpReOpenBuf, UINT
uStyle )
{
#ifdef _UNICODE
    char   achName[MAX_PATH];
    wcstombs( achName, tszFileName, MAX_PATH );
    return OpenFile( achName, lpReOpenBuf, uStyle );
#else
    return OpenFile( tszFileName, lpReOpenBuf, uStyle );
#endif

}
```

UTIL_A.CPP (DEFO2V Sample)

```
// ansiwrap.cpp
//
// Creates ANSI versions of all modules.  Any exported function get's "A"
// appended.
//
//

#ifdef UNICODE
#undef UNICODE
#endif
#ifdef _UNICODE
#undef _UNICODE
#endif

#include "precompa.h"
#include "util.cpp"
```

DFVIEW

This sample creates a tool for viewing OLE's implementation of structured storage. For information on compiling and building the program, see MAKEFILE (DFVIEW Sample).

MAKEFILE (DFVIEW Sample)

```
#+---------------------------------------------------------------------------
-
#
#  Microsoft Windows
#  Copyright (C) Microsoft Corporation, 1994.
#
#  File:        makefile
#
#----------------------------------------------------------------------------
-

!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

#
#       Makefile for OLE Sample DFView
#
#       builds DFView.EXE
#

OLEFLAGS = -I ..\idl -I ..\winhlprs
LINK = $(link)
LINKFLAGS = $(linkdebug) $(guilflags)
RCFLAGS = -DWIN32

OBJS = winmain.obj mwclass.obj strmvwr.obj about.obj bitmaps.obj message.obj
LIBS = $(olelibsmt) ..\winhlprs\winhlprs.lib

all: DFView.exe

clean:
    -del *.obj
    -del dfv.res
    -del DFView.map
    -del DFView.exe

winmain.obj: winmain.cxx        \
        dfv.h                   \
        mwclass.h               \
        message.h               \
        ..\winhlprs\cwindow.h
    $(cc) $(cvars) $(cflags) $(UFLAGS) $(cdebug) $(OLEFLAGS) $*.cxx

mwclass.obj: mwclass.cxx        \
        dfv.h                   \
        mwclass.h               \
        about.h                 \
        strmvwr.h               \
        bitmaps.h               \
        message.h               \
        ..\winhlprs\cdialog.h   \
        ..\winhlprs\cwindow.h
    $(cc) $(cvars) $(cflags) $(UFLAGS) $(cdebug) $(OLEFLAGS) $*.cxx
```

```
strmvwr.obj: strmvwr.cxx        \
        dfv.h                   \
        strmvwr.h               \
        message.h               \
        mwclass.h               \
        ..\winhlprs\cwindow.h
    $(cc) $(cvars) $(cflags) $(UFLAGS) $(cdebug) $(OLEFLAGS) $*.cxx

about.obj: about.cxx            \
        about.h                 \
        ..\winhlprs\cdialog.h
    $(cc) $(cvars) $(cflags) $(UFLAGS) $(cdebug) $(OLEFLAGS) $*.cxx

bitmaps.obj: bitmaps.cxx        \
        bitmaps.h
    $(cc) $(cvars) $(cflags) $(UFLAGS) $(cdebug) $(OLEFLAGS) $*.cxx

message.obj: message.cxx        \
        message.h
    $(cc) $(cvars) $(cflags) $(UFLAGS) $(cdebug) $(OLEFLAGS) $*.cxx

dfv.res: dfv.rc dfv.ico dfv.dlg document.bmp folder.bmp dfv.h
    rc $(RCFLAGS) -r -fo$@ $*.rc

DFView.exe: $(OBJS) dfv.res
    $(LINK) @<<
        $(LINKFLAGS)
        -out:$@
        -map:$*.map
        $(OBJS)
        dfv.res
        $(LIBS)
<<
```

ABOUT.H   (DFVIEW Sample)

```
//
+-------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       about.h
//
//  Contents:   definition for an about dialog box class
//
//  Classes:    CAbout
//
//  Functions:
//
//  History:    6-08-94   stevebl   Created
//
//------------------------------------------------------------------------
--

#ifndef __ABOUT_H__
#define __ABOUT_H__

#include <cdialog.h>

#ifdef __cplusplus

//
+-------------------------------------------------------------------------
//
//  Class:      CAbout
//
//  Purpose:    implements the about dialog box
//
//  Interface:  DialogProc -- dialog procedure
//
//  History:    6-08-94   stevebl   Created
//
//------------------------------------------------------------------------
--

class CAbout: public CHlprDialog
{
public:
    BOOL DialogProc(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM lParam);
};

#endif //__cplusplus

#endif //__ABOUT_H__
```

ABOUT.CXX   (DFVIEW Sample)

```
//
+---------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       about.cxx
//
//  Contents:   implementation for a simple about dialog box
//
//  Classes:    CAbout
//
//  Functions:
//
//  History:    6-08-94   stevebl   Created
//
//---------------------------------------------------------------------------
--

#include <windows.h>
#include "about.h"


//
+---------------------------------------------------------------------------
//
//  Member:     CAbout::DialogProc
//
//  Synopsis:   dialog proc for the About dialog box
//
//  Arguments:  [uMsg]   - message
//              [wParam] - first message parameter
//              [lParam] - second message parameter
//
//  History:    4-12-94   stevebl   Created for MFract
//              6-08-94   stevebl   Stolen from MFract
//
//---------------------------------------------------------------------------
--

BOOL CAbout::DialogProc(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{
    switch (uMsg)
    {
    case WM_INITDIALOG:
        return(TRUE);
    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK
            || LOWORD(wParam) == IDCANCEL)
        {
            EndDialog(hwndDlg, TRUE);
            return(TRUE);
        }
    }
    return(FALSE);
}
```

BITMAPS.H   (DFVIEW Sample)

```
//
+---------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       bitmaps.h
//
//  Contents:   bitmap helper functions
//
//  Classes:
//
//  Functions:  DDBChangeColor
//              LoadAndStretch
//
//  History:    6-24-94   stevebl   Created
//
//--------------------------------------------------------------------------
--

#ifndef __BITMAPS_H__
#define __BITMAPS_H__

#define DSPDxax 0x00E20746L

BOOL DDBChangeColor (
    HBITMAP hBitmap,
    COLORREF crFrom,
    COLORREF crTo);

BOOL LoadAndStretch (
    HINSTANCE hinst,
    HBITMAP hbmpDest,
    LPTSTR lpszResource,
    UINT cxBitmap,
    UINT cyBitmap,
    COLORREF crHigh,
    COLORREF crNorm);

#endif //__BITMAPS_H__
```

BITMAPS.CXX   (DFVIEW Sample)

```
//
+--------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       bitmaps.cxx
//
//  Contents:   bitmap helper functions
//
//  Classes:
//
//  Functions:  DDBChangeColor
//              LoadAndStretch
//
//  History:    6-24-94   stevebl   Created
//
//--------------------------------------------------------------------------
--

#include <windows.h>
#include "bitmaps.h"

//
+--------------------------------------------------------------------------
//
//  Function:   LoadAndStretch
//
//  Synopsis:   Loads a bitmap from our resources, and creates a bitmap
//              "array."  The red portions of the bitmap are considered
//              clear.  Each resource consists of two bitmaps side by
//              side:  an "open folder" and a "closed folder".  These
//              bitmaps are turned into 4 bitmaps:  two selected, and
//              two unselected bitmaps.
//
//  Arguments:  [hinst]         - instance containing the bitmap
//              [hbmpDest]      - destination bitmap array
//              [lpszResource] - name of the bitmap resource
//              [cxBitmap]      - width to stretch to
//              [cyBitmap]      - height to stretch to
//              [crHigh]        - color of the highlighted background
//              [crNorm]        - color of the normal background
//
//  Returns:    TRUE on success.
//              FALSE on failure.
//
//  History:    6-24-94   stevebl   Stolen and modified from original DFView
//
//--------------------------------------------------------------------------
--

BOOL LoadAndStretch (
    HINSTANCE hinst,
    HBITMAP hbmpDest,
    LPTSTR lpszResource,
    UINT cxBitmap,
    UINT cyBitmap,
    COLORREF crHigh,
    COLORREF crNorm)
{
    HBITMAP hbmpOld1, hbmpOld2, hbmp;
    BITMAP bm;
    int i;
    HDC hdcDest = CreateCompatibleDC(NULL);
    HDC hdcSrc = CreateCompatibleDC(NULL);

    if (NULL == hdcDest || NULL == hdcSrc)
    {
        DeleteDC(hdcSrc);
        DeleteDC(hdcDest);
        return(FALSE);
    }
    for (i=0;  i < 2;  i++)
    {
        hbmp = LoadBitmap(hinst, lpszResource);
        if (NULL == hbmp)
        {
            DeleteDC(hdcSrc);
            DeleteDC(hdcDest);
            return FALSE;
        }

        DDBChangeColor(hbmp, RGB (255,0,0), ((i == 0) ? crHigh : crNorm));
        GetObject(hbmp, sizeof (bm), &bm);

        hbmpOld1 = (HBITMAP) SelectObject(hdcDest, hbmpDest);
        hbmpOld2 = (HBITMAP) SelectObject(hdcSrc, hbmp);

        SetStretchBltMode(hdcDest, COLORONCOLOR);
        StretchBlt(
            hdcDest,
            i * cxBitmap * 2,
            0,
            cxBitmap * 2,
            cyBitmap,
            hdcSrc,
            0,
            0,
            bm.bmWidth,
            bm.bmHeight,
            SRCCOPY);

        SelectObject(hdcDest, hbmpOld1);
        SelectObject(hdcSrc,  hbmpOld2);
        DeleteObject(hbmp);
    }
    DeleteDC(hdcSrc);
    DeleteDC(hdcDest);
    return TRUE;
}



//
+---------------------------------------------------------------------------
//
//  Function:   DDBChangeColor
//
//  Synopsis:   Change a particular color in a bitmap to another color.
//              Strategy is to create a monochrome mask, where each pixel
//              in the source bitmap that matches the color we're converting
//              from is set to white (1), and all other colors to black.
//              We then Blt this mask into the original bitmap, with a ROP
//              code that does:
//
//                (~Mask&Source) | (~Mask&Pattern)
//
//              where Pattern is the color we're changing to.
//
//              In other words, wherever the Mask is 1, we want to put the
pattern;
//              wherever the Mask is 0, we want to leave the source alone.
By
//              using a Truth Table, you'll find that this ROP code is
equivalent
//              to DSPDxax or ROP code 0x00E20746.
//
//              For info on figuring out ROP codes given a set of boolean
ops,
//              check out the Windows 3.0 SDK, Reference Volume 2, chapter
11.
//
//  Arguments:  [hBitmap] - bitmap handle
//              [crFrom]  - color to change
//              [crTo]    - new color
//
//  Returns:    TRUE on success.  FALSE on failur
//
//  History:    6-24-94   stevebl   Stolen from original DFView code
//
//--------------------------------------------------------------------------
--

BOOL DDBChangeColor (HBITMAP hBitmap, COLORREF crFrom, COLORREF crTo)
{
    register int cx, cy;
    BITMAP       bm;
    HDC          hdcBmp, hdcMask;
    HBITMAP      hbmMask, hbmOld1, hbmOld2;
    HBRUSH       hBrush, hbrOld;
    if (!hBitmap)
          return FALSE;

    GetObject (hBitmap, sizeof (bm), &bm);
    cx = bm.bmWidth;
    cy = bm.bmHeight;

    hbmMask = CreateBitmap(cx, cy, 1, 1, NULL);
    hdcMask = CreateCompatibleDC(NULL);
    hdcBmp = CreateCompatibleDC(NULL);
    hBrush = CreateSolidBrush(crTo);

    if (!hdcMask || !hdcBmp || !hBrush || !hbmMask)
    {
        DeleteObject(hbmMask);
        DeleteObject(hBrush);
        DeleteDC(hdcMask);
        DeleteDC(hdcBmp);
        return FALSE;
    }

    hbmOld1 = (HBITMAP) SelectObject (hdcBmp,  hBitmap);
    hbmOld2 = (HBITMAP) SelectObject (hdcMask, hbmMask);
    hbrOld  = (HBRUSH) SelectObject (hdcBmp, hBrush);

    SetBkColor(hdcBmp, crFrom);
    BitBlt(hdcMask, 0, 0, cx, cy, hdcBmp,  0, 0, SRCCOPY);
    SetBkColor(hdcBmp, RGB(255,255,255));
    BitBlt(hdcBmp,  0, 0, cx, cy, hdcMask, 0, 0, DSPDxax);

    SelectObject(hdcBmp,  hbrOld);
    SelectObject(hdcBmp,  hbmOld1);
    SelectObject(hdcMask, hbmOld2);
    DeleteDC(hdcBmp);
    DeleteDC(hdcMask);
    DeleteObject(hBrush);
    DeleteObject(hbmMask);

    return TRUE;
}
```

DFV.DLG (DFVIEW Sample)

```
1 DLGINCLUDE "dfv.h"

IDM_ABOUT DIALOG 67, 46, 173, 53
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About DocFile Viewer"
FONT 8, "MS Sans Serif"
BEGIN
    ICON            "AppIcon", -1, 8, 6, 18, 20
    LTEXT           "Windows NT DocFile Viewer - Version 2.0", -1, 32, 7,
                    139, 8, NOT WS_GROUP
    LTEXT           "Copyright 1994 Microsoft Corporation", -1, 32, 18, 139,
                    8, NOT WS_GROUP
    PUSHBUTTON      "OK", IDOK, 66, 33, 40, 14
END
```

DFV.H (DFVIEW Sample)

```
//
+---------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       DFV.h
//
//  Contents:   globally defined constants, IDs and structures
//
//  History:    6-08-94   stevebl   Created
//
//---------------------------------------------------------------------------
--

#ifndef __DFV_H__
#define __DFV_H__

#include <windows.h>
#include <ole2.h>
#include "message.h"

// String Constants
#define VER_FILEDESCRIPTION_STR     "Windows NT Docfile Viewer"
#define VER_INTERNALNAME_STR        "DocFile Viewer"
#define VER_ORIGINALFILENAME_STR    "DFVIEW.exe"
#define HELPFILE_STR                "OLETools.hlp"
#define MAIN_WINDOW_CLASS_NAME      "MainDFVWindow"
#define MAIN_WINDOW_CLASS_MENU      MainDFVMenu
#define MAIN_WINDOW_CLASS_MENU_STR  "MainDFVMenu"
#define STREAM_VIEW_CLASS_NAME      "Stream Viewer"

// Menu Command Identifiers
#define IDM_EXIT                    1000
#define IDM_OPEN                    1001
#define IDM_CLOSE                   1002
#define IDM_EXPAND                  1100
#define IDM_EXPANDBRANCH            1101
#define IDM_EXPANDALL               1102
#define IDM_COLLAPSE                1103
#define IDM_COLLAPSEALL             1104
#define IDM_TOGGLE                  1105
#define IDM_HELP                    1200
#define IDM_ABOUT                   1201

// Listbox Identifiers
#define IDC_LISTBOX                 4000
#define IDC_STREAMVIEW              4001

// Bitmap Identifiers
#define BMP_STREAM                  3000
#define BMP_STORAGE                 3001
```

```
// String Identifiers
#define IDS_ERROR                    2000
#define IDS_NOHELPFILE               2001
#define IDS_OLEINCOMPATIBLE          2002
#define IDS_OLEINITFAILED            2003
#define IDS_ENUMSTATSTGFAILED        2004
#define IDS_OPENSTORAGEFAILED        2005
#define IDS_OPENSTREAMFAILED         2006
#define IDS_ENUMELEMENTSFAILED       2007
#define IDS_INSERTSTRINGFAILED       2008
#define IDS_LOADBITMAPSFAILED        2009
#define IDS_STGOPENSTORAGEFAILED     2010
#define IDS_STMSTATFAILED            2011
#define IDS_STMSEEKFAILED            2012
#define IDS_STMREADFAILED            2013
#define IDS_STMTITLETEXT             2014
#define IDS_OUTOFMEMORY              2015

#endif // __DFV_H__
```

DFV.RC   (DFVIEW Sample)

```
//
+--------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       DFV.rc
//
//  Contents:   resources for Stereo Shop
//
//  History:    6-08-94   stevebl   Created
//
//--------------------------------------------------------------------------
--

#include "dfv.h"

#define VER_PRODUCTVERSION_STR      "3.50"
#define VER_COMPANYNAME_STR         "Microsoft Corporation"
#define VER_PRODUCTNAME_STR         "Microsoft\256 Windows NT(TM) Operating
System"
#define VER_LEGALCOPYRIGHT_YEARS    "1981-1994"
#define VER_LEGALCOPYRIGHT_STR "Copyright \251 Microsoft Corp. "
VER_LEGALCOPYRIGHT_YEARS
#define VER_PRODUCTNAME_STR "Microsoft\256 Windows NT(TM) Operating System"
#define VER_PRODUCTVERSION 3,50,01,001

VS_VERSION_INFO VERSIONINFO
FILEVERSION    VER_PRODUCTVERSION
PRODUCTVERSION VER_PRODUCTVERSION
FILEFLAGSMASK  VS_FFI_FILEFLAGSMASK
FILEFLAGS      0
FILEOS         VOS_NT_WINDOWS32
FILETYPE       VFT_APP
FILESUBTYPE    VFT2_UNKNOWN
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904B0"   /* LANG_ENGLISH/SUBLANG_ENGLISH_US, Unicode CP */
        BEGIN
            VALUE "CompanyName",     VER_COMPANYNAME_STR
            VALUE "FileDescription", VER_FILEDESCRIPTION_STR
            VALUE "FileVersion",     VER_PRODUCTVERSION
            VALUE "InternalName",    VER_INTERNALNAME_STR
            VALUE "LegalCopyright",  VER_LEGALCOPYRIGHT_STR
            VALUE "OriginalFilename",VER_ORIGINALFILENAME_STR
            VALUE "ProductName",     VER_PRODUCTNAME_STR
            VALUE "ProductVersion",  VER_PRODUCTVERSION_STR
        END

    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x0409, 0x04B0
    END
END

AppIcon ICON DFV.ico
StreamIcon ICON stream.ico

BMP_STORAGE     BITMAP  DISCARDABLE "FOLDER.BMP"
BMP_STREAM      BITMAP  DISCARDABLE "DOCUMENT.BMP"

MAIN_WINDOW_CLASS_MENU MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Open DocFile...", IDM_OPEN
        MENUITEM "&Close", IDM_CLOSE
        MENUITEM SEPARATOR
        MENUITEM "E&xit",IDM_EXIT
    END
    POPUP "&Tree"
    BEGIN
        MENUITEM "E&xpand  \a+", IDM_EXPAND
        MENUITEM "Expand &Branch  \a*", IDM_EXPANDBRANCH
        MENUITEM "Expand &All  \aCtrl+*", IDM_EXPANDALL
        MENUITEM "&Collapse Branch  \a-", IDM_COLLAPSE
        MENUITEM "C&ollapse All \aCtrl+-", IDM_COLLAPSEALL
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&Index...  \aF1", IDM_HELP
        MENUITEM SEPARATOR
        MENUITEM "&About DocFile Viewer...", IDM_ABOUT
    END
END

#include "DFV.dlg"

STRINGTABLE
BEGIN
    IDS_ERROR                   "Error"
    IDS_NOHELPFILE              "Can't find help file."
    IDS_OLEINCOMPATIBLE         "This program is incompatible with the
version of OLE installed on this system.  This program requires at least
OLE2 version %i - %i"
    IDS_OLEINITFAILED           "OLE initizlization failed."
    IDS_ENUMSTATSTGFAILED       "IStorage::EnumSTATSTG failed with HRESULT:
0x%08X"
    IDS_OPENSTORAGEFAILED       "IStorage::OpenStorage failed with HRESULT:
0x%08X"
    IDS_OPENSTREAMFAILED        "IStorage::OpenStream failed with HRESULT:
0x%08X"
    IDS_ENUMELEMENTSFAILED      "IStorage::EnumElements failed with HRESULT:
0x%08X"
```

```
    IDS_INSERTSTRINGFAILED      "Tree is too large.  Collapse some
branches."
    IDS_LOADBITMAPSFAILED       "Couldn't load folder and document bitmaps."
    IDS_STGOPENSTORAGEFAILED    "Unable to open %ws as a DocFile."
    IDS_STMSTATFAILED           "IStream::Stat failed with HRESULT: 0x%08X"
    IDS_STMSEEKFAILED           "IStream::Seek failed with HRESULT: 0x%08X"
    IDS_STMREADFAILED           "IStream::Read failed with HRESULT: 0x%08X"
    IDS_STMTITLETEXT            "Stream:"
    IDS_OUTOFMEMORY             "DFView ran out of memory.  Free up more
resources."
END

AppAccel ACCELERATORS
BEGIN
    VK_F1,          IDM_HELP,           VIRTKEY
    VK_ADD,         IDM_EXPAND,         VIRTKEY
    VK_SUBTRACT,    IDM_COLLAPSE,       VIRTKEY
    VK_SUBTRACT,    IDM_COLLAPSEALL,    CONTROL,    VIRTKEY
    VK_MULTIPLY,    IDM_EXPANDALL,      CONTROL,    VIRTKEY
    VK_MULTIPLY,    IDM_EXPANDBRANCH,   VIRTKEY
    "+",            IDM_EXPAND
    "-",            IDM_COLLAPSE
    "*",            IDM_EXPANDBRANCH
END
```

MESSAGE.H   (DFVIEW Sample)

```
//
+----------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       message.h
//
//  Contents:   Helper functions for popup message boxes.
//
//  Classes:
//
//  Functions:  MessageBoxFromStringIds
//
//  History:    6-24-94   stevebl   Created
//
//----------------------------------------------------------------------------
--

#ifndef __MESSAGE_H__
#define __MESSAGE_H__

#define MAX_STRING_LENGTH           256

int MessageBoxFromStringIds(
    const HWND hwndOwner,
    const HINSTANCE hinst,
    const UINT idText,
    const UINT idTitle,
    const UINT fuStyle);

int MessageBoxFromStringIdsAndArgs(
    const HWND hwndOwner,
    const HINSTANCE hinst,
    const UINT idText,
    const UINT idTitle,
    const UINT fuStyle,
    ...);

#endif //__MESSAGE_H__
```

MESSAGE.CXX   (DFVIEW Sample)

```
//
+---------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       message.cxx
//
//  Contents:   Helper functions for popup message boxes.
//
//  Classes:
//
//  Functions:  MessageBoxFromSringIds
//
//  History:    6-24-94   stevebl   Created
//
//---------------------------------------------------------------------------
--

#include <windows.h>
#include "message.h"

//
+---------------------------------------------------------------------------
//
//  Function:   MessageBoxFromStringIds
//
//  Synopsis:   Displays a simple message box taking its text from a string
//              table instead of from user allocated strings.
//
//  Arguments:  [hwndOwner]      - window handle for the message box's owner
//              [hinst]          - instance associated with the string table
//              [idText]         - string id for the box's text string
//              [idTitle]        - string id for the box's title string
//              [fuStyle]        - style of message box
//                                 (see Windows function MessageBox for
styles)
//
//  Returns:    result from MessageBox
//
//  History:    6-24-94   stevebl   Created
//
//  Notes:      Each string is limited to MAX_STRING_LENGTH characters.
//
//---------------------------------------------------------------------------
--

int MessageBoxFromStringIds(
    const HWND hwndOwner,
    const HINSTANCE hinst,
    const UINT idText,
    const UINT idTitle,
    const UINT fuStyle)
{
    int iReturn = 0;
    TCHAR szTitle[MAX_STRING_LENGTH];
    TCHAR szText[MAX_STRING_LENGTH];
    if (LoadString(hinst, idTitle, szTitle, MAX_STRING_LENGTH))
    {
        if (LoadString(hinst, idText, szText, MAX_STRING_LENGTH))
        {
            iReturn = MessageBox(
                hwndOwner,
                szText,
                szTitle,
                fuStyle);
        }
    }
    return(iReturn);
}

//
+---------------------------------------------------------------------------
//
//  Function:   MessageBoxFromStringIdsAndArgs
//
//  Synopsis:   Creates a message box from a pair of string IDs similar
//              to MessageBoxFromStringIds.  The principle difference
//              is that idFormat is an ID for a string which is a printf
//              format string suitable for passing to wsprintf.
//              The variable argument list is combined with the format
//              string to create the text of the message box.
//
//  Arguments:  [hwndOwner] - window handle for the message box's owner
//              [hinst]     - instance associated with the string table
//              [idFormat]  - string id for the format of the box's text
//              [idTitle]   - string id for the box's title string
//              [fuStyle]   - style of the dialog box
//              [...]       - argument list for text format string
//
//  Returns:    result from MessageBox
//
//  History:    6-24-94   stevebl   Created
//
//  Notes:      Neither the composed text string nor the title must be
//              longer than MAX_STRING_LENGTH characters.
//
//----------------------------------------------------------------------------
--

int MessageBoxFromStringIdsAndArgs(
    const HWND hwndOwner,
    const HINSTANCE hinst,
    const UINT idFormat,
    const UINT idTitle,
    const UINT fuStyle, ...)
{
    int iReturn = 0;
    va_list arglist;
    va_start(arglist, fuStyle);
    TCHAR szTitle[MAX_STRING_LENGTH];
    TCHAR szText[MAX_STRING_LENGTH];
    TCHAR szFormat[MAX_STRING_LENGTH];
    if (LoadString(hinst, idTitle, szTitle, MAX_STRING_LENGTH))
    {
        if (LoadString(hinst, idFormat, szFormat, MAX_STRING_LENGTH))
        {
            wvsprintf(szText, szFormat, arglist);
            iReturn = MessageBox(
                hwndOwner,
                szText,
                szTitle,
                fuStyle);
        }
    }
    return(iReturn);
}
```

MWCLASS.H (DFVIEW Sample)

```
//
+---------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       mwclass.h
//
//  Contents:   definition of the main window class
//
//  Classes:    CMainWindow
//
//  Functions:  Exists
//
//  History:    6-08-94   stevebl   Created
//
//---------------------------------------------------------------------------
--

#ifndef __MWCLASS_H__
#define __MWCLASS_H__

#include <cwindow.h>
#include <commdlg.h>

#ifdef __cplusplus

int Exists(TCHAR *sz);


// List Box Element Structure
struct TAG_LISTBOXELEMENT
{
#define LBE_STORAGE     STGTY_STORAGE
#define LBE_STREAM      STGTY_STREAM
    unsigned uType;
    BOOL fOpen;
    void * pInterface;
    union {
        ULONG nChildren;
        HWND hwndView;
    };
    unsigned uIndent;
    LPOLESTR pwcsName;
};
typedef struct TAG_LISTBOXELEMENT LISTBOXELEMENT;

//
+---------------------------------------------------------------------------
//
//  Class:      CMainWindow
//
```

```
//  Purpose:    Code for the main docfile viewer window and the main menu.
//
//  Interface:  CMainWindow          -- constructor
//              InitInstance         -- instantiates the docfile viewer
window
//              Toggle               -- toggles a selection
//              Expand               -- expands a selection
//              Collapse             -- collapses a selection
//              ExpandAll            -- expands an entire tree
//              OpenFile             -- trys to load the docfile in
_szFileName
//
//  History:    6-27-94   stevebl   Created
//
//  Notes:      only the public interface is listed here
//
//----------------------------------------------------------------------------
--

class CMainWindow: public CHlprWindow
{
public:
    CMainWindow();
    BOOL InitInstance(HINSTANCE, int);
    void Toggle(long lIndex);
    void Expand(long lIndex);
    void Collapse(long lIndex);
    void ExpandAll(long lIndex);
    void OpenFile(void);
    TCHAR _szFileName[MAX_PATH];
protected:
    ~CMainWindow();
    LRESULT WindowProc(UINT uMsg, WPARAM wParam, LPARAM lParam);
private:
    LRESULT DoMenu(WPARAM wParam, LPARAM lParam);
    LISTBOXELEMENT * CreateStreamElement(IStream * pstm);
    LISTBOXELEMENT * CreateStorageElement(IStorage * pstg);
    void DeleteElement(LISTBOXELEMENT * plbe);
    BOOL LoadBitmaps(void);

    HWND _hlb;
    HBITMAP _hbmpStorage;
    HBITMAP _hbmpStream;
    COLORREF _crTextHigh;
    COLORREF _crBkHigh;
    COLORREF _crTextNorm;
    COLORREF _crBkNorm;
    WORD _wWidth;
    OPENFILENAME _ofnFile;
    TCHAR _szFileTitle[MAX_PATH];
    TEXTMETRIC _tm;
    UINT _cyBitmap;
    TCHAR _szHelpFile[MAX_PATH];
};
```

```
#endif // __cplusplus

#endif // __MWCLASS_H__
```

MWCLASS.CXX   (DFVIEW Sample)

```
//
+---------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       mwclass.cxx
//
//  Contents:   implementation for the main window class
//
//  Classes:    CMainWindow
//
//  Functions:  Exists
//
//  History:    6-08-94   stevebl   Created
//
//----------------------------------------------------------------------------
--

#include "dfv.h"
#include "mwclass.h"
#include "about.h"
#include "bitmaps.h"
#include "strmvwr.h"
#include <assert.h>

#define BITMAPWIDTH 16
#define BITMAPHEIGHT 16
#define INDENT BITMAPWIDTH
#define ROOTSTGSTR OLESTR(" \\ - ")

//
+---------------------------------------------------------------------------
//
//  Member:     CMainWindow::CMainWindow
//
//  Synopsis:   constructor
//
//  History:    6-27-94   stevebl   Created
//
//----------------------------------------------------------------------------
--

CMainWindow::CMainWindow()
{
    _hlb = NULL;
    _hbmpStorage = _hbmpStream = NULL;
}

//
+---------------------------------------------------------------------------
//
```

```
//  Member:     CMainWindow::~CMainWindow
//
//  Synopsis:   destructor
//
//  History:    6-27-94   stevebl   Created
//
//  Notes:      Destruction of the main window indicates that the app should
//              quit.
//
//----------------------------------------------------------------------------
--

CMainWindow::~CMainWindow()
{
    DeleteObject(_hbmpStorage);
    DeleteObject(_hbmpStream);
    PostQuitMessage(0);
}

//
+----------------------------------------------------------------------------
//
//  Member:     CMainWindow::InitInstance
//
//  Synopsis:   Instantiates an instance of the docfile viewer window.
//
//  Arguments:  [hInstance] - instance of the app
//              [nCmdShow]  - command to pass to ShowWindow
//
//  Returns:    TRUE on success
//              FALSE on failure
//
//  History:    6-27-94   stevebl   Created
//
//  Notes:      This method must be called only once, immediately after
//              class construction.
//
//----------------------------------------------------------------------------
--

BOOL CMainWindow::InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    // Note, the Create method sets the _hwnd member for me so I don't
    // need to set it myself.
    if (!Create(
        TEXT(MAIN_WINDOW_CLASS_NAME),
        TEXT(VER_INTERNALNAME_STR),
        WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX
            | WS_MAXIMIZEBOX | WS_THICKFRAME,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        NULL,
        NULL,
        hInstance))
    {
        return(FALSE);
    }

    ShowWindow(_hwnd, nCmdShow);
    UpdateWindow(_hwnd);

    // build a path to where the help file should be
    // (it should be in the same directory as the .EXE)
    // first get the exe's path
    DWORD cch = GetModuleFileName(hInstance, _szHelpFile, MAX_PATH);
    // then back up past the name of the application
    while (cch > 0 &&
        _szHelpFile[cch - 1] != TEXT('\\') &&
        _szHelpFile[cch - 1] != TEXT('/') &&
        _szHelpFile[cch - 1] != TEXT(':'))
    {
        cch--;
    }
    // finally copy the help file's name at the end of the path
    lstrcpy(&_szHelpFile[cch], TEXT(HELPFILE_STR));
    return(TRUE);
}

//
+---------------------------------------------------------------------------
//
//  Member:     CMainWindow::WindowProc
//
//  Synopsis:   main window procedure
//
//  Arguments:  [uMsg]   - Windows message
//              [wParam] - first message parameter
//              [lParam] - second message parameter
//
//  History:    6-27-94   stevebl   Created
//
//  Notes:      See CHlprWindow for a description of how this method gets
//              called by the global WinProc.
//
//--------------------------------------------------------------------------
--

LRESULT CMainWindow::WindowProc(UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch (uMsg)
    {
    case WM_CREATE:
        {
            RECT rc;
            GetClientRect(_hwnd, &rc);
            // record the width before we create the list box
            _wWidth = (WORD) rc.right;
            HDC hdc = GetDC(_hwnd);
            GetTextMetrics(hdc, &_tm);
            ReleaseDC(_hwnd, hdc);

            // create the child listbox
            _hlb = CreateWindow(
                TEXT("listbox"),
                TEXT(""),
                LBS_OWNERDRAWFIXED |
                    WS_CHILD |
                    WS_VISIBLE |
                    WS_VSCROLL |
                    LBS_NOTIFY |
                    LBS_NOINTEGRALHEIGHT |
                    LBS_WANTKEYBOARDINPUT,
                0,
                0,
                rc.right,
                rc.bottom,
                _hwnd,
                (HMENU) IDC_LISTBOX,
                _hInstance,
                NULL);
            if (NULL == _hlb)
            {
                // abort window creation
                return(-1);
            }
            SetFocus(_hlb);

            if (!LoadBitmaps())
            {
                MessageBoxFromStringIds(
                    _hwnd,
                    _hInstance,
                    IDS_LOADBITMAPSFAILED,
                    IDS_ERROR,
                    MB_OK | MB_ICONEXCLAMATION);
                return(-1);
            }
            _crTextHigh = GetSysColor(COLOR_HIGHLIGHTTEXT);
            _crTextNorm = GetSysColor(COLOR_WINDOWTEXT);
            _crBkHigh = GetSysColor(COLOR_HIGHLIGHT);
            _crBkNorm = GetSysColor(COLOR_WINDOW);

            // Fill in the File's OPENFILENAME structure
            _szFileName[0] = 0;
            _szFileTitle[0] = 0;
            _ofnFile.lStructSize = sizeof(OPENFILENAME);
            _ofnFile.hwndOwner = _hwnd;
            _ofnFile.lpstrFilter = TEXT("All Files (*.*)\0*.*\0Doc Files
(*.DFL;*.OL?)\0*.DFL;*.OL;\0");
            _ofnFile.lpstrCustomFilter = NULL;
            _ofnFile.nMaxCustFilter = 0;
            _ofnFile.nFilterIndex = 1;
            _ofnFile.lpstrFile = _szFileName;
            _ofnFile.nMaxFile = MAX_PATH;
            _ofnFile.lpstrInitialDir = NULL;
            _ofnFile.lpstrFileTitle = _szFileTitle;
            _ofnFile.nMaxFileTitle = MAX_PATH;
            _ofnFile.lpstrTitle = TEXT("Open DocFile");
            _ofnFile.lpstrDefExt = NULL;
            _ofnFile.Flags = OFN_HIDEREADONLY;
        }
        break;
    case WM_SETFOCUS:
        {
            if (_hlb)
            {
                SetFocus(_hlb);
            }
        break;
        }
    case WM_MEASUREITEM:
        {
            LPMEASUREITEMSTRUCT lpmis = (LPMEASUREITEMSTRUCT) lParam;
            if (BITMAPHEIGHT < _tm.tmHeight + _tm.tmExternalLeading)
            {
                lpmis->itemHeight = _tm.tmHeight + _tm.tmExternalLeading;
            }
            else
            {
                lpmis->itemHeight = BITMAPHEIGHT;
            }
            lpmis->itemWidth = _wWidth;
            return(TRUE);
        }
    case WM_DELETEITEM:
        if (IDC_LISTBOX == wParam)
        {
            DeleteElement((LISTBOXELEMENT *)((LPDELETEITEMSTRUCT)lParam)-
>itemData);
        }
        break;
    case WM_DRAWITEM:
        {
            LPDRAWITEMSTRUCT lpdis = (LPDRAWITEMSTRUCT) lParam;
            if (-1 == lpdis->itemID)
            {
                // no items in the listbox
                break;
            }
            // draw the item
            switch (lpdis->itemAction)
            {
            case ODA_SELECT:
            case ODA_DRAWENTIRE:
                {
                    // draw the listbox item
                    LISTBOXELEMENT * plbe = (LISTBOXELEMENT *) lpdis-
>itemData;
                    assert(plbe);

                    // compute the starting point
                    int x = lpdis->rcItem.left + plbe->uIndent * INDENT;
                    int y = lpdis->rcItem.top;

                    // paint the bitmap
                    HDC hMemDC = CreateCompatibleDC(lpdis->hDC);
                    HBITMAP hbmp;
                    switch (plbe->uType)
                    {
                    case LBE_STORAGE:
                        hbmp = _hbmpStorage;
                        break;
                    case LBE_STREAM:
                        hbmp = _hbmpStream;
                        break;
                    default:
                        assert(NULL == "Unsupported type");
                        hbmp = _hbmpStream;
                        break;
                    }
                    HBITMAP hbmpOld = (HBITMAP) SelectObject(hMemDC, hbmp);
                    int nBitmap = ((lpdis->itemState & ODS_SELECTED) ? 0 :
2) +

                        plbe->fOpen;
                    BitBlt(
                        lpdis->hDC,
                        x,
                        y,
                        BITMAPWIDTH,
                        _cyBitmap,
                        hMemDC,
                        nBitmap * BITMAPWIDTH, 0, SRCCOPY);
                    SelectObject(hMemDC, hbmpOld);
                    DeleteDC(hMemDC);
                    x += BITMAPWIDTH;

                    // paint the text
                    SIZE sizeText;
                    RECT rcItem = lpdis->rcItem;
                    rcItem.left = x;

                    // Note that pwcsName is an OLEStr and
                    // is therefore always wide under WIN32.
                    GetTextExtentPoint32W(
                        lpdis->hDC,
                        plbe->pwcsName,
                        wcslen(plbe->pwcsName),
                        &sizeText);
                    rcItem.right = x + sizeText.cx;
                    COLORREF crOldText = SetTextColor(
                        lpdis->hDC,
                        (lpdis->itemState & ODS_SELECTED) ? _crTextHigh :
_crTextNorm);
                    COLORREF crOldBack = SetBkColor(
                        lpdis->hDC,
                        (lpdis->itemState & ODS_SELECTED) ? _crBkHigh :
_crBkNorm);
                    ExtTextOutW(
                        lpdis->hDC,
                        x,
                        y,
                        ETO_OPAQUE | ETO_CLIPPED,
                        &rcItem,
                        plbe->pwcsName,
                        wcslen(plbe->pwcsName),
                        NULL);
                    SetTextColor(lpdis->hDC, crOldText);
                    SetBkColor(lpdis->hDC, crOldBack);
                }
                break;
            case ODA_FOCUS:
                break;
            }
            return(TRUE);
        }
    case WM_VKEYTOITEM:
        if ((HWND)lParam == _hlb)
        {
            if (LOWORD(wParam) == VK_RETURN)
            {
                if(SendMessage(_hlb, LB_GETCOUNT, 0, 0))
                {
                    Toggle(SendMessage(_hlb, LB_GETCURSEL, 0, 0));
                    InvalidateRect(_hlb, NULL, FALSE);
                }
                return(-2);
            }
            return(-1);
        }
        break;
    case WM_SIZE:
        {
            _wWidth = LOWORD(lParam);
            WORD wHeight = HIWORD(lParam);
            if (_hlb)
            {
                SetWindowPos(_hlb, NULL, 0, 0, _wWidth, wHeight, SWP_NOMOVE
| SWP_NOZORDER);
            }
            break;
        }
    case WM_COMMAND:
        if ((LOWORD(wParam) == IDC_LISTBOX) && ((HWND) lParam == _hlb))
        {
            switch (HIWORD(wParam))
            {
                case LBN_DBLCLK:
                    {
                        if(SendMessage(_hlb, LB_GETCOUNT, 0, 0))
                        {
                            Toggle(SendMessage(_hlb, LB_GETCURSEL, 0, 0));
                            InvalidateRect(_hlb, NULL, FALSE);
                        }
                    }
                    break;
                default:
                    break;
            }
        }
        return DoMenu(wParam, lParam);
    case WM_QUIT:
    case WM_CLOSE:
    default:
        return(DefWindowProc(_hwnd, uMsg, wParam, lParam));
    }
    return(FALSE);
}

//
+---------------------------------------------------------------------------
//
//  Member:     CMainWindow::DoMenu
//
//  Synopsis:   implements the main menu commands
//
//  Arguments:  [wParam] - first window parameter
//              [lParam] - second window parameter
//
//  History:    6-23-94   stevebl   Created
//
//--------------------------------------------------------------------------
--

LRESULT CMainWindow::DoMenu(WPARAM wParam, LPARAM lParam)
{
    switch (LOWORD(wParam))
    {
    case IDM_OPEN:
        if (!GetOpenFileName((LPOPENFILENAME)&_ofnFile))
        {
            return(FALSE);
        }
        OpenFile();
        break;
    case IDM_CLOSE:
        SendMessage(_hlb, LB_RESETCONTENT, 0, 0);
        break;
    case IDM_EXPAND:
        if(SendMessage(_hlb, LB_GETCOUNT, 0, 0))
        {
            Expand(SendMessage(_hlb, LB_GETCURSEL, 0, 0));
            InvalidateRect(_hlb, NULL, FALSE);
        }
        break;
    case IDM_EXPANDBRANCH:
        if(SendMessage(_hlb, LB_GETCOUNT, 0, 0))
        {
            ExpandAll(SendMessage(_hlb, LB_GETCURSEL, 0, 0));
            InvalidateRect(_hlb, NULL, FALSE);
        }
        break;
    case IDM_EXPANDALL:
        if(SendMessage(_hlb, LB_GETCOUNT, 0, 0))
        {
            ExpandAll(0);
            InvalidateRect(_hlb, NULL, FALSE);
        }
        break;
    case IDM_COLLAPSE:
        if(SendMessage(_hlb, LB_GETCOUNT, 0, 0))
        {
            Collapse(SendMessage(_hlb, LB_GETCURSEL, 0, 0));
            InvalidateRect(_hlb, NULL, FALSE);
        }
        break;
    case IDM_COLLAPSEALL:
        if(SendMessage(_hlb, LB_GETCOUNT, 0, 0))
        {
            Collapse(0);
            SendMessage(_hlb, LB_SETCURSEL, 0, 0);
            InvalidateRect(_hlb, NULL, FALSE);
        }
        break;
    case IDM_TOGGLE:
        if (SendMessage(_hlb, LB_GETCOUNT, 0, 0))
        {
            Toggle(SendMessage(_hlb, LB_GETCURSEL, 0, 0));
            InvalidateRect(_hlb, NULL, FALSE);
        }
    case IDM_EXIT:
        SendMessage(_hwnd, WM_CLOSE, 0, 0);
        break;
    case IDM_HELP:
        if (!Exists(_szHelpFile))
        {
            MessageBoxFromStringIds(
                _hwnd,
                _hInstance,
                IDS_NOHELPFILE,
                IDS_ERROR,
                MB_OK | MB_ICONEXCLAMATION);
        }
        else
        {
            WinHelp(_hwnd, _szHelpFile, HELP_CONTEXT, 3);
        }
        break;
    case IDM_ABOUT:
        {
            CAbout dlgAbout;
            dlgAbout.ShowDialog(_hInstance, MAKEINTRESOURCE(IDM_ABOUT),
_hwnd);
        }
        break;
    default:
        return(DefWindowProc(_hwnd, WM_COMMAND, wParam, lParam));
    }
    return(FALSE);
}

//
+---------------------------------------------------------------------------
//
//  Member:     CMainWindow::Expand
//
//  Synopsis:   Expands a level of the tree.
//
//  Arguments:  [lIndex] - index of the item that needs to be expanded
//
//  History:    6-23-94   stevebl   Created
//
//--------------------------------------------------------------------------
--

void CMainWindow::Expand(long lIndex)
{
    LISTBOXELEMENT * pElement = (LISTBOXELEMENT *) SendMessage(_hlb,
LB_GETITEMDATA, lIndex, 0);
    if (pElement->fOpen)
    {
        // this element has already been expanded
        return;
    }
    SendMessage(_hlb, WM_SETREDRAW, FALSE, 0);
    switch (pElement->uType)
    {
    case LBE_STORAGE:
        {
            // create entries for all of its children
            IStorage * pstg = (IStorage *) pElement->pInterface;
            IEnumSTATSTG * penumStatStg;
            STATSTG statstg;
            HRESULT hr = pstg->EnumElements(0, NULL, 0, &penumStatStg);
            if (SUCCEEDED(hr))
            {
                pElement->fOpen = TRUE;
                pElement->nChildren = 0;
                while (hr != S_FALSE)
                {
                    hr = penumStatStg->Next(1, &statstg, NULL);
                    if (FAILED(hr))
```

```
{
    MessageBoxFromStringIdsAndArgs(
        _hwnd,
        _hInstance,
        IDS_ENUMSTATSTGFAILED,
        IDS_ERROR,
        MB_OK | MB_ICONEXCLAMATION,
        hr);
}
else if (S_OK == hr)
{
    // retrieved another member from the list
    LISTBOXELEMENT * plbe;
    switch (statstg.type)
    {
    case STGTY_STORAGE:
        {
            IStorage * pstg2;
            hr = pstg->OpenStorage(
                statstg.pwcsName,
                NULL,
                STGM_READWRITE | STGM_SHARE_EXCLUSIVE,
                NULL,
                0,
                &pstg2);
            if (SUCCEEDED(hr))
            {
                plbe = CreateStorageElement(pstg2);
            }
            else
            {
                MessageBoxFromStringIdsAndArgs(
                    _hwnd,
                    _hInstance,
                    IDS_OPENSTORAGEFAILED,
                    IDS_ERROR,
                    MB_OK | MB_ICONEXCLAMATION,
                    hr);
                plbe = NULL;
            }
        }
        break;
    case STGTY_STREAM:
        {
            IStream * pstm;
            hr = pstg->OpenStream(
                statstg.pwcsName,
                0,
                STGM_READWRITE | STGM_SHARE_EXCLUSIVE,
                0,
                &pstm);
            if (SUCCEEDED(hr))
            {
                plbe = CreateStreamElement(pstm);
            }
```

```
                                else
                                {
                                    MessageBoxFromStringIdsAndArgs(
                                        _hwnd,
                                        _hInstance,
                                        IDS_OPENSTREAMFAILED,
                                        IDS_ERROR,
                                        MB_OK | MB_ICONEXCLAMATION,
                                        hr);
                                    plbe = NULL;
                                }
                            }
                            break;
                        }
                        if (plbe)
                        {
                            plbe->pwcsName = statstg.pwcsName;
                            plbe->uIndent = pElement->uIndent + 1;
                            pElement->nChildren++;
                            if (SendMessage(_hlb, LB_INSERTSTRING, lIndex +
pElement->nChildren, (LPARAM) plbe) == LB_ERRSPACE)
                            {
                                MessageBoxFromStringIds(
                                    _hwnd,
                                    _hInstance,
                                    IDS_INSERTSTRINGFAILED,
                                    IDS_ERROR,
                                    MB_OK | MB_ICONEXCLAMATION);
                                DeleteElement(plbe);
                                pElement->nChildren--;
                            }
                        }
                    }
                }
                penumStatStg->Release();
            }
            else
            {
                MessageBoxFromStringIdsAndArgs(
                    _hwnd,
                    _hInstance,
                    IDS_ENUMELEMENTSFAILED,
                    IDS_ERROR,
                    MB_OK | MB_ICONEXCLAMATION,
                    hr);
            }
        }
        break;
    case LBE_STREAM:
        // streams are only expanded if they are the currently selected list
item
        if (SendMessage(_hlb, LB_GETCURSEL, 0, 0) == lIndex)
        {
            // open stream
            CStreamView * psv = new CStreamView(_hlb, pElement);
            if (psv)
            {
                pElement->fOpen = psv->InitInstance(_hInstance,
SW_SHOWNORMAL);
                if (pElement->fOpen)
                {
                    pElement->hwndView = psv->GetHwnd();
                    SetFocus(pElement->hwndView);
                }
            }
        }
        break;
    }
    SendMessage(_hlb, WM_SETREDRAW, TRUE, 0);
}

//
+---------------------------------------------------------------------------
//
//  Member:     CMainWindow::Collapse
//
//  Synopsis:   collapses a level of the tree
//
//  Arguments:  [lIndex] - index of the item to be collapsed
//
//  History:    6-23-94   stevebl   Created
//
//--------------------------------------------------------------------------
--

void CMainWindow::Collapse(long lIndex)
{
    LISTBOXELEMENT * pElement = (LISTBOXELEMENT *) SendMessage(_hlb,
LB_GETITEMDATA, lIndex, 0);
    if (pElement->fOpen)
    {
        switch (pElement->uType)
        {
        case LBE_STORAGE:
            while (pElement->nChildren--)
            {
                Collapse(lIndex + 1);
                SendMessage(_hlb, LB_DELETESTRING, lIndex + 1, 0);
            }
            break;
        case LBE_STREAM:
            if (pElement->fOpen)
            {
                DestroyWindow(pElement->hwndView);
            }
            break;
        }
        pElement->fOpen = FALSE;
    }
}
```

```
//
+--------------------------------------------------------------------------
//
//  Member:     CMainWindow::ExpandAll
//
//  Synopsis:   expands the current item and all its children
//
//  Arguments:  [lIndex] - index of the item to be expanded
//
//  History:    6-23-94   stevebl   Created
//
//--------------------------------------------------------------------------
--

void CMainWindow::ExpandAll(long lIndex)
{
    LISTBOXELEMENT * pElement = (LISTBOXELEMENT *) SendMessage(_hlb,
LB_GETITEMDATA, lIndex, 0);
    Expand(lIndex);
    if (pElement->uType == LBE_STORAGE)
    {
        unsigned nChildren = pElement->nChildren;
        while (nChildren--)
        {
            ExpandAll(lIndex + 1 + nChildren);
        }
    }
}

//
+--------------------------------------------------------------------------
//
//  Member:     CMainWindow::Toggle
//
//  Synopsis:   toggles a node - expanding or collapsing as appropriate
//
//  Arguments:  [lIndex] - index of the node
//
//  History:    6-25-94   stevebl   Created
//
//--------------------------------------------------------------------------
--

void CMainWindow::Toggle(long lIndex)
{
    LISTBOXELEMENT * pElement = (LISTBOXELEMENT *) SendMessage(_hlb,
LB_GETITEMDATA, lIndex, 0);
    if (pElement->fOpen)
    {
        Collapse(lIndex);
    }
    else
    {
        Expand(lIndex);
    }
}

//
+---------------------------------------------------------------------------
//
//  Member:     CMainWindow::CreateStreamElement
//
//  Synopsis:   turns an IStream pointer into a listbox element
//
//  Arguments:  [pstm] - IStream pointer
//
//  Returns:    pointer to the newly created element
//              (NULL on failure)
//
//  History:    6-23-94   stevebl   Created
//
//--------------------------------------------------------------------------
--

LISTBOXELEMENT * CMainWindow::CreateStreamElement(IStream * pstm)
{
    LISTBOXELEMENT * plbe = new LISTBOXELEMENT;
    if (plbe)
    {
        plbe->uType = LBE_STREAM;
        plbe->fOpen = FALSE;
        plbe->pInterface = (void *) pstm;
        plbe->uIndent = 0;
        plbe->hwndView = NULL;
    }
    else
    {
        MessageBoxFromStringIds(
            _hwnd,
            _hInstance,
            IDS_OUTOFMEMORY,
            IDS_ERROR,
            MB_OK | MB_ICONEXCLAMATION);
    }
    return(plbe);
}

//
+---------------------------------------------------------------------------
//
//  Member:     CMainWindow::CreateStorageElement
//
//  Synopsis:   turns an IStorage pointer into a listbox element
//
//  Arguments:  [pstg] - IStorage pointer
//
//  Returns:    pointer to the newly created element
//              (NULL on failure)
//
```

```
//  History:    6-23-94   stevebl   Created
//
//----------------------------------------------------------------------------
--

LISTBOXELEMENT * CMainWindow::CreateStorageElement(IStorage * pstg)
{
    LISTBOXELEMENT * plbe = new LISTBOXELEMENT;
    if (plbe)
    {
        plbe->uType = LBE_STORAGE;
        plbe->fOpen = FALSE;
        plbe->pInterface = (void *) pstg;
        plbe->uIndent = 0;
        plbe->nChildren = 0;
    }
    else
    {
        MessageBoxFromStringIds(
            _hwnd,
            _hInstance,
            IDS_OUTOFMEMORY,
            IDS_ERROR,
            MB_OK | MB_ICONEXCLAMATION);
    }
    return(plbe);
}

//
+----------------------------------------------------------------------------
//
//  Member:     CMainWindow::DeleteElement
//
//  Synopsis:   deletes a listbox element and it's associated data
//
//  Arguments:  [plbe] - pointer to the element
//
//  History:    6-23-94   stevebl   Created
//
//  Notes:      This doesn't remove it from the list, it just frees up
//              the LISTBOXELEMENT pointer and it's data.
//
//----------------------------------------------------------------------------
--

void CMainWindow::DeleteElement(LISTBOXELEMENT * plbe)
{

    switch (plbe->uType)
    {
    case LBE_STORAGE:
        ((IStorage *) plbe->pInterface)->Release();
        break;
    case LBE_STREAM:
        if (plbe->fOpen)
        {
            DestroyWindow(plbe->hwndView);
        }
        ((IStream *) plbe->pInterface)->Release();
        break;
    }
    LPMALLOC pMalloc;
    CoGetMalloc(MEMCTX_TASK, &pMalloc);
    pMalloc->Free(plbe->pwcsName);
    pMalloc->Release();
    delete(plbe);
}

//
+---------------------------------------------------------------------------
//
//  Member:     CMainWindow::LoadBitmaps
//
//  Synopsis:   loads the Storage and Stream bitmap arrays
//
//  Returns:    TRUE on success
//              FALSE on failure
//
//  Modifies:   _hbmpStorage and _hbmpStream
//
//  History:    6-24-94   stevebl   Created from code used in original
DFView
//
//--------------------------------------------------------------------------
--

BOOL CMainWindow::LoadBitmaps(void)
{
    //
    // Get some metrics/info
    //
    UINT cxBitmap = BITMAPWIDTH;
    _cyBitmap = BITMAPHEIGHT;
    if (_cyBitmap < (UINT) _tm.tmHeight + _tm.tmExternalLeading)
    {
        _cyBitmap = _tm.tmHeight + _tm.tmExternalLeading;
    }
    UINT crHigh   = (COLORREF) GetSysColor(COLOR_HIGHLIGHT);
    UINT crNorm   = (COLORREF) GetSysColor(COLOR_WINDOW);

    HDC hdc = CreateCompatibleDC(NULL);
    if (!hdc)
    {
        goto ERRLOADBITMAP;
    }
    _hbmpStorage = CreateBitmap(
        4 * cxBitmap,
        _cyBitmap,
        GetDeviceCaps(hdc, PLANES),
        GetDeviceCaps(hdc, BITSPIXEL),
        NULL);
    _hbmpStream = CreateBitmap(
        4 * cxBitmap,
        _cyBitmap,
        GetDeviceCaps(hdc, PLANES),
        GetDeviceCaps (hdc, BITSPIXEL),
        NULL);
   DeleteDC(hdc);
   if (!_hbmpStorage || !_hbmpStream)
   {
        goto ERRLOADBITMAP;
   }

    //
    //  Load our bitmaps, and munge them to create
    //  our 4-bitmap "arrays".  Anything RED in the loaded
    //  bitmap will be considered clear.
    //
    if (!LoadAndStretch(
        _hInstance,
        _hbmpStorage,
        MAKEINTRESOURCE(BMP_STORAGE),
        cxBitmap,
        _cyBitmap,
        crHigh,
        crNorm))
    {
        goto ERRLOADBITMAP;
    }
    if (!LoadAndStretch(
        _hInstance,
        _hbmpStream,
        MAKEINTRESOURCE(BMP_STREAM),
        cxBitmap,
        _cyBitmap,
        crHigh,
        crNorm))
    {
        goto ERRLOADBITMAP;
    }
    return TRUE;

ERRLOADBITMAP:
    //
    // Bitmap loading error.
    //
    DeleteObject(_hbmpStorage);
    DeleteObject(_hbmpStream);
    _hbmpStorage = NULL;
    _hbmpStream = NULL;

    return FALSE;
}
```

```
//
+---------------------------------------------------------------------------
//
//  Member:     CMainWindow::OpenFile
//
//  Synopsis:   attempts to open whatever docfile is named by _szFileName;
//
//  History:    6-24-94   stevebl   Created
//
//----------------------------------------------------------------------------
--

void CMainWindow::OpenFile(void)
{
    SendMessage(_hlb, LB_RESETCONTENT, 0, 0);
    IStorage * pstg;
#ifndef UNICODE
    // convert file name to a wide string
    OLECHAR wszFileName[MAX_PATH];
    mbstowcs(wszFileName, _szFileName, MAX_PATH);
#endif
    // get IStorage
    HRESULT hr = StgOpenStorage(
#ifdef UNICODE
        _szFileName,
#else
        wszFileName,
#endif
        NULL,
        STGM_TRANSACTED | STGM_READWRITE | STGM_SHARE_EXCLUSIVE,
        NULL,
        0,
        &pstg);
    if (FAILED(hr))
    {
        MessageBoxFromStringIdsAndArgs(
            _hwnd,
            _hInstance,
            IDS_STGOPENSTORAGEFAILED,
            IDS_ERROR,
            MB_OK | MB_ICONEXCLAMATION,
#ifdef UNICODE
            _szFileName
#else
            wszFileName
#endif
            );
        return;
    }
    // Storage was opened successfully.
    LISTBOXELEMENT * plbe = CreateStorageElement(pstg);
    if (plbe)
    {
        IMalloc * pMalloc;
        CoGetMalloc(MEMCTX_TASK, &pMalloc);
        plbe->pwcsName = (LPOLESTR) pMalloc->Alloc(sizeof(OLECHAR) *
(lstrlen(_szFileName) + wcslen(ROOTSTGSTR) + 1));
        pMalloc->Release();
        if (plbe->pwcsName)
        {
            wcscpy(plbe->pwcsName, ROOTSTGSTR);
#ifdef UNICODE
            wcscpy(&plbe->pwcsName[wcslen(ROOTSTGSTR)], _szFileName);
#else
            mbstowcs(&plbe->pwcsName[wcslen(ROOTSTGSTR)], _szFileName,
strlen(_szFileName) + 1);
#endif
        }
        if (SendMessage(_hlb, LB_INSERTSTRING, 0, (LPARAM) plbe) ==
LB_ERRSPACE)
        {
            MessageBoxFromStringIds(
                _hwnd,
                _hInstance,
                IDS_INSERTSTRINGFAILED,
                IDS_ERROR,
                MB_OK | MB_ICONEXCLAMATION);
            DeleteElement(plbe);
        }
        else
        {
            SendMessage(_hlb, LB_SETCURSEL, 0, 0);
        }
    }
}

//
+----------------------------------------------------------------------------
//
//  Function:   Exists
//
//  Synopsis:   simple function to test for the existance of a file
//
//  History:    6-16-93   stevebl   Created
//
//----------------------------------------------------------------------------
--

int Exists(TCHAR *sz)
{
    HANDLE h;
    h = CreateFile(sz,
        GENERIC_READ,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL,
        OPEN_EXISTING,
        0,
        0);
    if (h != INVALID_HANDLE_VALUE)
    {
        CloseHandle(h);
        return(1);
    }
    return (0);
}
```

STRMVWR.H (DFVIEW Sample)

```
//
+---------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       strmvwr.h
//
//  Contents:   Stream viewer window class definition.
//
//  Classes:    CStreamView
//
//  Functions:
//
//  History:    6-27-94   stevebl   Created
//
//--------------------------------------------------------------------------
--

#ifndef __STRMVWR_H__
#define __STRMVWR_H__
#include <cwindow.h>
#include "dfv.h"
#include "mwclass.h"

#ifdef __cplusplus

//
+---------------------------------------------------------------------------
//
//  Class:      CStreamView
//
//  Purpose:    implements a stream viewer window
//
//  Interface:  CStreamView  -- constructor
//              InitInstance -- instantiates an instance of the viewer
//
//  History:    6-27-94   stevebl   Created
//
//  Notes:      only the public interface is described by this comment
//
//--------------------------------------------------------------------------
--

class CStreamView: public CHlprWindow
{
public:
    CStreamView(HWND hlb, LISTBOXELEMENT * ple);
    BOOL InitInstance(HINSTANCE hInst, int nCmdShow);
protected:
    ~CStreamView();
    LRESULT WindowProc(UINT uMsg, WPARAM wParam, LPARAM lParam);
private:
    HWND _hlbParent;
    HWND _hlb;
    LONG _lIndex;
    LISTBOXELEMENT * _ple;
    COLORREF _crTextHigh;
    COLORREF _crBkHigh;
    COLORREF _crTextNorm;
    COLORREF _crBkNorm;
    WORD _wWidth;
    DWORD _cbSize;
    TEXTMETRIC _tm;
    HFONT _hfListBox;
};

#endif // __cplusplus
#endif // __STRMVWR_H__
```

STRMVWR.CXX   (DFVIEW Sample)

```
//
+---------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       stmvwr.cxx
//
//  Contents:   implementation for the stream viewer window class
//
//  Classes:    CStreamView
//
//  Functions:
//
//  History:    6-27-94   stevebl   Created
//
//--------------------------------------------------------------------------
--

#include <windows.h>
#include <ole2.h>
#include "strmvwr.h"

#define BYTESPERLINE 16
#define DIVISIONPOINTS 8

//
+---------------------------------------------------------------------------
//
//  Member:     CStreamView::CStreamView
//
//  Synopsis:   constructor
//
//  Arguments:  [hlb]    - handle to CMainWindow's listbox
//              [ple]    - pointer to the LISTBOXELEMENT structure for this
//                         stream
//
//  History:    6-27-94   stevebl   Created
//
//--------------------------------------------------------------------------
--

CStreamView::CStreamView(HWND hlb, LISTBOXELEMENT * ple)
{
    _hlbParent = hlb;
    _ple = ple;
    _hfListBox = NULL;
}

//
+---------------------------------------------------------------------------
//
```

```
//  Member:     CStreamView::~CStreamView
//
//  Synopsis:   destructor
//
//  History:    6-27-94   stevebl   Created
//
//----------------------------------------------------------------------------
--

CStreamView::~CStreamView()
{
    _ple->fOpen = FALSE;
    if (_hfListBox)
    {
        DeleteObject(_hfListBox);
    }
    InvalidateRect(_hlbParent, NULL, FALSE);
}

//
+----------------------------------------------------------------------------
//
//  Member:     CStreamView::InitInstance
//
//  Synopsis:   creates and shows the stream viewer window
//
//  Arguments:  [hInstance] - the application's instance handle
//              [nCmdShow]  - command to pass to ShowWindow
//
//  Returns:    TRUE on success
//              FALSE on failure
//
//  History:    6-27-94   stevebl   Created
//
//  Notes:      must only be called once immediately after class
construction
//
//----------------------------------------------------------------------------
--

BOOL CStreamView::InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    IStream * pstm = (IStream * ) _ple->pInterface;
    STATSTG statstg;
    HRESULT hr = pstm->Stat(&statstg, STATFLAG_NONAME);
    if (FAILED(hr))
    {
        MessageBoxFromStringIdsAndArgs(
            _hwnd,
            _hInstance,
            IDS_STMSTATFAILED,
            IDS_ERROR,
            MB_OK | MB_ICONEXCLAMATION,
            hr
            );
        return(FALSE);
    }
    _cbSize = statstg.cbSize.LowPart;

    TCHAR szStreamTitleText[MAX_STRING_LENGTH];
    if (!LoadString(_hInstance, IDS_STMTITLETEXT, szStreamTitleText,
MAX_STRING_LENGTH))
    {
        lstrcpy(szStreamTitleText, TEXT(""));
    }

    TCHAR * szTitle = new TCHAR[lstrlen(TEXT("  [0x00000000 bytes]"))
        + lstrlen(szStreamTitleText) + wcslen(_ple->pwcsName) + 1];
    if (szTitle)
    {
        wsprintf(szTitle, TEXT("%s %ws [0x%08X bytes]"), szStreamTitleText,
_ple->pwcsName, _cbSize);
    }
    // Note, the Create method sets the _hwnd member for me so I don't
    // need to set it myself.
    HWND hwnd = Create(
        TEXT(STREAM_VIEW_CLASS_NAME),
        szTitle,
        WS_OVERLAPPED
            | WS_MINIMIZEBOX
            | WS_MAXIMIZEBOX
            | WS_SYSMENU
            | WS_THICKFRAME,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        NULL,
        NULL,
        hInstance);
    delete[] szTitle;
    if (!hwnd)
    {
        return(FALSE);
    }
    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);
    return(TRUE);
}

//
+---------------------------------------------------------------------------
//
//  Member:     CStreamView::WindowProc
//
//  Synopsis:   window procedure for the stream viewer window
//
//  Arguments:  [uMsg]    - Window's message
//              [wParam] - first message parameter
//              [lParam] - second message parameter
```

```
//
//  History:    6-27-94   stevebl   Created
//
//---------------------------------------------------------------------------
--

LRESULT CStreamView::WindowProc(UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch (uMsg)
    {
    case WM_CREATE:
        {
            RECT rc;
            GetClientRect(_hwnd, &rc);
            // record the width before we create the list box
            _wWidth = (WORD) rc.right;

            HDC hdc = GetDC(_hwnd);
            LOGFONT lf;
            memset(&lf, 0, sizeof(LOGFONT));

            // Get an 8 point Courier font.
            lf.lfHeight = -MulDiv(8, GetDeviceCaps(hdc, LOGPIXELSY), 72);
            lstrcpy(lf.lfFaceName, TEXT("Courier"));
            _hfListBox = CreateFontIndirect(&lf);
            HFONT hfOld = (HFONT) SelectObject(hdc, _hfListBox);
            GetTextMetrics(hdc, &_tm);
            SelectObject(hdc, hfOld);
            ReleaseDC(_hwnd, hdc);

            // create the child listbox
            _hlb = CreateWindow(
                TEXT("listbox"),
                TEXT(""),
                LBS_OWNERDRAWFIXED |
                    WS_CHILD |
                    WS_VISIBLE |
                    WS_VSCROLL |
                    LBS_NOINTEGRALHEIGHT |
                    LBS_NODATA,
                0,
                0,
                rc.right,
                rc.bottom,
                _hwnd,
                (HMENU) IDC_LISTBOX,
                _hInstance,
                NULL);
            if (NULL == _hlb)
            {
                // abort window creation
                return(-1);
            }
            SetFocus(_hlb);
            SendMessage(
                _hlb,
                LB_SETCOUNT,
                (WPARAM)((_cbSize / BYTESPERLINE) + (_cbSize %
BYTESPERLINE ? 1 : 0)),
                0);
            SendMessage(_hlb, WM_SETFONT, (WPARAM) _hfListBox, 0);
            SendMessage(_hlb, LB_SETCURSEL, 0, 0);
            _crTextHigh = GetSysColor(COLOR_HIGHLIGHTTEXT);
            _crTextNorm = GetSysColor(COLOR_WINDOWTEXT);
            _crBkHigh = GetSysColor(COLOR_HIGHLIGHT);
            _crBkNorm = GetSysColor(COLOR_WINDOW);
        }
        break;
    case WM_SETFOCUS:
        {
            if (_hlb)
            {
                SetFocus(_hlb);
            }
        break;
        }
    case WM_MEASUREITEM:
        {
            LPMEASUREITEMSTRUCT lpmis = (LPMEASUREITEMSTRUCT) lParam;
            lpmis->itemHeight = _tm.tmHeight + _tm.tmExternalLeading;
            lpmis->itemWidth = _wWidth;
            return(TRUE);
        }
    case WM_DRAWITEM:
        {
            LPDRAWITEMSTRUCT lpdis = (LPDRAWITEMSTRUCT) lParam;
            if (-1 == lpdis->itemID)
            {
                // no items in the listbox
                break;
            }
            // draw the item
            switch (lpdis->itemAction)
            {
            case ODA_SELECT:
            case ODA_DRAWENTIRE:
                {
                    // draw the listbox item
                    LONG lStartByte = lpdis->itemID * BYTESPERLINE;

                    // NOTE:  The following string is actually a bit larger
                    // than it has to be.  The minimum size for this string
                    // is acutally somewhere between 5 and 6 bytes for each
                    // byte (plus 12) in the stream that is to be displayed.
                    // Doing it this way wastes a little space but it's much
                    // simpler, it's faster, and it's less prone to error
                    // than dynamically computing it each time.
                    TCHAR szLine[BYTESPERLINE * 6 + 12];
                    unsigned cch;
                    wsprintf(szLine, TEXT("0x%08X:"), lStartByte);
                    for (cch = BYTESPERLINE * 6 + 12; (unsigned)
lstrlen(TEXT("0x12345678:")) < cch--; )
                    {
                        szLine[cch] = TEXT(' ');
                    }

                    IStream * pstm = (IStream *) _ple->pInterface;
                    BYTE rgb[BYTESPERLINE];
                    LARGE_INTEGER dlibMove;
                    dlibMove.HighPart = 0;
                    // If we ever get a stream that's big enough to require
                    // that the HiPart get set, then nobody would want to
                    // browse that much data anyway!
                    dlibMove.LowPart = lStartByte;

                    HRESULT hr = pstm->Seek(dlibMove, STREAM_SEEK_SET,
NULL);
                    if (FAILED(hr))
                    {
                        MessageBoxFromStringIdsAndArgs(
                            _hwnd,
                            _hInstance,
                            IDS_STMSEEKFAILED,
                            IDS_ERROR,
                            MB_OK | MB_ICONEXCLAMATION,
                            hr
                            );
                        return(FALSE);
                    }
                    ULONG cbRead;
                    hr = pstm->Read(rgb, BYTESPERLINE, &cbRead);
                    if (FAILED(hr))
                    {
                        MessageBoxFromStringIdsAndArgs(
                            _hwnd,
                            _hInstance,
                            IDS_STMREADFAILED,
                            IDS_ERROR,
                            MB_OK | MB_ICONEXCLAMATION,
                            hr
                            );
                        return(FALSE);
                    }

                    for (cch = 0; cch < cbRead; cch++)
                    {
                        TCHAR szTemp[3];
                        wsprintf(szTemp, TEXT("%02X"), rgb[cch]);
                        szLine[cch * 3 + cch / DIVISIONPOINTS + 12] =
szTemp[0];
                        szLine[cch * 3 + cch / DIVISIONPOINTS + 13] =
szTemp[1];
                        char ch = rgb[cch];
                        if (ch < ' ')
                        {
                            ch = '.';
                        }
                        // this is a little trick to get the character in
the
                        // correct type of string (UNICODE or not)
                        wsprintf(szTemp, TEXT("%c"), ch);
                        szLine[cch + 13 + cch / DIVISIONPOINTS + 3 *
BYTESPERLINE + BYTESPERLINE / DIVISIONPOINTS] = szTemp[0];
                    }
                    szLine[cch + 13 + cch / DIVISIONPOINTS + 3 *
BYTESPERLINE + BYTESPERLINE / DIVISIONPOINTS] = 0;

                    SIZE sizeText;
                    RECT rcItem = lpdis->rcItem;

                    GetTextExtentPoint32(
                        lpdis->hDC,
                        szLine,
                        lstrlen(szLine),
                        &sizeText);
                    rcItem.right = rcItem.left + sizeText.cx;
                    COLORREF crOldText = SetTextColor(
                        lpdis->hDC,
                        (lpdis->itemState & ODS_SELECTED) ? _crTextHigh :
_crTextNorm);
                    COLORREF crOldBack = SetBkColor(
                        lpdis->hDC,
                        (lpdis->itemState & ODS_SELECTED) ? _crBkHigh :
_crBkNorm);
                    ExtTextOut(
                        lpdis->hDC,
                        rcItem.left,
                        rcItem.top,
                        ETO_OPAQUE | ETO_CLIPPED,
                        &rcItem,
                        szLine,
                        lstrlen(szLine),
                        NULL);
                    SetTextColor(lpdis->hDC, crOldText);
                    SetBkColor(lpdis->hDC, crOldBack);
                }
                break;
            case ODA_FOCUS:
                break;
            }
            return(TRUE);
        }
    case WM_SIZE:
        {
            _wWidth = LOWORD(lParam);
            WORD wHeight = HIWORD(lParam);
            if (_hlb)
            {
                SetWindowPos(_hlb, NULL, 0, 0, _wWidth, wHeight, SWP_NOMOVE
| SWP_NOZORDER);
            }
            break;
        }
    case WM_COMMAND:
    case WM_QUIT:
    case WM_CLOSE:
    default:
        return(DefWindowProc(_hwnd, uMsg, wParam, lParam));
    }
    return(FALSE);
}
```

WINMAIN.CXX (DFVIEW Sample)

```
//
+---------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       winmain.cxx
//
//  Contents:   main entry point for Stereo Shop
//
//  Classes:
//
//  Functions:  WinMain
//
//  History:    6-08-94   stevebl   Created
//
//---------------------------------------------------------------------------
--

#include "dfv.h"
#include "mwclass.h"
#include <ole2ver.h>

//
+---------------------------------------------------------------------------
//
//  Function:   InitApplication
//
//  Synopsis:   initializes the application and registers its window class
//              (called once for all instances)
//
//  Arguments:  [hInstance] - handle to the first instance
//
//  Returns:    TRUE on success
//
//  History:    4-11-94   stevebl   Created for MFract
//              6-08-94   stevebl   Stolen from MFract
//
//---------------------------------------------------------------------------
--

BOOL InitApplication(HINSTANCE hInstance)
{
    WNDCLASS wc;

    wc.style = 0;
    wc.lpfnWndProc = &WindowProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(hInstance, TEXT("StreamIcon"));
    wc.hCursor = (HCURSOR) LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName = NULL;
    wc.lpszClassName = TEXT(STREAM_VIEW_CLASS_NAME);
    ATOM aReturn = RegisterClass(&wc);
    if (!aReturn)
    {
        return(aReturn);
    }
    wc.style = CS_DBLCLKS;
    wc.hIcon = LoadIcon(hInstance, TEXT("AppIcon"));
    wc.lpszMenuName = TEXT(MAIN_WINDOW_CLASS_MENU_STR);
    wc.lpszClassName = TEXT(MAIN_WINDOW_CLASS_NAME);
    return(RegisterClass(&wc));
}

//
+---------------------------------------------------------------------------
//
//  Function:   WinMain
//
//  Synopsis:   main window proceedure
//
//  Arguments:  [hInstance]     - instance handle
//              [hPrevInstance] - handle of the previous instance (if any)
//              [lpCmdLine]     - pointer to the command line
//              [nCmdShow]      - show state
//
//  History:    4-11-94   stevebl   Created for MFract
//              6-08-94   stevebl   Stolen from MFract
//
//  Notes:      initializes application and starts message loop
//
//---------------------------------------------------------------------------
--

extern "C" int PASCAL WinMain(HINSTANCE hInstance,
            HINSTANCE hPrevInstance,
            LPSTR lpCmdLine,
            int nCmdShow)
{
    DWORD dwBuildVersion = CoBuildVersion();
    if (HIWORD(dwBuildVersion) != rmm || LOWORD(dwBuildVersion) < rup)
    {
        // alert the caller that the OLE version is incompatible
        // with this build.
        MessageBoxFromStringIdsAndArgs(
            NULL,
            hInstance,
            IDS_OLEINCOMPATIBLE,
            IDS_ERROR,
            MB_OK | MB_ICONEXCLAMATION,
            rmm, rup);
        return(FALSE);
    }
    if (FAILED(CoInitialize(NULL)))
    {
        // alert the caller that OLE couldn't be initialized
        MessageBoxFromStringIds(
            NULL,
            hInstance,
            IDS_OLEINITFAILED,
            IDS_ERROR,
            MB_OK | MB_ICONEXCLAMATION);
        return(FALSE);
    }
    if (!hPrevInstance)
    {
        if (!InitApplication(hInstance))
        {
            CoUninitialize();
            return(FALSE);
        }
    }
    CMainWindow * pw = new CMainWindow;
    if (pw == NULL)
    {
        CoUninitialize();
        return(FALSE);
    }
    if (!pw->InitInstance(hInstance, nCmdShow))
    {
        // Note, if InitInstance has failed then it would have
        // already deleted pw for me so I don't delete it here.
        // This is because when WM_CREATE returns -1 (failure)
        // Windows sends the WM_DESTROY message to the window
        // and the the CHlprWindow class destroys itself whenever
        // it receives this message.
        CoUninitialize();
        return(FALSE);
    }

    // check command line and try and load as a file

    if (strlen(lpCmdLine) > 0)
    {
#ifdef UNICODE
        mbstowcs(pw->_szFileName, lpCmdLine, MAX_PATH - 1);
#else
        strncpy(pw->_szFileName, lpCmdLine, MAX_PATH - 1);
#endif
        pw->OpenFile();
    }

    MSG msg;
    HACCEL haccel = LoadAccelerators(hInstance, TEXT("AppAccel"));
    if (haccel == NULL)
    {
        CoUninitialize();
        return(FALSE);
    }
    while (GetMessage(&msg, NULL, 0, 0))
    {
        if (!TranslateAccelerator(
            pw->GetHwnd(),
            haccel,
            &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
    CoUninitialize();
    return(msg.wParam);
}
```

DISPCALC

-------------------------------------- OLE Automation Sample Program: Dispcalc --------------------------------------

DispCalc is a simple accumulator-based calculator. Its user interface consists of buttons for the numbers (0-9), the operators (+, -, *, /), and some other necessary buttons (C for Clear, = for evaluation). Its programmability interface consists of one object, which is described below.

The ProgID for dispcalc's only object is "DispCalc.Application". An instance of this object can be created by executing the following lines of code in Visual Basic or DispTest:

```
Sub Foo
  Dim MyCalculator as Object

  Set MyCalculator = CreateObject("DispCalc.Application")
  . . .
End Sub
```

---------------- Program Structure ---------------- Dispcalc uses INTERFACEDATA and CreateStdDispatch in order to implement the IDispatch interface.

------------------------ Properties for the object ------------------------

Name Type Description --------------------------- Value VT_I4 Same as the value for the accumulator.

Accum VT_I4 The value that is in the accumulator of the calculator.

Opnd VT_I4 The operand. This is the number which is currently being entered.

Op VT_I2 The operator that is currently being used. This is an enumeration:
const OP_NONE = 0 const OP_PLUS = 1 const
OP_MINUS = 2 const OP_MULT = 3

---------------------------- Methods defined on the object ----------------------------

Name Description -------------------------- Eval() as Boolean If there is an operator, apply it to accumulator and the operand, placing the result in the accumulator.

The return value indicates success or failure.

Clear() Resets the calculator. This sets Op to OP_NONE, and both Accum and Opnd to 0. Display() Updates the display of the calculator. (Other operations do not do this.)

Quit() Close the calculator.

Button(b as string) as Boolean Press the indicated button and return
success or failure. Valid string values are: +, -, *,
+ 0-9 c, C =
Note that you may also pass the numbers 0-9 and these will be
converted to strings automatically.

-------------------------- Shortcomings of this sample -------------------------- 1. Property and method names should not be abbreviated. For example, the "Opnd" property should be the "Operand" property.

2. Since the object is the application object, it should have Name and Version properties, which are read-only.

MAKEFILE   (DISPCALC Sample)

```
############################################################################
##
#
#  (c) Copyright Microsoft Corp. 1992-1993 All Rights Reserved
#
#  File:
#
#    makefile - makefile for spoly2.exe
#
#  Purpose:
#
#    Builds the OLE 2.0 sample IDispatch server, dispcalc.exe.
#
#
#  Usage:
#
#     NMAKE                     ; build with defaults
#     or: NMAKE option          ; build with the given option(s)
#     or: NMAKE clean           ; erase all compiled files
#
#     option:
#         dev = [win16 | win32] ; dev=win32 is the default
#         DEBUG=[0|1]           ; DEBUG=1 is the default
#
#  Notes:
#
#    This makefile assumes that the PATH, INCLUDE and LIB environment
#    variables are setup properly.
#
############################################################################
##



##########################################################################
#
# Default Settings
#

!if "$(dev)" == ""
dev = win32
!endif

!if !("$(dev)" == "win16" || "$(dev)" == "win32")
!error Invalid dev option, choose from [win16 | win32]
!endif

!if "$(dev)" == "win16"
TARGET  = WIN16
!endif

!if "$(dev)" == "win32"
```

```
TARGET  = WIN32
!endif

!ifdef NODEBUG
DEBUG = 0
!endif

!if "$(DEBUG)" == "0"
NODEBUG = 1
!endif

!if "$(DEBUG)" == ""
DEBUG = 1
!endif


##########################################################################
#
# WIN16 Settings
#
!if "$(TARGET)" == "WIN16"

CC   = cl
LINK = link

RCFLAGS = -dWIN16
CFLAGS = -c -W3 -AM -GA -GEs -DWIN16
LINKFLAGS = /NOD /NOI /BATCH /ONERROR:NOEXE

LIBS = libw.lib mlibcew.lib

!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS) -Od -Zi -D_DEBUG $(CL)
LINKFLAGS = $(LINKFLAGS) /COD
!else
CFLAGS = $(CFLAGS) -Ox $(CL)
LINKFLAGS = $(LINKFLAGS) /FAR /PACKC
!endif
!endif


##########################################################################
#
# WIN32 Settings
#
!if "$(TARGET)" == "WIN32"

!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

CC = $(cc)
CFLAGS = $(cflags) $(cvarsmt) -DINC_OLE2 -DUNICODE $(cdebug)

!ifndef NODEBUG
CFLAGS = $(CFLAGS) -D_DEBUG
!endif
```

```
LINK = $(link)
LINKFLAGS = $(linkdebug) $(guilflags)
RCFLAGS = -DWIN32

!endif


##########################################################################
#
# Build rules
#

.cpp.obj:
    @echo Compiling $<...
    $(CC) $<

.c.obj:
    @echo Compiling $<...
    $(CC) $<


##########################################################################
#
# Application Settings
#

APPS = dispcalc


!if "$(TARGET)" == "WIN16"
LIBS = ole2.lib compobj.lib ole2disp.lib $(LIBS)
!else
!if "$(TARGET)" == "WIN32"
LIBS = $(olelibsmt)
!endif
!endif

OBJS = \
        winmain.obj     \
        idata.obj       \
        dispcalc.obj    \
        clsid.obj


##########################################################################
#
# Default Goal
#

goal : setflags $(APPS).exe

setflags :
        set CL=$(CFLAGS)
```

```
###########################################################################
#
# Clean (erase) generated files
#
clean :
    if exist *.obj       del *.obj
    if exist $(APPS).exe del $(APPS).exe
    if exist $(APPS).map del $(APPS).map
    if exist $(APPS).res del $(APPS).res


###########################################################################
#
# Application Build (WIN16 Specific)
#

!if "$(TARGET)" == "WIN16"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
        link $(LINKFLAGS) @<<
$(OBJS),
$@,,
$(LIBS),
$(APPS).def
<<
        rc -k -t $(APPS).res $@
!endif


###########################################################################
#
# Application Build (WIN32 Specific)
#
!if "$(TARGET)" == "WIN32"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
      $(LINK) @<<
        $(LINKFLAGS)
        -out:$@
        -map:$*.map
        $(OBJS)
        $(APPS).res
        $(LIBS)
<<
!endif


###########################################################################
#
# Application Build (Common)
#

$(APPS).res : $(APPS).rc
        rc  $(RCFLAGS) -r -fo$@ $?
```

```
##########################################################################
#
# Dependencies
#

winmain.obj: winmain.cpp dispcalc.h
    $(CC) winmain.cpp

idata.obj: idata.cpp dispcalc.h
    $(CC) idata.cpp

dispcalc.obj: dispcalc.cpp dispcalc.h
    $(CC) dispcalc.cpp

clsid.obj: clsid.c clsid.h
    $(CC) clsid.c
```

CLSID.H   (DISPCALC Sample)

```
/***
*clsid.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This file defines the CLSIDs
*
*Implementation Notes:
*
*****************************************************************************
*/

DEFINE_GUID(CLSID_CCalc, 0x00020467, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);


#ifdef INITGUID
# ifdef _MAC
DEFINE_GUID(GUID_NULL, 0L, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
DEFINE_GUID(IID_IDispatch, 0x00020400, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);
DEFINE_GUID(IID_IEnumVARIANT, 0x00020404, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0,
0x46);
DEFINE_GUID(IID_IUnknown, 0x00000000, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);
DEFINE_GUID(IID_IClassFactory, 0x00000001, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0,
0x46);
# endif
#endif
```

CLSID.C (DISPCALC Sample)

```
/***
*clsid.c
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This file allocates and initializes the CLSIDs.
*
*****************************************************************************
*/

#ifdef _PPCMAC
#pragma data_seg ("_FAR_DATA")
#pragma data_seg ( )
#endif //_PPCMAC

#ifdef _MAC
# include <Types.h>
#ifdef _MSC_VER
# include <Processe.h>
# include <AppleEve.h>
#else //_MSC_VER
# include <Processes.h>
# include <AppleEvents.h>
#endif //_MSC_VER
#else
# include <windows.h>
#endif

#ifndef WIN32
#include <compobj.h>
#endif //!WIN32

// this redefines the DEFINE_GUID() macro to do allocation.
//
#include <initguid.h>

#ifndef INITGUID
# define INITGUID
#endif

// due to the previous header, including this causes the DEFINE_GUID
// definitions in the following header(s) to actually allocate data.
//
#include "clsid.h"
```

DISPCALC.DEF (DISPCALC Sample)

```
NAME          DISPCALC

DESCRIPTION   'IDispatch Calculator'

EXETYPE       WINDOWS

STUB          'WINSTUB.EXE'

CODE          PRELOAD MOVEABLE DISCARDABLE
DATA          PRELOAD MULTIPLE

HEAPSIZE      4096
STACKSIZE     8192
```

DISPCALC.H   (DISPCALC Sample)

```
/***
*dispcalc.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  UNDONE
*
*
*Implementation Notes:
*
****************************************************************************
*/

#include "hostenv.h"
#include "resource.h"
#include "clsid.h"

#ifndef CLASS
# ifdef __TURBOC__
#  define CLASS class huge
# else
#  define CLASS class FAR
# endif
#endif

#pragma warning(disable:4355)

#ifdef _MAC
# define NEARDATA
#else
# ifdef WIN32
#  define NEARDATA
# else
#  define NEARDATA __near
# endif
#endif

#ifdef _MAC
# define UNUSED(X) ((void)(void*)&(X))
#else
# define UNUSED(X) (X)
#endif

#define DIM(X) (sizeof(X)/sizeof(X[0]))



// forward decl
CLASS CCalc;
```

```
// Introduced "calculator" interface
//
// This class implementes core arithmetic functionality
// (such as it is) *and* is the interface that will be
// exposed via IDispatch for external programmability.

#if defined(_MAC) && !(!defined(applec) || defined(__SC__) ||
defined(_MSC_VER))

// Mac Note: The default implementation of IDispatch places
// a couple of requirements on the layout of an instance that
// in can invoke on.
//
//   1. It assumes that the vtable pointer is at offset 0
//      from the beginning of the instance. This appears
//      to always be the case if the class derives from
//      an interface, and hence the games I play below
//      creating a stub _CArith interface containing a
//      single pure virtual function. The only reason for
//      this is to ensure that the vtable ptr is at offset
//      0 when this class is a member of CCalc. If your
//      class derives from an Ole interface, then you are ok.
//
//   2. It assumes that the vtable is a simple array of
//      function pointers, with index zero reserved. In
//      order to ensure this, the class must derive from
//      SingleObject. Note that IUnknown derives from SingleObject,
//      so if your class derives from any Ole interface, then
//      you are also ok - otherwise you need to explicitly
//      derive from SingleObject as I have done below.
//
//  (The above comments apply to MPW C++ v3.3)
//

interface _CArith : public SingleObject
{
    STDMETHOD_(void,  put_Accum)(long l) PURE;
};
CLASS CArith : public _CArith
#else
CLASS CArith
#endif
{
public:
#if defined(_MAC)
    BEGIN_INTERFACE
#endif //_MAC

    STDMETHOD_(void,  put_Accum)(long l);
    STDMETHOD_(long,  get_Accum)(void);
    STDMETHOD_(void,  put_Opnd)(long l);
    STDMETHOD_(long,  get_Opnd)(void);
    STDMETHOD_(void,  put_Op)(short op);
    STDMETHOD_(short, get_Op)(void);
    STDMETHOD_(short, Eval)(void);
    STDMETHOD_(void,  Clear)(void);
    STDMETHOD_(void,  Display)(void);
    STDMETHOD_(void,  Quit)(void);
    STDMETHOD_(short, Button)(BSTR button);

    // the following method is internal, and not exposed for programmability
    int ButtonPush(int button);

    CArith::CArith(CCalc FAR* pcalc){
      m_pcalc = pcalc;
      Clear();
    }
    enum states { STATE_LOPND, STATE_OP, STATE_ROPND, STATE_EVAL };

private:
    CCalc FAR*  m_pcalc;

    short  m_op;
    long   m_opnd;
    long   m_accum;
    enum states m_state;
};


CLASS CCalc : public IUnknown {
public:
    friend CArith;

    static CCalc FAR* Create();


    STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppv);
    STDMETHOD_(unsigned long, AddRef)(void);
    STDMETHOD_(unsigned long, Release)(void);

    CCalc::CCalc();

#ifdef _MAC
    DialogPtr m_pdlg;
#else
    HWND m_hwnd;
#endif
    CArith m_arith;

private:
    unsigned long m_refs;
    IUnknown FAR* m_punkStdDisp;
};

enum operators {
    OP_NONE = 0,
    OP_PLUS,
    OP_MINUS,
    OP_MULT,
    OP_DIV
```

```
};

// the following enum defines method indices used by the
// default IDispatch implementation - DispInvoke().
//
// Note: these must match the order of the preceeding declarations
//
enum IMETH_CARITH {
    IMETH_PUTACCUM = 0,
    IMETH_GETACCUM,
    IMETH_PUTOPERAND,
    IMETH_GETOPERAND,
    IMETH_PUTOPERATOR,
    IMETH_GETOPERATOR,
    IMETH_EVAL,
    IMETH_CLEAR,
    IMETH_DISPLAY,
    IMETH_QUIT,
    IMETH_BUTTON,

    // Define the "property" indices. these are defined to be
    // the first index in a set/get property method pair. These
    // definitions are used to build the METHODDATA that drives
    // our implementation of IDispatch. see cdisp.cpp.
    //
    IMETH_ACCUM    = IMETH_PUTACCUM,
    IMETH_OPERAND  = IMETH_PUTOPERAND,
    IMETH_OPERATOR = IMETH_PUTOPERATOR
};

// the following enum defines the IDs used by IDispatch
//
// Note: these values do *not* depend on order of declaration,
// but are sensitive to the kind of the method - ie, if a get/set
// method pair implements a property, then they need to share
// an ID.
//
// Note: by assigning "accum" the ID 'DISPID_VALUE', we are
// choosing to expose it as the default "value" property.
//
enum IDMEMBER_CARITH {
    IDMEMBER_ACCUM = DISPID_VALUE, // the default property
    IDMEMBER_OPERAND,
    IDMEMBER_OPERATOR,
    IDMEMBER_EVAL,
    IDMEMBER_CLEAR,
    IDMEMBER_DISPLAY,
    IDMEMBER_QUIT,
    IDMEMBER_BUTTON
};


// the CCalc Class Factory
//
CLASS CCalcCF : public IClassFactory {
```

```
public:
    static IClassFactory FAR* Create();

    STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppv);
    STDMETHOD_(unsigned long, AddRef)(void);
    STDMETHOD_(unsigned long, Release)(void);

    STDMETHOD(CreateInstance)(
      IUnknown FAR* punkOuter, REFIID riid, void FAR* FAR* ppv);
#ifdef _MAC
    STDMETHOD(LockServer)(unsigned long fLock);
#else
    STDMETHOD(LockServer)(BOOL fLock);
#endif

    CCalcCF() { m_refs = 1; }

private:
    unsigned long m_refs;
};

extern HRESULT InitOle(void);
extern HRESULT UninitOle(void);

extern CCalc FAR* g_pcalc;
```

DISPCALC.RC (DISPCALC Sample)

```
#include <windows.h>
#include "resource.h"

DispCalc ICON dispcalc.ico

DispCalc DIALOG DISCARDABLE  0, 0, 92, 114
STYLE    WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX
CLASS    "DispCalc"
#ifdef WIN32
CAPTION  "DispCalc (32-bit)"
#else //WIN32
CAPTION  "DispCalc"
#endif //WIN32
BEGIN
    PUSHBUTTON "0", IDC_ZERO,           9, 90, 14, 15
    PUSHBUTTON "1", IDC_ONE,            9, 70, 15, 16
    PUSHBUTTON "2", IDC_TWO,           29, 70, 16, 15
    PUSHBUTTON "3", IDC_THREE,         49, 70, 15, 15
    PUSHBUTTON "4", IDC_FOUR,           9, 50, 15, 14
    PUSHBUTTON "5", IDC_FIVE,          29, 50, 15, 15
    PUSHBUTTON "6", IDC_SIX,           49, 50, 15, 15
    PUSHBUTTON "7", IDC_SEVEN,          9, 30, 15, 15
    PUSHBUTTON "8", IDC_EIGHT,         29, 30, 15, 15
    PUSHBUTTON "9", IDC_NINE,          49, 30, 15, 15
    PUSHBUTTON "=", IDC_EQUALS,        49, 90, 14, 15
    PUSHBUTTON "+", IDC_PLUS,          69, 30, 15, 15
    PUSHBUTTON "-", IDC_MINUS,         69, 50, 14, 15
    PUSHBUTTON "*", IDC_MULT,          69, 70, 15, 15
    PUSHBUTTON "/", IDC_DIV,           69, 90, 15, 15
    PUSHBUTTON "C", IDC_CLEAR,         29, 90, 16, 15
    EDITTEXT        IDC_DISPLAY,        9,  6, 76, 15, ES_MULTILINE |
ES_RIGHT | ES_AUTOHSCROLL | ES_READONLY
END
```

DISPCALC.REG (DISPCALC Sample)

```
REGEDIT


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; registration info DispCalc.Application (defaults to DispCalc.Application.1

HKEY_CLASSES_ROOT\Dispcalc.Application = OLE Automation Dispcalc Application
HKEY_CLASSES_ROOT\Dispcalc.Application\Clsid = {00020467-0000-0000-C000-
000000000046}


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; registration info DispCalc 1.0

HKEY_CLASSES_ROOT\Dispcalc.Application.1 = OLE Automation Dispcalc 1.0
Application
HKEY_CLASSES_ROOT\Dispcalc.Application.1\Clsid = {00020467-0000-0000-C000-
000000000046}


HKEY_CLASSES_ROOT\CLSID\{00020467-0000-0000-C000-000000000046} = OLE
Automation Dispcalc 1.0 Application
HKEY_CLASSES_ROOT\CLSID\{00020467-0000-0000-C000-000000000046}\ProgID =
Dispcalc.Application.1
HKEY_CLASSES_ROOT\CLSID\{00020467-0000-0000-C000-
000000000046}\VersionIndependentProgID = Dispcalc.Application
HKEY_CLASSES_ROOT\CLSID\{00020467-0000-0000-C000-000000000046}\LocalServer32
= dispcalc.exe /Automation
```

DISPCALC.CPP   (DISPCALC Sample)

```
/***
*dispcalc.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This module implements the basic user interface and arithmetic
*  functionality of the IDispatch calculator.
*
*  The implementation of IDispatch it via aggregation with an
*  instance of the "standard" IDispatch implementation, which is
*  initialized with a DispTypeInfo constructed from an INTERFACEDATA
*  description.
*
*Implementation Notes:
*
****************************************************************************
*/

#include "dispcalc.h"

CCalc FAR* g_pcalc = NULL;

unsigned long g_dwCCalcCF = 0;
unsigned long g_dwRegisterCCalc = 0;

#ifdef _MAC
extern Boolean g_fQuit;
#endif //_MAC


CCalc::CCalc() : m_arith(this)
{
    m_refs = 0;
#ifdef _MAC
    m_pdlg = nil;
#else
    m_hwnd = NULL;
#endif
    m_punkStdDisp = NULL;
}

/***
*CCalc *CCalc::Create(void)
*Purpose:
*  Create an instance of the IDispatch calculator, build a
*  TypeInfo that describes the exposed functionality and
*  aggregate with an instance of CStdDispatch that has been
*  initialized with this TypeInfo.
*
*Entry:
```

```
*  None
*
*Exit:
*  return value = CCalc*, NULL if the creation failed.
*
***********************************************************************/
CCalc FAR*
CCalc::Create()
{
    HRESULT hresult;
    CCalc FAR* pcalc;
    CArith FAR* parith;
    ITypeInfo FAR* ptinfo;
    IUnknown FAR* punkStdDisp;
extern INTERFACEDATA NEARDATA g_idataCCalc;

    if((pcalc = new FAR CCalc()) == NULL)
      return NULL;
    pcalc->AddRef();

    parith = &(pcalc->m_arith);

    // Build a TypeInfo for the functionality on this object that
    // is being exposing for external programmability.
    //
    hresult = CreateDispTypeInfo(
      &g_idataCCalc, LOCALE_SYSTEM_DEFAULT, &ptinfo);
    if(hresult != NOERROR)
      goto LError0;

    // Create and aggregate with an instance of the default
    // implementation of IDispatch that is initialized with our
    // TypeInfo.
    //
    hresult = CreateStdDispatch(
      pcalc,                    // controlling unknown
      parith,                   // instance to dispatch on
      ptinfo,                   // typeinfo describing the instance
      &punkStdDisp);

    ptinfo->Release();

    if(hresult != NOERROR)
      goto LError0;

    pcalc->m_punkStdDisp = punkStdDisp;

    return pcalc;

LError0:;
    pcalc->Release();

    return NULL;
}
```

```
//---------------------------------------------------------------------
//                        IUnknown methods
//---------------------------------------------------------------------

STDMETHODIMP
CCalc::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    if(IsEqualIID(riid, IID_IUnknown)){
      *ppv = this;
    }else
    if(IsEqualIID(riid, IID_IDispatch)){
      return m_punkStdDisp->QueryInterface(riid, ppv);
    }else {
      *ppv = NULL;
      return ResultFromScode(E_NOINTERFACE);
    }

    AddRef();
    return NOERROR;
}

STDMETHODIMP_(unsigned long)
CCalc::AddRef()
{
    return ++m_refs;
}


STDMETHODIMP_(unsigned long)
CCalc::Release()
{
    if(--m_refs == 0){
      if(m_punkStdDisp != NULL)
        m_punkStdDisp->Release();
#ifndef _MAC
      UninitOle();
      PostQuitMessage(0);
#endif
      delete this;
      return 0;
    }
    return m_refs;
}


//---------------------------------------------------------------------
//                       Arithmetic features
//---------------------------------------------------------------------

STDMETHODIMP_(void)
CArith::Clear()
{
    m_opnd = 0;
    m_accum = 0;
    m_op = OP_NONE;
    m_state = STATE_LOPND;
}

STDMETHODIMP_(void)
CArith::put_Accum(long l)
{
    m_accum = l;
}


STDMETHODIMP_(long)
CArith::get_Accum()
{
    return m_accum;
}

STDMETHODIMP_(void)
CArith::put_Opnd(long l)
{
    m_opnd = l;
}

STDMETHODIMP_(long)
CArith::get_Opnd()
{
    return m_opnd;
}

STDMETHODIMP_(void)
CArith::put_Op(short op)
{
    m_op = op;
}

STDMETHODIMP_(short)
CArith::get_Op()
{
    return m_op;
}

STDMETHODIMP_(short)
CArith::Eval()
{
    if(m_op == OP_NONE)
      return 0;

    switch(m_op){
    case OP_PLUS:
      m_accum += m_opnd;
      break;
    case OP_MINUS:
      m_accum -= m_opnd;
      break;
    case OP_MULT:
      m_accum *= m_opnd;
      break;
    case OP_DIV:
      m_accum = (m_opnd == 0) ? 0 : (m_accum / m_opnd);
      break;
    default:
      // ASSERT(UNREACHED);
      return 0;

    }

    m_state = STATE_EVAL;

    return 1;
}


//---------------------------------------------------------------------
//                       User Interface features
//---------------------------------------------------------------------

/***
*void CArith::Display()
*Purpose:
*  Display the contents of the register currently being edited.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
STDMETHODIMP_(void)
CArith::Display()
{
    VARIANT var;

    VariantInit(&var);

    V_VT(&var) = VT_I4;
    V_I4(&var) = (m_state == STATE_ROPND) ? m_opnd : m_accum;
    VariantChangeType(&var, &var, 0, VT_BSTR);

#ifdef _MAC
    {
      Rect rcItem;
      Handle hItem;
      char str[255];
      short sItemKind;

      strcpy(str, V_BSTR(&var));
      GetDItem(m_pcalc->m_pdlg, IDC_DISPLAY, &sItemKind, &hItem, &rcItem);
      SetIText(hItem, c2pstr(str));
    }
#else
```

```
    SetDlgItemText(m_pcalc->m_hwnd, IDC_DISPLAY, STRING(V_BSTR(&var)));
#endif

    VariantClear(&var);
}

STDMETHODIMP_(short)
CArith::Button(BSTR bstrButton)
{
    int i, button;

static struct {
    OLECHAR ch;
    int idc;
} NEARDATA rgIdcOfCh[] = {
      { OLESTR('+'), IDC_PLUS   }
    , { OLESTR('-'), IDC_MINUS  }
    , { OLESTR('*'), IDC_MULT   }
    , { OLESTR('/'), IDC_DIV    }
    , { OLESTR('C'), IDC_CLEAR  }
    , { OLESTR('c'), IDC_CLEAR  }
    , { OLESTR('='), IDC_EQUALS }
    , { OLESTR('0'), IDC_ZERO   }
    , { OLESTR('1'), IDC_ONE    }
    , { OLESTR('2'), IDC_TWO    }
    , { OLESTR('3'), IDC_THREE  }
    , { OLESTR('4'), IDC_FOUR   }
    , { OLESTR('5'), IDC_FIVE   }
    , { OLESTR('6'), IDC_SIX    }
    , { OLESTR('7'), IDC_SEVEN  }
    , { OLESTR('8'), IDC_EIGHT  }
    , { OLESTR('9'), IDC_NINE   }
    , { (OLECHAR)-1 , -1         }
};

    // if the string is more that 1 character long, then we know its wrong.
    if(SysStringLen(bstrButton) > 1)
      return 0;

    // translate button string into control ID
    for(i = 0;; ++i){
      if(rgIdcOfCh[i].ch == -1)
        return 0;
      if(rgIdcOfCh[i].ch == bstrButton[0]){
        button = rgIdcOfCh[i].idc;
        break;
      }
    }

    return ButtonPush(button);
}

// the following method is internal, and not exposed for programmability
int
```

```
CArith::ButtonPush(int button)
{
    if(button >= IDC_ZERO && button <= IDC_NINE){

      long lVal = button - IDC_ZERO;

      switch(m_state){
      case STATE_EVAL:
        m_accum = lVal;
        m_state = STATE_LOPND;
        break;
      case STATE_OP:
        m_opnd = lVal;
        m_state = STATE_ROPND;
        break;
      case STATE_LOPND:
        m_accum = (m_accum * 10) + lVal;
        break;
      case STATE_ROPND:
        m_opnd  = (m_opnd * 10) + lVal;
        break;
      }

    }else if(button >= IDC_PLUS && button <= IDC_DIV){

      if(m_state == STATE_LOPND){
        m_opnd  = m_accum;
        m_state = STATE_OP;
        m_op    = button - IDC_PLUS + OP_PLUS;
      }

    }else if(button == IDC_EQUALS){

      if(m_state > STATE_LOPND)
        Eval();

    }else if (button == IDC_CLEAR){

      Clear();

    }else{

      return 0; // unknown button

    }

    // Flash the button

#ifdef _MAC
    {
      Rect rcItem;
      long lDummy;
      Handle hItem;
      short sItemKind;
      GetDItem(m_pcalc->m_pdlg, button, &sItemKind, &hItem, &rcItem);
      HiliteControl((ControlHandle)hItem, 1);
      Delay(6, &lDummy);
      HiliteControl((ControlHandle)hItem, 0);
    }
#else
    SendMessage(m_pcalc->m_hwnd, BM_SETSTATE, 1, 0L);
    SendMessage(m_pcalc->m_hwnd, BM_SETSTATE, 0, 0L);
#endif

    // Update the calculator display

    Display();

    return 1;
}

/***
*void CArith::Quit()
*Purpose:
*
*Entry:
*  None
*
*Exit:
*  None
*
************************************************************************/
STDMETHODIMP_(void)
CArith::Quit()
{
#ifndef _MAC
    UninitOle();
    PostQuitMessage(0);
#else
    g_fQuit = 1;
#endif
}


//----------------------------------------------------------------------
//                      The CCalc Class Factory
//----------------------------------------------------------------------

IClassFactory FAR*
CCalcCF::Create()
{
    return new FAR CCalcCF();
}

STDMETHODIMP
CCalcCF::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    if(IsEqualIID(riid, IID_IUnknown) ||
       IsEqualIID(riid, IID_IClassFactory)){
      AddRef();
      *ppv = this;
      return NOERROR;
    }

    *ppv = NULL;
    return ResultFromScode(E_NOINTERFACE);
}

STDMETHODIMP_(unsigned long)
CCalcCF::AddRef()
{
    return ++m_refs;
}

STDMETHODIMP_(unsigned long)
CCalcCF::Release()
{
    if(--m_refs == 0){
      delete this;
      return 0;
    }
    return m_refs;
}

STDMETHODIMP
CCalcCF::CreateInstance(
    IUnknown FAR* punkOuter,
    REFIID riid,
    void FAR* FAR* ppv)
{
    if(punkOuter != NULL)
      return ResultFromScode(CLASS_E_NOAGGREGATION);
    return g_pcalc->QueryInterface(riid, ppv);
}

STDMETHODIMP
#ifdef _MAC
CCalcCF::LockServer(unsigned long fLock)
#else
CCalcCF::LockServer(BOOL fLock)
#endif
{
    UNUSED(fLock);
    return NOERROR;
}


//---------------------------------------------------------------------
//                        Ole Init/Uninit
//---------------------------------------------------------------------

HRESULT
UninitOle()
```

```
{
    if(g_dwRegisterCCalc != 0)
      RevokeActiveObject(g_dwRegisterCCalc, NULL);

    if(g_dwCCalcCF != 0)
      CoRevokeClassObject(g_dwCCalcCF);

    if(g_pcalc != NULL)
      g_pcalc->Release();

    OleUninitialize();

    return NOERROR;
}

#ifdef _MAC
struct regentry{
    char *szKey;
    char *szValue;
} g_rgregentry[] = {

      { "CLSID\\{00020467-0000-0000-C000-000000000046}",
        "OLE Automation DispCalc 1.0 Application" }

    , { "CLSID\\{00020467-0000-0000-C000-000000000046}\\LocalServer",
        "DCLC" }

    , { "CLSID\\{00020467-0000-0000-C000-000000000046}\\ProgID",
        "Dispcalc.Application" }

    , { "CLSID\\{00020467-0000-0000-C000-000000000046}\\InprocHandler",
        "OLE2:Def$DefFSet" }

    , { "DCLC", "{00020467-0000-0000-C000-000000000046}" }

    , { "Dispcalc.Application\\CLSID",
        "{00020467-0000-0000-C000-000000000046}" }

};

HRESULT
EnsureRegistration()
{
    HKEY hkey;

    if(RegOpenKey(HKEY_CLASSES_ROOT, "DCLC", &hkey) == NOERROR){
      RegCloseKey(hkey);
      return NOERROR;
    }

    for(int i = 0; i < DIM(g_rgregentry); ++i){
      if(RegSetValue(HKEY_CLASSES_ROOT, g_rgregentry[i].szKey, REG_SZ,
g_rgregentry[i].szValue, 0) != ERROR_SUCCESS)
        return ResultFromScode(E_FAIL);
    }
    return NOERROR;
}
#endif

/***
*HRESULT InitOle(void)
*Purpose:
*  Initialize Ole, and register our class factories.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
HRESULT
InitOle()
{
    HRESULT hresult;
    IClassFactory FAR* pcf;

    if((hresult = OleInitialize(NULL)) != NOERROR)
      goto LError0;

#ifdef _MAC
    if((hresult = EnsureRegistration()) != NOERROR)
      goto LError0;
#endif

    // create the single global instance of CCalc
    if((g_pcalc = CCalc::Create()) == NULL){
      hresult = ResultFromScode(E_OUTOFMEMORY);
      goto LError0;
    }

    if((pcf = CCalcCF::Create()) == NULL)
      goto LError1;

    hresult = CoRegisterClassObject(
      CLSID_CCalc,
      pcf,
      CLSCTX_LOCAL_SERVER,
      REGCLS_MULTIPLEUSE,
      &g_dwCCalcCF);
    if(hresult != NOERROR)
      goto LError2;

    hresult = RegisterActiveObject(
      g_pcalc, CLSID_CCalc, NULL, &g_dwRegisterCCalc);
    if(hresult != NOERROR)
      goto LError2;
    pcf->Release();

    return NOERROR;

LError2:;
    pcf->Release();

LError1:;
    UninitOle();

LError0:;
    return hresult;
}
```

HOSTENV.H   (DISPCALC Sample)

```
/***
*hostenv.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*  Generic host specific includes.
*
*Implementation Notes:
*
****************************************************************************
*/

#if defined(_MAC)

#if defined(_PPCMAC)
#pragma data_seg("_FAR_DATA")
#pragma data_seg( )
#define MAXLONG 0x7fffffff
#define EventHandlerProcPtr AEEventHandlerUPP
#else //_PPCMAC
#define  GetMenuItemText(mApple,menuItem,daName)
GetItem(mApple,menuItem,daName)
#endif //_PPCMAC

#endif //_MAC

#if defined(_MAC)

#if defined(_MSC_VER)

# include <values.h>
# include <types.h>
# include <string.h>
# include <quickdra.h>
# include <fonts.h>
# include <events.h>
# include <resource.h>
# include <menus.h>
# include <textedit.h>
# include <dialogs.h>
# include <desk.h>
# include <toolutil.h>
# include <memory.h>
# include <files.h>
# include <osutils.h>
# include <osevents.h>
# include <diskinit.h>
# include <packages.h>
# include <traps.h>
# include <AppleEve.h>
```

```
#else //_MSC_VER

# include <values.h>
# include <types.h>
# include <strings.h>
# include <quickdraw.h>
# include <fonts.h>
# include <events.h>
# include <resources.h>
# include <windows.h>
# include <menus.h>
# include <textedit.h>
# include <dialogs.h>
# include <desk.h>
# include <toolutils.h>
# include <memory.h>
# include <files.h>
# include <osutils.h>
# include <osevents.h>
# include <diskinit.h>
# include <packages.h>
# include <traps.h>
# include <AppleEvents.h>

#endif //_MSC_VER

# include <ole2.h>
# include <olenls.h>
# include <dispatch.h>

# define TCHAR               char
# define TSTR(str)           str
# define STRING(str)           (str)
# define WIDESTRING(str)        (str)

#elif defined(WIN32)

# include <windows.h>
# include <ole2.h>
# include <oleauto.h>

# if defined(UNICODE)
    #define TCHAR      WCHAR
    #define TSTR(str)        L##str
    #define STRING(str)        (str)
    #define WIDESTRING(str)  (str)
# else
    #define TCHAR      char
    #define TSTR(str)        str
    #define STRING(str)        AnsiString(str)
    #define WIDESTRING(str)  WideString(str)
    extern "C" char FAR* AnsiString(OLECHAR FAR* strIn);
    extern "C" OLECHAR FAR* WideString(char FAR* strIn);
# endif
```

```
#else /* WIN16 */

# include <windows.h>
# include <ole2.h>
# include <olenls.h>
# include <dispatch.h>

# define TCHAR                char
# define TSTR(str)            str
# define STRING(str)            (str)
# define WIDESTRING(str)         (str)
#endif
```

IDATA.CPP   (DISPCALC Sample)

```
/***
*idata.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This file contains the INTERFACEDATA definitions for the methods
*  and properties on the CCalc object that are exposed for external
*  programmability via IDispatch.
*
*
*Implementation Notes:
*
*****************************************************************************
*/

#include "dispcalc.h"


//---------------------------------------------------------------------
//                    INTERFACEDATA definitions
//---------------------------------------------------------------------

/* The INTERFACEDATA table describes the methods and properties that
 * are being exposed for external programmability via IDispatch.
 * This table is used to construct a CDispTypeInfo for this interface,
 * and that TypeInfo is in turn used to initialize and drive the
 * default implementation of IDispatch.
 */


// PDATA1() declares the PARAMDATA for a methods that takes a single param
//
#define PDATA1(NAME, TYPE) \
    static PARAMDATA NEARDATA rgpdata ## NAME = {OLESTR(#NAME), TYPE}

// MDATA() declares a single METHODDATA entry
//
#define MDATA(NAME, PDATA, IDMEMBER, IMETH, CARGS, KIND, TYPE) \
    { OLESTR(#NAME), PDATA, IDMEMBER, IMETH, CC_CDECL, CARGS, KIND, TYPE }

// The following macro defines the METHODDATA entries for a
// property Put/Get method pair.
//
// Note: this macro *assumes* that the Put/Get pair are adjacent
// in the vtable, and that the Put method comes first.
//
#define PROPERTY(NAME, IMETH, ID, TYPE) \
    MDATA(NAME, &rgpdata ## NAME, ID, IMETH,
1,DISPATCH_PROPERTYPUT,VT_EMPTY), \
    MDATA(NAME, NULL, ID, IMETH+1, 0, DISPATCH_PROPERTYGET, TYPE)
```

```
// The following macro is used to define a METHODDATA entry for
// a method that takes zero parameters.
//
#define METHOD0(NAME, IMETH, ID, TYPE) \
    MDATA(NAME, NULL, ID, IMETH, 0, DISPATCH_METHOD, TYPE)

// and for one param
#define METHOD1(NAME, IMETH, ID, TYPE) \
    MDATA(NAME, &rgpdata ## NAME, ID, IMETH, 1, DISPATCH_METHOD, TYPE)


PDATA1(VALUE,  VT_I4);
PDATA1(ACCUM,  VT_I4);
PDATA1(OPND,   VT_I4);
PDATA1(OP,     VT_I2);
PDATA1(BUTTON, VT_BSTR);

static METHODDATA NEARDATA rgmdataCCalc[] =
{
      PROPERTY(VALUE,   IMETH_ACCUM,    IDMEMBER_ACCUM,    VT_I4)
    , PROPERTY(ACCUM,   IMETH_ACCUM,    IDMEMBER_ACCUM,    VT_I4)
    , PROPERTY(OPND,    IMETH_OPERAND,  IDMEMBER_OPERAND,  VT_I4)
    , PROPERTY(OP,      IMETH_OPERATOR, IDMEMBER_OPERATOR, VT_I2)
    ,  METHOD0(EVAL,    IMETH_EVAL,     IDMEMBER_EVAL,     VT_BOOL)
    ,  METHOD0(CLEAR,   IMETH_CLEAR,    IDMEMBER_CLEAR,    VT_EMPTY)
    ,  METHOD0(DISPLAY, IMETH_DISPLAY,  IDMEMBER_DISPLAY,  VT_EMPTY)
    ,  METHOD0(QUIT,    IMETH_QUIT,     IDMEMBER_QUIT,     VT_EMPTY)
    ,  METHOD1(BUTTON,  IMETH_BUTTON,   IDMEMBER_BUTTON,   VT_BOOL)
};

INTERFACEDATA NEARDATA g_idataCCalc =
{
    rgmdataCCalc, DIM(rgmdataCCalc)
};
```

RESOURCE.H   (DISPCALC Sample)

```
/***
*resource.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*
*Implementation Notes:
*
****************************************************************************
*/

#ifdef _MAC

#define kMinSize 500   /* minimum size (in K) */
#define kPrefSize500   /* preferred size (in K) */

#define kMinHeap 21 * 1024
#define kMinSpace8 * 1024

#define    rMenuBar    128   /* menu bar */
#define    rAboutAlert 128   /* about alert */
#define    rUserAlert  129   /* error alert */

#define    rCalc       130

#define    mApple            128   /* Apple menu */
#define    iAbout            1

#define    mFile       129   /* File menu */
#define iClose         4
#define    iQuit       12

#define    mEdit       130   /* Edit menu */
#define    iUndo       1
#define    iCut        3
#define    iCopy       4
#define    iPaste            5
#define    iClear            6

#endif

// Note: there is code that depends on all of the digits being contiguous
// and in the following order.

// Mac Note: On the mac these IDs correspond to the control indices
// in the DITL array.

#define IDC_ZERO     1
#define IDC_ONE      2
#define IDC_TWO      3
#define IDC_THREE    4
```

```
#define IDC_FOUR     5
#define IDC_FIVE     6
#define IDC_SIX      7
#define IDC_SEVEN    8
#define IDC_EIGHT    9
#define IDC_NINE     10

// Note: there is code that depends on the operators being contiguous
// and in the following order.
//
#define IDC_PLUS     11
#define IDC_MINUS    12
#define IDC_MULT     13
#define IDC_DIV      14

#define IDC_CLEAR    15
#define IDC_EQUALS   16

#define IDC_DISPLAY  17
```

WINMAIN.CPP   (DISPCALC Sample)

```
/***
*main.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This module is the main entry point of the sample IDispatch
*  calculator, dispcalc.exe
*
*  This program is intended to demonstrate an implementation of
*  the IDispatch interface.
*
*Implementation Notes:
*
*****************************************************************************
*/

#include "dispcalc.h"

TCHAR g_szAppName[] = TSTR("DispCalc");

BOOL InitApplication(HANDLE);
BOOL InitInstance(HANDLE, int);


extern "C" {
long FAR PASCAL WndProc(HWND, UINT, WPARAM, LPARAM);
int PASCAL WinMain(HANDLE, HANDLE, LPSTR, int);
}


extern "C" int PASCAL
WinMain(
    HANDLE hinst,
    HANDLE hinstPrev,
    LPSTR lpCmdLine,
    int nCmdShow)
{
    MSG msg;

    if(!hinstPrev)
      if(!InitApplication(hinst))
        return FALSE;

    if(InitOle() != NOERROR)
      return FALSE;

    if(!InitInstance(hinst, nCmdShow)){
      UninitOle();
      return FALSE;
    }
    while(GetMessage(&msg, NULL, NULL, NULL)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }

    return msg.wParam;
}


BOOL
InitApplication(HANDLE hinst)
{
    WNDCLASS  wc;

    wc.style            = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc      = WndProc;
    wc.cbClsExtra       = 0;
    wc.cbWndExtra       = DLGWINDOWEXTRA;
    wc.hInstance        = hinst;
    wc.hIcon            = LoadIcon(hinst, g_szAppName);
    wc.hCursor          = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground    = (HBRUSH)(COLOR_APPWORKSPACE+1);
    wc.lpszMenuName     = NULL;
    wc.lpszClassName    = g_szAppName;

    if(!RegisterClass(&wc))
      return FALSE;

    return TRUE;
}


BOOL
InitInstance(HANDLE hinst, int nCmdShow)
{
    g_pcalc->m_hwnd = CreateDialog(hinst, g_szAppName, 0, NULL);

    ShowWindow(g_pcalc->m_hwnd, nCmdShow);

    g_pcalc->m_arith.Display();

    return TRUE;
}


extern "C" long FAR PASCAL
WndProc(
    HWND hwnd,
    UINT message,
    WPARAM wParam,
    LPARAM lParam)
{
    switch(message){
    case WM_COMMAND:
      switch(wParam){
      case IDC_ZERO:
      case IDC_ONE:
      case IDC_TWO:
      case IDC_THREE:
      case IDC_FOUR:
      case IDC_FIVE:
      case IDC_SIX:
      case IDC_SEVEN:
      case IDC_EIGHT:
      case IDC_NINE:
      case IDC_PLUS:
      case IDC_MINUS:
      case IDC_MULT:
      case IDC_DIV:
      case IDC_CLEAR:
      case IDC_EQUALS:
        g_pcalc->m_arith.ButtonPush(wParam);
        return 0;
      }
      break;

    case WM_DESTROY:
      UninitOle();
      PostQuitMessage(0);
      return 0;
    }
    return DefWindowProc(hwnd, message, wParam, lParam);
}


#if defined(WIN32)

extern "C" char FAR*
ConvertStrWtoA(OLECHAR FAR* strIn, char FAR* buf, UINT size)
{
  int badConversion = FALSE;

  WideCharToMultiByte(CP_ACP, NULL,
                      strIn, -1,
                      buf, size,
                      NULL, &badConversion);
  return buf;
}

extern "C" char FAR*
AnsiString(OLECHAR FAR* strIn)
{
  static char buf[256];

  return (ConvertStrWtoA(strIn, buf, 256));
}

#endif
```

DISPDEMO

--------------------------------------- OLE Automation Sample Program: DispDemo ---------------------------------------

DispDemo is a sample OLE Automation controller.   DispDemo drives the two polygon servers which are also provided as samples.

----------------- Program Structure ----------------- The interesting part of DispDemo is in the remote polygon class.   The remote polygon class was constructed so that   the IDispatch members could be accessed conveniently from   C++.   The remote polygon class does this by transforming the C++ calls into IDispatch calls.   The code that calls these routines has no idea that it is going through IDispatch.

MAKEFILE   (DISPDEMO Sample)

```
############################################################################
##
#
#  (c) Copyright Microsoft Corp. 1992-1993 All Rights Reserved
#
#  File:
#
#    makefile - makefile for dispdemo.exe
#
#  Purpose:
#
#    Builds the OLE 2.0 IDispatch sample client application, dispdemo.exe.
#
#
#  Usage:
#
#     NMAKE                      ; build with defaults
#     or: NMAKE option           ; build with the given option(s)
#     or: NMAKE clean            ; erase all compiled files
#
#     option:
#         dev = [win16 | win32] ; dev=win32 is the default
#         DEBUG=[0|1]           ; DEBUG=1 is the default
#
#
#  Notes:
#
#    This makefile assumes that the PATH, INCLUDE and LIB environment
#    variables are setup properly.
#
############################################################################
##



##########################################################################
#
# Default Settings
#

!if "$(dev)" == ""
dev = win32
!endif

!if !("$(dev)" == "win16" || "$(dev)" == "win32")
!error Invalid dev option, choose from [win16 | win32]
!endif

!if "$(dev)" == "win16"
TARGET  = WIN16
!endif
```

```
!if "$(dev)" == "win32"
TARGET  = WIN32
!endif

!ifdef NODEBUG
DEBUG = 0
!endif

!if "$(DEBUG)" == "0"
NODEBUG = 1
!endif

!if "$(DEBUG)" == ""
DEBUG = 1
!endif


##########################################################################
#
# WIN16 Settings
#
!if "$(TARGET)" == "WIN16"

CC   = cl
LINK = link

RCFLAGS = -dWIN16
CFLAGS = -W3 -AM -GA -GEs -DWIN16 -c
LINKFLAGS = /NOD /NOI /BATCH /ONERROR:NOEXE

LIBS = libw.lib mlibcew.lib

!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS) -Od -Zi -D_DEBUG $(CL)
LINKFLAGS = $(LINKFLAGS) /COD
!else
CFLAGS = $(CFLAGS) -Ox $(CL)
LINKFLAGS = $(LINKFLAGS) /FAR /PACKC
!endif
!endif


##########################################################################
#
# WIN32 Settings
#
!if "$(TARGET)" == "WIN32"

!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

CC = $(cc)
CFLAGS = $(cflags) $(cvarsmt) -DINC_OLE2 -DUNICODE $(cdebug)

!ifndef NODEBUG
CFLAGS = $(CFLAGS) -D_DEBUG
```

```
!endif

LINK = $(link)
LINKFLAGS = $(linkdebug) $(guilflags)
RCFLAGS = -DWIN32

!endif


##########################################################################
#
# Build rules
#

.cpp.obj:
    @echo Compiling $<...
    $(CC) $<

.c.obj:
    @echo Compiling $<...
    $(CC) $<


##########################################################################
#
# Application Settings
#

APPS = dispdemo


!if "$(TARGET)" == "WIN16"
LIBS = ole2.lib compobj.lib ole2disp.lib $(LIBS)
!endif
!if "$(TARGET)" == "WIN32"
LIBS = $(olelibsmt)
!endif

OBJS = \
         winmain.obj \
         misc.obj \
         crempoly.obj \
         clsid.obj


##########################################################################
#
# Default Goal
#

goal : setflags $(APPS).exe

setflags :
        set CL=$(CFLAGS)
```

```
###########################################################################
#
# Clean (erase) generated files
#
clean :
    if exist *.obj       del *.obj
    if exist $(APPS).exe del $(APPS).exe
    if exist $(APPS).map del $(APPS).map
    if exist $(APPS).res del $(APPS).res


###########################################################################
#
# Application Build (WIN16 Specific)
#

!if "$(TARGET)" == "WIN16"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
        link $(LINKFLAGS) @<<
$(OBJS),
$@,,
$(LIBS),
$(APPS).def
<<
        rc -k -t $(APPS).res $@
!endif


###########################################################################
#
# Application Build (WIN32 Specific)
#
!if "$(TARGET)" == "WIN32"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
      $(LINK) @<<
        $(LINKFLAGS)
        -out:$@
        -map:$*.map
        $(OBJS)
        $(APPS).res
        $(LIBS)
<<
!endif


###########################################################################
#
# Application Build (Common)
#

$(APPS).res : $(APPS).rc
        rc $(RCFLAGS) -r -fo$@ $?
```

```
##########################################################################
#
# Dependencies
#

winmain.obj: winmain.cpp dispdemo.h
    $(CC) winmain.cpp

misc.obj: misc.cpp dispdemo.h
    $(CC) misc.cpp

crempoly.obj: crempoly.cpp crempoly.h
    $(CC) crempoly.cpp

clsid.obj: clsid.c clsid.h
    $(CC) clsid.c
```

CLSID.H (DISPDEMO Sample)

```
/***
*clsid.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This file defines the CLSIDs referenced by the IDispatch demo app.
*
*Implementation Notes:
*
*****************************************************************************
*/

DEFINE_GUID(CLSID_CPoly,  0x00020462, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);
DEFINE_GUID(CLSID_CPoly2, 0x00020464, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);


#ifdef INITGUID
# ifdef _MAC
DEFINE_GUID(GUID_NULL, 0L, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
DEFINE_GUID(IID_IDispatch, 0x00020400, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);
DEFINE_GUID(IID_IEnumVARIANT, 0x00020404, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0,
0x46);
DEFINE_GUID(IID_IUnknown, 0x00000000, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);
DEFINE_GUID(IID_IClassFactory, 0x00000001, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0,
0x46);
# endif
#endif
```

CLSID.C (DISPDEMO Sample)

```
/***
*clsid.c
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This file allocates and initializes the CLSIDs.
*
****************************************************************************
*/

#ifdef _PPCMAC
#pragma data_seg ("_FAR_DATA")
#pragma data_seg ( )
#endif

#ifdef _MAC
# include <Types.h>
#ifdef _MSC_VER
# include <Processe.h>
# include <AppleEve.h>
#else //_MSC_VER
# include <Processes.h>
# include <AppleEvents.h>
#endif //_MSC_VER
#else
# include <windows.h>
#endif

#ifndef WIN32
#include <compobj.h>
#endif //!WIN32

// this redefines the DEFINE_GUID() macro to do allocation.
//
#include <initguid.h>

#ifndef INITGUID
# define INITGUID
#endif

// due to the previous header, including this causes the DEFINE_GUID
// definitions in the following header(s) to actually allocate data.
//
#include "clsid.h"
```

CREMPOLY.H   (DISPDEMO Sample)

```
/***
*crempoly.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This file defines the CRemPoly remote polygon object.
*
*Implementation Notes:
*
*****************************************************************************
*/

class CRemPoly : public IUnknown {
public:
    static HRESULT Create(CLSID clsid, CRemPoly FAR* FAR*);

    // IUnknown methods
    //
    STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppvObj);
    STDMETHOD_(unsigned long, AddRef)(void);
    STDMETHOD_(unsigned long, Release)(void);

    // Introduced methods
    //
    HRESULT Draw(void);
    HRESULT Dump(void);
    HRESULT Reset(void);
    HRESULT AddPoint(short x, short y);
    HRESULT EnumPoints(IEnumVARIANT FAR* FAR* ppenum);
    HRESULT GetXOrigin(short FAR* pxorg);
    HRESULT SetXOrigin(short xorg);
    HRESULT GetYOrigin(short FAR* pyorg);
    HRESULT SetYOrigin(short yorg);
    HRESULT GetWidth(short FAR* pwidth);
    HRESULT SetWidth(short width);

    HRESULT get_red(short FAR* psRed);
    HRESULT set_red(short sRed);
    HRESULT get_green(short FAR* psGreen);
    HRESULT set_green(short sGreen);
    HRESULT get_blue(short FAR* psBlue);
    HRESULT set_blue(short sBlue);

private:
    CRemPoly();

    HRESULT get_i2(DISPID dispid, short FAR* ps);
    HRESULT set_i2(DISPID dispid, short s);

    unsigned long m_refs;
    IDispatch FAR* m_pdisp;

    // NOTE: this enumeration exists simply to allow us to symbolicly
    // index the member name and id arrays (m_rgid and m_rgszMethods).
    // This doesn't (necessarrily) have any connection to the vtable
    // indices, it *only* needs to correspond correctly to the m_rgid
    // and m_rgszMethods arrays.
    //
    enum CREMPOLY_METHODS {
     IMETH_CREMPOLY_DRAW = 0,
     IMETH_CREMPOLY_DUMP,
     IMETH_CREMPOLY_RESET,
     IMETH_CREMPOLY_ADDPOINT,
     IMETH_CREMPOLY_ENUMPOINTS,
     IMETH_CREMPOLY_GETXORIGIN,
     IMETH_CREMPOLY_SETXORIGIN,
     IMETH_CREMPOLY_GETYORIGIN,
     IMETH_CREMPOLY_SETYORIGIN,
     IMETH_CREMPOLY_GETWIDTH,
     IMETH_CREMPOLY_SETWIDTH,
     IMETH_CREMPOLY_GETRED,
     IMETH_CREMPOLY_SETRED,
     IMETH_CREMPOLY_GETGREEN,
     IMETH_CREMPOLY_SETGREEN,
     IMETH_CREMPOLY_GETBLUE,
     IMETH_CREMPOLY_SETBLUE,
     IMETH_CREMPOLY_MAX
    };

    // member IDs - these are used by IDispatch::Invoke to identify the
    // method or property on the remote object we accessing.
    //
    DISPID m_rgdispid[IMETH_CREMPOLY_MAX];

    // member names - these are used to learn the member IDs when we
    // connect to the remote object.
    //
    static OLECHAR FAR* m_rgszMethods[IMETH_CREMPOLY_MAX];
};
```

CREMPOLY.CPP   (DISPDEMO Sample)

```
/***
*crempoly.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This file contains the implementation of CRemPoly, the remote polygon
*  class. This class presents a standard C++ vtable interface to the
*  rest of the application, and hides the details of talking to the
*  actual remote CPoly class exposed by the SPoly server. Each of
*  the introduced methods is simply a cover for an IDispatch invocation
*  of the actual method on the remote object.
*
*Implementation Notes:
*
*****************************************************************************
*/

#include "dispdemo.h"
#include "crempoly.h"

extern int g_fTrace;

// method names on the CPoly class.
//
OLECHAR FAR* CRemPoly::m_rgszMethods[] = {
    OLESTR("draw"),
    OLESTR("dump"),
    OLESTR("reset"),
    OLESTR("addpoint"),
    OLESTR("enumpoints"),
    OLESTR("getxorigin"),
    OLESTR("setxorigin"),
    OLESTR("getyorigin"),
    OLESTR("setyorigin"),
    OLESTR("getwidth"),
    OLESTR("setwidth"),
    OLESTR("get_red"),
    OLESTR("set_red"),
    OLESTR("get_green"),
    OLESTR("set_green"),
    OLESTR("get_blue"),
    OLESTR("set_blue")
};

#ifdef _MAC
# define IfMac(X) (X)
# define IfWin(X)
#else
# define IfMac(X)
# define IfWin(X) (X)
```

```
#endif


CRemPoly::CRemPoly()
{
    m_refs = 0;
    m_pdisp = (IDispatch FAR*)NULL;
}


// A useful pre-initialized DISPATCHPARAMS, used on all the methods that
// take 0 arguments.
//
DISPPARAMS NEAR g_dispparamsNoArgs = {NULL, NULL, 0, 0};


/***
*HRESULT CRemPoly::Create(clsid, CRemPoly**)
*
*Purpose:
*  This function creates an instance of the CRemPoly class, connects
*  it to the IDispatch interface of the remote CPoly class, and learns
*  the DISPIDs for the members (that we know about) exposed by that
*  class.
*
*Entry:
*  clsid = The CLSID of the CPoly we are to create. (taking this as a
*    param is a bit weird, but allows us to connect to several remote
*    versions.
*
*Exit:
*  return value = HRESULT
*
*  *pprempoly = pointer to the newly created CRemPoly, if successfyl.
*
***********************************************************************/
HRESULT
CRemPoly::Create(CLSID clsid, CRemPoly FAR* FAR* pprempoly)
{
    int i;
    HRESULT hresult;
    IUnknown FAR* punk;
    CRemPoly FAR* prempoly;


    prempoly = new FAR CRemPoly();
    if(prempoly == (CRemPoly FAR*)NULL){
      hresult = ResultFromScode(E_OUTOFMEMORY);
      goto LError;
    }
    prempoly->AddRef();

    // create an instance of the remote CPoly class.
    //
    IfMac(DbPrintf("CoCreateInstance(CLSID_CPoly)"));
    hresult = CoCreateInstance(
      clsid, NULL, CLSCTX_LOCAL_SERVER, IID_IUnknown, (void FAR*
FAR*)&punk);
    if(hresult != NOERROR){
      IfMac(DbPrintf("CoCreateInstance() = 0x%x", hresult));
      IfWin(MessageBox(NULL, TSTR("Unable to create polygon object"),
                    NULL, MB_OK));
      goto LFreeCRemPoly;
    }

    // were going to talk to this remote instance via IDispatch.
    //
    IfMac(DbPrintf("QueryInterface(IID_IDispatch)"));
    hresult = punk->QueryInterface(
      IID_IDispatch, (void FAR* FAR*)&prempoly->m_pdisp);
    if(hresult != NOERROR){
      IfMac(DbPrintf("QueryInterface(IID_IDispatch) = 0x%x", hresult));
      IfWin(MessageBox(NULL, TSTR("Unable to QueryInterface to IDispatch"),
                    NULL, MB_OK));
      goto LReleaseUnk;
    }


    // We learn *all* the member IDs up front. A more sophisticated
    // implementation might defer learning about the IDs for a given
    // method until the first time the method is invoked, thereby
    // amortizing the creation costs.
    //
    IfMac(DbPrintf("GetIDsOfNames()"));
    for(i = 0; i < IMETH_CREMPOLY_MAX; ++i){
      hresult = prempoly->m_pdisp->GetIDsOfNames(
     IID_NULL,
        &prempoly->m_rgszMethods[i],
     1, LOCALE_USER_DEFAULT,
     &prempoly->m_rgdispid[i]);
      if(hresult != NOERROR){
     IfMac(DbPrintf("GetIDsOfNames() = 0x%x", hresult));
     IfWin(MessageBox(NULL, TSTR("Unrecognized member name"),
                    NULL, MB_OK));
     goto LReleaseUnk;
      }
    }

    punk->Release();

    *pprempoly = prempoly;

    IfMac(DbPrintf("Object created."));

    return NOERROR;

LReleaseUnk:;
    punk->Release();

LFreeCRemPoly:;
```

```
    prempoly->Release();

LError:;
    return hresult;
}


//---------------------------------------------------------------------
//                     IUnknown methods
//---------------------------------------------------------------------


/***
*HRESULT CRemPoly::QueryInterface(REFIID, void**)
*
*Purpose:
*  Standard Ole2 implementation of QueryInterface. This class
*  supports the IUnknown interface, and introduces a number of
*  nonvirtual members.
*
*Entry:
*  riid = reference to the requested interface id
*
*Exit:
*  return value = HRESULT
*  *ppv = pointer to the requested interface, if successful.
*
***********************************************************************/
STDMETHODIMP
CRemPoly::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    if(IsEqualIID(riid, IID_IUnknown)){
      *ppv = this;
      AddRef();
      return NOERROR;
    }
    *ppv = NULL;
    return ResultFromScode(E_NOINTERFACE);
}


/***
*unsigned long CRemPoly::AddRef(void)
*
*Purpose:
*  Add a reference to the instance.
*
*Entry:
*  None
*
*Exit:
*  return value = unsigned long. The resulting reference count.
*
***********************************************************************/
STDMETHODIMP_(unsigned long)
```

```
CRemPoly::AddRef(void)
{
    return ++m_refs;
}


/***
*unsigned long CRemPoly::Release(void)
*
*Purpose:
*  Release a reference to the instance. If the reference count goes
*  to zero, delete the instance.
*
*Entry:
*  None
*
*Exit:
*  return value = unsigned long. The resulting reference count.
*
***********************************************************************/
STDMETHODIMP_(unsigned long)
CRemPoly::Release(void)
{
    if(--m_refs == 0){
      if(m_pdisp != (IDispatch FAR*)NULL){
        m_pdisp->Release();
      }
      delete this;
      return 0;
    }
    return m_refs;
}


//---------------------------------------------------------------------
//                    Introduced methods
//---------------------------------------------------------------------


/*
 * Each of these methods is simply a cover for an IDispatch Invocation
 * of the actual method on the remote CPoly class. This allows CRemPoly
 * to present an interface that looks and acts just like the CPoly
 * object, even though the actual work is being done in another process.
 *
 */

/***
*HRESULT CRemPoly::Draw(void)
*
*Purpose:
*  Invoke the Draw method on the remote CPoly instance.
*
*Entry:
```

```
*  None
*
*Exit:
*  return value = HRESULT
*
***********************************************************************/
HRESULT
CRemPoly::Draw()
{
    return m_pdisp->Invoke(
      m_rgdispid[IMETH_CREMPOLY_DRAW],
      IID_NULL,
      LOCALE_USER_DEFAULT,
      DISPATCH_METHOD,
      &g_dispparamsNoArgs, NULL, NULL, NULL);
}


/***
*HRESULT CRemPoly::Dump(void)
*
*Purpose:
*  Invoke the Dump() method on the remote CPoly instance. This method
*  dumps the contained CPoints and writes the properties of the remote
*  CPoly instance to the debug window.
*
*Entry:
*  None
*
*Exit:
*  return value = HRESULT
*
***********************************************************************/
HRESULT
CRemPoly::Dump()
{
    return m_pdisp->Invoke(
      m_rgdispid[IMETH_CREMPOLY_DUMP],
      IID_NULL,
      LOCALE_USER_DEFAULT,
      DISPATCH_METHOD,
      &g_dispparamsNoArgs, NULL, NULL, NULL);
}


/***
*HRESULT CRemPoly::Reset(void)
*
*Purpose:
*  Invoke the Reset() method on the remote CPoly instance. The Reset()
*  method causes the remote CPoly to release all contained CPoints.
*
*Entry:
*  None
*
```

```
*Exit:
*  return value = HRESULT
*
***********************************************************************/
HRESULT
CRemPoly::Reset()
{
    return m_pdisp->Invoke(
      m_rgdispid[IMETH_CREMPOLY_RESET],
      IID_NULL,
      LOCALE_USER_DEFAULT,
      DISPATCH_METHOD,
      &g_dispparamsNoArgs, NULL, NULL, NULL);
}


/***
*HRESULT CRemPoly::AddPoint(short, short)
*
*Purpose:
*  Invoke the AddPoint method in the remote CPoly object to add a
*  new point with the given coordinates to this instance.
*
*Entry:
*  x,y = the x and y coordinates of the new point.
*
*Exit:
*  return value = HRESULT
*
***********************************************************************/
HRESULT
CRemPoly::AddPoint(short x, short y)
{
    HRESULT hresult;
    VARIANTARG varg[2];
    DISPPARAMS dispparams;

    V_VT(&varg[0]) = VT_I2;
    V_I2(&varg[0]) = y;

    V_VT(&varg[1]) = VT_I2;
    V_I2(&varg[1]) = x;

    dispparams.cArgs = 2;
    dispparams.rgvarg = varg;
    dispparams.cNamedArgs = 0;
    dispparams.rgdispidNamedArgs = NULL;

    hresult = m_pdisp->Invoke(
      m_rgdispid[IMETH_CREMPOLY_ADDPOINT],
      IID_NULL,
      LOCALE_USER_DEFAULT,
      DISPATCH_METHOD,
      &dispparams, NULL, NULL, NULL);
    return hresult;
}


/***
*HRESULT CRemPoly::EnumPoints(IEnumVARIANT**)
*Purpose:
*  Inoke the EnumPoints() method in the remote object to
*  get a enumerator for the points contained in the current poly.
*
*Entry:
*  None
*
*Exit:
*  return value = HRESULT
*
*  *ppenum = pointer to the point enumerator
*
***********************************************************************/
HRESULT
CRemPoly::EnumPoints(IEnumVARIANT FAR* FAR* ppenum)
{
    HRESULT hresult;
    IEnumVARIANT FAR* penum;
    VARIANT varResult, FAR* pvarResult;


    pvarResult = &varResult;
    VariantInit(pvarResult);
    hresult = m_pdisp->Invoke(
      m_rgdispid[IMETH_CREMPOLY_ENUMPOINTS],
      IID_NULL,
      LOCALE_USER_DEFAULT,
      DISPATCH_METHOD,
      &g_dispparamsNoArgs, pvarResult, NULL, NULL);

    if(hresult != NOERROR)
      return hresult;

    if(V_VT(pvarResult) != VT_UNKNOWN)
      return ResultFromScode(E_FAIL);

    hresult = V_UNKNOWN(pvarResult)->QueryInterface(
      IID_IEnumVARIANT, (void FAR* FAR*)&penum);

    if(hresult == NOERROR)
      *ppenum = penum;

    VariantClear(pvarResult);

    return NOERROR;
}
```

```
/***
*HRESULT CRemPoly::GetXOrigin(short*)
*
*Purpose:
*  Invoke the GetXOrigin() method on the remote object to extract
*  the current value of the XOrigin property.
*
*Entry:
*  None
*
*Exit:
*  return value = HRESULT
*
*  *pxorg = the current X origin of the polygon.
*
************************************************************************/
HRESULT
CRemPoly::GetXOrigin(short FAR* pxorg)
{
    HRESULT hresult;
    VARIANT varResult;

    VariantInit(&varResult);
    hresult = m_pdisp->Invoke(
      m_rgdispid[IMETH_CREMPOLY_GETXORIGIN],
      IID_NULL,
      LOCALE_USER_DEFAULT,
      DISPATCH_METHOD,
      &g_dispparamsNoArgs, &varResult, NULL, NULL);

    if(hresult != NOERROR)
      return hresult;

    *pxorg = V_I2(&varResult);
    VariantClear(&varResult);

    return NOERROR;
}


/***
*HRESULT CRemPoly::SetXOrigin(short)
*
*Purpose:
*  Invoke the SetXOrigin method on the remote object to set the
*  XOrigin property of the polygon to the given value.
*
*Entry:
*  xorg = the new X origin
*
*Exit:
*  return value = HRESULT
*
************************************************************************/
HRESULT
```

```
CRemPoly::SetXOrigin(short xorg)
{
    VARIANTARG varg;
    DISPPARAMS dispparams;


    V_VT(&varg) = VT_I2;
    V_I2(&varg) = xorg;

    dispparams.cArgs = 1;
    dispparams.cNamedArgs = 0;
    dispparams.rgvarg = &varg;

    return m_pdisp->Invoke(
      m_rgdispid[IMETH_CREMPOLY_SETXORIGIN],
      IID_NULL,
      LOCALE_USER_DEFAULT,
      DISPATCH_METHOD,
      &dispparams, NULL, NULL, NULL);
}


/***
*HRESULT CRemPoly::GetYOrigin(short*)
*
*Purpose:
*  Invoke the GetYOrigin() method on the remote object to extract
*  the current value of the YOrigin property.
*
*Entry:
*  None
*
*Exit:
*  return value = HRESULT
*
*  *pyorg = the current Y origin of the polygon
*
***********************************************************************/
HRESULT
CRemPoly::GetYOrigin(short FAR* pyorg)
{
    HRESULT hresult;
    VARIANT varResult;


    VariantInit(&varResult);
    hresult = m_pdisp->Invoke(
      m_rgdispid[IMETH_CREMPOLY_GETYORIGIN],
      IID_NULL,
      LOCALE_USER_DEFAULT,
      DISPATCH_METHOD,
      &g_dispparamsNoArgs, &varResult, NULL, NULL);

    if(hresult != NOERROR)
```

```
      return hresult;

    *pyorg = V_I2(&varResult);
    VariantClear(&varResult);

    return hresult;
}


/***
*HRESULT CRemPoly::SetYOrigin(short)
*
*Purpose:
*  Invoke the SetYOrigin method on the remote object to set the
*  YOrigin property of the polygon to the given value.
*
*Entry:
*  yorg = the new Y origin
*
*Exit:
*  return value = HRESULT
*
***********************************************************************/
HRESULT
CRemPoly::SetYOrigin(short yorg)
{
    VARIANTARG varg;
    DISPPARAMS dispparams;

    V_VT(&varg) = VT_I2;
    V_I2(&varg) = yorg;

    dispparams.cArgs = 1;
    dispparams.cNamedArgs = 0;
    dispparams.rgvarg = &varg;

    return m_pdisp->Invoke(
      m_rgdispid[IMETH_CREMPOLY_SETYORIGIN],
      IID_NULL,
      LOCALE_USER_DEFAULT,
      DISPATCH_METHOD,
      &dispparams, NULL, NULL, NULL);
}


/***
*HRESULT CRemPoly::GetWidth(short*)
*
*Purpose:
*  Invoke the GetWidth() method on the remote object to extract
*  the current value of the line width property.
*
*Entry:
*  None
*
```

```
*Exit:
*  return value = HRESULT
*
*  *pwidth = short, the current line width of the polygon
*
************************************************************************/
HRESULT
CRemPoly::GetWidth(short FAR* pwidth)
{
    HRESULT hresult;
    VARIANT varResult;

    VariantInit(&varResult);
    hresult = m_pdisp->Invoke(
      m_rgdispid[IMETH_CREMPOLY_GETWIDTH],
      IID_NULL,
      LOCALE_USER_DEFAULT,
      DISPATCH_METHOD,
      &g_dispparamsNoArgs, &varResult, NULL, NULL);

    if(hresult != NOERROR)
      return hresult;

    *pwidth = V_I2(&varResult);
    VariantClear(&varResult);
    return NOERROR;
}


/***
*HRESULT CRemPoly::SetWidth(short)
*
*Purpose:
*  Invoke the SetWidth method on the remote object to set the
*  line width property of the polygon to the given value.
*
*Entry:
*  width = the new value for the line width property.
*
*Exit:
*  return value = HRESULT
*
************************************************************************/
HRESULT
CRemPoly::SetWidth(short width)
{
    VARIANTARG varg;
    DISPPARAMS dispparams;


    V_VT(&varg) = VT_I2;
    V_I2(&varg) = width;

    dispparams.cArgs = 1;
    dispparams.cNamedArgs = 0;
    dispparams.rgvarg = &varg;

    return m_pdisp->Invoke(
      m_rgdispid[IMETH_CREMPOLY_SETWIDTH],
      IID_NULL,
      LOCALE_USER_DEFAULT,
      DISPATCH_METHOD,
      &dispparams, NULL, NULL, NULL);
}


HRESULT CRemPoly::get_red(short FAR* psRed)
{
    return get_i2(m_rgdispid[IMETH_CREMPOLY_GETRED], psRed);
}

HRESULT CRemPoly::set_red(short sRed)
{
    return set_i2(m_rgdispid[IMETH_CREMPOLY_SETRED], sRed);
}


HRESULT CRemPoly::get_green(short FAR* psGreen)
{
    return get_i2(m_rgdispid[IMETH_CREMPOLY_GETGREEN], psGreen);
}

HRESULT CRemPoly::set_green(short sGreen)
{
    return set_i2(m_rgdispid[IMETH_CREMPOLY_SETGREEN], sGreen);
}


HRESULT CRemPoly::get_blue(short FAR* psBlue)
{
    return get_i2(m_rgdispid[IMETH_CREMPOLY_GETBLUE], psBlue);
}

HRESULT CRemPoly::set_blue(short sBlue)
{
    return set_i2(m_rgdispid[IMETH_CREMPOLY_SETBLUE], sBlue);
}

HRESULT
CRemPoly::get_i2(DISPID dispid, short FAR* ps)
{
    HRESULT hresult;
    VARIANT varResult;

    VariantInit(&varResult);

    hresult = m_pdisp->Invoke(
      dispid,
      IID_NULL,
      LOCALE_SYSTEM_DEFAULT,
      DISPATCH_METHOD,
      &g_dispparamsNoArgs,
      &varResult, NULL, NULL);

    if(hresult != NOERROR)
      return hresult;

    hresult = VariantChangeType(&varResult, &varResult, 0, VT_I2);
    if(hresult != NOERROR){
      VariantClear(&varResult);
      return hresult;
    }

    *ps = V_I2(&varResult);
    VariantClear(&varResult);
    return NOERROR;
}

HRESULT
CRemPoly::set_i2(DISPID dispid, short s)
{
    VARIANTARG varg;
    DISPPARAMS dispparams;

    V_VT(&varg) = VT_I2;
    V_I2(&varg) = s;

    dispparams.cArgs = 1;
    dispparams.cNamedArgs = 0;
    dispparams.rgvarg = &varg;

    return m_pdisp->Invoke(
      dispid,
      IID_NULL,
      LOCALE_SYSTEM_DEFAULT,
      DISPATCH_METHOD,
      &dispparams, NULL, NULL, NULL);
}


/***
*void DoPoly(CLSID)
*
*Purpose:
*  This function simply exercises our CRemPoly class by creating an
*  instance and invoking a number of its methods.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
```

```
STDAPI
DoPoly(CLSID clsid)
{
    HRESULT hr;
    int numpoly, i, j;

static struct {
    short x;
    short y;
} rgptPoly[] = {
      { 25,   0}
    , { 75,   0}
    , {100,  25}
    , {100,  75}
    , { 75, 100}
    , { 25, 100}
    , {  0,  75}
    , {  0,  25}
};

static struct {
    short red;
    short green;
    short blue;
} rgrgbColors[] = {
#ifdef _MAC
      {     0,      0,      0}
    , {     0,      0, 0x7fff}
    , {     0, 0x7fff,      0}
    , {0x7fff,      0,      0}
    , {0x7fff,      0, 0x7fff}
    , {0x7fff, 0x7fff,      0}
    , {0x7fff, 0x7fff, 0x7fff}
#else
      {  0,   0,   0}
    , {  0,   0, 127}
    , {  0, 127,   0}
    , {127,   0,   0}
    , {127,   0, 127}
    , {127, 127,   0}
    , {127, 127, 127}
#endif
};

    CRemPoly FAR* rgprempoly[DIM(rgrgbColors)];

    numpoly = DIM(rgprempoly);

    // init
    for(i = 0; i < numpoly; ++i)
      rgprempoly[i] = (CRemPoly FAR*)NULL;

    for(i = 0; i < numpoly; ++i){
      hr = CRemPoly::Create(clsid, &rgprempoly[i]);
      if(hr != NOERROR)
```

```
        goto LError0;
      IfMac(DbPrintf("CRemPoly::Create()"));


      for(j = 0; j < DIM(rgptPoly); ++j){
        short x = rgptPoly[j].x;
     short y = rgptPoly[j].y;
        IfMac(DbPrintf("CRemPoly::AddPoint(%d,%d)", x, y));
        hr = rgprempoly[i]->AddPoint(x, y);
     ASSERT(hr == NOERROR);
      }

      for(j = 0; j < DIM(rgrgbColors); ++j){
        hr = rgprempoly[i]->SetWidth(i + j);
     ASSERT(hr == NOERROR);
     IfMac(DbPrintf("CRemPoly::SetWidth()"));

        hr = rgprempoly[i]->set_red(rgrgbColors[j].red);
     ASSERT(hr == NOERROR);
     IfMac(DbPrintf("CRemPoly::set_red()"));

        hr = rgprempoly[i]->set_green(rgrgbColors[j].green);
     ASSERT(hr == NOERROR);
     IfMac(DbPrintf("CRemPoly::set_green()"));

        hr = rgprempoly[i]->set_blue(rgrgbColors[j].blue);
     ASSERT(hr == NOERROR);
     IfMac(DbPrintf("CRemPoly::set_blue()"));

        hr = rgprempoly[i]->SetXOrigin((2*i) + j << 4);
     ASSERT(hr == NOERROR);
     IfMac(DbPrintf("CRemPoly::SetXOrigin()"));

        hr = rgprempoly[i]->SetYOrigin(j << 4);
     ASSERT(hr == NOERROR);
     IfMac(DbPrintf("CRemPoly::SetYOrigin()"));

        hr = rgprempoly[i]->Draw();
     ASSERT(hr == NOERROR);
     IfMac(DbPrintf("CRemPoly::Draw()"));
      }
    }

    hr = NOERROR;

LError0:;
    for(i = 0; i < numpoly; ++i){
      if(rgprempoly[i] != (CRemPoly FAR*)NULL){
     rgprempoly[i]->Release();
      }
    }

    return hr;
}
```

DISPDEMO.DEF (DISPDEMO Sample)

```
NAME          DispDemo

DESCRIPTION   'IDispatch Demo Application'

EXETYPE       WINDOWS

STUB          'WINSTUB.EXE'

CODE          PRELOAD MOVEABLE DISCARDABLE
DATA          PRELOAD MULTIPLE

HEAPSIZE      1024
STACKSIZE     5120
```

DISPDEMO.H   (DISPDEMO Sample)

```
/***
*dispdemo.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  IDispatch Demo App definitions.
*
*Implementation Notes:
*
****************************************************************************
*/

#include "hostenv.h"
#include "resource.h"
#include "clsid.h"

#ifdef _MAC
# define UNUSED(X) ((void)(void*)&(X))
#else
# define UNUSED(X) (X)
#endif

#ifndef NEAR
# define NEAR
#endif

#define DIM(X) (sizeof(X) / sizeof(X[0]))

#define ASSERT(X) Assert(X, __FILE__, __LINE__)


#ifdef __cplusplus
extern "C" {
#endif

STDAPI InitOle(void);
STDAPI UninitOle(void);

STDAPI DoPoly(CLSID);

void   Assert(int, char FAR*, int);

#ifdef _MAC
void DbPrintf(char*, ...);
#endif

#ifdef __cplusplus
}
#endif
```

DISPDEMO.RC   (DISPDEMO Sample)

```
#define NOKERNEL
#define NOGDI
#define NOSOUND
#define NOCOMM
#define NODRIVERS
#include "windows.h"
#include "dispdemo.h"

DISPDEMO ICON dispdemo.ico

DispDemoMenu MENU
BEGIN
    MENUITEM     "&CPoly",               IDM_POLY

    MENUITEM     "CPoly&2",              IDM_POLY2

    POPUP        "&Options"
    BEGIN
        MENUITEM "&Trace",               IDM_TRACE
    END

    POPUP        "&Help"
    BEGIN
        MENUITEM "&About IDispatch Demo...",   IDM_ABOUT
    END
END

AboutBox DIALOG 22, 17, 144, 75
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "About IDispatch Demo App"
BEGIN
    CTEXT "Microsoft Windows"        -1,  0,  5, 144,  8
    CTEXT "IDispatch Demo Application"      -1,  0, 14, 144,  8
    CTEXT "Version 2.0"              -1,  0, 34, 144,  8
    DEFPUSHBUTTON "OK"             IDOK, 53, 59,  32, 14, WS_GROUP
END
```

HOSTENV.H   (DISPDEMO Sample)

```
/***
*hostenv.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*  Generic host specific includes.
*
*Implementation Notes:
*
****************************************************************************
*/

#if defined(_MAC)

#if defined(_PPCMAC)
#pragma data_seg("_FAR_DATA")
#pragma data_seg( )
#define MAXLONG 0x7fffffff
#define EventHandlerProcPtr AEEventHandlerUPP
#else //_PPCMAC
#define  GetMenuItemText(mApple,menuItem,daName)
GetItem(mApple,menuItem,daName)
#endif //_PPCMAC

#endif //_MAC

#if defined(_MAC)

#if defined(_MSC_VER)

# include <values.h>
# include <types.h>
# include <string.h>
# include <quickdra.h>
# include <fonts.h>
# include <events.h>
# include <resource.h>
# include <menus.h>
# include <textedit.h>
# include <dialogs.h>
# include <desk.h>
# include <toolutil.h>
# include <memory.h>
# include <files.h>
# include <osutils.h>
# include <osevents.h>
# include <diskinit.h>
# include <packages.h>
# include <traps.h>
# include <AppleEve.h>
```

```
#else //_MSC_VER

# include <values.h>
# include <types.h>
# include <strings.h>
# include <quickdraw.h>
# include <fonts.h>
# include <events.h>
# include <resources.h>
# include <windows.h>
# include <menus.h>
# include <textedit.h>
# include <dialogs.h>
# include <desk.h>
# include <toolutils.h>
# include <memory.h>
# include <files.h>
# include <osutils.h>
# include <osevents.h>
# include <diskinit.h>
# include <packages.h>
# include <traps.h>
# include <AppleEvents.h>

#endif //_MSC_VER

# include <ole2.h>
# include <olenls.h>
# include <dispatch.h>

# define TCHAR               char
# define TSTR(str)           str
# define STRING(str)           (str)
# define WIDESTRING(str)        (str)

#elif defined(WIN32)

# include <windows.h>
# include <ole2.h>
# include <oleauto.h>

# if defined(UNICODE)
    #define TCHAR      WCHAR
    #define TSTR(str)        L##str
    #define STRING(str)        (str)
    #define WIDESTRING(str)  (str)
# else
    #define TCHAR      char
    #define TSTR(str)        str
    #define STRING(str)        AnsiString(str)
    #define WIDESTRING(str)  WideString(str)
    extern "C" char FAR* AnsiString(OLECHAR FAR* strIn);
    extern "C" OLECHAR FAR* WideString(char FAR* strIn);
# endif
```

```
#else /* WIN16 */

# include <windows.h>
# include <ole2.h>
# include <olenls.h>
# include <dispatch.h>

# define TCHAR               char
# define TSTR(str)           str
# define STRING(str)           (str)
# define WIDESTRING(str)        (str)
#endif
```

MISC.CPP   (DISPDEMO Sample)

```
/***
*misc.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*
*Implementation Notes:
*
*****************************************************************************
*/

// Use ANSI strings for assertions
#ifdef UNICODE
#  undef UNICODE
#endif

#include "dispdemo.h"

#include <stdio.h>
#include <stdarg.h>


int g_fTrace = 0;

STDAPI
InitOle(void)
{
    HRESULT hresult;

    if((hresult = OleInitialize(NULL)) != NOERROR)
      return hresult;

// UNDONE: temporary to try to get symbols for the DLL
    VARIANT var;
    VariantInit(&var);

    return NOERROR;
}

STDAPI
UninitOle()
{
    OleUninitialize();

    return NOERROR;
}

extern "C" void
Assert(int fCond, char FAR* file, int line)
{
    char buf[128];

    if(fCond)
      return;

    sprintf(buf, "Assertion failed: %s(%d)\n", file, line);

#ifdef _MAC
    DebugStr(c2pstr(buf));
#else
    OutputDebugString(buf);
    DebugBreak();
#endif
}

#ifdef _MAC

extern "C" {
extern WindowPtr g_pwndDebug;

void
DbPrintf(char *sz, ...)
{
    va_list args;
    WindowPtr pwndSaved;
static char rgchOut[256];

    if(g_pwndDebug == nil)
      return;

    GetPort(&pwndSaved);
    SetPort(g_pwndDebug);

    va_start(args, sz);
    vsprintf(rgchOut, sz, args);
    rgchOut[79] = '\0';

    EraseRect(&g_pwndDebug->portRect);
    TextFont(systemFont);
    MoveTo(10,20);
    DrawString(c2pstr(rgchOut));

    SetPort(pwndSaved);
}

}

#endif
```

RESOURCE.H (DISPDEMO Sample)

```
/***
*resource.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*
*Implementation Notes:
*
****************************************************************************
*/

#ifdef _MAC

#define kMinSize 500   /* minimum size (in K) */
#define kPrefSize500   /* preferred size (in K) */

#define    rMenuBar    128   /* menu bar */

#define    rAboutAlert 128   /* about alert */

#define    rUserAlert  129   /* error alert */

#define    rWindow           128   /* application's window */
#define    rDebugWindow      129   /* debug window */

#define    mApple            128   /* Apple menu */
#define    iAbout            1

#define    mFile       129   /* File menu */
#define    iNew        1
#define    iClose            4
#define    iQuit       12

#define    mEdit       130   /* Edit menu */
#define    iUndo       1
#define    iCut        3
#define    iCopy       4
#define    iPaste            5
#define    iClear            6

#define mSpoly         131
#define iSpoly         1

#define mSpoly2        132
#define iSpoly2        1

#define kMinHeap 21 * 1024
#define kMinSpace8 * 1024

#else /* WIN16 || WIN32 */
```

```
# define IDM_ABOUT      1
# define IDM_POLY2
# define IDM_POLY2      3
# define IDM_TRACE      4
# define IDM_MAX 5

#endif
```

WINMAIN.CPP (DISPDEMO Sample)

```
/***
*dispdemo.c - IDespatch demo/sample client application.
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This module is the main entry point for the sample IDispatch client,
*  dispdemo.exe.
*
*  This program is intended to demonstrate a client invoking methods
*  and referencing properties on a remote object via the IDispatch
*  interface.
*
*  The bulk of the sample can be found in the file crempoly.cpp, which
*  implements CRemPoly, the remote polygon class.
*
*Implementation Notes:
*
*****************************************************************************
*/

#include "dispdemo.h"

extern BOOL g_fTrace;

BOOL InitApplication(HANDLE);
BOOL InitInstance(HANDLE, int);
BOOL FAR PASCAL About(HWND, unsigned, WORD, LONG);

extern "C" {
int PASCAL WinMain(HANDLE, HANDLE, LPSTR, int);
LRESULT FAR PASCAL MainWndProc(HWND, UINT, WPARAM, LPARAM);
}

HANDLE g_hInst;

TCHAR g_szDispDemoWClass[] = TSTR("DispDemoWClass");

extern "C" int PASCAL
WinMain(
    HANDLE hinst,
    HANDLE hPrevInstance,
    LPSTR lpCmdLine,
    int nCmdShow)
{
    MSG msg;


    (lpCmdLine); // UNUSED

    if(!hPrevInstance)
```

```
      if(!InitApplication(hinst))
        return FALSE;

    if(InitOle() != NOERROR)
      return FALSE;

    if(!InitInstance(hinst, nCmdShow))
      return FALSE;

    while(GetMessage(&msg, NULL, 0, 0)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }


    return msg.wParam;
}


BOOL
InitApplication(HANDLE hinst)
{
    WNDCLASS  wc;

    wc.style            = NULL;
    wc.lpfnWndProc      = MainWndProc;
    wc.cbClsExtra       = 0;
    wc.cbWndExtra       = 0;
    wc.hInstance        = hinst;
    wc.hIcon            = LoadIcon(hinst, TSTR("DISPDEMO"));
    wc.hCursor          = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground    = GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName     = TSTR("DispDemoMenu");
    wc.lpszClassName    = g_szDispDemoWClass;
    if(!RegisterClass(&wc))
      return FALSE;

    return TRUE;
}

#ifdef WIN32
#define szAppTitle TSTR("IDispatch Demo App (32-bit)")
#else //WIN32
#define szAppTitle TSTR("IDispatch Demo App")
#endif //WIN32

BOOL
InitInstance(HANDLE hinst, int nCmdShow)
{
    HWND hWnd;

    g_hInst = hinst;

    hWnd = CreateWindow(
      g_szDispDemoWClass,
      szAppTitle,
      WS_OVERLAPPED|WS_CAPTION|WS_SYSMENU|WS_MINIMIZEBOX,
      CW_USEDEFAULT, CW_USEDEFAULT, 300, 100,
      NULL, NULL, hinst, NULL);

    if(!hWnd)
      return FALSE;

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    return TRUE;
}


BOOL FAR PASCAL
About(HWND hDlg, unsigned message, WORD wParam, LONG lParam)
{
    switch(message){
    case WM_INITDIALOG:
      return TRUE;

    case WM_COMMAND:
      switch(wParam){
      case IDOK:
      case IDCANCEL:
        EndDialog(hDlg, TRUE);
        return TRUE;
      }
      break;
    }
    return FALSE;
}



extern "C" LRESULT FAR PASCAL
MainWndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    HMENU hmenu;
static FARPROC pfnAbout;

    switch(message){
    case WM_COMMAND:
      switch(wParam){
      case IDM_TRACE:
        /* enable/disable trace */
        g_fTrace = (g_fTrace) ? FALSE : TRUE;
        hmenu = GetMenu(hwnd);
        CheckMenuItem(hmenu, IDM_TRACE, g_fTrace ? MF_CHECKED :
MF_UNCHECKED);
        return 0;

      case IDM_POLY:
        DoPoly(CLSID_CPoly);
```

```
          return 0;

        case IDM_POLY2:
          DoPoly(CLSID_CPoly2);
          return 0;

        case IDM_ABOUT:
          pfnAbout = (FARPROC)MakeProcInstance((FARPROC)About, g_hInst);
          DialogBox(g_hInst, TSTR("AboutBox"), hwnd, pfnAbout);
          FreeProcInstance(pfnAbout);
          return 0;
        }
        break;

      case WM_CLOSE:
        DestroyWindow(hwnd);
        return 0;

      case WM_DESTROY:
        UninitOle();
        PostQuitMessage(0);
        return 0;
      }
      return DefWindowProc(hwnd, message, wParam, lParam);
  }
```

DSPCALC2

-------------------------------------- OLE Automation Sample Program: Dispcalc --------------------------------------

DspCalc2 is a simple accumulator-based calculator. Its user interface consists of buttons for the numbers (0-9), the operators (+, -, *, /), and some other necessary buttons (C for Clear, = for evaluation). Its programmability interface consists of one object, which is described below.

The ProgID for dspcalc2's only object is "DspCalc2.Application". An instance of this object can be created by executing the following lines of code in Visual Basic or DispTest:

```
Sub Foo
  Dim MyCalculator as Object

  Set MyCalculator = CreateObject("DspCalc2.Application")
  . . .
End Sub
```

---------------- Program Structure ---------------- DspCalc2 uses a type library and CreateStdDispatch in order to implement the IDispatch interface.

------------------------ Properties for the object ------------------------

Name Type Description --------------------------- Value VT_I4 Same as the value for the accumulator.

Accum VT_I4 The value that is in the accumulator of the calculator.

Opnd VT_I4 The operand. This is the number which is currently being entered.

Op VT_I2 The operator that is currently being used. This is an enumeration:
const OP_NONE = 0 const OP_PLUS = 1 const
OP_MINUS = 2 const OP_MULT = 3

---------------------------- Methods defined on the object ----------------------------

Name Description -------------------------- Eval() as Boolean If there is an operator, apply it to accumulator and the operand, placing the result in the accumulator.

The return value indicates success or failure.

Clear() Resets the calculator. This sets Op to OP_NONE, and both Accum and Opnd to 0. Display() Updates the display of the calculator. (Other operations do not do this.)

Quit() Close the calculator.

Button(b as string) as Boolean Press the indicated button and return
success or failure. Valid string values are: +, -, *,
+ 0-9 c, C =
Note that you may also pass the numbers 0-9 and these will be
converted to strings automatically.

-------------------------- Shortcomings of this sample -------------------------- 1. Property and method names should not be abbreviated. For example, the "Opnd" property should be the "Operand" property.

2. Since the object is the application object, it should have Name and Version properties, which are read-only.

MAKEFILE   (DSPCALC2 Sample)

```
####
#makefile - makefile for dspcalc2.exe
#
#       Copyright (C) 1992, Microsoft Corporation
#
#Purpose:
#  Builds the OLE 2.0 sample IDispatch server, dspcalc2.exe.
#
#
#  Usage: NMAKE                 ; build with defaults
#     or: NMAKE option          ; build with the given option(s)
#     or: NMAKE clean           ; erase all compiled files
#
#     option: dev = [win16 | win32]    ; dev=win32 is the default
#             DEBUG=[0|1]              ; DEBUG=1 is the default
#             HOST=[DOS | NT]          ; HOST=DOS (for win16)
#                                      ; HOST=NT  (for win32)
#
#Notes:
#  This makefile assumes that the PATH, INCLUDE and LIB environment
#  variables are setup properly.
#
############################################################################
##


##########################################################################
#
# Default Settings
#

!if "$(dev)" == ""
dev = win32
!endif

!if !("$(dev)" == "win16" || "$(dev)" == "win32")
!error Invalid dev option, choose from [win16 | win32]
!endif

!if "$(dev)" == "win16"
TARGET  = WIN16
!if "$(HOST)" == ""
HOST  = DOS
!endif
!endif

!if "$(dev)" == "win32"
TARGET  = WIN32
HOST  = NT
!endif

!ifdef NODEBUG
```

```
DEBUG = 0
!endif

!if "$(DEBUG)" == "0"
NODEBUG = 1
!endif

!if "$(DEBUG)" == ""
DEBUG = 1
!endif

###########################################################################
#
# WIN16 Settings
#
!if "$(TARGET)" == "WIN16"

CC   = cl
LINK = link

!if "$(HOST)" == "DOS"
WX   = wx /w
!else
WX   =
!endif

RCFLAGS = -dWIN16
CFLAGS = -c -W3 -AM -GA -GEs -DWIN16
LINKFLAGS = /NOD /NOI /BATCH /ONERROR:NOEXE

LIBS = libw.lib mlibcew.lib

!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS) -Od -Zi -D_DEBUG $(CL)
LINKFLAGS = $(LINKFLAGS) /COD
!else
CFLAGS = $(CFLAGS) -Ox $(CL)
LINKFLAGS = $(LINKFLAGS) /FAR /PACKC
!endif
!endif


###########################################################################
#
# WIN32 Settings
#
!if "$(TARGET)" == "WIN32"

!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

CC = $(cc)
CFLAGS = $(cflags) $(cvarsmt) -DUNICODE $(cdebug)

!ifndef NODEBUG
CFLAGS = $(CFLAGS) -D_DEBUG
```

```
!endif

LINK = $(link)
LINKFLAGS = $(linkdebug) $(guilflags)
RCFLAGS = -DWIN32

WX =

!endif


###########################################################################
#
# Build rules
#

.cpp.obj:
    @echo Compiling $<...
    $(CC) $<

.c.obj:
    @echo Compiling $<...
    $(CC) $<


###########################################################################
#
# Application Settings
#

APPS = dspcalc2


!if "$(TARGET)" == "WIN16"
LIBS = ole2.lib compobj.lib ole2disp.lib typelib.lib $(LIBS)
!endif
!if "$(TARGET)" == "WIN32"
LIBS = $(olelibsmt)
!endif

OBJS = \
        main.obj        \
        dspcalc2.obj    \
        clsid.obj


###########################################################################
#
# Default Goal
#

goal : setflags $(APPS).exe

setflags :
        set CL=$(CFLAGS)


##########################################################################
#
# Clean (erase) generated files
#
clean :
    if exist *.obj       del *.obj
    if exist $(APPS).exe del $(APPS).exe
    if exist $(APPS).tlb del $(APPS).tlb
    if exist $(APPS).map del $(APPS).map
    if exist $(APPS).res del $(APPS).res
    if exist calctype.h  del calctype.h
    if exist *.log       del *.log
    if exist *.pdb       del *.pdb


##########################################################################
#
# Application Build (WIN16 Specific)
#

!if "$(TARGET)" == "WIN16"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
        link $(LINKFLAGS) @<<
$(OBJS),
$@,,
$(LIBS),
$(APPS).def
<<
        rc -k -t $(APPS).res $@
!endif


##########################################################################
#
# Application Build (WIN32 Specific)
#
!if "$(TARGET)" == "WIN32"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
      $(LINK) @<<
        $(LINKFLAGS)
        -out:$@
        -map:$*.map
        $(OBJS)
        $(APPS).res
        $(LIBS)
<<
!endif


##########################################################################
#
# Application Build (Common)
```

```
#

$(APPS).res : $(APPS).rc
        rc $(RCFLAGS) -r -fo$@ $?


##########################################################################
#
# Dependencies
#

calctype.h : calctype.odl
        if exist calctype.h  del calctype.h
        if exist dspcalc2.tlb  del dspcalc2.tlb
        $(WX) mktyplib /D$(TARGET) /h calctype.h /o calctype.log /tlb
dspcalc2.tlb calctype.odl

main.obj : main.cpp dspcalc2.h calctype.h
        $(CC) main.cpp

dspcalc2.obj : dspcalc2.cpp dspcalc2.h calctype.h
        $(CC) dspcalc2.cpp

clsid.obj : clsid.c clsid.h
        $(CC) clsid.c
```

CALCTYPE.ODL   (DSPCALC2 Sample)

```
/***
*calctype.odl
*
*  Copyright (C) 1993-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This file contains the definitions of the objects exposed for OLE
Automation
*  by the dspcalc2
*  example program. MkTypLib uses this file to produce a type library and
*  a header file. The header file is then used in compiling dspcalc2.
*
*Implementation Notes:
*
*****************************************************************************
*/

[
  uuid(00020470-0000-0000-C000-000000000046),
  helpstring("OLE Automation DspCalc2 1.0 Type Library"),
  lcid(0x0409),
  version(1.0)
]
library DspCalc2
{

#if defined(WIN32)
    importlib("stdole32.tlb");
#else
#if defined(MAC) && !defined(_PPCMAC)
    importlib("Standard OLE Types");
#else
    importlib("stdole.tlb");
#endif
#endif

    typedef enum operators {
      OP_NONE = 0,
      OP_PLUS,
      OP_MINUS,
      OP_MULT,
      OP_DIV
    } OPERATORS;


    [
      uuid(00020441-0000-0000-C000-000000000046),
      helpstring("DspCalc2"),
      odl
    ]
    interface _ICalculator : IUnknown
    {
      [propput]
      void Accum([in] long l);
      [propget, helpstring("The value stored in the calculator")]
      long Accum();

      [propput]
      void Opnd([in] long l);
      [propget]
      long Opnd();

      [propput]
      void Op([in] OPERATORS op);
      [propget]
      OPERATORS Op();

      boolean Eval(void);
      void Clear(void);
      void Display(void);
      void Quit(void);
      [vararg] boolean Button([in]SAFEARRAY(VARIANT) psa);
    }


    [
      uuid(00020442-0000-0000-C000-000000000046),
      helpstring("DspCalc2")
    ]
    dispinterface _DCalculator
    {
      interface _ICalculator;
    }


    [
      uuid(00020469-0000-0000-C000-000000000046),
      helpstring("DspCalc2"),
      appobject
    ]
    coclass Calculator
    {
      dispinterface _DCalculator;
      interface _ICalculator;
    }
}
```

CLSID.H   (DSPCALC2 Sample)

```
/***
*clsid.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This file defines the CLSIDs
*
*Implementation Notes:
*
****************************************************************************
*/

DEFINE_GUID(CLSID_CCalc2,    0x00020469, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0,
0x46);
DEFINE_GUID(LIBID_DspCalc2,  0x00020470, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0,
0x46);
DEFINE_GUID(IID_ICalculator, 0x00020441, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0,
0x46);
DEFINE_GUID(IID_DCalculator, 0x00020442, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0,
0x46);

#ifdef INITGUID
# ifdef _MAC
DEFINE_GUID(GUID_NULL, 0L, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
DEFINE_GUID(IID_IDispatch, 0x00020400, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);
DEFINE_GUID(IID_IEnumVARIANT, 0x00020404, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0,
0x46);
DEFINE_GUID(IID_IUnknown, 0x00000000, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);
DEFINE_GUID(IID_IClassFactory, 0x00000001, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0,
0x46);
# endif
#endif
```

CLSID.C   (DSPCALC2 Sample)

```
/***
*clsid.c
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This file allocates and initializes the CLSIDs.
*
****************************************************************************
*/

#ifdef _PPCMAC
#pragma data_seg ("_FAR_DATA")
#pragma data_seg ( )
#endif //_PPCMAC

#ifdef _MAC
# include <Types.h>
#ifdef _MSC_VER
# include <Processe.h>
# include <AppleEve.h>
#else //_MSC_VER
# include <Processes.h>
# include <AppleEvents.h>
#endif //_MSC_VER
#else
# include <windows.h>
#endif

#ifndef WIN32
#include <compobj.h>
#endif //!WIN32

// this redefines the DEFINE_GUID() macro to do allocation.
//
#include <initguid.h>

#ifndef INITGUID
# define INITGUID
#endif

// due to the previous header, including this causes the DEFINE_GUID
// definitions in the following header(s) to actually allocate data.
//
#include "clsid.h"
```

DSPCALC2.DEF (DSPCALC2 Sample)

```
NAME         DSPCALC2

DESCRIPTION  'IDispatch Calculator w/ Type Library'

EXETYPE      WINDOWS

STUB         'WINSTUB.EXE'

CODE         PRELOAD MOVEABLE DISCARDABLE
DATA         PRELOAD MULTIPLE

HEAPSIZE     4096
STACKSIZE    8192
```

DSPCALC2.H (DSPCALC2 Sample)

```
/***
*dspcalc2.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*
*
*Implementation Notes:
*
*****************************************************************************
*/

#include "hostenv.h"
#include "resource.h"
#include "clsid.h"

#ifndef CLASS
# ifdef __TURBOC__
#  define CLASS class huge
# else
#  define CLASS class FAR
# endif
#endif

#pragma warning(disable:4355)

#ifdef _MAC
typedef void * LPVOID;
typedef unsigned long ULONG;
typedef int BOOL;
typedef unsigned int UINT;
typedef unsigned short WORD;
typedef unsigned char BYTE;
typedef unsigned long DWORD;
typedef long LONG;
#define FALSE 0
#define TRUE  1
#define NEAR
#endif

#ifdef _MAC
# define UNUSED(X) ((void)(void*)&(X))
#else
# define UNUSED(X) (X)
#endif


/*
 * The .h file included below is created via MkTypLib
 */
```

```
#include "calctype.h"

#pragma warning(disable:4355)

#define DIM(X) (sizeof(X)/sizeof(X[0]))


// forward decl
CLASS CCalc;


// Introduced "calculator" interface
//
// This nested class implementes core arithmetic functionality
// (such as it is) *and* is the interface that will be exposed via
// IDispatch for external programmability.
//
CLASS CArith : public _ICalculator {
 public:

  /* IUnknown methods */
  STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR * ppvObj);
  STDMETHOD_(ULONG, AddRef)(THIS);
  STDMETHOD_(ULONG, Release)(THIS);

  /* ICalculator methods */
  STDMETHOD_(void, put_Accum)(long l);
  STDMETHOD_(long, get_Accum)(void);
  STDMETHOD_(void, put_Opnd)(long l);
  STDMETHOD_(long, get_Opnd)(void);
  STDMETHOD_(void, put_Op)(OPERATORS op);
  STDMETHOD_(OPERATORS, get_Op)(void);
  STDMETHOD_(VARIANT_BOOL, Eval)(void);
  STDMETHOD_(void, Clear)(void);
  STDMETHOD_(void, Display)(void);
  STDMETHOD_(void, Quit)(void);
  STDMETHOD_(VARIANT_BOOL, Button)(SAFEARRAY FAR* psa);

  // the following method is internal, and not exposed for programmability
  BOOL ButtonPush(int button);

  CArith(CCalc FAR* pcalc){
     m_pcalc = pcalc;
     Clear();
  }

  enum states { STATE_LOPND, STATE_OP, STATE_ROPND, STATE_EVAL };

 private:
  CCalc FAR* m_pcalc;
  OPERATORSm_op;
  long     m_opnd;
  long     m_accum;
  enum states m_state;
};
```

```
CLASS CCalc : public IUnknown {
public:
    friend CArith;

    static CCalc FAR* Create();

    STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppv);
    STDMETHOD_(ULONG, AddRef)(void);
    STDMETHOD_(ULONG, Release)(void);

    CCalc() : m_arith(this)
    {
      m_refs = 0;
#ifdef _MAC
      m_pdlg = nil;
#else
      m_hwnd = 0;
#endif
      m_punkStdDisp = NULL;
    }

#ifdef _MAC
    DialogPtr m_pdlg;
#else
    HWND m_hwnd;
#endif
    CArith m_arith;

private:
    ULONG m_refs;
    IUnknown FAR* m_punkStdDisp;
};



// the CCalc Class Factory
//
CLASS CCalcCF : public IClassFactory {
public:
    static IClassFactory FAR* Create();

    STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppv);
    STDMETHOD_(ULONG, AddRef)(void);
    STDMETHOD_(ULONG, Release)(void);

    STDMETHOD(CreateInstance)(
      IUnknown FAR* punkOuter, REFIID riid, void FAR* FAR* ppv);
#ifdef _MAC
    STDMETHOD(LockServer)(unsigned long fLock);
#else
    STDMETHOD(LockServer)(BOOL fLock);
#endif
```

```
    CCalcCF() { m_refs = 1; }

private:
    ULONG m_refs;
};

extern HRESULT InitOle(void);
extern HRESULT UninitOle(void);

extern CCalc FAR* g_pcalc;
```

DSPCALC2.RC (DSPCALC2 Sample)

```
#include <windows.h>
#include "resource.h"

DspCalc2 ICON dspcalc2.ico

DspCalc2 DIALOG DISCARDABLE  0, 0, 92, 114
STYLE    WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX
CLASS    "DspCalc2"
#ifdef WIN32
CAPTION  "DspCalc2 (32-bit)"
#else //WIN32
CAPTION  "DspCalc2"
#endif //WIN32
BEGIN
    PUSHBUTTON "0", IDC_ZERO,           9, 90, 14, 15
    PUSHBUTTON "1", IDC_ONE,            9, 70, 15, 16
    PUSHBUTTON "2", IDC_TWO,           29, 70, 16, 15
    PUSHBUTTON "3", IDC_THREE,         49, 70, 15, 15
    PUSHBUTTON "4", IDC_FOUR,           9, 50, 15, 14
    PUSHBUTTON "5", IDC_FIVE,          29, 50, 15, 15
    PUSHBUTTON "6", IDC_SIX,           49, 50, 15, 15
    PUSHBUTTON "7", IDC_SEVEN,          9, 30, 15, 15
    PUSHBUTTON "8", IDC_EIGHT,         29, 30, 15, 15
    PUSHBUTTON "9", IDC_NINE,          49, 30, 15, 15
    PUSHBUTTON "=", IDC_EQUALS,        49, 90, 14, 15
    PUSHBUTTON "+", IDC_PLUS,          69, 30, 15, 15
    PUSHBUTTON "-", IDC_MINUS,         69, 50, 14, 15
    PUSHBUTTON "*", IDC_MULT,          69, 70, 15, 15
    PUSHBUTTON "/", IDC_DIV,           69, 90, 15, 15
    PUSHBUTTON "C", IDC_CLEAR,         29, 90, 16, 15
    EDITTEXT        IDC_DISPLAY,        9,  6, 76, 15, ES_MULTILINE |
ES_RIGHT | ES_AUTOHSCROLL | ES_READONLY
END
```

DSPCALC2.REG (DSPCALC2 Sample)

```
REGEDIT


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; registration info DspCalc2.Application (defaults to DspCalc2.Application.1

HKEY_CLASSES_ROOT\Dspcalc2.Application = OLE Automation Dspcalc2 Application
HKEY_CLASSES_ROOT\Dspcalc2.Application\Clsid = {00020469-0000-0000-C000-
000000000046}


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; registration info DspCalc2.Application.1

HKEY_CLASSES_ROOT\Dspcalc2.Application.1 = OLE Automation Dspcalc2 1.0
Application
HKEY_CLASSES_ROOT\Dspcalc2.Application.1\Clsid = {00020469-0000-0000-C000-
000000000046}


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; registration info DspCalc2 1.0

HKEY_CLASSES_ROOT\CLSID\{00020469-0000-0000-C000-000000000046} = OLE
Automation Dspcalc2 1.0 Application
HKEY_CLASSES_ROOT\CLSID\{00020469-0000-0000-C000-000000000046}\ProgID =
Dspcalc2.Application.1
HKEY_CLASSES_ROOT\CLSID\{00020469-0000-0000-C000-
000000000046}\VersionIndependentProgID = Dspcalc2.Application
HKEY_CLASSES_ROOT\CLSID\{00020469-0000-0000-C000-000000000046}\LocalServer32
= dspcalc2.exe /Automation
```

DSPCALC2.CPP (DSPCALC2 Sample)

```
/***
*dspcalc2.cpp
*
*  Copyright (C) 1993-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This module implements the basic user interface and arithmetic
*  functionality of the IDispatch calculator.
*
*  The implementation of IDispatch is via aggregation with an
*  instance of the "standard" IDispatch implementation, which is
*  initialized with a TypeInfo loaded from the TypeLib that was
*  constructed from the ODL description of the calculator.
*
*Implementation Notes:
*
****************************************************************************
*/

#include "dspcalc2.h"

CCalc FAR* g_pcalc = NULL;

unsigned long g_dwCCalcCF = 0;
unsigned long g_dwRegisterCCalc = 0;

#ifdef _MAC
extern Boolean g_fQuit;
#endif //_MAC


/***
*CCalc *CCalc::Create(void)
*Purpose:
*  Create an instance of the IDispatch calculator, load the
*  TypeInfo that describes the exposed functionality and
*  aggregate with an instance of CStdDispatch that has been
*  initialized with this TypeInfo.
*
*Entry:
*  None
*
*Exit:
*  return value = CCalc*, NULL if the creation failed.
*
***********************************************************************/
CCalc FAR*
CCalc::Create()
{
    HRESULT hresult;
    CCalc FAR* pcalc;
    ITypeLib FAR* ptlib;
    ITypeInfo FAR* ptinfo;
    IUnknown FAR* punkStdDisp;

    ptlib = NULL;
    ptinfo = NULL;

    if((pcalc = new FAR CCalc()) == NULL)
      return NULL;
    pcalc->AddRef();

    // first try to load the type library from the information in the
registry
    if((hresult = LoadRegTypeLib(LIBID_DspCalc2, 1, 0, 0x0409, &ptlib)) !=
NOERROR){

      #define TLB_NAME OLESTR("dspcalc2.tlb")

      // if it wasn't registered, try to load it from the path/current
directory
      // if this succeeds, it will have registered the type library for us
      // for the next time.
      if((hresult = LoadTypeLib(TLB_NAME, &ptlib)) != NOERROR){
#ifndef _MAC
        MessageBox(NULL, TSTR("error loading TypeLib"),
              TSTR("dspcalc2"), MB_OK);
#endif
        goto LError0;
      }

    }

    if((hresult = ptlib->GetTypeInfoOfGuid(IID_ICalculator, &ptinfo)) !=
NOERROR){
#ifndef _MAC
      MessageBox(NULL, TSTR("error accessing TypeInfo"),
              TSTR("dspcalc2"), MB_OK);
#endif
      goto LError0;
    }

    // Create and aggregate with an instance of the default
    // implementation of IDispatch that is initialized with our
    // TypeInfo.
    //
    hresult = CreateStdDispatch(
      pcalc,                            // controlling unknown
      &(pcalc->m_arith),                // vtable* to dispatch on
      ptinfo,
      &punkStdDisp);

    if(hresult != NOERROR)
      goto LError0;

    pcalc->m_punkStdDisp = punkStdDisp;
    ptinfo->Release();
    ptlib->Release();

    return pcalc;

LError0:;
    pcalc->Release();
    if(ptinfo != NULL)
      ptinfo->Release();
    if(ptlib != NULL)
      ptlib->Release();
    return NULL;
}


//---------------------------------------------------------------------
//                        IUnknown methods
//---------------------------------------------------------------------


STDMETHODIMP
CCalc::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{

    if(IsEqualIID(riid, IID_IUnknown)){
      *ppv = this;
    }else
    if(IsEqualIID(riid, IID_IDispatch) ||
       IsEqualIID(riid, IID_DCalculator)){
      return m_punkStdDisp->QueryInterface(IID_IDispatch, ppv);
    }else
    if(IsEqualIID(riid, IID_ICalculator)){
      *ppv = &m_arith;
    }else {
      *ppv = NULL;
      return ResultFromScode(E_NOINTERFACE);
    }

    AddRef();
    return NOERROR;
}


STDMETHODIMP_(ULONG)
CCalc::AddRef()
{
    return ++m_refs;
}


STDMETHODIMP_(ULONG)
CCalc::Release()
{
    if(--m_refs == 0){
      if(m_punkStdDisp != NULL)
     m_punkStdDisp->Release();
#ifndef _MAC
      PostQuitMessage(0);
#endif
      delete this;
      return 0;
    }
    return m_refs;
}


STDMETHODIMP
CArith::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    return m_pcalc->QueryInterface(riid, ppv);
}


STDMETHODIMP_(ULONG)
CArith::AddRef()
{
    return m_pcalc->AddRef();
}


STDMETHODIMP_(ULONG)
CArith::Release()
{
    return m_pcalc->Release();
}


//----------------------------------------------------------------------
//                       Arithmetic features
//----------------------------------------------------------------------


STDMETHODIMP_(void)
CArith::Clear()
{
    m_opnd = 0;
    m_accum = 0;
    m_op = OP_NONE;
    m_state = STATE_LOPND;
}

STDMETHODIMP_(void)
CArith::put_Accum(long l)
{
    m_accum = l;
}


STDMETHODIMP_(long)
```

```
CArith::get_Accum()
{
    return m_accum;
}


STDMETHODIMP_(void)
CArith::put_Opnd(long l)
{
    m_opnd = l;
}


STDMETHODIMP_(long)
CArith::get_Opnd()
{
    return m_opnd;
}


STDMETHODIMP_(void)
CArith::put_Op(OPERATORS op)
{
    m_op = op;
}


STDMETHODIMP_(OPERATORS)
CArith::get_Op()
{
    return m_op;
}


STDMETHODIMP_(VARIANT_BOOL)
CArith::Eval()
{
    if(m_op == OP_NONE)
      return FALSE;

    switch(m_op){
    case OP_PLUS:
      m_accum += m_opnd;
      break;
    case OP_MINUS:
      m_accum -= m_opnd;
      break;
    case OP_MULT:
      m_accum *= m_opnd;
      break;
    case OP_DIV:
      m_accum = (m_opnd == 0) ? 0 : (m_accum / m_opnd);
      break;
    default:
      // ASSERT(UNREACHED);
      return FALSE;

    }

    m_state = STATE_EVAL;

    return TRUE;
}


//---------------------------------------------------------------------
//                       User Interface features
//---------------------------------------------------------------------


/***
*void CArith::Display()
*Purpose:
*  Display the contents of the register currently being edited.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
STDMETHODIMP_(void)
CArith::Display()
{
    VARIANT var;

    VariantInit(&var);
    V_VT(&var) = VT_I4;
    V_I4(&var) = (m_state == STATE_ROPND) ? m_opnd : m_accum;
    VariantChangeType(&var, &var, 0, VT_BSTR);

    #ifdef _MAC
    {
      Rect rcItem;
      Handle hItem;
      char str[255];
      short sItemKind;

      strcpy(str, V_BSTR(&var));
      GetDItem(m_pcalc->m_pdlg, IDC_DISPLAY, &sItemKind, &hItem, &rcItem);
      SetIText(hItem, c2pstr(str));
    }
#else
    SetDlgItemText(m_pcalc->m_hwnd, IDC_DISPLAY, STRING(V_BSTR(&var)));
#endif

    VariantClear(&var);
}
```

```
STDMETHODIMP_(VARIANT_BOOL)
CArith::Button(SAFEARRAY FAR * psa)
{
    int i, button;

static struct {
    OLECHAR ch;
    int idc;
} NEAR rgIdcOfCh[] = {
      { OLESTR('+'), IDC_PLUS   }
    , { OLESTR('-'), IDC_MINUS  }
    , { OLESTR('*'), IDC_MULT   }
    , { OLESTR('/'), IDC_DIV    }
    , { OLESTR('C'), IDC_CLEAR  }
    , { OLESTR('c'), IDC_CLEAR  }
    , { OLESTR('='), IDC_EQUALS }
    , { OLESTR('0'), IDC_ZERO   }
    , { OLESTR('1'), IDC_ONE    }
    , { OLESTR('2'), IDC_TWO    }
    , { OLESTR('3'), IDC_THREE  }
    , { OLESTR('4'), IDC_FOUR   }
    , { OLESTR('5'), IDC_FIVE   }
    , { OLESTR('6'), IDC_SIX    }
    , { OLESTR('7'), IDC_SEVEN  }
    , { OLESTR('8'), IDC_EIGHT  }
    , { OLESTR('9'), IDC_NINE   }
    , {  (OLECHAR)-1 , -1         }
};
    LONG saIndex, saUbound;
    VARIANT varButton;

    // Since this is a vararg function, we should be given a 1-dimensional
    // array with 0 for the lower bound. The array could be uninitialized
    // if 0 args were passed to us -- this call will give an error in this
case.
    if (SafeArrayGetUBound(psa, 1, &saUbound) != NOERROR)
      return FALSE;    // most likely 0 args were passed to us

    for (saIndex = 0; saIndex <= saUbound; saIndex++) {

      // get next parameter
      if (SafeArrayGetElement(psa, &saIndex, &varButton) != NOERROR)
     return FALSE;

      // convert it to a string in-place
      if (VariantChangeType(&varButton, &varButton, 0, VT_BSTR) != NOERROR)
     goto Error;

      // if the string is more that 1 character long, then we know its
wrong.
      if(SysStringLen(varButton.bstrVal) > 1)
        goto Error;
      // translate button string into control ID
      for(i = 0;; ++i){
        if(rgIdcOfCh[i].ch == -1)
       goto Error;
        if(rgIdcOfCh[i].ch == varButton.bstrVal[0]){
       button = rgIdcOfCh[i].idc;
       break;
        }
      }

      VariantClear(&varButton);          // done with the parameter

      if (!ButtonPush(button))
     return FALSE;

    } // for

    return TRUE; // success

Error:
    VariantClear(&varButton);
    return FALSE;// failure
}


// the following method is internal, and not exposed for programmability
BOOL
CArith::ButtonPush(int button)
{
    if(button >= IDC_ZERO && button <= IDC_NINE){

      long lVal = button - IDC_ZERO;


      switch(m_state){
      case STATE_EVAL:
     m_accum = lVal;
     m_state = STATE_LOPND;
     break;
      case STATE_OP:
     m_opnd = lVal;
     m_state = STATE_ROPND;
     break;
      case STATE_LOPND:
     m_accum = (m_accum * 10) + lVal;
     break;
      case STATE_ROPND:
     m_opnd  = (m_opnd * 10) + lVal;
     break;
      }

    }else if(button >= IDC_PLUS && button <= IDC_DIV){

      if(m_state == STATE_LOPND){
     m_opnd  = m_accum;
     m_state = STATE_OP;
     m_op    = (OPERATORS)(button - IDC_PLUS + OP_PLUS);
      }

    }else if(button == IDC_EQUALS){

      if(m_state > STATE_LOPND)
        Eval();

    }else if (button == IDC_CLEAR){

      Clear();

    } else {

      return 0; // unknown button

    }


    // Flash the button

#ifdef _MAC
    {
      Rect rcItem;
      long lDummy;
      Handle hItem;
      short sItemKind;

      GetDItem(m_pcalc->m_pdlg, button, &sItemKind, &hItem, &rcItem);
      HiliteControl((ControlHandle)hItem, 1);
      Delay(6, &lDummy);
      HiliteControl((ControlHandle)hItem, 0);
    }
#else
    SendMessage(m_pcalc->m_hwnd, BM_SETSTATE, 1, 0L);
    SendMessage(m_pcalc->m_hwnd, BM_SETSTATE, 0, 0L);
#endif

    // Update the calculator display

    Display();

    return TRUE;
}

/***
*void CArith::Quit()
*Purpose:
*
*Entry:
*  None
*
*Exit:
*  None
```

```
*
***********************************************************************/
STDMETHODIMP_(void)
CArith::Quit()
{
#ifndef _MAC
    PostQuitMessage(0);
#else
    g_fQuit = TRUE;
#endif
}


//----------------------------------------------------------------------
//                      The CCalc Class Factory
//----------------------------------------------------------------------


IClassFactory FAR*
CCalcCF::Create()
{
    return new FAR CCalcCF();
}


STDMETHODIMP
CCalcCF::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    if(IsEqualIID(riid, IID_IUnknown) || IsEqualIID(riid,
IID_IClassFactory)){
      AddRef();
      *ppv = this;
      return NOERROR;
    }
    *ppv = NULL;
    return ResultFromScode(E_NOINTERFACE);
}


STDMETHODIMP_(ULONG)
CCalcCF::AddRef()
{
    return ++m_refs;
}


STDMETHODIMP_(ULONG)
CCalcCF::Release()
{
    if(--m_refs == 0){
      delete this;
      return 0;
    }
    return m_refs;
}
```

```
STDMETHODIMP
CCalcCF::CreateInstance(
    IUnknown FAR* punkOuter,
    REFIID riid,
    void FAR* FAR* ppv)
{
extern CCalc FAR* g_pcalc;

    UNUSED(punkOuter);
    return g_pcalc->QueryInterface(riid, ppv);
}



STDMETHODIMP
#ifdef _MAC
CCalcCF::LockServer(unsigned long fLock)
#else
CCalcCF::LockServer(BOOL fLock)
#endif
{
    UNUSED(fLock);
    return NOERROR;
}

#ifdef _MAC
struct regentry{
    char *szKey;
    char *szValue;
} g_rgregentry[] = {

      { "CLSID\\{00020469-0000-0000-C000-000000000046}",
     "OLE Automation DspCalc2 1.0 Application" }

    , { "CLSID\\{00020469-0000-0000-C000-000000000046}\\LocalServer",
     "DCL2" }

    , { "CLSID\\{00020469-0000-0000-C000-000000000046}\\ProgID",
     "Dspcalc2.Application" }

    , { "CLSID\\{00020469-0000-0000-C000-000000000046}\\InprocHandler",
     "OLE2:Def$DefFSet" }

    , { "DCL2", "{00020469-0000-0000-C000-000000000046}" }

    , { "Dspcalc2.Application\\CLSID",
     "{00020469-0000-0000-C000-000000000046}" }

};

HRESULT
EnsureRegistration()
{
    HKEY hkey;

    if(RegOpenKey(HKEY_CLASSES_ROOT, "DCL2", &hkey) == NOERROR){
      RegCloseKey(hkey);
      return NOERROR;
    }

    for(int i = 0; i < DIM(g_rgregentry); ++i){
      if(RegSetValue(HKEY_CLASSES_ROOT, g_rgregentry[i].szKey, REG_SZ,
g_rgregentry[i].szValue, 0) != ERROR_SUCCESS)
        return ResultFromScode(E_FAIL);
    }

    return NOERROR;
}
#endif //_MAC


/***
*HRESULT InitOle(void)
*Purpose:
*  Initialize Ole, and register our class factories.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
HRESULT
InitOle()
{
    HRESULT hresult;
    IClassFactory FAR* pcf;


    if((hresult = OleInitialize(NULL)) != NOERROR)
      goto LError0;

#ifdef _MAC
    if((hresult = EnsureRegistration()) != NOERROR)
      goto LError0;
#endif

    // create the single global instance of CCalc
    if((g_pcalc = CCalc::Create()) == NULL){
      hresult = ResultFromScode(E_OUTOFMEMORY);
      goto LError0;
    }

    if((pcf = CCalcCF::Create()) == NULL)
      goto LError1;

    hresult = CoRegisterClassObject(
      CLSID_CCalc2,
      pcf,
      CLSCTX_LOCAL_SERVER,
      REGCLS_MULTIPLEUSE,
      &g_dwCCalcCF);
    if(hresult != NOERROR)
      goto LError2;

    hresult = RegisterActiveObject(
      g_pcalc, CLSID_CCalc2, NULL, &g_dwRegisterCCalc);
    if(hresult != NOERROR)
      goto LError2;

    pcf->Release();

    return NOERROR;

LError2:;
    pcf->Release();

LError1:;
    UninitOle();

LError0:;
    return hresult;
}


HRESULT
UninitOle()
{
    if(g_dwRegisterCCalc != 0)
      RevokeActiveObject(g_dwRegisterCCalc, NULL);

    if(g_dwCCalcCF != 0)
      CoRevokeClassObject(g_dwCCalcCF);

    // cause the remaining typeinfo to be released
    if(g_pcalc != NULL)
      g_pcalc->Release();

    OleUninitialize();

    return NOERROR;
}
```

HOSTENV.H   (DSPCALC2 Sample)

```
/***
*hostenv.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*  Generic host specific includes.
*
*Implementation Notes:
*
****************************************************************************
*/

#if defined(_MAC)

#if defined(_PPCMAC)
#pragma data_seg("_FAR_DATA")
#pragma data_seg( )
#define MAXLONG 0x7fffffff
#define EventHandlerProcPtr AEEventHandlerUPP
#else //_PPCMAC
#define  GetMenuItemText(mApple,menuItem,daName)
GetItem(mApple,menuItem,daName)
#endif //_PPCMAC

#endif //_MAC

#if defined(_MAC)

#if defined(_MSC_VER)

# include <values.h>
# include <types.h>
# include <string.h>
# include <quickdra.h>
# include <fonts.h>
# include <events.h>
# include <resource.h>
# include <menus.h>
# include <textedit.h>
# include <dialogs.h>
# include <desk.h>
# include <toolutil.h>
# include <memory.h>
# include <files.h>
# include <osutils.h>
# include <osevents.h>
# include <diskinit.h>
# include <packages.h>
# include <traps.h>
# include <AppleEve.h>
```

```
#else //_MSC_VER

# include <values.h>
# include <types.h>
# include <strings.h>
# include <quickdraw.h>
# include <fonts.h>
# include <events.h>
# include <resources.h>
# include <windows.h>
# include <menus.h>
# include <textedit.h>
# include <dialogs.h>
# include <desk.h>
# include <toolutils.h>
# include <memory.h>
# include <files.h>
# include <osutils.h>
# include <osevents.h>
# include <diskinit.h>
# include <packages.h>
# include <traps.h>
# include <AppleEvents.h>

#endif //_MSC_VER

# include <ole2.h>
# include <olenls.h>
# include <dispatch.h>

# define TCHAR               char
# define TSTR(str)           str
# define STRING(str)           (str)
# define WIDESTRING(str)        (str)

#elif defined(WIN32)

# include <windows.h>
# include <ole2.h>
# include <oleauto.h>

# if defined(UNICODE)
    #define TCHAR      WCHAR
    #define TSTR(str)        L##str
    #define STRING(str)        (str)
    #define WIDESTRING(str)  (str)
# else
    #define TCHAR      char
    #define TSTR(str)        str
    #define STRING(str)        AnsiString(str)
    #define WIDESTRING(str)  WideString(str)
    extern "C" char FAR* AnsiString(OLECHAR FAR* strIn);
    extern "C" OLECHAR FAR* WideString(char FAR* strIn);
# endif
```

```
#else /* WIN16 */

# include <windows.h>
# include <ole2.h>
# include <olenls.h>
# include <dispatch.h>

# define TCHAR               char
# define TSTR(str)           str
# define STRING(str)           (str)
# define WIDESTRING(str)        (str)
#endif
```

MAIN.CPP (DSPCALC2 Sample)

```
/***
*main.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This module is the main entry point of the sample IDispatch
*  calculator, dspcalc2.exe.
*
*  This program is intended to demonstrate an implementation of
*  the IDispatch interface.
*
*Implementation Notes:
*
****************************************************************************
*/

#include "dspcalc2.h"

TCHAR g_szAppName[] = TSTR("DspCalc2");

BOOL InitApplication(HANDLE);
BOOL InitInstance(HANDLE, int);


extern "C" {
long FAR PASCAL WndProc(HWND, UINT, WPARAM, LPARAM);
int PASCAL WinMain(HANDLE, HANDLE, LPSTR, int);
}


extern "C" int PASCAL
WinMain(
    HANDLE hinst,
    HANDLE hPrevInstance,
    LPSTR lpCmdLine,
    int nCmdShow)
{
    MSG msg;

    if(!hPrevInstance)
      if(!InitApplication(hinst))
     return FALSE;

    if(InitOle() != NOERROR)
      return FALSE;

    if(!InitInstance(hinst, nCmdShow)){
      UninitOle();
      return FALSE;
    }
    while(GetMessage(&msg, NULL, NULL, NULL)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }

    UninitOle();

    return msg.wParam;
}


BOOL
InitApplication(HANDLE hinst)
{
    WNDCLASS  wc;

    wc.style           = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc     = WndProc;
    wc.cbClsExtra= 0;
    wc.cbWndExtra= DLGWINDOWEXTRA;
    wc.hInstance = hinst;
    wc.hIcon           = LoadIcon(hinst, g_szAppName);
    wc.hCursor         = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground   = (HBRUSH)(COLOR_APPWORKSPACE+1);
    wc.lpszMenuName    = NULL;
    wc.lpszClassName   = g_szAppName;

    if(!RegisterClass(&wc))
      return FALSE;

    return TRUE;
}


BOOL
InitInstance(HANDLE hinst, int nCmdShow)
{
    g_pcalc->m_hwnd = CreateDialog(hinst, g_szAppName, 0, NULL);

    ShowWindow(g_pcalc->m_hwnd, nCmdShow);

    g_pcalc->m_arith.Display();

    return TRUE;
}


extern "C" long FAR PASCAL
WndProc(
    HWND hwnd,
    UINT message,
    WPARAM wParam,
    LPARAM lParam)
{
    switch(message){
    case WM_COMMAND:
      switch(wParam){
      case IDC_ZERO:
      case IDC_ONE:
      case IDC_TWO:
      case IDC_THREE:
      case IDC_FOUR:
      case IDC_FIVE:
      case IDC_SIX:
      case IDC_SEVEN:
      case IDC_EIGHT:
      case IDC_NINE:
      case IDC_PLUS:
      case IDC_MINUS:
      case IDC_MULT:
      case IDC_DIV:
      case IDC_CLEAR:
      case IDC_EQUALS:
     g_pcalc->m_arith.ButtonPush(wParam);
     return 0;
      }
      break;

    case WM_DESTROY:
      PostQuitMessage(0);
      return 0;
    }
    return DefWindowProc(hwnd, message, wParam, lParam);
}


#if defined(WIN32)

extern "C" char FAR*
ConvertStrWtoA(OLECHAR FAR* strIn, char FAR* buf, UINT size)
{
  int badConversion = FALSE;

  WideCharToMultiByte(CP_ACP, NULL,
                   strIn, -1,
                 buf, size,
                 NULL, &badConversion);
  return buf;
}

extern "C" char FAR*
AnsiString(OLECHAR FAR* strIn)
{
  static char buf[256];

  return (ConvertStrWtoA(strIn, buf, 256));
}
```

```
#endif
```

RESOURCE.H (DSPCALC2 Sample)

```
/***
*resource.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*
*Implementation Notes:
*
****************************************************************************
*/

#ifdef _MAC

#define kMinSize 500   /* minimum size (in K) */
#define kPrefSize500   /* preferred size (in K) */

#define kMinHeap 21 * 1024
#define kMinSpace8 * 1024

#define    rMenuBar    128   /* menu bar */
#define    rAboutAlert 128   /* about alert */
#define    rUserAlert  129   /* error alert */

#define    rCalc       130

#define    mApple            128   /* Apple menu */
#define    iAbout            1

#define    mFile       129   /* File menu */
#define iClose         4
#define    iQuit       12

#define    mEdit       130   /* Edit menu */
#define    iUndo       1
#define    iCut        3
#define    iCopy       4
#define    iPaste            5
#define    iClear            6

#endif

// Note: there is code that depends on all of the digits being contiguous
// and in the following order.

// Mac Note: On the mac these IDs correspond to the control indices
// in the DITL array.

#define IDC_ZERO     1
#define IDC_ONE      2
#define IDC_TWO      3
#define IDC_THREE    4
```

```
#define IDC_FOUR     5
#define IDC_FIVE     6
#define IDC_SIX      7
#define IDC_SEVEN    8
#define IDC_EIGHT    9
#define IDC_NINE     10

// Note: there is code that depends on the operators being contiguous
// and in the following order.
//
#define IDC_PLUS     11
#define IDC_MINUS    12
#define IDC_MULT     13
#define IDC_DIV      14

#define IDC_CLEAR    15
#define IDC_EQUALS   16

#define IDC_DISPLAY  17
```

GIZMOBAR

This sample demonstrates an implementation of the GizmoBar custom control. For information on compiling and building the sample, see MAKEFILE (GIZMOBAR Sample).

MAKEFILE   (GIZMOBAR Sample)

```
#
# MAKEFILE
# GizmoBar Version 1.00, March 1993
#
#

!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

APP=gizmobar

all: $(APP).dll

OLE_FLAGS = -DUNICODE

!ifndef NO_DEBUG
OLE_FLAGS = $(OLE_FLAGS) /D_DEBUG /DDEBUG /D_DEBUGTRACE=0
!endif

.SUFFIXES: .h .c .obj .exe .dll .cpp .res .rc

INCLS    = ..\include\$(APP).h gizmoint.h gizmo.h win1632.h

OBJS1    = $(APP).obj init.obj
OBJS2    = api.obj paint.obj gizmo.obj
OBJS     = $(OBJS1) $(OBJS2)


RCFILES1 = $(APP).rcv
RCFILES  = $(RCFILES1)


#
# Tool Directives
#
.c.obj:
    $(cc) $(cvars) $(cflags) $(cdebug) $(OLE_FLAGS) $*.c


.rc.res:
    rc -r -DWIN32 -DDEBUG $*.rc


clean:
    -del *.obj
    -del *.dll
    -del *.res
    -del *.lib
    -del *.map
    -del *.exp

$(APP).dll: $(OBJS) $(APP).lib $(APP).def $(APP).res
```

```
    $(link) $(linkdebug) $(dlllflags) $(APP).exp $(OBJS) $*.res -out:$@
-map:$*.map $(guilibsdll)  bttncur.lib
    if not exist ..\bin mkdir ..\bin
    copy $(APP).dll ..\bin

$(APP).lib: $(OBJS) $(APP).def
    $(implib) $(OBJS) -out:$(APP).lib -def:$(APP).def -machine:$(CPU)
    if not exist ..\lib mkdir ..\lib
    copy $(APP).lib ..\lib

##### Dependencies #####
$(APP).obj     : $(APP).c       $(INCLS)
paint.obj      : paint.c        $(INCLS)
init.obj       : init.c         $(INCLS)
api.obj        : api.c          $(INCLS)
gizmo.obj      : gizmo.c        $(INCLS)


$(APP).res  : $(APP).rc   $(INCLS) $(RCFILES)
```

API.C   (GIZMOBAR Sample)

```
/*
 * API.C
 * GizmoBar Version 1.01
 *
 * API functions affecting a GizmoBar and a message processing
 * function to handle the equivalent called through messages.
 *
 * Copyright (c)1993-1994 Microsoft Corporation, All Rights Reserved
 *
 * Kraig Brockschmidt, Software Design Engineer
 * Microsoft Systems Developer Relations
 *
 * Internet  :  kraigb@microsoft.com
 * Compuserve:  >INTERNET:kraigb@microsoft.com
 */


#include <windows.h>
#include <malloc.h>
#include "gizmoint.h"




/*
 * GBMessageHandler
 *
 * Purpose:
 *  Processes control messages that are equivalents of available
 *  control API.  The data passed with these messages is simply
 *  extracted from structures and passed as parameters to their
 *  equivalent function.
 *
 * Parameters:
 *  <Standard Message Parameters> plus
 *  pGB             PGIZMOBAR providing control-specific data.
 *
 * Return Value:
 *  LRESULT         Return value from equivalent API function.
 */

LRESULT GBMessageHandler(HWND hWnd, UINT iMsg, WPARAM wParam
    , LPARAM lParam, PGIZMOBAR pGB)
    {
    LRESULT         lRet=0L;
    LPCREATEGIZMOW   pCGW;
    LPCREATEGIZMOA   pCGA;
    LPGBMSG         pMsg;
    LPGBGETTEXTW    pGTW;
    LPGBGETTEXTA    pGTA;
    LPGBGETINT      pGI;
    LPGBSETINT      pSI;
```

```
if (NULL==pGB)
    return 0L;

switch (iMsg)
    {
    case GBM_HWNDASSOCIATESET:
        lRet=(LRESULT)(UINT)GBHwndAssociateSet(hWnd
            , (HWND)wParam);
        break;

    case GBM_HWNDASSOCIATEGET:
        lRet=(LRESULT)(UINT)GBHwndAssociateGet(hWnd);
        break;

    case GBM_GIZMOADDW:
        pCGW=(LPCREATEGIZMOW)lParam;
        lRet=(LRESULT)GBGizmoAddW(pCGW->hWndParent, pCGW->iType
            , pCGW->iGizmo, pCGW->uID, pCGW->dx, pCGW->dy
            , pCGW->pszText, pCGW->hBmp, pCGW->iImage, pCGW->uState);
        break;

    case GBM_GIZMOADDA:
        pCGA=(LPCREATEGIZMOA)lParam;
        lRet=(LRESULT)GBGizmoAddA(pCGA->hWndParent, pCGA->iType
            , pCGA->iGizmo, pCGA->uID, pCGA->dx, pCGA->dy
            , pCGA->pszText, pCGA->hBmp, pCGA->iImage, pCGA->uState);
        break;

    case GBM_GIZMOREMOVE:
        lRet=(LRESULT)GBGizmoRemove(hWnd, wParam);
        break;

    case GBM_GIZMOSENDMESSAGE:
        pMsg=(LPGBMSG)lParam;
        lRet=GBGizmoSendMessage(hWnd, wParam, pMsg->iMsg
            , pMsg->wParam, pMsg->lParam);
        break;

    case GBM_GIZMOSHOW:
        lRet=(LRESULT)GBGizmoShow(hWnd, wParam
            , (BOOL)LOWORD(lParam));
        break;

    case GBM_GIZMOENABLE:
        lRet=(LRESULT)GBGizmoEnable(hWnd, wParam
            , (BOOL)LOWORD(lParam));
        break;

    case GBM_GIZMOCHECK:
        lRet=(LRESULT)GBGizmoCheck(hWnd, wParam
            , (BOOL)LOWORD(lParam));
        break;

    case GBM_GIZMOFOCUSSET:
        lRet=(LRESULT)GBGizmoFocusSet(hWnd, wParam);
        break;

    case GBM_GIZMOEXIST:
        lRet=(LRESULT)GBGizmoExist(hWnd, wParam);
        break;

    case GBM_GIZMOTYPEGET:
        lRet=(LRESULT)GBGizmoTypeGet(hWnd, wParam);
        break;

    case GBM_GIZMODATASET:
        lRet=(LRESULT)GBGizmoDataSet(hWnd, wParam
            , (DWORD)lParam);
        break;

    case GBM_GIZMODATAGET:
        lRet=(LRESULT)GBGizmoDataGet(hWnd, wParam);
        break;

    case GBM_GIZMONOTIFYSET:
        lRet=(LRESULT)GBGizmoNotifySet(hWnd, wParam
            , (BOOL)LOWORD(lParam));
        break;

    case GBM_GIZMONOTIFYGET:
        lRet=(LRESULT)GBGizmoNotifyGet(hWnd, wParam);
        break;

    case GBM_GIZMOTEXTGETW:
        pGTW=(LPGBGETTEXTW)lParam;
        lRet=(LRESULT)GBGizmoTextGetW(hWnd, wParam, pGTW->psz
            , pGTW->cch);
        break;
    case GBM_GIZMOTEXTGETA:
        pGTA=(LPGBGETTEXTA)lParam;
        lRet=(LRESULT)GBGizmoTextGetA(hWnd, wParam, pGTA->psz
            , pGTA->cch);
        break;

    case GBM_GIZMOTEXTSETW:
        GBGizmoTextSetW(hWnd, wParam, (LPWSTR)lParam);
        break;

    case GBM_GIZMOTEXTSETA:
        GBGizmoTextSetA(hWnd, wParam, (LPSTR)lParam);
        break;

    case GBM_GIZMOINTGET:
        pGI=(LPGBGETINT)lParam;
        lRet=(LRESULT)GBGizmoIntGet(hWnd, wParam, &pGI->fSuccess
            , pGI->fSigned);
        break;


    case GBM_GIZMOINTSET:
```

```
            pSI=(LPGBSETINT)lParam;
            GBGizmoIntSet(hWnd, wParam, pSI->uValue, pSI->fSigned);
            break;

        default:
            break;
        }

    return lRet;
    }
```

```
/*
 * PGizmoFromHwndID
 *
 * Purpose:
 *  Retrieves the pGizmo for the given GizmoBar and the gizmo ID.
 *
 * Parameters:
 *  hWnd            HWND of a GizmoBar.
 *  uID             UINT gizmo identifier.
 *
 * Return Value:
 *  PGIZMO          NULL if the gizmo does not exist or hWnd is
 *                  invalid.  Non-NULL PGIZMO otherwise.
 */

PGIZMO PGizmoFromHwndID(HWND hWnd, UINT uID)
    {
    PGIZMOBAR     pGB;

    if (!IsWindow(hWnd))
        return FALSE;

    pGB=(PGIZMOBAR)GetWindowLong(hWnd, GBWL_STRUCTURE);

    if (NULL==pGB)
        return FALSE;

    return GizmoPFind(&pGB->pGizmos, uID);
    }
```

```
/*
 * GBHwndAssociateSet
 *
 * Purpose:
 *  Changes the associate window of a GizmoBar.
 *
 * Parameters:
 *  hWnd            HWND of the control window.
 *
 * Set Parameters:
 *  hWndAssociate   HWND of new associate.
 *
 * Return Value:
 *  HWND            Handle of previous associate.
 */

HWND WINAPI GBHwndAssociateSet(HWND hWnd, HWND hWndNew)
    {
    HWND        hWndOld=NULL;
    PGIZMOBAR   pGB;

    pGB=(PGIZMOBAR)GetWindowLong(hWnd, GBWL_STRUCTURE);

    if (NULL!=pGB)
        {
        hWndOld=pGB->hWndAssociate;
        pGB->hWndAssociate=hWndNew;

        if (NULL!=hWndOld)
            SendCommand(hWndOld, pGB->uID, GBN_ASSOCIATELOSS, hWnd);

        if (NULL!=hWndNew)
            SendCommand(hWndNew, pGB->uID, GBN_ASSOCIATEGAIN, hWnd);
        }

    return hWndOld;
    }





/*
 * GBHwndAssociateGet
 *
 * Purpose:
 *  Retrieves the associate window of a GizmoBar
 *
 * Parameters:
 *  hWnd            HWND of the control window.
 *
 * Set Parameters:
 *  hWndAssociate   HWND of new associate.
 *
 * Return Value:
```

```
 *  HWND            Handle of current associate.
 */

HWND WINAPI GBHwndAssociateGet(HWND hWnd)
    {
    HWND        hWndOld=NULL;
    PGIZMOBAR   pGB;

    pGB=(PGIZMOBAR)GetWindowLong(hWnd, GBWL_STRUCTURE);

    if (NULL!=pGB)
        hWndOld=pGB->hWndAssociate;

    return hWndOld;
    }





/*
 * GBGizmoAdd
 *
 * Purpose:
 *  Creates a new gizmo on the GizmoBar.  Subsequent operations
 *  should be done using the identifier, uID, for this gizmo.
 *
 * Parameters:
 *  hWnd            HWND of the GizmoBar.
 *  iType           UINT type of the gizmo to create.
 *  iGizmo          UINT position (zero-based) at which to place the
 *                  gizmo.
 *  uID             UINT identifier for WM_COMMAND from this gizmo.
 *  dx, dy          UINT dimensions of the gizmo.
 *  pszText         LPTSTR initial text for edit, list, combo, and
 *                  text gizmos.
 *  hBitmap         HBITMAP for gizmos of the button types (COMMAND
 *                  or ATTRIBUTE) specifies a source bitmap from
 *                  which the button image is taken.
 *  iImage          UINT index into hBitmap for the image for this
 *                  button.
 *  uState          UINT initial state of the gizmo.
 *
 * Return Value:
 *  BOOL            TRUE if creation succeeded, FALSE otherwise.
 */

BOOL WINAPI GBGizmoAddW(HWND hWnd, UINT iType, UINT iGizmo, UINT uID
    , UINT dx, UINT dy, LPWSTR pszText, HBITMAP hBmp, UINT iImage
    , UINT uState)
    {
    BOOL        fSuccess;
    PGIZMOBAR   pGB;
    PGIZMO      pGizmo;
    if (!IsWindow(hWnd))
        return FALSE;

    pGB=(PGIZMOBAR)GetWindowLong(hWnd, GBWL_STRUCTURE);

    if (NULL==pGB)
        return FALSE;

    /*
     * This automatically creates the windows, allocates structures,
     * includes the gizmo in pGB->pGizmos, and so forth.
     */
    pGizmo=GizmoPAllocate(&fSuccess, &pGB->pGizmos, hWnd, iType
        , iGizmo, uID, dx, dy, pszText, hBmp, iImage, uState);

    if (fSuccess)
        {
        if (NULL!=pGB->hWndAssociate)
            {
            SendCommand(pGB->hWndAssociate,GBN_GIZMOADDED, pGB->uID
                , hWnd);
            }

        InvalidateRect(hWnd, NULL, TRUE);
        UpdateWindow(hWnd);
        }
    else
        GizmoPFree(&pGB->pGizmos, pGizmo);

    return fSuccess;
    }

BOOL WINAPI GBGizmoAddA(HWND hWnd, UINT iType, UINT iGizmo, UINT uID
    , UINT dx, UINT dy, LPSTR pszText, HBITMAP hBmp, UINT iImage
    , UINT uState)
{
    LPWSTR  pszTemp;
    BOOL    retVal;

    if(pszText != NULL)
    {
        pszTemp = malloc((1+strlen(pszText))*sizeof(WCHAR));

        if (pszTemp)
        {
            MultiByteToWideChar(CP_ACP, 0, pszText, -1, pszTemp,
strlen(pszText)+1);
            retVal=GBGizmoAddW(hWnd, iType, iGizmo, uID, dx, dy, pszTemp,
hBmp, iImage, uState);

            free(pszTemp);
            return retVal;
        }

    }
    else
        return GBGizmoAddW(hWnd, iType, iGizmo, uID, dx, dy, NULL, hBmp,
iImage, uState);
}


/*
 * GBGizmoRemove
 *
 * Purpose:
 *  Removes an existing gizmo from the GizmoBar.
 *
 * Parameters:
 *  hWnd            HWND of the GizmoBar.
 *  uID             UINT identifier for this gizmo.
 *
 * Return Value:
 *  BOOL            TRUE if deletion succeeded, FALSE otherwise.
 */

BOOL WINAPI GBGizmoRemove(HWND hWnd, UINT uID)
    {
    PGIZMOBAR   pGB;
    PGIZMO      pGizmo;

    if (!IsWindow(hWnd))
        return FALSE;

    pGB=(PGIZMOBAR)GetWindowLong(hWnd, GBWL_STRUCTURE);

    if (NULL==pGB)
        return FALSE;

    pGizmo=GizmoPFind(&pGB->pGizmos, uID);

    if (NULL==pGizmo)
        return FALSE;

    GizmoPFree(&pGB->pGizmos, pGizmo);

    if (NULL!=pGB->hWndAssociate)
        {
        SendCommand(pGB->hWndAssociate, GBN_GIZMOREMOVED, pGB->uID
            , hWnd);
        }

    InvalidateRect(hWnd, NULL, TRUE);
    UpdateWindow(hWnd);
    return TRUE;
    }
```

```
/*
 * GBGizmoSendMessage
 *
 * Purpose:
 *  Implements the equivalent of SendMessage to a gizmo in the
 *  GizmoBar.  Separators, command buttons, and attribute buttons
 *  do not accept messages.
 *
 * Parameters:
 *  hWnd            HWND of the GizmoBar.
 *  uID             UINT identifier of the gizmo to affect.
 *  iMsg            UINT message to send.
 *  wParam          WPARAM of the message.
 *  lParam          LPARAM of the message.
 *
 * Return Value:
 *  LRESULT         Return value from the message.  0L if the
 *                  gizmo does not accept messages.
 */

LRESULT WINAPI GBGizmoSendMessage(HWND hWnd, UINT uID, UINT iMsg
    , WPARAM wParam, LPARAM lParam)
    {
    PGIZMO      pGizmo;
    LONG        lRet=0L;

    pGizmo=PGizmoFromHwndID(hWnd, uID);

    if (NULL!=pGizmo && NULL!=pGizmo->hWnd)
        lRet=SendMessage(pGizmo->hWnd, iMsg, wParam, lParam);

    return lRet;
    }






/*
 * GBGizmoShow
 *
 * Purpose:
 *  Shows or hides a control, adjusting the positions of all others
 *  to make room for or reuse the space for this control.
 *
 * Parameters:
 *  hWnd            HWND of the GizmoBar.
 *  uID             UINT identifier of the gizmo to affect.
 *  fShow           BOOL TRUE to show the gizmo, FALSE to hide it.
 *
 * Return Value:
 *  BOOL            TRUE if the function was successful, FALSE
 *                  otherwise.
 */

BOOL WINAPI GBGizmoShow(HWND hWnd, UINT uID, BOOL fShow)
    {
    BOOL        fRet=FALSE;
    PGIZMO      pGizmo;

    pGizmo=PGizmoFromHwndID(hWnd, uID);

    if (NULL!=pGizmo)
        {
        if (fShow && pGizmo->fHidden)
            {
            if (NULL!=pGizmo->hWnd)
                ShowWindow(pGizmo->hWnd, SW_SHOWNORMAL);

            GizmosExpand(pGizmo);
            }

        if (!fShow && !pGizmo->fHidden)
            {
            if (NULL!=pGizmo->hWnd)
                ShowWindow(pGizmo->hWnd, SW_HIDE);

            GizmosCompact(pGizmo);
            }

        //This will be right even if we didn't change anything.
        pGizmo->fHidden=!fShow;
        }

    InvalidateRect(hWnd, NULL, TRUE);
    UpdateWindow(hWnd);
    return fRet;
    }




/*
 * GBGizmoEnable
 *
 * Purpose:
 *  Enables or disables a control on the GizmoBar.
 *
 * Parameters:
 *  hWnd            HWND of the GizmoBar.
 *  uID             UINT identifier of the gizmo to affect.
 *  fEnable         BOOL TRUE to enable the gizmo, FALSE otherwise.
 *
 * Return Value:
 *  BOOL            TRUE if the gizmo was previously disabled, FALSE
 *                  otherwise.
 */

BOOL WINAPI GBGizmoEnable(HWND hWnd, UINT uID, BOOL fEnable)
    {
    PGIZMO      pGizmo;
    BOOL        fRet=FALSE;

    pGizmo=PGizmoFromHwndID(hWnd, uID);

    if (NULL==pGizmo)
        return FALSE;

    fRet=(BOOL)(BUTTONGROUP_DISABLED & pGizmo->uState);

    //Use windows to enable or disable window gizmos
    if (NULL!=pGizmo->hWnd)
        EnableWindow(pGizmo->hWnd, fEnable);
    else
        {
        /*
         * If we're not down, command and attribute buttons act
         * the same.
         */
        if (!(BUTTONGROUP_DOWN & pGizmo->uState))
            {
            GizmoPStateSet(hWnd, pGizmo, fEnable
                ? COMMANDBUTTON_UP : COMMANDBUTTON_DISABLED);
            }
        else
            {
            /*
             * Attribute buttons are a little more sensitive with
             * DOWNDISABLED
             */
            GizmoPStateSet(hWnd, pGizmo, fEnable
                ? ATTRIBUTEBUTTON_DOWN
                : ATTRIBUTEBUTTON_DOWNDISABLED);
            }
        }

    return fRet;
    }







/*
 * GBGizmoCheck
 *
 * Purpose:
 *  Checks or unchecks an attribute button in the GizmoBar.  If the
 *  gizmo is part of a group of mutually exclusive attributes, then
 *  other gizmos are unchecked when this one is checked.  If this is
 *  the only one checked in these circumstances, this function is a
 *  NOP.
 *
 * Parameters:
 *  hWnd            HWND of the GizmoBar.
 *  uID             UINT identifier of the gizmo to affect.
 *  fCheck          BOOL TRUE to check this gizmo, FALSE to uncheck.
 *
 * Return Value:
 *  BOOL            TRUE if the change took place.  FALSE otherwise.
 */

BOOL WINAPI GBGizmoCheck(HWND hWnd, UINT uID, BOOL fCheck)
    {
    PGIZMOBAR   pGB;
    PGIZMO      pGizmo;

    if (!IsWindow(hWnd))
        return FALSE;

    pGB=(PGIZMOBAR)GetWindowLong(hWnd, GBWL_STRUCTURE);

    if (NULL==pGB)
        return FALSE;

    pGizmo=GizmoPFind(&pGB->pGizmos, uID);

    if (NULL!=pGizmo)
        GizmoPCheck(hWnd, pGizmo, fCheck);

    return TRUE;
    }





/*
 * GBGizmoFocusSet
 *
 * Purpose:
 *  Sets the focus to a partuclar gizmo in the gizmo if that gizmo
 *  can accept the focus.  Separators, attribute buttons, text,
 *  and command buttons cannot have the focus.
 *
 * Parameters:
 *  hWnd            HWND of the GizmoBar.
 *  uID             UINT identifier of the gizmo to affect.
 *
 * Return Value:
 *  BOOL            TRUE if the focus was set.  FALSE otherwise,
 *                  such as when uID identifies a control that cannot
 *                  have focus.
 */

UINT WINAPI GBGizmoFocusSet(HWND hWnd, UINT uID)
    {
    PGIZMO      pGizmo;
    BOOL        fRet=FALSE;

    pGizmo=PGizmoFromHwndID(hWnd, uID);

    if (NULL!=pGizmo && NULL!=pGizmo->hWnd)
        {
        fRet=TRUE;
        SetFocus(pGizmo->hWnd);
        }

    return fRet;
    }





/*
 * GBGizmoExist
 *
 * Purpose:
 *  Determines if a gizmo of a given identifier exists in the
 *  GizmoBar.
 *
 * Parameters:
 *  hWnd            HWND of the GizmoBar.
 *  uID             UINT identifier to verify.
 *
 * Return Value:
 *  BOOL            TRUE if the gizmo exists, FALSE otherwise.
 */

BOOL WINAPI GBGizmoExist(HWND hWnd, UINT uID)
    {
    return (NULL!=PGizmoFromHwndID(hWnd, uID));
    }





/*
 * GBGizmoTypeGet
 *
 * Purpose:
 *  Returns the type of the gizmo specified by the given identifer.
 *
 * Parameters:
 *  hWnd            HWND of the GizmoBar.
 *  uID             UINT identifier to find.
 *
 * Return Value:
 *  int                 A GIZMOTYPE_* value if the function is
 *                      successful, otherwise -1.
 */

int WINAPI GBGizmoTypeGet(HWND hWnd, UINT uID)
    {
    int         iRet=-1;
    PGIZMO      pGizmo;

    pGizmo=PGizmoFromHwndID(hWnd, uID);

    if (NULL!=pGizmo)
        iRet=pGizmo->iType;

    return iRet;
    }





/*
 * GBGizmoDataSet
 * GBGizmoDataGet
 *
 * Purpose:
 *  Sets or retrieves an extra DWORD value associated with the given
 *  gizmo.  Applications can store any information here they please.
 *
 * Parameters:
 *  hWnd                HWND of the GizmoBar.
 *  uID                 UINT identifier of the gizmo.
 *  dwData              (Set only) DWORD data to store with the gizmo.
 *
 * Return Value:
 *  DWORD               Set:  Previous value
 *                      Get:  Current value
 */

DWORD WINAPI GBGizmoDataSet(HWND hWnd, UINT uID, DWORD dwData)
    {
    PGIZMO      pGizmo;
    DWORD       dw=0L;

    pGizmo=PGizmoFromHwndID(hWnd, uID);

    if (NULL!=pGizmo)
        {
        dw=pGizmo->dwData;
        pGizmo->dwData=dwData;
        }

    return dw;
    }



DWORD WINAPI GBGizmoDataGet(HWND hWnd, UINT uID)
    {
    PGIZMO      pGizmo;
    DWORD       dw=0L;

    pGizmo=PGizmoFromHwndID(hWnd, uID);

    if (NULL!=pGizmo)
        dw=pGizmo->dwData;

    return dw;
    }






/*
 * GBGizmoNotifySet
 * GBGizmoNotifyGet
 *
 * Purpose:
 *  Sets or retrieves the notify status of a gizmo.  If notify is
 *  FALSE, the no WM_COMMAND messages are sent from the GizmoBar to
 *  the parent window when this gizmo is used.
 *
 * Parameters:
 *  hWnd            HWND of the GizmoBar.
 *  uID             UINT identifier of the gizmo.
 *  fNotify         (Set only) BOOL new notify status to set.
 *
 * Return Value:
 *  BOOL            Set:  Previous value of the notify flag.
 *                  Get:  Current value of the notify flag.
 */

BOOL WINAPI GBGizmoNotifySet(HWND hWnd, UINT uID, BOOL fNotify)
    {
    PGIZMO      pGizmo;
    BOOL        fRet=FALSE;

    pGizmo=PGizmoFromHwndID(hWnd, uID);

    if (NULL!=pGizmo)
        {
        fRet=pGizmo->fNotify;
        pGizmo->fNotify=fNotify;
        }

    return fRet;
    }


BOOL WINAPI GBGizmoNotifyGet(HWND hWnd, UINT uID)
    {
    PGIZMO      pGizmo;
    BOOL        fRet=FALSE;

    pGizmo=PGizmoFromHwndID(hWnd, uID);

    if (NULL!=pGizmo)
        fRet=pGizmo->fNotify;

    return fRet;
    }






/*
 * GBGizmoTextSet
 * GBGizmoTextGet
 *
 * Purpose:
 *  Retrieves or sets text in a GizmoBar gizmo.  Separators, command
 *  buttons, and attribute buttons are not affected by this call.
 *
 * Parameters:
 *  hWnd            HWND of the GizmoBar.
 *  uID             UINT identifying the gizmo.
 *  psz             LPTSTR (Set) providing the text to show in the
 *                  window or (Get) pointing to a buffer to receive
 *                  the text.
 *  cch             (Get only) UINT maximum number of chars to copy
 *                  to psz.
 *
 * Return Value:
 *  int             Number of characters copied to psz.
 */

void WINAPI GBGizmoTextSetW(HWND hWnd, UINT uID, LPWSTR psz)
    {
    //This fails on non-windowed gizmos anyway, so we don't check.
    SetDlgItemTextW(hWnd, uID, psz);
    return;
    }

void WINAPI GBGizmoTextSetA(HWND hWnd, UINT uID, LPSTR psz)
    {
    //This fails on non-windowed gizmos anyway, so we don't check.
    SetDlgItemTextA(hWnd, uID, psz);
    return;
    }


int WINAPI GBGizmoTextGetW(HWND hWnd, UINT uID, LPWSTR psz, UINT cch)
    {
    //This fails on non-windowed gizmos anyway, so we don't check.
    return GetDlgItemTextW(hWnd, uID, psz, cch);
    }

int WINAPI GBGizmoTextGetA(HWND hWnd, UINT uID, LPSTR psz, UINT cch)
    {
    //This fails on non-windowed gizmos anyway, so we don't check.
    return GetDlgItemTextA(hWnd, uID, psz, cch);
    }




/*
 * GBGizmoIntSet
 * GBGizmoIntGet
 *
 * Purpose:
 *  Retrieves or sets an integer in a GizmoBar gizmo.  Separators,
 *  command buttons, and attribute buttons are not affected by this
 *  call.
 *
 * Parameters:
 *  hWnd            HWND of the GizmoBar.
 *  uID             UINT identifying the gizmo.
 *
 *  (Set only)
 *  u               UINT value to set in the gizmo.
 *  fSigned         BOOL TRUE to indicate if the value is signed.
 *
 *  (Get only)
 *  pfTrans         BOOL FAR * in which the success of the function
 *                  is returned.
 *  fSigned         BOOL TRUE to indicate if the value is signed.
 *
 * Return Value:
 *  (Set): None
 *  (Get): UINT     Integer translation of the gizmo's text.
 */

void WINAPI GBGizmoIntSet(HWND hWnd, UINT uID, UINT u, BOOL fSigned)
    {
    //This fails on non-windowed gizmos anyway, so we don't check.
    SetDlgItemInt(hWnd, uID, u, fSigned);
    return;
    }



UINT WINAPI GBGizmoIntGet(HWND hWnd, UINT uID, BOOL FAR *pfTrans
    , BOOL fSigned)
```

```
{
//This fails on non-windowed gizmos anyway, so we don't check.
return GetDlgItemInt(hWnd, uID, pfTrans, fSigned);
}
```

GIZMO.H   (GIZMOBAR Sample)

```
/*
 * GIZMO.H
 * GizmoBar Version 1.01
 *
 * Data structure and type definitions for the GIZMO data structure.
 * Each gizmo on a gizmobar has one of these structures associated
 * with it.
 *
 * Copyright (c)1993-1994 Microsoft Corporation, All Rights Reserved
 *
 * Kraig Brockschmidt, Software Design Engineer
 * Microsoft Systems Developer Relations
 *
 * Internet  :  kraigb@microsoft.com
 * Compuserve:  >INTERNET:kraigb@microsoft.com
 */


#ifndef _GIZMO_H_
#define _GIZMO_H_

#ifdef __cplusplus
extern "C"
    {
#endif


typedef struct tagGIZMO
    {
    struct tagGIZMO     *pPrev;
    struct tagGIZMO     *pNext;
    UINT                 iType;
    HWND                 hWnd;       //Texts, edits, lists, combos
    UINT                 uID;
    UINT                 x, y;
    UINT                 dx, dy;
    UINT                 cxImage;    //From UIToolConfigureForDisplay
    UINT                 cyImage;
    HBITMAP              hBmp;       //Buttons only.
    UINT                 iBmp;
    BOOL                 fNotify;    //Send WM_COMMANDs?
    BOOL                 fHidden;    //Independent of state
    BOOL                 fDisabled;
    UINT                 uState;
    UINT                 uStateOrg;
    DWORD                dwData;     //Application-supplied data.
    } GIZMO, * PGIZMO;

typedef PGIZMO *PPGIZMO;
#define CBGIZMO sizeof(GIZMO)

//Property name we attach to controls in a gizmo to identify type
```

```
#define SZTYPEPROP      TEXT("iType")

//Number of controls we subclass
#define CSUBGIZMOS       4

//ID of edit controls in comboboxes
#define ID_COMBOEDIT     1001


/*
 * Conversion of iType (a positioned bit) into its position.
 * The BITPOSITION macro does not need to be fast because we only
 * use it once when creating a gizmo.  POSITIONBIT does, however,
 * since we use it in subclass procedures.
 */
#define BITPOSITION(i, j)  {int k=i; for (j=0; k>>=1; j++);}
#define POSITIONBIT(i)     (1 << i)

//Control classifications.  GIZMOBAR.H must be included first.
#define GIZMOTYPE_WINDOWS   (GIZMOTYPE_TEXT | GIZMOTYPE_EDIT \
                            | GIZMOTYPE_LISTBOX              \
                            | GIZMOTYPE_COMBOBOX             \
                            | GIZMOTYPE_BUTTONNORMAL)

#define GIZMOTYPE_BUTTONS   (GIZMOTYPE_BUTTONATTRIBUTEIN     \
                            | GIZMOTYPE_BUTTONATTRIBUTEEX    \
                            | GIZMOTYPE_BUTTONCOMMAND        \
                            | GIZMOTYPE_BUTTONNORMAL)

#define GIZMOTYPE_DRAWN     (GIZMOTYPE_BUTTONATTRIBUTEIN     \
                            | GIZMOTYPE_BUTTONATTRIBUTEEX    \
                            | GIZMOTYPE_BUTTONCOMMAND)


//These must stay in sync with GIZMOBAR.H
#define GIZMOTYPE_MIN               GIZMOTYPE_EDIT
#define GIZMOTYPE_MAX               GIZMOTYPE_BUTTONCOMMAND


//Enumeration callback
typedef BOOL (CALLBACK *PFNGIZMOENUM)(PGIZMO, UINT, DWORD);


//GIZMO.C
PGIZMO   GizmoPAllocate(int *, PPGIZMO, HWND, UINT, UINT, UINT
             , UINT, UINT, LPTSTR, HBITMAP, UINT, UINT);
void     GizmosExpand(PGIZMO);
PGIZMO   GizmoPFree(PPGIZMO, PGIZMO);
void     GizmosCompact(PGIZMO);
PGIZMO   GizmoPFind(PPGIZMO, UINT);
PGIZMO   GizmoPEnum(PPGIZMO, PFNGIZMOENUM, DWORD);
UINT     GizmoPStateSet(HWND, PGIZMO, UINT);
BOOL     GizmoPCheck(HWND, PGIZMO, BOOL);
```

```
LRESULT APIENTRY GenericSubProc(HWND, UINT, WPARAM, LPARAM);


#ifdef __cplusplus
    }
#endif

#endif //_GIZMO_H_
```

GIZMO.C (GIZMOBAR Sample)

```
/*
 * GIZMO.C
 * GizmoBar Version 1.01
 *
 * Allocate, free, find, and enumerate functions for the GIZMO
 * structure and a generic subclass procedure to handle tabbing
 * between gizmos.
 *
 * Copyright (c)1993-1994 Microsoft Corporation, All Rights Reserved
 *
 * Kraig Brockschmidt, Software Design Engineer
 * Microsoft Systems Developer Relations
 *
 * Internet  :  kraigb@microsoft.com
 * Compuserve:  >INTERNET:kraigb@microsoft.com
 */


#include <windows.h>
#include "gizmoint.h"


/*
 * In order to control tabbing in the gizmos, we need to subclass
 * real pushbuttons, edit controls, listboxes, and comboboxes.  So
 * we keep an array of the four original procs for such controls.
 */
WNDPROC     pfnOrg[CSUBGIZMOS]={NULL, NULL, NULL, NULL};


TCHAR szStatic[]=TEXT("static");
TCHAR szEdit[]=TEXT("edit");
TCHAR szCombobox[]=TEXT("combobox");
TCHAR szListbox[]=TEXT("listbox");
TCHAR szButton[]=TEXT("button");


//Here so PAINT.C can get at it.
TOOLDISPLAYDATA tdd;



/*
 * GizmoPAllocate
 *
 * Purpose:
 *  Allocates and initializes a GIZMO data structure.
 *
 * Parameters:
 *  pfSuccess       int * flag indicating success of failure.
 *  ppFirst         PPGIZMO providing the first gizmo in this list.
 *  hWndParent      HWND of the parent of this gizmo.  Can be NULL
 *                  for iType==GIZMOTYPE_BUTTON* or
 *                  GIZMOTYPE_SEPARATOR.
 *  iType           UINT gizmo control type.
 *  iGizmo          UINT index of this gizmo in the GizmoBar.
 *  uID             UINT identifier to send with WM_COMMAND for this
 *                  control.
 *  dx, dy          UINT width and height of the gizmo.
 *  pszText         LPTSTR to the text for edits, listboxes, combobox,
 *                  and text.
 *  dwStyle         DWORD style for edits, lists, and combos, and
 *                  texts.
 *  hBmp            HBITMAP for button gizmos, is applicable.
 *  iImage          UINT index into hBmp for the button image, if
 *                  applicable.
 *  uState          UINT initial state of the control.
 *
 * Return Value:
 *  PGIZMO          If NULL returned then GizmoPAllocate could not
 *                  allocate memory.  If a non-NULL pointer is
 *                  returned with *pfSuccess, then call GizmoPFree
 *                  immediately.  If you get a non-NULL pointer and
 *                  *pfSuccess==TRUE then the function succeeded.
 */

PGIZMO GizmoPAllocate(int *pfSuccess, PPGIZMO ppFirst
    , HWND hWndParent, UINT iType, UINT iGizmo, UINT uID, UINT dx
    , UINT dy, LPWSTR pszText, HBITMAP hBmp, UINT iImage, UINT uState)
    {
    PGIZMO          pGizmo;
    PGIZMO          pCur, pPrev;
    LPTSTR          pszClass;
    HINSTANCE       hInst;
    UINT            i;
    DWORD           dwStyle;
    HWND            hWndE;

    if (NULL==pfSuccess)
        return NULL;

    //Make sure we know of this gizmo type.
    if (GIZMOTYPE_MIN > iType || GIZMOTYPE_MAX < iType)
        return NULL;

    *pfSuccess=FALSE;

    //Allocate the structure
    pGizmo=(PGIZMO)(TCHAR *)LocalAlloc(LPTR, CBGIZMO);

    if (NULL==pGizmo)
        return NULL;


    //Store the necessary information for this gizmo.
    pGizmo->iType   =iType;
    pGizmo->uID     =uID;
```

```
pGizmo->hBmp    =hBmp;
pGizmo->iBmp    =iImage;
pGizmo->uState  =uState;
pGizmo->fNotify =TRUE;


/*
 * Insert this structure into our gizmo list.  Each time we scan
 * we increment the index counter (starting at zero) comparing it
 * to the desired index of insertion.  We then know exactly where
 * to insert this new gizmo.  Note that we insert the new gizmo
 * in the list appropriately for the given owner, so enumerations
 * will come out ordered in the same way for that owner.
 */

i=0;
pCur=*ppFirst;
pPrev=NULL;

while (NULL!=pCur && i++ < iGizmo)
    {
    pPrev=pCur;
    pCur =pCur->pNext;
    }

//Point to our neighbors
pGizmo->pPrev=pPrev;
pGizmo->pNext=pCur;


//Point out neighbors to us.
if (NULL==pPrev)
    *ppFirst=pGizmo;
else
    pPrev->pNext=pGizmo;

if (NULL!=pCur)
    pCur->pPrev=pGizmo;


//Our x-coordinate is the x of the previous gizmo plus its width.
if (NULL!=pPrev)
    pGizmo->x=pGizmo->pPrev->x+pGizmo->pPrev->dx;
else
    pGizmo->x=4;    //First gizmo is at x=4


//If we're a separator or image button, force standards on dx.
UIToolConfigureForDisplay(&tdd);
pGizmo->cxImage=tdd.cxImage;
pGizmo->cyImage=tdd.cyImage;

if ((GIZMOTYPE_DRAWN & iType) && NULL==hBmp)
    dx=tdd.cxButton;
```

```
if (GIZMOTYPE_SEPARATOR==iType)
    dx=6;

/*
 * Now create windows for edits, texts, lists, and comboboxes.
 * First calculate the most often defaults used in the switch.
 */
pGizmo->dx=dx+6;
pGizmo->dy=min(dy, tdd.cyButton);
pGizmo->y=2;
pszClass=NULL;

//If this is new gizmo is a window, create it.
switch (iType)
        {
        case GIZMOTYPE_TEXT:
            pGizmo->dx=dx;

            //Center vertically.
            pGizmo->y=(tdd.cyBar-1-pGizmo->dy) >> 1;
            pszClass=szStatic;
            dwStyle=SS_LEFT;
            break;

        case GIZMOTYPE_EDIT:
            //Center vertically.
            pGizmo->y=(tdd.cyBar-1-pGizmo->dy) >> 1;
            pszClass=szEdit;
            dwStyle=ES_LEFT | WS_BORDER | WS_TABSTOP;
            break;

        case GIZMOTYPE_LISTBOX:
            pGizmo->dy=dy;
            pszClass=szCombobox;
            dwStyle=CBS_DROPDOWNLIST | WS_TABSTOP;
            break;

        case GIZMOTYPE_COMBOBOX:
            pGizmo->dy=dy;
            pszClass=szCombobox;
            dwStyle=CBS_DROPDOWN | WS_TABSTOP;
            break;

        case GIZMOTYPE_BUTTONNORMAL:
            pGizmo->dy=dy;
            pszClass=szButton;
            dwStyle=BS_PUSHBUTTON | WS_TABSTOP;
            break;

        case GIZMOTYPE_SEPARATOR:
            pGizmo->dx=dx;
            pGizmo->y=3;
            break;

        case GIZMOTYPE_BUTTONATTRIBUTEIN:
```

```
        case GIZMOTYPE_BUTTONATTRIBUTEEX:
        case GIZMOTYPE_BUTTONCOMMAND:
            pGizmo->dx=dx;
            pGizmo->y=3;
            break;
        }


//If we matched a classname, create a window.
if (GIZMOTYPE_WINDOWS & iType)
    {
    if (!IsWindow(hWndParent))
        return pGizmo;

    hInst=GETWINDOWINSTANCE(hWndParent);    //Macro in book1632.h

    pGizmo->hWnd=CreateWindowW(pszClass, pszText
        , dwStyle | WS_CHILD | WS_VISIBLE, pGizmo->x, pGizmo->y
        , dx, pGizmo->dy, hWndParent, (HMENU)uID, hInst, NULL);

    if (NULL==pGizmo->hWnd)
        return pGizmo;

    /*
     * Subclass comboboxes, listboxes, edits, and windowed
     * buttons.  We use iType to index the original proc array
     * so we can use a single subclass procedure for all
     * controls.  If you mess with the gizmo type definitions,
     * this is going to break.
     */

    if (GIZMOTYPE_WINDOWS & iType && GIZMOTYPE_TEXT!=iType)
        {
        //Give the window its type.
        BITPOSITION(iType, i);
        SetProp(pGizmo->hWnd, SZTYPEPROP, (HANDLE)i);

        if (NULL==pfnOrg[i])
            {
            pfnOrg[i]=(WNDPROC)GetWindowLong(pGizmo->hWnd
                , GWL_WNDPROC);
            }

        SetWindowLong(pGizmo->hWnd, GWL_WNDPROC
            , (LONG)GenericSubProc);

        //If we're a combobox, subclass edit control
        if (GIZMOTYPE_COMBOBOX==iType)
            {
            hWndE=GetDlgItem(pGizmo->hWnd, ID_COMBOEDIT);
            SetProp(hWndE, SZTYPEPROP, (HANDLE)-1);

            if (NULL==pfnOrg[0])
                {
                pfnOrg[0]=(WNDPROC)GetWindowLong(pGizmo->hWnd
                        , GWL_WNDPROC);
                    }

                SetWindowLong(hWndE, GWL_WNDPROC
                    , (LONG)GenericSubProc);
                }
            }
        }


    //Finally, move all our neighbors to the right to accomodate us.
    GizmosExpand(pGizmo);

    *pfSuccess=TRUE;
    return pGizmo;
    }






/*
 * GizmoPFree
 *
 * Purpose:
 *  Reverses all initialization done by GizmoPAllocate, cleaning up
 *  any allocations including the application structure itself.
 *
 * Parameters:
 *  ppFirst         PPGIZMO providing the first gizmo in this list.
 *  pGizmo          PGIZMO to the structure
 *
 * Return Value:
 *  PGIZMO          NULL if successful, pGizmo if not, meaning we
 *                  couldn't free something.
 */

PGIZMO GizmoPFree(PPGIZMO ppFirst, PGIZMO pGizmo)
    {
    int     i;

    if (NULL==pGizmo)
        return NULL;

    //Move other gizmos to fill in this gap.
    GizmosCompact(pGizmo);

    //Unsubclass
    if (GIZMOTYPE_WINDOWS & pGizmo->iType
        && GIZMOTYPE_TEXT!=pGizmo->iType)
        {
        i=(int)GetProp(pGizmo->hWnd, SZTYPEPROP);
        RemoveProp(pGizmo->hWnd, SZTYPEPROP);
        if (GIZMOTYPE_COMBOBOX==pGizmo->iType)
            {
            HWND        hWndE;

            hWndE=GetDlgItem(pGizmo->hWnd, ID_COMBOEDIT);
            RemoveProp(hWndE, SZTYPEPROP);
            }

        SetWindowLong(pGizmo->hWnd, GWL_WNDPROC, (LONG)pfnOrg[i]);
        }

    //If this was a window gizmo, destroy the window.
    if (NULL!=pGizmo->hWnd && IsWindow(pGizmo->hWnd))
        DestroyWindow(pGizmo->hWnd);

    //Unlink ourselves.
    if (NULL!=pGizmo->pNext)
        pGizmo->pNext->pPrev=pGizmo->pPrev;

    if (NULL!=pGizmo->pPrev)
        pGizmo->pPrev->pNext=pGizmo->pNext;
    else
        *ppFirst=pGizmo->pNext;

    return (PGIZMO)LocalFree((HLOCAL)(UINT)(LONG)pGizmo);
    }





/*
 * GizmosExpand
 *
 * Purpose:
 *  Given a starting gizmo and a width, moves it and all gizmos to
 *  its right to the right by the width to make space for showing
 *  or creating a new gizmo.
 *
 * Parameters:
 *  pGizmo          PGIZMO specifying the gizmo that was inserted.
 *
 * Return Value:
 *  None
 */

void GizmosExpand(PGIZMO pGizmo)
    {
    int         cx;

    cx=(int)pGizmo->dx;

    /*
     * If we and the next control are buttons, use our width-1 to
     * expand so we overlap borders with our neighboring button.
     */

    if (NULL!=pGizmo->pNext)
        {
        if ((GIZMOTYPE_BUTTONS & pGizmo->pNext->iType)
            && (GIZMOTYPE_BUTTONS & pGizmo->iType))
            cx-=1;
        }

    //Walk the gizmo list moving them right by our width.
    pGizmo=pGizmo->pNext;

    while (NULL!=pGizmo)
        {
        pGizmo->x+=cx;

        //hWnd is NULL for buttons and separators.
        if (NULL!=pGizmo->hWnd)
            {
            SetWindowPos(pGizmo->hWnd, NULL, pGizmo->x, pGizmo->y
                , 0, 0, SWP_NOZORDER | SWP_NOSIZE);
            }

        pGizmo=pGizmo->pNext;
        }

    return;
    }






/*
 * GizmosCompact
 *
 * Purpose:
 *  Given a gizmo, moves all other gizmos to the right of it to the
 *  left by its width on the GizmoBar.  Used when removing or hiding
 *  the gizmo.
 *
 * Parameters:
 *  pGizmo          PGIZMO that is going away, visibly or physically.
 *
 * Return Value:
 *  None
 */

void GizmosCompact(PGIZMO pGizmo)
    {
    UINT        cx;
    PGIZMO      pCur;
    //Move all the gizmos beyond us back by our width.
    if (NULL!=pGizmo->pNext)
        {
        cx=pGizmo->pNext->x - pGizmo->x;
        pCur=pGizmo->pNext;

        while (NULL!=pCur)
            {
            pCur->x-=cx;

            if (NULL!=pCur->hWnd)
                {
                SetWindowPos(pCur->hWnd, NULL, pCur->x, pCur->y
                             , 0, 0, SWP_NOZORDER | SWP_NOSIZE);
                }

            pCur=pCur->pNext;
            }
        }

    return;
    }





/*
 * GizmoPFind
 *
 * Purpose:
 *  Given a GIZMO identifier, locates and returns a pointer to the
 *  structure for that position.
 *
 * Parameters:
 *  ppFirst         PPGIZMO providing the first gizmo in this list.
 *  uID             UINT identifier to find.
 *
 * Return Value:
 *  PGIZMO          A pointer to a GIZMO structure allocated through
 *                  GizmoPAllocate, NULL if iGizmo is out of range.
 */

PGIZMO GizmoPFind(PPGIZMO ppFirst, UINT uID)
    {
    PGIZMO       pGizmo;

    pGizmo=*ppFirst;

    /*
     * Yep, linear search, but a better search algorithm won't
     * improve things appreciably.  The better thing to optimize
     * is what the caller passes as ppFirst.
     */
    while (NULL!=pGizmo && uID!=pGizmo->uID)
        pGizmo=pGizmo->pNext;

    return pGizmo;
    }






/*
 * GizmoFEnum
 *
 * Purpose:
 *  Enumerates the list of GIZMO structures, passing each one to
 *  an application-defined callback.
 *
 * Parameters:
 *  ppFirst         PPGIZMO providing the first gizmo in this list.
 *  pfnEnum         PFNGIZMOENUM to call for each enumerated
 *                  structure.
 *  dw              DWORD extra data to pass to the enumeration
 *                  function.
 *
 * Return Value:
 *  PGIZMO          NULL if the enumeration completed.  Otherwise a
 *                  pointer to the gizmo that enumeration stopped on.
 */

PGIZMO GizmoPEnum(PPGIZMO ppFirst, PFNGIZMOENUM pfnEnum, DWORD dw)
    {
    PGIZMO  pGizmo;
    UINT    i=0;

    pGizmo=*ppFirst;

    while (NULL!=pGizmo)
        {
        if (!(*pfnEnum)(pGizmo, i++, dw))
            break;

        pGizmo=pGizmo->pNext;
        }

    return pGizmo;
    }




/*
 * GizmoPStateSet
 *
 * Purpose:
 *  State maniuplation functions.  Set and Clear also invalidate
 *  this gizmo's rectangle on the given window and forces a repaint.
 *
 * Parameters:
 *  hWnd            HWND of the window to repaint.
 *  pGizmo          PGIZMO affected.
 *  dwNew           DWORD new state flags.
 *
 * Return Value:
 *  UINT            Previous state.
 */

UINT  GizmoPStateSet(HWND hWnd, PGIZMO pGizmo, UINT uNew)
    {
    UINT        uRet;
    RECT        rc;

    if (GIZMOTYPE_SEPARATOR==pGizmo->iType)
        return pGizmo->uState;

    //Preserve the color conversion flags across this state change.
    uRet=pGizmo->uState;
    pGizmo->uState=(uNew & 0x00FF) | (uRet & 0xFF00);

    //Adjust the rectangle by  one to avoid repainting  borders.
    SetRect(&rc, pGizmo->x+1, pGizmo->y+1, pGizmo->x+pGizmo->dx-1
        , pGizmo->y+pGizmo->dy-1);
    InvalidateRect(hWnd, &rc, FALSE);
    UpdateWindow(hWnd);

    return uRet;
    }








/*
 * GizmoPCheck
 *
 * Purpose:
 *  Handles checking a single button in a group of attribute buttons.
 *  If the gizmo belongs to a group of mutually exclusive buttons
 *  then the others surrounding it are unchecked appropriately.
 *
 * Parameters:
 *  hWnd            HWND of the GizmoBar.
 *  pGizmo          PGIZMO of the gizmo affected.
 *  fCheck          BOOL TRUE to check the button, FALSE to uncheck.
 *
 * Return Value:
```

```
 *  BOOL            TRUE if the gizmo was previously checked, FALSE
 *                  otherwise.
 */

BOOL GizmoPCheck(HWND hWnd, PGIZMO pGizmo, BOOL fCheck)
    {
    BOOL        fPrevCheck;
    PGIZMO      pCur;


    //Ignore command buttons.
    if (GIZMOTYPE_BUTTONCOMMAND==pGizmo->iType)
        return FALSE;

    //Get the previous state
    fPrevCheck=(BOOL)(BUTTONGROUP_DOWN & pGizmo->uState);


    //Simply set the state for inclusive attribute buttons.
    if (GIZMOTYPE_BUTTONATTRIBUTEIN==pGizmo->iType)
        {
        if (pGizmo->fDisabled)
            {
            GizmoPStateSet(hWnd, pGizmo
                , fCheck ? ATTRIBUTEBUTTON_DOWNDISABLED
                : ATTRIBUTEBUTTON_DISABLED);
            }
        else
            {
            GizmoPStateSet(hWnd, pGizmo, fCheck
                ? ATTRIBUTEBUTTON_DOWN : ATTRIBUTEBUTTON_UP);
            }
        }


    if (GIZMOTYPE_BUTTONATTRIBUTEEX==pGizmo->iType)
        {
        //We cannot uncheck an exclusive attribute
        if (!fCheck)
            return fPrevCheck;

        /*
         * For exclusive buttons we have to do more work.  First, if
         * we're already checked (incliding DOWN and MOUSEDOWN) then
         * we set DOWN and exit.  If we're not already checked, then
         * we look for the gizmo around us, backwards and forwards,
         * that is checked and uncheck him.
         */

        //Search  backwards.
        pCur=pGizmo->pPrev;

        while (NULL!=pCur)
            {
            //Stop at any non-exclusive attribute.
            if (GIZMOTYPE_BUTTONATTRIBUTEEX!=pCur->iType)
                {
                pCur=NULL;
                break;
                }

            //If it's down, set it up and we've finished.
            if (BUTTONGROUP_DOWN & pCur->uState)
                break;

            pCur=pCur->pPrev;
            }


        //If we didn't find a previous one, pCur is NULL, look ahead.
        if (NULL==pCur)
            {
            pCur=pGizmo->pNext;

            while (NULL!=pCur)
                {
                //Stop at any non-exclusive attribute.
                if (GIZMOTYPE_BUTTONATTRIBUTEEX!=pCur->iType)
                    {
                    pCur=NULL;
                    break;
                    }

                //If it's down, set it up and we've finished.
                if (BUTTONGROUP_DOWN & pCur->uState)
                    break;

                pCur=pCur->pNext;
                }
            }

        //If pCur is non-NULL, we found a neighbor, so uncheck it
        if (NULL!=pCur)
            {
            GizmoPStateSet(hWnd, pCur, (pGizmo->fDisabled)
                ? ATTRIBUTEBUTTON_DISABLED : ATTRIBUTEBUTTON_UP);
            }

        //Always set ourselves down
        GizmoPStateSet(hWnd, pGizmo, (pGizmo->fDisabled)
            ? ATTRIBUTEBUTTON_DOWNDISABLED : ATTRIBUTEBUTTON_DOWN);
        }

    return fPrevCheck;
    }
```

```
/*
 * GenericSubProc
 *
 * Purpose:
 *  Subclasses window controls in Gizmos so we can trap the tab key
 *  and tab to the next control.  We can have one shared generic
 *  subclass procedure because we save the type index for this
 *  control in the property "iType."  This allows us to look up the
 *  original procedure in the pfnOrg array.
 *
 * Parameters:
 *  Standard
 *
 * Return Value:
 *  Standard
 */

LRESULT APIENTRY GenericSubProc(HWND hWnd, UINT iMsg
    , WPARAM wParam, LPARAM lParam)
    {
    LONG        lRet;
    RECT        rc;
    RECT        rcE;
    HWND        hWndE;
    HBRUSH      hBr;
    HDC         hDC;
    UINT        dx;
    UINT        iType, i;

    i=(int)GetProp(hWnd, SZTYPEPROP);
    iType=POSITIONBIT(i);

    //Special:  paint the gap in drop-down comboboxes.
    if (GIZMOTYPE_COMBOBOX==iType && WM_PAINT==iMsg)
        {
        //Do default painting.
        lRet=(*pfnOrg[i])(hWnd, iMsg, wParam, lParam);

        hWndE=GetDlgItem(hWnd, ID_COMBOEDIT);

        GetClientRect(hWnd, &rc);
        GetClientRect(hWndE, &rcE);

        //The width of the button is the scroll bar width.
        dx=GetSystemMetrics(SM_CXVSCROLL);

        //Calculate the rectangle
        rc.right -=dx;
        rc.left   =rcE.right;
        rc.bottom+=1;

        //Paint the gap
        hDC=GetDC(hWnd);   //Already did BeginPaint and EndPaint
        hBr=CreateSolidBrush(GetSysColor(COLOR_BTNFACE));
        FillRect(hDC, &rc, hBr);
        DeleteObject(hBr);

        ReleaseDC(hWnd, hDC);
        return lRet;
        }

    //Control tabbing to the next or previous control
    if (WM_KEYDOWN==iMsg && VK_TAB==wParam)
        {
        hWndE=hWnd;

        if (-1==i)
            hWndE=GetParent(hWnd);

        hWndE=GetNextDlgTabItem(GetParent(hWndE), hWnd
            , (BOOL)(GetKeyState(VK_SHIFT)));
        SetFocus(hWndE);
        return 0L;
        }

    if (-1==i) i=0;

    //Eat tab chars in edit controls to prevent beeping.
    if (0==i && WM_CHAR==iMsg && VK_TAB==wParam)
        return 0L;


    //Do this or edit controls bomb big-time.
    return CallWindowProc(pfnOrg[i], hWnd, iMsg, wParam, lParam);
    }
```

GIZMOBAR.DEF (GIZMOBAR Sample)

```
LIBRARY     GIZMOBAR
DESCRIPTION 'GizmoBar 1.00, By Kraig Brockschmidt, (c)1993 Microsoft
Corporation'
CODE        EXECUTE READ SHARED
EXPORTS     GizmoBarWndProc         @2

            GBHwndAssociateSet      @3
            GBHwndAssociateGet      @4

            GBGizmoAddA             @5
            GBGizmoAddW             @12
            GBGizmoRemove           @6

            GBGizmoSendMessage      @7

            GBGizmoShow             @8
            GBGizmoEnable           @9
            GBGizmoCheck            @10
            GBGizmoFocusSet         @11

            GBGizmoTypeGet          @13

            GBGizmoDataSet          @14
            GBGizmoDataGet          @15
            GBGizmoNotifySet        @16
            GBGizmoNotifyGet        @17

            GBGizmoTextGetA         @18
            GBGizmoTextGetW         @19
            GBGizmoTextSetA         @20
            GBGizmoTextSetW         @21
            GBGizmoIntGet           @22
            GBGizmoIntSet           @23
            GBGizmoExist            @24
```

GIZMOBAR.RC (GIZMOBAR Sample)

```
/*
 * GIZMOBAR.RC
 * GizmoBar Version 1.01
 *
 * Strings and other resources for the GizmoBar.
 *
 * Copyright (c)1993-1994 Microsoft Corporation, All Rights Reserved
 *
 * Kraig Brockschmidt, Software Design Engineer
 * Microsoft Systems Developer Relations
 *
 * Internet  :  kraigb@microsoft.com
 * Compuserve:  >INTERNET:kraigb@microsoft.com
 */


#include <windows.h>
#include "gizmoint.h"

rcinclude gizmobar.rcv
```

GIZMOBAR.RCV   (GIZMOBAR Sample)

```
/*
 * GIZMOBAR.RCV
 * GizmoBar Version 1.01
 *
 * Version resource file for the GizmoBar Control DLL
 *
 * Copyright (c)1993-1994 Microsoft Corporation, All Rights Reserved
 *
 * Kraig Brockschmidt, Software Design Engineer
 * Microsoft Systems Developer Relations
 *
 * Internet  :  kraigb@microsoft.com
 * Compuserve:  >INTERNET:kraigb@microsoft.com
 */

#ifdef WIN32
#include <winver.h>
#else
#include <ver.h>
#endif

//Default is nodebug
#ifndef DEBUG
#define VER_DEBUG                   0
#else
#define VER_DEBUG                   VS_FF_DEBUG
#endif

VS_VERSION_INFO VERSIONINFO
 FILEVERSION        1,0,1,0
 PRODUCTVERSION     1,0,1,0
 FILEFLAGSMASK      VS_FFI_FILEFLAGSMASK
 FILEFLAGS          VER_DEBUG
#ifdef WIN32
 FILEOS             VOS_NT_WINDOWS32
#else
 FILEOS             VOS_DOS_WINDOWS16
#endif
 FILETYPE           VFT_DLL
 FILESUBTYPE        VFT_UNKNOWN

 BEGIN
   BLOCK "StringFileInfo"
    BEGIN
     BLOCK "040904E4"
      BEGIN
       VALUE "CompanyName",     "Microsoft Corporation\0", "\0"
       VALUE "FileDescription", "Kraig Brockschmidt's GizmoBar", "\0"
       VALUE "FileVersion",     "1.01\0", "\0"
       VALUE "InternalName",    "GIZMOBAR.DLL", "\0"
       VALUE "LegalCopyright",  "Copyright \2511993-1994osoft Corp.", "\0"
       VALUE "OriginalFilename","GIZMOBAR.DLL", "\0"
       VALUE "ProductName",     "Kraig Brockschmidt's GizmoBar", "\0"
       VALUE "ProductVersion",  "1.01\0"
      END
   END

   BLOCK "VarFileInfo"
    BEGIN
     VALUE "Translation", 0x0409, 0x04E4
    END
 END
```

_

GIZMOBAR.C   (GIZMOBAR Sample)

```
/*
 * GIZMOBAR.C
 * GizmoBar Version 1.01
 *
 * Contains the main window procedure of the GizmoBar control
 * that handles mouse logic and Windows messages.
 *
 * Copyright (c)1993-1994 Microsoft Corporation, All Rights Reserved
 *
 * Kraig Brockschmidt, Software Design Engineer
 * Microsoft Systems Developer Relations
 *
 * Internet  :  kraigb@microsoft.com
 * Compuserve:  >INTERNET:kraigb@microsoft.com
 */


#include <windows.h>
#include "gizmoint.h"



/*
 * GizmoBarWndProc
 *
 * Purpose:
 *  Window Procedure for the GizmoBar custom control.  Handles all
 *  messages like WM_PAINT just as a normal application window would.
 *  Any message not processed here should go to DefWindowProc.
 *
 * Parameters:
 *  Standard
 *
 * Return Value:
 *  Standard
 */

LRESULT WINAPI GizmoBarWndProc(HWND hWnd, UINT iMsg
    , WPARAM wParam, LPARAM lParam)
    {
    BOOL            fSuccess;
    BOOL            fTemp;
    LPCREATESTRUCT  pCreate;
    PGIZMOBAR       pGB;
    PGIZMO          pGizmo;
    RECT            rc;
    POINT           pt;
    short           x, y;
    COLORREF        cr;

    COMMANDPARAMS(wID, wCode, hWndMsg);
```

```
pGB=(PGIZMOBAR)GetWindowLong(hWnd, GBWL_STRUCTURE);

//Pass control messages onto another function for processing.
if (iMsg >= WM_USER)
    return GBMessageHandler(hWnd, iMsg, wParam, lParam, pGB);

switch (iMsg)
    {
    case WM_NCCREATE:
        pCreate=(LPCREATESTRUCT)lParam;

        pGB=GizmoBarPAllocate(&fSuccess, hWnd
            , pCreate->hInstance, pCreate->hwndParent
            , pCreate->style, 0, (UINT)pCreate->hMenu);

        if (!fSuccess)
            {
            GizmoBarPFree(pGB);
            return -1L;
            }
        else
            SetWindowLong(hWnd, GBWL_STRUCTURE, (LONG)pGB);

        return DefWindowProc(hWnd, iMsg, wParam, lParam);


    case WM_DESTROY:
        /*
         * We want to clean up before DestroyWindow nukes all the
         * children, so WM_DESTROY is a better to do it than
         * WM_NCDESTROY.
         */
        GizmoBarPFree(pGB);
        break;


    case WM_ERASEBKGND:
        /*
         * Eat this message to avoid erasing portions that
         * we are going to repaint in WM_PAINT.  Part of a
         * change-state-and-repaint strategy is to rely on
         * WM_PAINT to do anything visual, which includes
         * erasing invalid portions.  Letting WM_ERASEBKGND
         * erase the background is redundant.
         */
        return TRUE;

   #ifdef WIN32
    case WM_CTLCOLORBTN:
    case WM_CTLCOLORSTATIC:
        fTemp=TRUE;
   #else
    case WM_CTLCOLOR:
        //Change the color of static text on the GizmoBar.
        fTemp=(HIWORD(lParam)==CTLCOLOR_STATIC
            || HIWORD(lParam)==CTLCOLOR_BTN);
   #endif

        if (fTemp)
            {
            cr=GetSysColor(COLOR_BTNFACE);
            SetTextColor((HDC)wParam
                , GetSysColor(COLOR_BTNTEXT));
            SetBkColor((HDC)wParam, cr);

            /*
             * If the system colors have changed, then crFace
             * will not be equal to COLOR_BTNFACE, so we
             * reinitialize the background brush.  This scheme
             * handles system color changes appropriately
             * without processing WM_WININICHANGE and without
             * blindly creating a new brush on every WM_CTLCOLOR
             * message.
             */
            if (cr!=pGB->crFace)
                {
                pGB->crFace=cr;

                if (NULL!=pGB->hBrFace)
                    DeleteObject(pGB->hBrFace);

                pGB->hBrFace=CreateSolidBrush(pGB->crFace);
                }

            return (LONG)(UINT)pGB->hBrFace;
            }

        return DefWindowProc(hWnd, iMsg, wParam, lParam);


    case WM_PAINT:
        GizmoBarPaint(hWnd, pGB);
        break;


    case WM_SETFONT:
        /*
         * wParam has the new font that we now send to all other
         * windows controls in us.  We control repaints here to
         * prevent a lot of repainting for each control.
         */
        DefWindowProc(hWnd, WM_SETREDRAW, FALSE, 0L);

        if ((WPARAM)NULL!=wParam)
            {
            pGB->hFont=(HFONT)wParam;
            GizmoPEnum(&pGB->pGizmos, FEnumChangeFont
                , (DWORD)(LPTSTR)pGB);

            DefWindowProc(hWnd, WM_SETREDRAW, TRUE, 0L);
```

```
            InvalidateRect(hWnd, NULL, FALSE);
            UpdateWindow(hWnd);
            }

        break;


    case WM_GETFONT:
        return (LRESULT)(UINT)pGB->hFont;


    case WM_ENABLE:
        /*
         * wParam has the new enable flag that we use to enable
         * or disable ALL controls in us at one time.  We also
         * turn the redraw off to prevent a lot of flicker.
         */
        DefWindowProc(hWnd, WM_SETREDRAW, FALSE, 0L);

        pGB->fEnabled=(BOOL)wParam;
        GizmoPEnum(&pGB->pGizmos, FEnumEnable
            , (DWORD)(LPTSTR)pGB);

        DefWindowProc(hWnd, WM_SETREDRAW, TRUE, 0L);
        InvalidateRect(hWnd, NULL, FALSE);
        UpdateWindow(hWnd);
        break;


    case WM_CANCELMODE:
        pGizmo=pGB->pGizmoTrack;

        pGB->fTracking=FALSE;
        pGB->fMouseOut=FALSE;

        if (NULL!=pGizmo)
            GizmoPStateSet(hWnd, pGizmo, COMMANDBUTTON_UP);

        ReleaseCapture();
        break;


    case WM_LBUTTONDBLCLK:
    case WM_LBUTTONDOWN:
        //Get the mouse coordinates.
        x=LOWORD(lParam);
        y=HIWORD(lParam);


        /*
         * See if we hit a command or attribute gizmo or not.
         * Anything else that is a control will get the message
         * instead of us anyway, so we don't have to check.
         * FEnumHitTest also validates drawn gizmos, enabled, and
         * visible, so we don't.
```

```
             */
            pGizmo=GizmoPEnum(&pGB->pGizmos, FEnumHitTest, lParam);

            if (NULL==pGizmo)
                break;          //Didn't hit one matching our needs.

            /*
             * Inform the associate that a command was hit like a
             * menu item.
             */
            if (NULL!=pGB->hWndAssociate)
                {
                if (pGizmo->fNotify)
                    {
                    SendMenuSelect(pGB->hWndAssociate, pGizmo->uID
                        , 0, 0);
                    }
                }

            /*
             * We hit a button.  If it's a command or attribute,
             * then change the state and start tracking.
             */
            pGB->fTracking=TRUE;
            pGB->pGizmoTrack=pGizmo;
            pGB->fMouseOut=FALSE;
            SetCapture(hWnd);

            pGizmo->uStateOrg=pGizmo->uState;
            GizmoPStateSet(hWnd, pGizmo, ATTRIBUTEBUTTON_MOUSEDOWN);

            break;


        case WM_MOUSEMOVE:
            POINTFROMLPARAM(pt, lParam);

            if (!pGB->fTracking)
                break;

            pGizmo=pGB->pGizmoTrack;
            SetRect(&rc, pGizmo->x, pGizmo->y, pGizmo->x+pGizmo->dx
                , pGizmo->y+pGizmo->dy);

            fTemp=pGB->fMouseOut;
            pGB->fMouseOut=!PtInRect(&rc, pt);

            //If the mouse went out, change state to the original.
            if (!fTemp && pGB->fMouseOut)
                {
                GizmoPStateSet(hWnd, pGizmo, pGizmo->uStateOrg);

                if (NULL!=pGB->hWndAssociate)
                    {
                    //Notify that we left the button
                    if (pGizmo->fNotify)
                        SendMenuSelect(pGB->hWndAssociate, 0, ~0, 0);
                    }
                }

            if (fTemp && !pGB->fMouseOut)
                {
                GizmoPStateSet(hWnd, pGizmo
                    , ATTRIBUTEBUTTON_MOUSEDOWN);

                if (NULL!=pGB->hWndAssociate)
                    {
                    //Notify that we pressed down again
                    if (pGizmo->fNotify)
                        {
                        SendMenuSelect(pGB->hWndAssociate
                            , pGizmo->uID , 0, 0);
                        }
                    }
                }

            break;


        case WM_LBUTTONUP:
            if (!pGB->fTracking)
                break;

            pGB->fTracking=FALSE;
            pGizmo=pGB->pGizmoTrack;
            ReleaseCapture();


            /*
             * Repaint if we were actually below the mouse when this
             * occurred.  For command buttons, pop the button up.
             * For attributes, either toggle the state (inclusive
             * buttons) or check the selected one (exclusive buttons)
             */

            if (!pGB->fMouseOut)
                {
                //Command buttons always come up.
                if (GIZMOTYPE_BUTTONCOMMAND==pGizmo->iType)
                    GizmoPStateSet(hWnd, pGizmo, COMMANDBUTTON_UP);

                //Attribute inclusive buttons toggle
                if (GIZMOTYPE_BUTTONATTRIBUTEIN==pGizmo->iType)
                    {
                    GizmoPCheck(hWnd, pGizmo, !(BUTTONGROUP_DOWN
                        & pGizmo->uStateOrg));
                    }

                //Attribure exclusive buttons are always checked.
                if (GIZMOTYPE_BUTTONATTRIBUTEEX==pGizmo->iType)
```

```
                    GizmoPCheck(hWnd, pGizmo, TRUE);

                //Only send messages if notify is ON.
                if (NULL!=pGB->hWndAssociate && pGizmo->fNotify)
                    {
                    SendMenuSelect(pGB->hWndAssociate, 0, ~0, 0);
                    SendCommand(pGB->hWndAssociate, pGizmo->uID
                        , BN_CLICKED, hWnd);
                    }
                }

            break;


        case WM_COMMAND:
            //Pass control messages on if the gizmo's notify is ON.
            if (NULL!=pGB->hWndAssociate)
                {
                pGizmo=PGizmoFromHwndID(hWnd, wID);

                if (NULL!=pGizmo)
                    {
                    if (pGizmo->fNotify)
                        {
                        SendMessage(pGB->hWndAssociate, iMsg, wParam
                            , lParam);
                        }
                    }
                }
            break;

        default:
            return DefWindowProc(hWnd, iMsg, wParam, lParam);
        }

    return 0L;
    }




/*
 * FEnumChangeFont
 *
 * Purpose:
 *  Enumeration callback for all the gizmos we know about in order to
 *  send a new font to them that's stored in PGIZMOBAR in dw.
 *
 * Parameters:
 *  pGizmo          PGIZMO to draw.
 *  iGizmo          UINT index on the GizmoBar of this gizmo.
 *  dw              DWORD extra data passed to GizmoPEnum, in our
 *                  case the GizmoBar's pGB.
 *
 * Return Value:
 *  BOOL            TRUE to continue the enumeration, FALSE otherwise.
 */

BOOL WINAPI FEnumChangeFont(PGIZMO pGizmo, UINT iGizmo, DWORD dw)
    {
    PGIZMOBAR   pGB=(PGIZMOBAR)dw;

    //Only need to change fonts in windowed controls using WM_SETFONT
    if (NULL!=pGizmo->hWnd)
        {
        SendMessage(pGizmo->hWnd, WM_SETFONT
            , (WPARAM)pGB->hFont, 1L);
        }

    return TRUE;
    }





/*
 * FEnumEnable
 *
 * Purpose:
 *  Enumeration callback for all the gizmos we know about in order to
 *  enable or disable them from the WM_ENABLE message.
 *
 * Parameters:
 *  pGizmo          PGIZMO to draw.
 *  iGizmo          UINT index on the GizmoBar of this gizmo.
 *  dw              DWORD extra data passed to GizmoPEnum, in our
 *                  case the GizmoBar's pGB.
 *
 * Return Value:
 *  BOOL            TRUE to continue the enumeration, FALSE
 *                  otherwise.
 */

BOOL WINAPI FEnumEnable(PGIZMO pGizmo, UINT iGizmo, DWORD dw)
    {
    PGIZMOBAR   pGB=(PGIZMOBAR)dw;
    BOOL        fEnable=pGB->fEnabled;

    //NOTE:  This code is duplicated in GBGizmoEnable in API.C
    if (NULL!=pGizmo->hWnd)
        EnableWindow(pGizmo->hWnd, fEnable);
    else
        {
        //If we're not down, command and attribute buttons act same.
        if (!(BUTTONGROUP_DOWN & pGizmo->uState))
            {
            GizmoPStateSet(pGB->hWnd, pGizmo, fEnable
                ? COMMANDBUTTON_UP : COMMANDBUTTON_DISABLED);
            }
        else
            {
            /*
             * Attribute buttons are a little more sensitive
             * with DOWNDISABLED
             */
            GizmoPStateSet(pGB->hWnd, pGizmo, fEnable
                ? ATTRIBUTEBUTTON_DOWN
                : ATTRIBUTEBUTTON_DOWNDISABLED);
            }
        }

    return TRUE;
    }
```

```
/*
 * FEnumHitTest
 *
 * Purpose:
 *  Enumeration callback for all the gizmos we know about in order to
 *  hit-test them.
 *
 * Parameters:
 *  pGizmo          PGIZMO to draw.
 *  iGizmo          UINT index on the GizmoBar of this gizmo.
 *  dw              DWORD extra data passed to GizmoPEnum, in our
 *                  case the hDC on which to draw.
 *
 * Return Value:
 *  BOOL            TRUE to continue the enumeration, FALSE
 *                  otherwise.
 */

BOOL WINAPI FEnumHitTest(PGIZMO pGizmo, UINT iGizmo, DWORD dw)
    {
    RECT  rc;
    POINT pt;

    POINTFROMLPARAM(pt, dw);

    /*
     * Hit tests have to happen on visible, enabled, and drawn
     * controls only.
     */
    if (GIZMOTYPE_DRAWN & pGizmo->iType && !pGizmo->fHidden
        && !(BUTTONGROUP_DISABLED & pGizmo->uState))
        {
```

```
    SetRect(&rc, pGizmo->x, pGizmo->y
        , pGizmo->x+pGizmo->dx, pGizmo->y+pGizmo->dy);

    //Stop enumeration if we have a hit.
    return !PtInRect(&rc, pt);
    }

return TRUE;
}
```

GIZMOINT.H (GIZMOBAR Sample)

```
/*
 * GIZMOINT.H
 * GizmoBar Version 1.01
 *
 * Internal definitions for the GizmoBar DLL
 *
 * Copyright (c)1993-1994 Microsoft Corporation, All Rights Reserved
 *
 * Kraig Brockschmidt, Software Design Engineer
 * Microsoft Systems Developer Relations
 *
 * Internet  :  kraigb@microsoft.com
 * Compuserve:  >INTERNET:kraigb@microsoft.com
 */


#ifndef _GIZMOINT_H_
#define _GIZMOINT_H_

#include "gizmobar.h"
#include "win1632.h"
#include "gizmo.h"

#ifdef __cplusplus
extern "C"
    {
#endif


/*
 * The main gizmobar structure itself.  There's only one of these,
 * but it references the first GIZMO in the list.
 */

typedef struct tagGIZMOBAR
    {
    PGIZMO      pGizmos;            //List of gizmos we own.
    HWND        hWnd;               //Window handle of ourselves.
    HINSTANCE   hInst;
    HWND        hWndAssociate;      //Associate who gets messages.
    DWORD       dwStyle;            //Copy of GWL_STYLE
    UINT        uState;             //State flags
    UINT        uID;                //Control ID.

    HBRUSH      hBrFace;            //Static control background color
    COLORREF    crFace;             //Color of hBrFace
    HFONT       hFont;              //Font in use
    BOOL        fEnabled;           //Are we enabled?

    PGIZMO      pGizmoTrack;        //Current pressed button.
    BOOL        fTracking;
    BOOL        fMouseOut;
    } GIZMOBAR, * PGIZMOBAR;

#define CBGIZMOBAR sizeof(GIZMOBAR)


//Extra bytes for the window if the size of a local handle.
#define CBWINDOWEXTRA       sizeof(PGIZMOBAR)

#define GBWL_STRUCTURE      0


//Structure for passing paint info to a gizmo enumeration callback.
typedef struct
    {
    HDC     hDC;
    BOOL    fPaint;
    } PAINTGIZMO, * PPAINTGIZMO;



//Private functions specific to the control.

//INIT.C
BOOL                FRegisterControl(HINSTANCE);
PGIZMOBAR           GizmoBarPAllocate(int *, HWND, HINSTANCE, HWND
                        , DWORD, UINT, UINT);
PGIZMOBAR           GizmoBarPFree(PGIZMOBAR);


//PAINT.C
void                GizmoBarPaint(HWND, PGIZMOBAR);
BOOL WINAPI         FEnumPaintGizmos(PGIZMO, UINT, DWORD);


//GIZMOBAR.C
LRESULT WINAPI    GizmoBarWndProc(HWND, UINT, WPARAM, LPARAM);
BOOL    WINAPI    FEnumChangeFont(PGIZMO, UINT, DWORD);
BOOL    WINAPI    FEnumEnable(PGIZMO, UINT, DWORD);
BOOL    WINAPI    FEnumHitTest(PGIZMO, UINT, DWORD);


//API.C  Also see GIZMOBAR.H for others
LRESULT    GBMessageHandler(HWND, UINT, WPARAM, LPARAM, PGIZMOBAR);
PGIZMO     PGizmoFromHwndID(HWND, UINT);


#endif //_GIZMOINT_H_
```

INIT.C (GIZMOBAR Sample)

```
/*
 * INIT.C
 * GizmoBar Version 1.01
 *
 * LibMain entry point and initialization code for the GizmoBar
 * DLL that is likely to be used once or very infrequently.
 *
 * Copyright (c)1993-1994 Microsoft Corporation, All Rights Reserved
 *
 * Kraig Brockschmidt, Software Design Engineer
 * Microsoft Systems Developer Relations
 *
 * Internet  :  kraigb@microsoft.com
 * Compuserve:  >INTERNET:kraigb@microsoft.com
 */


#include <windows.h>
#include "gizmoint.h"


/*
 * LibMain(32)
 *
 * Purpose:
 *  Entry point conditionally compiled for Windows NT and Windows
 *  3.1.  Provides the proper structure for each environment.
 */

#ifdef WIN32

BOOL WINAPI DllMain(HINSTANCE hInstance, ULONG ulReason
    , PCONTEXT pContext)
    {
    UNREFERENCED_PARAMETER(pContext);

    if (DLL_PROCESS_ATTACH==ulReason)
        return FRegisterControl(hInstance);

    return TRUE;
    }

#else

int WINAPI LibMain(HANDLE hInstance, WORD wDataSeg
    , WORD cbHeapSize, LPTSTR lpCmdLine)
    {
     //Perform global initialization.
    if (FRegisterControl(hInstance))
        {
        if (0!=cbHeapSize)
            UnlockData(0);
        }

    return (int)hInstance;
    }

#endif




/*
 * FRegisterControl
 *
 * Purpose:
 *  Registers the GizmoBar control class, including CS_GLOBALCLASS
 *  to make the control available to all applications in the system.
 *
 * Parameters:
 *  hInst           HINSTANCE of the DLL that will own this class.
 *
 * Return Value:
 *  BOOL            TRUE if the class is registered, FALSE otherwise.
 */

BOOL FRegisterControl(HINSTANCE hInst)
    {
    static BOOL     fRegistered=FALSE;
    WNDCLASS        wc;

    if (!fRegistered)
        {
        wc.lpfnWndProc   =GizmoBarWndProc;
        wc.cbClsExtra    =0;
        wc.cbWndExtra    =CBWINDOWEXTRA;
        wc.hInstance     =hInst;
        wc.hIcon         =NULL;
        wc.hCursor       =LoadCursor(NULL, IDC_ARROW);
        wc.hbrBackground =(HBRUSH)(COLOR_BTNFACE+1);
        wc.lpszMenuName  =NULL;
        wc.lpszClassName =CLASS_GIZMOBAR;
        wc.style         =CS_DBLCLKS | CS_GLOBALCLASS
                          | CS_VREDRAW | CS_HREDRAW;

        fRegistered=RegisterClass(&wc);
        }

    return fRegistered;
    }







/*
 * GizmoBarPAllocate
 *
 * Purpose:
 *  Allocates and initializes the control's primary data structure
 *  for each window that gets created.
 *
 * Parameters:
 *  pfSuccess       int * indicating success of the function.
 *  hWnd            HWND that is tied to this structure.
 *  hInst           HINSTANCE of the DLL.
 *  hWndAssociate   HWND to which we send messages.
 *  dwStyle         DWORD initial style.
 *  uState          UINT initial state.
 *  uID             UINT identifier for this window.
 *
 * Return Value:
 *  PGIZMOBAR       If NULL returned then GizmoBarPAllocate could not
 *                  allocate memory.  If a non-NULL pointer is
 *                  returned with *pfSuccess, then call GizmoBarPFree
 *                  immediately.  If you get a non-NULL pointer and
 *                  *pfSuccess==TRUE then the function succeeded.
 */

PGIZMOBAR GizmoBarPAllocate(int *pfSuccess, HWND hWnd
    , HINSTANCE hInst, HWND hWndAssociate, DWORD dwStyle
    , UINT uState , UINT uID)
    {
    PGIZMOBAR     pGB;

    if (NULL==pfSuccess)
        return NULL;

    *pfSuccess=FALSE;

    //Allocate the structure
    pGB=(PGIZMOBAR)(void *)LocalAlloc(LPTR, CBGIZMOBAR);

    if (NULL==pGB)
        return NULL;

    //Initialize LibMain parameter holders.
    pGB->hWnd         =hWnd;
    pGB->hInst        =hInst;
    pGB->hWndAssociate=hWndAssociate;
    pGB->dwStyle      =dwStyle;
    pGB->uState       =uState;
    pGB->uID          =uID;
    pGB->fEnabled     =TRUE;

    pGB->crFace=GetSysColor(COLOR_BTNFACE);
    pGB->hBrFace=CreateSolidBrush(pGB->crFace);

    if (NULL==pGB->hBrFace)
        return pGB;

    pGB->hFont=GetStockObject(SYSTEM_FONT);
    *pfSuccess=TRUE;
    return pGB;
    }




/*
 * GizmoBarPFree
 *
 * Purpose:
 *  Reverses all initialization done by GizmoBarPAllocate, cleaning
 *  up any allocations including the application structure itself.
 *
 * Parameters:
 *  pGB             PGIZMOBAR to the control's structure
 *
 * Return Value:
 *  PGIZMOBAR       NULL if successful, pGB if not, meaning we
 *                  couldn't free some allocation.
 */

PGIZMOBAR GizmoBarPFree(PGIZMOBAR pGB)
    {
    if (NULL==pGB)
        return NULL;

    /*
     * Free all the gizmos we own.  When we call GizmoPFree we always
     * free the first one in the list which updates pGB->pGizmos for
     * us, so we just have to keep going until pGizmos is NULL,
     * meaning we're at the end of the list.
     */
    while (NULL!=pGB->pGizmos)
        GizmoPFree(&pGB->pGizmos, pGB->pGizmos);

    if (NULL!=pGB->hBrFace)
        DeleteObject(pGB->hBrFace);

    /*
     * Notice that since we never create a font, we aren't
     * responsible for our hFont member.
     */

    return (PGIZMOBAR)(void *)LocalFree((HLOCAL)(UINT)(LONG)pGB);
    }
```

PAINT.C (GIZMOBAR Sample)

```
/*
 * PAINT.C
 * GizmoBar Version 1.01
 *
 * Contains any code related to GizmoBar visuals, primarily
 * the WM_PAINT handler.
 *
 * Copyright (c)1993-1994 Microsoft Corporation, All Rights Reserved
 *
 * Kraig Brockschmidt, Software Design Engineer
 * Microsoft Systems Developer Relations
 *
 * Internet  :  kraigb@microsoft.com
 * Compuserve:  >INTERNET:kraigb@microsoft.com
 */


#include <windows.h>
#include "gizmoint.h"


//In GIZMO.C
extern TOOLDISPLAYDATA tdd;


/*
 * GizmoBarPaint
 *
 * Purpose:
 *  Handles all WM_PAINT messages for the control and paints either
 *  the entire thing or just one GizmoBar button if pGB->pGizmoPaint
 *  is non-NULL.
 *
 * Parameters:
 *  hWnd            HWND Handle to the control.
 *  pGB             PGIZMOBAR control data pointer.
 *
 * Return Value:
 *  None
 */

void GizmoBarPaint(HWND hWnd, PGIZMOBAR pGB)
    {
    PAINTSTRUCT ps;
    RECT        rc;
    HDC         hDC;
    HBRUSH      hBr=NULL;
    HPEN        hPen=NULL;


    hDC=BeginPaint(hWnd, &ps);
    GetClientRect(hWnd, &rc);
    /*
     * The only part of the frame we need to draw is the bottom line,
     * so we inflate the rectangle such that all other parts are
     * outside the visible region.
     */
    hBr =CreateSolidBrush(GetSysColor(COLOR_BTNFACE));

    if (NULL!=hBr)
        SelectObject(hDC, hBr);

    hPen=CreatePen(PS_SOLID, 1, GetSysColor(COLOR_WINDOWFRAME));

    if (NULL!=hPen)
        SelectObject(hDC, hPen);

    Rectangle(hDC, rc.left-1, rc.top-1, rc.right+1, rc.bottom);


    /*
     * All that we have to do to draw the controls is start through
     * the list, ignoring anything but buttons, and calling BTTNCUR's
     * UIToolButtonDraw for buttons.  Since we don't even have to
     * track positions of things, we can just use an enum.
     */
    GizmoPEnum(&pGB->pGizmos, FEnumPaintGizmos, (DWORD)(LPTSTR)&ps);

    //Clean up
    EndPaint(hWnd, &ps);

    if (NULL!=hBr)
        DeleteObject(hBr);

    if (NULL!=hPen)
        DeleteObject(hPen);

    return;
    }





/*
 * FEnumPaintGizmos
 *
 * Purpose:
 *  Enumeration callback for all the gizmos we know about in order to
 *  draw them.
 *
 * Parameters:
 *  pGizmo          PGIZMO to draw.
 *  iGizmo          UINT index on the GizmoBar of this gizmo.
 *  dw              DWORD extra data passed to GizmoPEnum, in our
 *                  case a pointer to the PAINTSTRUCT.
 *
 * Return Value:
 *  BOOL            TRUE to continue the enumeration, FALSE
 *                  otherwise.
 */

BOOL WINAPI FEnumPaintGizmos(PGIZMO pGizmo, UINT iGizmo, DWORD dw)
    {
    LPPAINTSTRUCT   pps=(LPPAINTSTRUCT)dw;
    RECT            rc, rcI;

    //Only draw those marked for repaint.
    if ((GIZMOTYPE_DRAWN & pGizmo->iType))
        {
        SetRect(&rc, pGizmo->x, pGizmo->y
            , pGizmo->x+pGizmo->dx, pGizmo->y+pGizmo->dy);

        //Only draw gizmos in the repaint area
        if (IntersectRect(&rcI, &rc, &pps->rcPaint))
            {
            UIToolButtonDrawTDD(pps->hdc, pGizmo->x, pGizmo->y
                , pGizmo->dx, pGizmo->dy, pGizmo->hBmp
                , pGizmo->cxImage, pGizmo->cyImage, pGizmo->iBmp
                , (UINT)pGizmo->uState, &tdd);
            }
        }

    return TRUE;
    }
```

WIN1632.H (GIZMOBAR Sample)

```
/*
 * BOOK1632.H
 *
 * Macros and other definitions that assist in porting between Win16
 * and Win32 applications.  Defines WIN32 to enable 32-bit versions.
 *
 * Copyright (c)1993-1994 Microsoft Corporation, All Rights Reserved
 *
 * Kraig Brockschmidt, Software Design Engineer
 * Microsoft Systems Developer Relations
 *
 * Internet  :  kraigb@microsoft.com
 * Compuserve:  INTERNET>kraigb@microsoft.com
 */


#ifndef _BOOK1632_H_
#define _BOOK1632_H_

#ifdef WIN32

#ifndef COMMANDPARAMS
#define COMMANDPARAMS(wID, wCode, hWndMsg)                          \
    WORD        wID     = LOWORD(wParam);                           \
    WORD        wCode   = HIWORD(wParam);                           \
    HWND        hWndMsg = (HWND)(UINT)lParam;
#endif  //COMMANDPARAMS

#ifndef SendCommand
#define SendCommand(hWnd, wID, wCode, hControl)                     \
            SendMessage(hWnd, WM_COMMAND, MAKEWPARAM(wID, wCode)    \
                        , (LPARAM)hControl)
#endif  //SendCommand

#ifndef MENUSELECTPARAMS
#define MENUSELECTPARAMS(wID, wFlags, hMenu)                        \
    WORD        wID     = LOWORD(wParam);                           \
    WORD        wFlags  = HIWORD(wParam);                           \
    HMENU       hMenu   = (HMENU)lParam;
#endif  //MENUSELECTPARAMS


#ifndef SendMenuSelect
#define SendMenuSelect(hWnd, wID, wFlags, hMenu)                    \
            SendMessage(hWnd, WM_MENUSELECT                         \
                , MAKEWPARAM((WORD)wID, (WORD)wFlags), (LPARAM)hMenu)

#endif  //SendMenuSelect

#ifndef SendScrollPosition
#define SendScrollPosition(hWnd, iMsg, iPos)                        \
            SendMessage(hWnd, iMsg, MAKEWPARAM(SB_THUMBPOSITION, iPos), 0)
```

```
#endif //SendScrollPosition

#ifndef ScrollThumbPosition
#define ScrollThumbPosition(w, l) HIWORD(w)
#endif //ScrollThumbPosition

#ifndef GETWINDOWINSTANCE
#define GETWINDOWINSTANCE(h) (HINSTANCE)GetWindowLong(h, GWL_HINSTANCE)
#endif  //GETWINDOWINSTANCE

#ifndef GETWINDOWID
#define GETWINDOWID(h) (UINT)GetWindowLong(h, GWW_ID)
#endif  //GETWINDOWID

#ifndef POINTFROMLPARAM
#define POINTFROMLPARAM(p, l) {p.x=(LONG)(SHORT)LOWORD(l); \
                               p.y=(LONG)(SHORT)HIWORD(l);}
#endif  //POINTEFROMLPARAM

#ifndef MDIREFRESHMENU
#define MDIREFRESHMENU(h) SendMessage(h, WM_MDIREFRESHMENU, 0, 0L)
#endif  //MDIREFRESHMENU


//Activated child window in WM_MDIACTIVATE
#ifndef NEWMDIACTIVE
#define NEWMDIACTIVE lParam
#endif  //NEWMDIACTIVE

#ifndef UNICODETOANSI
#ifdef UNICODE
#define UNICODETOANSI(s, d, cch) WideCharToMultiByte(CP_ACP \
            , 0, s, -1, d, cch, NULL, NULL)
#else  //ANSI
#define UNICODETOANSI(s, d, cch) lstrcpy(d, (LPTSTR)s)
#endif
#endif //UNICODETOANSI


//****END WIN32



#else



//****START !WIN32

#ifndef POINTS
typedef POINT POINTS;
#endif  //POINTS

#ifndef LPPOINTS
typedef POINTS FAR *LPPOINTS;
```

```
#endif  //LPPOINTS

#ifndef MAKEPOINTS
#define MAKEPOINTS MAKEPOINT
#endif  //MAKEPOINTS


#ifndef COMMANDPARAMS
#define COMMANDPARAMS(wID, wCode, hWndMsg)                           \
    WORD        wID     = LOWORD(wParam);                            \
    WORD        wCode   = HIWORD(lParam);                            \
    HWND        hWndMsg = (HWND)(UINT)lParam;
#endif  //COMMANDPARAMS

#ifndef SendCommand
#define SendCommand(hWnd, wID, wCode, hControl)                      \
            SendMessage(hWnd, WM_COMMAND, wID                        \
                        , MAKELONG(hControl, wCode))
#endif  //SendCommand

#ifndef MENUSELECTPARAMS
#define MENUSELECTPARAMS(wID, wFlags, hMenu)                         \
    WORD        wID     = LOWORD(wParam);                            \
    WORD        wFlags  = LOWORD(lParam);                            \
    HMENU       hMenu   = (HMENU)HIWORD(lParam);
#endif  //MENUSELECTPARAMS

#ifndef SendMenuSelect
#define SendMenuSelect(hWnd, wID, wFlags, hMenu)                     \
            SendMessage(hWnd, WM_MENUSELECT, wID                     \
                        , MAKELONG((WORD)wFlags, (WORD)hMenu))
#endif  //SendMenuSelect

#ifndef SendScrollPosition
#define SendScrollPosition(hWnd, iMsg, iPos)                         \
            SendMessage(hWnd, iMsg, SB_THUMBPOSITION, MAKELONG(iPos, 0))
#endif //Send ScrollPosition

#ifndef ScrollThumbPosition
#define ScrollThumbPosition(w, l) LOWORD(l)
#endif //ScrollThumbPosition

#ifndef GETWINDOWINSTANCE
#define GETWINDOWINSTANCE(h) (HINSTANCE)GetWindowWord(h, GWW_HINSTANCE)
#endif  //GETWINDOWINSTANCE

#ifndef GETWINDOWID
#define GETWINDOWID(h) (UINT)GetWindowWord(h, GWW_ID)
#endif  //GETWINDOWID

#ifndef POINTFROMLPARAM
#define POINTFROMLPARAM(p, l) {p.x=LOWORD(l); p.y=HIWORD(l);}
#endif  //POINTEFROMLPARAM

#ifndef MDIREFRESHMENU
```

```
#define MDIREFRESHMENU(h) SendMessage(h, WM_MDISETMENU, TRUE, 0L)
#endif  //MDIREFRESHMENU

//Activated child window in WM_MDIACTIVATE
#ifndef NEWMDIACTIVE
#define NEWMDIACTIVE wParam
#endif  //NEWMDIACTIVE


//Things not present in Win3.1 SDK but present in Win32 SDK.

#ifndef APIENTRY
#define APIENTRY __export FAR PASCAL
#endif  //APIENTRY

#ifndef USHORT
typedef unsigned short USHORT;
#endif  //USHORT

//These are so we can write ANSI/UNICODE portable code.
#ifndef TCHAR
typedef char TCHAR;
#endif  //TCHAR

#ifndef LPTSTR
typedef TCHAR *LPTSTR;
#endif  //LPTSTR

#ifndef LPCTSTR
typedef const TCHAR *LPCTSTR;
#endif  //LPCTSTR

#ifndef _tcsncpy
#define _tcsncpy strncpy
#endif  //_tcsncpy

#ifndef lstrcpyA
#define lstrcpyA lstrcpy
#endif  //lstrcpyA

#ifndef lstrcmpiA
#define lstrcmpiA lstrcmpi
#endif  //lstrcmpiA

#ifndef TEXT
#define TEXT(a) a
#endif  //TEXT

#ifndef DeleteFile
#define DeleteFile(f)                       \
            {                               \
            OFSTRUCT    of;                 \
            OpenFile(f, &of, OF_DELETE);    \
            }
#endif  //DeleteFile
```

```
#ifndef UNICODETOANSI
#define UNICODETOANSI(s, d, cch) lstrcpy(d, (LPTSTR)s)
#endif  //UNICODETOANSI

#ifndef SetForegroundWindow
#define SetForegroundWindow(w) SetActiveWindow(w)
#endif  //SetForegroundWindow

#endif  //!WIN32


//These definitions we need regardless of Win16 or Win32

typedef struct
    {
    short   left;
    short   top;
    short   right;
    short   bottom;
    } RECTS, FAR *LPRECTS;

#define RECTSTORECT(rs, r) {(r).left=(rs).left;(r).top=(rs).top; \
    (r).right=(rs).right;(r).bottom=(rs).bottom;};

#define RECTTORECTS(r, rs) {(rs).left=(short)(r).left;    \
    (rs).top=(short)(r).top;(rs).right=(short)(r).right; \
    (rs).bottom=(short)(r).bottom;};

#endif  //_BOOK1632_H_
```

HELLO

------------------------------------- OLE Automation Sample Program: Hello -------------------------------------

This is the "hello world" sample application. It is an OLE object which is programmable via IDispatch.

The application has one window which contains a button and a text box. When the button is pressed, the string "Hello world." is displayed in the text box.
The hello application to the SayHello method by performing the same action as when the "Say Hello" button in the user interface is pressed. The SayHello method returns the same string.

This is the simplest OLE automation sample application and is a good place to start if you have not done any OLE Automation programming.

The ProgID for hello's only object is "Hello.Application". An instance of this object can be created by executing the following lines of code in Visual Basic or DispTest:

```
Sub Foo
  Dim MyCalculator as Object

  Set MyCalculator = CreateObject("Hello.Application")
  . . .
End Sub
```

---------------- Program Structure ---------------- Hello uses a type library and CreateStdDispatch in order to implement the IDispatch interface.

----------------------- Properties for the object -----------------------

Name Type Description ---
HelloMessage VT_BSTR the message that is printed when the
SayHello method is called or when the SayHello button is pressed.

--------------------------- Methods defined on the object ---------------------------

Name Description -- SayHello() as String
Print the hello message and return it.

MAKEFILE   (HELLO Sample)

```
####
#makefile - makefile for hello.exe
#
#       Copyright (C) 1992, Microsoft Corporation
#
#Purpose:
#  Builds the OLE 2.0 sample IDispatch server, hello.exe.
#
#
#  Usage: NMAKE                 ; build with defaults
#     or: NMAKE option          ; build with the given option(s)
#     or: NMAKE clean           ; erase all compiled files
#
#     option: dev = [win16 | win32]    ; dev=win32 is the default
#             DEBUG=[0|1]              ; DEBUG=1 is the default
#             HOST=[DOS | NT]          ; HOST=DOS (for win16)
#                                      ; HOST=NT (for win32)
#
#Notes:
#  This makefile assumes that the PATH, INCLUDE and LIB environment
#  variables are setup properly.
#
############################################################################
##


##########################################################################
#
# Default Settings
#

!if "$(dev)" == ""
dev = win32
!endif

!if !("$(dev)" == "win16" || "$(dev)" == "win32")
!error Invalid dev option, choose from [win16 | win32]
!endif

!if "$(dev)" == "win16"
TARGET  = WIN16
!if "$(HOST)" == ""
HOST  = DOS
!endif
!endif

!if "$(dev)" == "win32"
TARGET  = WIN32
HOST  = NT
!endif
```

```
!ifdef NODEBUG
DEBUG = 0
!endif

!if "$(DEBUG)" == "0"
NODEBUG = 1
!endif

!if "$(DEBUG)" == ""
DEBUG = 1
!endif


###########################################################################
#
# WIN16 Settings
#
!if "$(TARGET)" == "WIN16"

CC   = cl
LINK = link
!if "$(HOST)" == "DOS"
WX   = wx /w
!else
WX   =
!endif

RCFLAGS = -dWIN16
CFLAGS = -c -W3 -AM -GA -GEs -DWIN16
LINKFLAGS = /NOD /NOI /BATCH /ONERROR:NOEXE

LIBS = libw.lib mlibcew.lib

!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS) -Od -Zi -D_DEBUG $(CL)
LINKFLAGS = $(LINKFLAGS) /COD
!else
CFLAGS = $(CFLAGS) -Ox $(CL)
LINKFLAGS = $(LINKFLAGS) /FAR /PACKC
!endif
!endif


###########################################################################
#
# WIN32 Settings
#
!if "$(TARGET)" == "WIN32"


WX =

!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

CC = $(cc)
```

```
CFLAGS = $(cflags) $(cvarsmt) -DUNICODE $(cdebug)

!ifndef NODEBUG
CFLAGS = $(CFLAGS) -D_DEBUG
!endif

LINK = $(link)
LINKFLAGS = $(linkdebug) $(guilflags)
RCFLAGS = -DWIN32

!endif

##########################################################################
#
# Build rules
#

.cpp.obj:
    @echo Compiling $<...
    $(CC) $<

.c.obj:
    @echo Compiling $<...
    $(CC) $<


##########################################################################
#
# Application Settings
#

APPS = hello


!if "$(TARGET)" == "WIN16"
LIBS = ole2.lib compobj.lib ole2disp.lib typelib.lib $(LIBS)
!endif
!if "$(TARGET)" == "WIN32"
LIBS = $(olelibsmt)
!endif

OBJS = \
        main.obj     \
        hello.obj    \
        hellopro.obj \
        clsid.obj


##########################################################################
#
# Default Goal
#

goal : setflags $(APPS).exe
```

```
setflags :
        set CL=$(CFLAGS)


##########################################################################
#
# Clean (erase) generated files
#
clean :
    if exist *.obj       del *.obj
    if exist $(APPS).exe del $(APPS).exe
    if exist $(APPS).tlb del $(APPS).tlb
    if exist $(APPS).map del $(APPS).map
    if exist $(APPS).res del $(APPS).res
    if exist ihello.h    del ihello.h
    if exist *.log       del *.log
    if exist *.pdb       del *.pdb


##########################################################################
#
# Application Build (WIN16 Specific)
#

!if "$(TARGET)" == "WIN16"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
        link $(LINKFLAGS) @<<
$(OBJS),
$@,,
$(LIBS),
$(APPS).def
<<
        rc -k -t $(APPS).res $@
!endif


##########################################################################
#
# Application Build (WIN32 Specific)
#
!if "$(TARGET)" == "WIN32"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
      $(LINK) @<<
        $(LINKFLAGS)
        -out:$@
        -map:$*.map
        $(OBJS)
        $(APPS).res
        $(LIBS)
<<
!endif


##########################################################################
#
```

```
# Application Build (Common)
#

$(APPS).res : $(APPS).rc
        rc $(RCFLAGS) -r -fo$@ $?


##########################################################################
#
# Dependencies
#

ihello.h : hello.odl
        if exist ihello.h  del ihello.h
        if exist hello.tlb  del hello.tlb
        $(WX) mktyplib /D$(TARGET) /h ihello.h /o hello.log /tlb hello.tlb
hello.odl

main.obj : main.cpp hello.h ihello.h
        $(CC) main.cpp

hello.obj : hello.cpp hello.h resource.h
        $(CC) hello.cpp

hellopro.obj : hellopro.cpp hello.h ihello.h
        $(CC) hellopro.cpp

clsid.obj : clsid.c clsid.h
        $(CC) clsid.c
```

CLSID.H (HELLO Sample)

```
/***
*clsid.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This file defines the CLSIDs for the class which we
*  will register with OLE.
*
*  You can get a universally unique identifier (uuid) by running
*  the uuidgen tool supplied with OLE2.  When you run it, a huge
*  hex number comes out.  Do some pattern matching and stick the
*  numbers into this macro.  A long (l) is 8 he digits, a word (w)
*  is 4 hex digits and a byte is two hex digits.
*
*  You only have to do this once for each class you create and register
*  with OLE.
*
*Implementation Notes:
*
*****************************************************************************
*/


/*
 *
  IDispatch*     Hello.Application {D3CE6D43-F1AF-1068-9FBB-08002B32372A}
  Type library:  Hello.tlb         {D3CE6D44-F1AF-1068-9FBB-08002B32372A}
  Interface:     _IHello                 {D3CE6D45-F1AF-1068-
9FBB-08002B32372A}
  DispInterface:_DHello            {D3CE6D46-F1AF-1068-9FBB-08002B32372A}
 *
 */

DEFINE_GUID(CLSID_CHello,
                0xD3CE6D43,
                0xF1AF,
                0x1068,
                0x9F,
                0xBB,
                0x08,
                0x00,
                0x2B,
                0x32,
                0x37,
                0x2A);


DEFINE_GUID(IID_IHello,
                0xD3CE6D45,
                0xF1AF,
                0x1068,
                0x9F,
                0xBB,
                0x08,
                0x00,
                0x2B,
                0x32,
                0x37,
                0x2A);



DEFINE_GUID(IID_DHello,
                0xD3CE6D46,
                0xF1AF,
                0x1068,
                0x9F,
                0xBB,
                0x08,
                0x00,
                0x2B,
                0x32,
                0x37,
                0x2A);


DEFINE_GUID(LIBID_HELLOTLB,
                0xD3CE6D44,
                0xF1AF,
                0x1068,
                0x9F,
                0xBB,
                0x08,
                0x00,
                0x2B,
                0x32,
                0x37,
                0x2A);
```

CLSID.C (HELLO Sample)

```
/***
*clsid.c
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This file allocates and initializes the CLSIDs.
*
*****************************************************************************
*/



// initguid.h requires this.
//

#ifdef _MAC
# include <Types.h>
# include <Processes.h>
# include <AppleEvents.h>
#else
# include <windows.h>
#endif

#ifndef WIN32
#include <compobj.h>
#endif //!WIN32

// this redefines the DEFINE_GUID() macro to do allocation.
//
#include <initguid.h>

// due to the previous header, including this causes the DEFINE_GUID
// definitions in the following header(s) to actually allocate data.
//
#include "clsid.h"
```

COMMON.H   (HELLO Sample)

```
/***
*
*  Copyright (C) 1993-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*  File:
*    common.h
*
*  Purpose:
*
*    Common definitions across Win16/Win32
*
****************************************************************************
*/

#ifndef __Common_h_
#define __Common_h_

#include <windows.h>
#include <ole2.h>
#ifndef WIN32
#include <olenls.h>
#include <dispatch.h>
#else
#include <oleauto.h>
#endif


#ifdef WIN32
# define STRLEN        strlen
# define STRICMP _stricmp
# define MEMCPY        memcpy
# define MEMCMP        memcmp
# define MEMSET        memset
# define STRSTR        strstr
# if defined(UNICODE)
    #define TCHAR      WCHAR
    #define TSTR(str)        L##str
    #define STRING(str)      (str)
# else
    #define TCHAR      char
    #define TSTR(str)        str
    #define STRING(str)         AnsiString(str)
    extern "C" char FAR* AnsiString(OLECHAR FAR* strIn);
# endif

#else
# define STRLEN        _fstrlen
# define STRICMP _fstricmp
# define MEMCPY        _fmemcpy
# define MEMCMP        _fmemcmp
# define MEMSET        _fmemset
# define STRSTR         _fstrstr
# define TCHAR          char
# define TSTR(str)      str
# define STRING(str)     (str)
#endif

#ifndef CLASS
# ifdef __TURBOC__
#  define CLASS class huge
# else
#  define CLASS class FAR
# endif
#endif


#endif // __Common_h_
```

HELLO.DEF (HELLO Sample)

```
NAME          HELLO

DESCRIPTION   'Hello world OLE object'

EXETYPE       WINDOWS

STUB          'WINSTUB.EXE'

CODE          PRELOAD MOVEABLE DISCARDABLE
DATA          PRELOAD MULTIPLE

HEAPSIZE      4096
STACKSIZE     8192
```

HELLO.H (HELLO Sample)

```
/***
*
*  Copyright (C) 1993-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*  File:
*    hello.h
*
*  Purpose:
*    Contains the following:
*
*    * The CHello class
*          - Creation
*          - Creating the standard dispatch object
*          - Deletion
*          - Reference count maintenance
*          - IUnknown methods
*      * The CHelloCF class, which creates instances of CHello.
*          - Creation
*          - Deletion
*          - Reference count maintenance
*          - IUnknown methods
*
****************************************************************************
*/

#include "common.h"
#include "clsid.h"
#include "hellopro.h"

#pragma warning(disable:4355)



/*   The CHello class.
 *
 */
CLASS CHello : public IUnknown
{

 public:
    CHello();
    ~CHello();
    static CHello FAR* Create();
    void ProcessCommand(WPARAM wparam); // Process a Windows WM_COMMAND
message

    // Standard OLE stuff.
    STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppv);
    STDMETHOD_(ULONG, AddRef)(void);
    STDMETHOD_(ULONG, Release)(void);
    HWND m_hwnd;             // The window to which we draw.
    BSTR m_bstrHelloMsg;

 private:
    ULONG m_refs;            // Reference count.
    IUnknown FAR* m_disp_interface;// Pointer to the standard dispatch
object.
    CHelloPro FAR* m_prog_interface;  // What a programmer sees.
};





/*   The class factory for CHello.
 *
 */
CLASS CHelloCF : public IClassFactory
{
  public:
    CHelloCF();
    static IClassFactory FAR* Create();

    STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppv);
    STDMETHOD_(ULONG, AddRef)(void);
    STDMETHOD_(ULONG, Release)(void);

    STDMETHOD(CreateInstance)(     IUnknown FAR* punkOuter,
                       REFIID riid,
                       void FAR* FAR* ppv);
    STDMETHOD(LockServer)(BOOL fLock);

  private:
    ULONG m_refs;
};
```

HELLO.ODL (HELLO Sample)

```
/*
 *   hello.odl
 *
 *
 *
  IDispatch*     Hello.Application {D3CE6D43-F1AF-1068-9FBB-08002B32372A}
  Type library:  hello.tlb         {D3CE6D44-F1AF-1068-9FBB-08002B32372A}
  Interface:     IHello                  {D3CE6D45-F1AF-1068-
9FBB-08002B32372A}

 *
 *
 *
 */

[
  uuid(D3CE6D44-F1AF-1068-9FBB-08002B32372A),
  helpstring("OLE Automation Hello 1.0 Type Library"),
  lcid(0x0409),
  version(1.0)
]
library Hello
{
#ifdef WIN32
    importlib("stdole32.tlb");
#else
    importlib("stdole.tlb");
#endif

    [
     // The uuid for the interface IID_IHelloPro.
     odl,
     uuid(D3CE6D45-F1AF-1068-9FBB-08002B32372A),
     helpstring("Hello")
    ]
    interface _IHello : IUnknown
    {
     [id(0), propput]
     void HelloMessage([in] BSTR b);

           [id (0), propget, helpstring("The message that will be
displayed.")]
           BSTR HelloMessage();

     BSTR SayHello(void);
    }


    [
     uuid(D3CE6D46-F1AF-1068-9FBB-08002B32372A),
     helpstring("Hello DispInterface")
    ]
    dispinterface _DHello
    {
      interface _IHello;
    }


    [ // The uuid for the class we expose.  Same as registry
      // entry for hello.hello
      uuid(D3CE6D43-F1AF-1068-9FBB-08002B32372A),
      helpstring("Hello")
    ]
    coclass Hello
    {
      dispinterface _DHello;
      interface _IHello;
    }
};
```

HELLO.RC   (HELLO Sample)

```
//Microsoft App Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
/////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#define APSTUDIO_HIDDEN_SYMBOLS
#include "windows.h"
#undef APSTUDIO_HIDDEN_SYMBOLS
#include "resource.h"

////////////////////////////////////////////////////////////////////////////
/////////
#undef APSTUDIO_READONLY_SYMBOLS


////////////////////////////////////////////////////////////////////////////
//
//
// Icon
//

HELLO                   ICON    DISCARDABLE     "HELLO.ICO"

////////////////////////////////////////////////////////////////////////////
//
//
// Dialog
//

HELLO DIALOG DISCARDABLE  20, 20, 214, 144
STYLE WS_MINIMIZEBOX | WS_CAPTION | WS_SYSMENU
CAPTION "Hello"
CLASS "Hello"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON      "Say Hello",IDC_SAYHELLO,7,6,52,15
    EDITTEXT        IDC_HELLOAREA,86,7,119,117,ES_MULTILINE | ES_AUTOHSCROLL
|
                    ES_READONLY
END

#ifdef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////
//
//
// TEXTINCLUDE
//
```

```
1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resrc1.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#define APSTUDIO_HIDDEN_SYMBOLS\r\n"
    "#include ""windows.h""\r\n"
    "#undef APSTUDIO_HIDDEN_SYMBOLS\r\n"
    "#include ""resource.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "\r\n"
    "\0"
END

////////////////////////////////////////////////////////////////////////////
/////////
#endif    // APSTUDIO_INVOKED


#ifndef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////
////
//
// Generated from the TEXTINCLUDE 3 resource.
//


////////////////////////////////////////////////////////////////////////////
/////////
#endif    // not APSTUDIO_INVOKED
```

HELLO.REG   (HELLO Sample)

```
REGEDIT

; Registration information for the hello application
;
; IDispatch*     Hello.Application {D3CE6D43-F1AF-1068-9FBB-08002B32372A}
; Type library:  hello.tlb         {D3CE6D44-F1AF-1068-9FBB-08002B32372A}
; Interface:     IHello                  {D3CE6D45-F1AF-1068-
9FBB-08002B32372A}
;


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; registration info Hello.Application (defaults to Hello.Application.1)

HKEY_CLASSES_ROOT\Hello.Application = OLE Automation Hello Application
HKEY_CLASSES_ROOT\Hello.Application\Clsid = {D3CE6D43-F1AF-1068-
9FBB-08002B32372A}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; registration info Hello.Application.1

HKEY_CLASSES_ROOT\Hello.Application.1 = OLE Automation Hello 1.0 Application
HKEY_CLASSES_ROOT\Hello.Application.1\Clsid = {D3CE6D43-F1AF-1068-
9FBB-08002B32372A}


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; registration info Hello 1.0

HKEY_CLASSES_ROOT\CLSID\{D3CE6D43-F1AF-1068-9FBB-08002B32372A} = IDispatch
Hello Example
HKEY_CLASSES_ROOT\CLSID\{D3CE6D43-F1AF-1068-9FBB-08002B32372A}\ProgID =
Hello.Application.1
HKEY_CLASSES_ROOT\CLSID\{D3CE6D43-F1AF-1068-9FBB-08002B32372A}
\VersionIndependentProgID = Hello.Application
HKEY_CLASSES_ROOT\CLSID\{D3CE6D43-F1AF-1068-9FBB-08002B32372A}\LocalServer32
= hello.exe /Automation
```

HELLO.CPP (HELLO Sample)

```
/***
*
*  Copyright (C) 1993-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*  File:
*    hello.cpp
*
*  Purpose:
*    See comment in hello.h
*
*****************************************************************************
*/


#include "common.h"
#include "resource.h"
#include "hello.h"





IUnknown FAR*
CreateDispatchInterface(
  IUnknown FAR*  punkController,
  void FAR* pProgInterface)
{
    HRESULT hresult;
    ITypeLib FAR* ptlib;
    ITypeInfo FAR* ptinfo;
    IUnknown FAR* punkStdDisp;

    // Load the the hellotlb type library.
    hresult = LoadRegTypeLib(LIBID_HELLOTLB, 1, 0, 0x0409, &ptlib);
    if (hresult != NOERROR) {
      #define TLB_NAME OLESTR("hello.tlb")

      // if it wasn't registered, try to load it from the path/current
directory
      // if this succeeds, it will have registered the type library for us
      // for the next time.
      if((hresult = LoadTypeLib(TLB_NAME, &ptlib)) != NOERROR)
      return NULL;
    }

    // Get Dispatch TypeInfo from the type library.
    // Exit if an error occurs.
    hresult = ptlib->GetTypeInfoOfGuid(IID_IHello, &ptinfo);
    if (hresult != NOERROR)
     return NULL;
    ptlib->Release();

    // Create a standard dispatch object.
    // Release the pointer to the previously created TypeInfo.
    // Exit if an error occurred.
    hresult = CreateStdDispatch(punkController,
                       pProgInterface,
                       ptinfo,
                             &punkStdDisp);
    ptinfo->Release();
    if (hresult != NOERROR)
     return NULL;


    // If execution has reached this point, then no errors occurred.
    return punkStdDisp;
}





/*
 *   CHello::CHello(void)
 *
 */
CHello::CHello()
{
    m_refs = 1;
    m_disp_interface = NULL;
    m_prog_interface = new CHelloPro;
    m_prog_interface->m_phello = this;
    m_bstrHelloMsg = SysAllocString(OLESTR("Hello, world"));
}





/*
 *   CHello::~CHello(void)
 *
 */
CHello::~CHello()
{
    SysFreeString(m_bstrHelloMsg);
    delete m_prog_interface;
}






/*
 *   CHello *CHello::Create(void)
 *
 *   Creates a new instance of the CHello object.
 *
 *   Returns a pointer to the newly created instance or
 *   NULL if the creation fails.
 *
 */
CHello FAR*
CHello::Create()
{
    CHello FAR* phello;
    IUnknown FAR* punkStdDisp;


    // Create an instance of CHello.  Exit if an error occurs.
    phello = new FAR CHello();
    if(phello == NULL)
      return NULL;

    punkStdDisp = CreateDispatchInterface((IUnknown FAR*) phello,
                               phello->m_prog_interface);
    if (punkStdDisp == NULL) {
      phello->Release();
      return NULL;
    }


    // If execution has reached this point, then no errors have occurred.
    // Save the standard dispatch item and return the new instance.
    phello->m_disp_interface = punkStdDisp;
    return phello;
}





/*
 *   void CHello::ProcessCommand(WPARAM param)
 *
 *   Process a windows WM_COMMAND with the specified parameter.
 *
 */
void CHello::ProcessCommand(WPARAM wparam)
{
    switch(wparam) {
     case IDC_SAYHELLO:
     // The user pressed the "Say Hello" button.
     m_prog_interface->SayHello();
     break;
    }
}
```

```
//---------------------------------------------------------------------
//                        IUnknown methods
//---------------------------------------------------------------------


/*   CHello::QueryInterface(...)
 *
 *   Someone wants to know if CHello support the interface identified
 *   by riid.  CHello supports IUnknown and we defer to a member
 *      (m_disp_interfaceatch) and thus support IDispatch.
 *
 *   CHello doesn't support any other interfaces.
 *
 */
STDMETHODIMP
CHello::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    // riid identifies an interface type.  Do we support it?

    if (IsEqualIID(riid, IID_IUnknown))
     // Someone is asking us for our controlling IUnknown.
     // CHello is that IUnknown, so "this" is the answer.
     *ppv = this;
    else if (IsEqualIID(riid, IID_IDispatch) ||
          IsEqualIID(riid, IID_DHello))
        // Someone is asking us for our IDispatch or dispinterface.
     // We simply defer to the standard dispatch interface
     // we created earlier.
     return m_disp_interface->QueryInterface(IID_IDispatch, ppv);
    else if (IsEqualIID(riid, IID_IHello))
     // Someone is asking us for our IHelloPro interface.
     // We simply defer to the standard dispatch interface
     // we created earlier.
     *ppv = &m_prog_interface;
    else {
     // They must have asked for another interface -- something
     // we don't support.  Report the error.
     *ppv = NULL;
     return ResultFromScode(E_NOINTERFACE);
    }

    // If execution reached this point then no error occurred.
    AddRef();
    return NOERROR;
}





STDMETHODIMP_(ULONG)
CHello::AddRef()
```

```
{
    return ++m_refs;
}




STDMETHODIMP_(ULONG)
CHello::Release()
{
    if(--m_refs == 0)
    {
      if(m_disp_interface != NULL)
           m_disp_interface->Release();
      PostQuitMessage(0);
      delete this;
      return 0;
    }
    return m_refs;
}



//----------------------------------------------------------------------
//                      The CHello Class Factory
//----------------------------------------------------------------------


CHelloCF::CHelloCF()
{
    m_refs = 1;
}




IClassFactory FAR*
CHelloCF::Create()
{
    return new FAR CHelloCF();
}




STDMETHODIMP
CHelloCF::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    if(IsEqualIID(riid, IID_IUnknown) ||
       IsEqualIID(riid, IID_IClassFactory)) {
      AddRef();
      *ppv = this;
      return NOERROR;
    }
    *ppv = NULL;
    return ResultFromScode(E_NOINTERFACE);
}






STDMETHODIMP_(ULONG)
CHelloCF::AddRef()
{
    return ++m_refs;
}






STDMETHODIMP_(ULONG)
CHelloCF::Release()
{
    if(--m_refs == 0) {
       delete this;
       return 0;
    }
    return m_refs;
}







STDMETHODIMP
CHelloCF::CreateInstance(
    IUnknown FAR* punkOuter,
    REFIID riid,
    void FAR* FAR* ppv)
{
    extern CHello FAR* g_phello;


    return g_phello->QueryInterface(riid, ppv);
}






STDMETHODIMP
CHelloCF::LockServer(BOOL fLock)
{
    return NOERROR;
```

```
}
```

HELLOPRO.H (HELLO Sample)

```
/***
*
*  Copyright (C) 1993-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*  File:
*    hellopro.h
*
*  Purpose:
*    This is the header for CHelloProgrammability, which is the
*    programmability interface for CHello objects.
*
*    This is the user interface that a programmer who uses our
*    object will be using.
*
****************************************************************************
*/

#include "common.h"


// Forward declaration.
CLASS CHello;

#include "ihello.h"    // Include MkTypLib-created interface
                 // description of _IHello.



CLASS CHelloPro : public _IHello
{
  public:

    // Standard OLE stuff.
    STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR * ppvObj);
    STDMETHOD_(ULONG, AddRef)(THIS);
    STDMETHOD_(ULONG, Release)(THIS);

    STDMETHOD_(void, put_HelloMessage)(THIS_ BSTR b);
    STDMETHOD_(BSTR, get_HelloMessage)(THIS);
    STDMETHOD_(BSTR, SayHello)(THIS);

    CHello FAR*  m_phello;
};
```

HELLOPRO.CPP (HELLO Sample)

```
/***
*
*  Copyright (C) 1993-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*  File:
*    hellopro.cpp
*
*  Purpose:
*    CHelloProgrammability is the programmability interface for
*    the CHello object.  it contains the implementation of every
*    member of the CHello class which is visible to a programmer
*    using the CHello object.
*
****************************************************************************
*/

#include "common.h"
#include "resource.h"
#include "hello.h" // This also gets hellopro.h



/*      SetHelloMsg
 *
 *      SetHelloMsg sets the message that the hello object
 *      says when the SayHello method is called.
 *
 */
STDMETHODIMP_(void)
CHelloPro::put_HelloMessage(BSTR b)
{
     SysReAllocString(&(m_phello->m_bstrHelloMsg), b);
}




/*      GetHelloMsg
 *
 *   GetHelloMsg gets the message that the hello object
 *   says when the SayHello method is called.
 *
 */
STDMETHODIMP_(BSTR)
CHelloPro::get_HelloMessage()
{
     return SysAllocString(m_phello->m_bstrHelloMsg);
}
```

```
/*      SayHello
 *
 *   SayHello writes the string "Hello world." in the dialog's
 *   text box and returns the same string.
 *
 */
STDMETHODIMP_(BSTR)
CHelloPro::SayHello()
{
     BSTR b;

     b = SysAllocString(m_phello->m_bstrHelloMsg);
     SetDlgItemText(m_phello->m_hwnd, IDC_HELLOAREA, STRING(b));
     return b;
}




/*   IUnknown methods
 *
 *   Simply defer to pur controlling IUnknown.
 *
 */
STDMETHODIMP
CHelloPro::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
     return m_phello->QueryInterface(riid, ppv);
}

STDMETHODIMP_(ULONG)
CHelloPro::AddRef()
{
     return m_phello->AddRef();
}

STDMETHODIMP_(ULONG)
CHelloPro::Release()
{
     return m_phello->Release();
}
```

MAIN.CPP (HELLO Sample)

```
/***
*
*  Copyright (C) 1993-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*  File:
*    main.cpp
*
*  Purpose:
*    This file contains:
*
*      WinMain() and WndProc(), which are called by Windows when the app is
run.
*      Code to initialize and uninitialize OLE.
*      Code to create and destroy the window for the application.
*
*****************************************************************************
*/

#include "common.h"
#include "resource.h"
#include "hello.h"

#include <string.h>


/*   Global variables.
 */

DWORD g_dwCHelloCF = 0;  // Holds the return code for class factory
creation.

CHello FAR* g_phello = NULL; // Pointer to a CHello object.

TCHAR g_szAppName[] = TSTR("Hello"); // Name of the application.



/*   Forward declarations.
 */
HRESULT InitOle(void);
void UninitOle(void);
BOOL InitApplication(HANDLE);
BOOL InitInstance(HANDLE, int);



/*   Let Windows call WinMain and WndProc.
 */
extern "C"
{
     long FAR PASCAL WndProc(HWND, UINT, WPARAM, LPARAM);
     int PASCAL WinMain(HANDLE, HANDLE, LPSTR, int);
}




/*   WinMain
 *
 *   Windows calls WinMain when the application starts.
 *
 */
extern "C" int PASCAL
WinMain(
    HANDLE hinst,
    HANDLE hPrevInstance,
    LPSTR lpCmdLine,
    int nCmdShow)
{
    MSG msg;


    if(!hPrevInstance)
      if(!InitApplication(hinst))
     return FALSE;

    if(InitOle() != NOERROR)
      return FALSE;

    if(!InitInstance(hinst, nCmdShow)){
      UninitOle();
      return FALSE;
    }


    while(GetMessage(&msg, NULL, NULL, NULL)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }

    // Uninitialize OLE.
    UninitOle();

    return msg.wParam;
}




/*   InitApplication
 *
 *   Create a window and register it with Windows.
 *
 *   Return FALSE if an error occurs and TRUE otherwise.
 */
BOOL
InitApplication(HANDLE hinst)
{
    WNDCLASS  wc;

    wc.style           = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc     = WndProc;
    wc.cbClsExtra= 0;
    wc.cbWndExtra= DLGWINDOWEXTRA;
    wc.hInstance = hinst;
    wc.hIcon           = LoadIcon(hinst, g_szAppName);     // Loads
hello.ico
    wc.hCursor         = LoadCursor(NULL, IDC_ARROW);      // Normal arrow
pointer
    wc.hbrBackground    = (HBRUSH) (COLOR_APPWORKSPACE+1);
    wc.lpszMenuName    = NULL;                       // No menus.
    wc.lpszClassName= g_szAppName;

    // Tell Windows about the window class we just created.
    // Exit if an error occurs.
    if(!RegisterClass(&wc))
      return FALSE;

    return TRUE;
}




/*   InitInstance
 *
 *   Create an instance of CHello and make g_phello
 *   point to it..
 *
 *   Return FALSE if an error occurs and TRUE otherwise.
 */
BOOL
InitInstance(HANDLE hinst, int nCmdShow)
{
    HWND hwnd;

    // Create the window and show it
    hwnd = CreateDialog(hinst, g_szAppName, 0, NULL);
    ShowWindow(hwnd, nCmdShow);
    g_phello->m_hwnd = hwnd;

    return TRUE;
}
```

```
extern "C" long FAR PASCAL
WndProc(
    HWND hwnd,
    UINT message,
    WPARAM wParam,
    LPARAM lParam)
{
    switch(message) {
      case WM_COMMAND:
       g_phello->ProcessCommand(wParam);
       return 0;

      case WM_DESTROY:
        PostQuitMessage(0);
              return 0;
    }
    return DefWindowProc(hwnd, message, wParam, lParam);
}
```

```
HRESULT
InitOle()
{
    HRESULT hresult;
    IClassFactory FAR* pcf;

    if((hresult = OleInitialize(NULL)) != NOERROR)
      return hresult;

    // create the single global instance of CHello
    if((g_phello = CHello::Create()) == NULL){
      hresult = ResultFromScode(E_OUTOFMEMORY);
      return hresult;
    }

    // Create an instance of the class factory for CHello.
    // Exit if an error occurs.
    pcf = CHelloCF::Create();
    if (pcf == NULL) {
      UninitOle();
      return hresult;
    }

    // Register the class factroy.  Exit if an error occurs.
    hresult = CoRegisterClassObject(CLSID_CHello,
                           pcf,
                           CLSCTX_LOCAL_SERVER,
                           REGCLS_MULTIPLEUSE,
                           &g_dwCHelloCF);
    if (hresult != NOERROR) {
      pcf->Release();
      UninitOle();
      return hresult;
    }

    pcf->Release();

    // If execution has reached this spot, then no errors have occurred.
    return NOERROR;
}





/*   UninitOLE
 *
 *   Tell OLE that we are going away.
 *
 */
void
UninitOle()
{
    // If a class factory was successfully created earlier then
    // tell OLE that the object is no longer available.
    if(g_dwCHelloCF != 0)
      CoRevokeClassObject(g_dwCHelloCF);

    // cause the remaining typeinfo to be released
    if (g_phello != NULL)
      g_phello->Release();

    // Tell OLE we are done using them.
    OleUninitialize();
}


#if defined(WIN32)

extern "C" char FAR*
ConvertStrWtoA(OLECHAR FAR* strIn, char FAR* buf, UINT size)
{
  int badConversion = FALSE;

  WideCharToMultiByte(CP_ACP, NULL,
                   strIn, -1,
                 buf, size,
                 NULL, &badConversion);
  return buf;
}

extern "C" char FAR*
AnsiString(OLECHAR FAR* strIn)
{
  static char buf[256];
  return (ConvertStrWtoA(strIn, buf, 256));
}

#endif
```

RESOURCE.H (HELLO Sample)

```
//{{NO_DEPENDENCIES}}
// App Studio generated include file.
// Used by HELLO.RC
//


#define IDC_SAYHELLO    1000
#define IDC_HELLOAREA  1001


// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS

#define _APS_NEXT_RESOURCE_VALUE        101
#define _APS_NEXT_COMMAND_VALUE         101
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

INCLUDE

These topics contain header files included in the various other OLE code samples.

ANSIAPI.H   (INCLUDE Sample)

```
/*
 * ansiapi.h
 * This file contains prototypes of ANSI version of OLE apis
 * and mapping fooA to foo
 */

#ifndef _ANSIAPI_INCL
#define _ANSIAPI_INCL

#define UNICODEOLE32

#ifdef WIN32S
#if !defined(UNICODEOLE32)
#pragma message("Compiling for 32bit ANSI Ole!\n");
#endif
#endif

#define WASANSI() {
\
                   OutputDebugString("\n\r\t\tUnicode string was ANSI!
\r\n"); \
                   Assert(0);
\
                   _asm { int 3 }
\
                  }



#ifndef UNICODEOLE32

#define W2A(w, a, cb)     lstrcpy (a, w)
#define A2W(a, w, cb)     lstrcpy (w, a)

#define FREELOCALSTRING(p)
#define MAKE_STR_LOCAL_COPYW2A(s, l)  l = s
#define MAKE_STR_LOCAL_COPYA2W(s, l)  l = s

#define OLESTRCPY                lstrcpy
#define OLESTRCAT                lstrcat
#define OLESTRLEN                lstrlen
#define OLESTRCMP                lstrcmp
#define OLESTRCMPI               lstrcmpi

#define DeleteFile_AW            DeleteFile
#define _lopen_AW                _lopen
#define _lcreat_AW               _lcreat
#define GlobalAddAtom_AW         GlobalAddAtom
#define GlobalGetAtomName_AW     GlobalGetAtomName
#define RegOpenKey_AW            RegOpenKey

#define CoLoadLibraryA   CoLoadLibrary
```

```
#define StringFromCLSID2A    StringFromCLSID2
#define StringFromIID2A      StringFromIID2
#define StringFromGUID2A     StringFromGUID2
#define CLSIDFromProgIDA     CLSIDFromProgID
#define CLSIDFromStringA     CLSIDFromString
#define ProgIDFromCLSIDA     ProgIDFromCLSID
#define StringFromCLSIDA     StringFromCLSID

#define UtDupStringA2W       UtDupString

// Storage APIs
#define StgOpenStorageA      StgOpenStorage

// IPersistFile ansi translation

#define IPersistFile_LoadA(pf, file, mode) (pf)->Load(file, mode)

// IMoniker ansi translation

#define IMoniker_GetDisplayNameA(pm, p1, p2, p3) (pm)-
>GetDisplayName(p1,p2,p3)

#define CreateFileMonikerA   CreateFileMoniker

#else  // UNICODEOLE32


#define W2A(w, a, cb)
WideCharToMultiByte(                              \
                                               CP_ACP,
\
                                               0,
\
                                               w,
\
                                               -1,
\
                                               a,
\
                                               cb,
\
                                               NULL,
\
                                               NULL)

#define A2W(a, w, cb)
MultiByteToWideChar(                              \
                                               CP_ACP,
\
                                               0,
\
                                               a,
\
                                                -1,
\
                                                w,
\
                                                cb)


#define FREELOCALSTRING(p)      delete (p)
#define MAKE_STR_LOCAL_COPYW2A(s, l)  {                           \
                                      l = UtDupStringW2A(s);      \
                                      if (!l) {                   \
                                         return ResultFromScode(S_OOM);   \
                                      }                           \
                                   }

#define MAKE_STR_LOCAL_COPYA2W(s, l)  {                           \
                                      l = UtDupStringA2W(s);      \
                                      if (!l) {                   \
                                         return ResultFromScode(S_OOM);   \
                                      }                           \
                                   }

#define OLESTRCPY                lstrcpyW
#define OLESTRCAT                lstrcatW
#define OLESTRLEN                lstrlenW
#define OLESTRCMP                lstrcmpW
#define OLESTRCMPI               lstrcmpiW

#define DeleteFile_AW            DeleteFileW
#define _lopen_AW                _lopenW
#define _lcreat_AW               _lcreatW
#define GlobalAddAtom_AW         GlobalAddAtomW
#define GlobalGetAtomName_AW     GlobalGetAtomNameW
#define RegOpenKey_AW            RegOpenKeyW

STDAPI_(HINSTANCE) CoLoadLibraryA(LPSTR lpszLibName, BOOL bAutoFree);

#define StringFromCLSID2A(rclsid, lpsz, cbMax) \
    StringFromGUID2A(rclsid, lpsz, cbMax)

#define StringFromIID2A(riid, lpsz, cbMax) \
    StringFromGUID2A(riid, lpsz, cbMax)


STDAPI_(int)  StringFromGUID2A(REFGUID rguid, LPSTR lpsz, int cbMax);
STDAPI        CLSIDFromProgIDA(LPCSTR szProgID, LPCLSID pclsid);
STDAPI        CLSIDFromStringA(LPSTR lpsz, LPCLSID lpclsid);
STDAPI        StringFromCLSIDA(REFCLSID rclsid, LPSTR FAR* lplpsz);
STDAPI        ProgIDFromCLSIDA (REFCLSID clsid, LPSTR FAR* lplpszProgID);

LPWSTR UtDupStringA2W(LPCSTR pSrc);
LPSTR  UtDupStringW2A(LPCWSTR pSrc);

// Storage APIs
```

```
STDAPI StgOpenStorageA(LPCSTR pwcsName,IStorage FAR *pstgPriority, DWORD
grfMode, SNB snbExclude, DWORD reserved, IStorage FAR * FAR *ppstgOpen);

// IPersistFile ansi translation

HRESULT IPersistFile_LoadA(LPPERSISTFILE pIPF, LPSTR szFile, DWORD dwMode);

// IMoniker ansi translation

HRESULT IMoniker_GetDisplayNameA(LPMONIKER pm, LPBC p1, LPMONIKER p2, LPSTR
FAR *p3);

//OLEAPI CreateFileMonikerA ( LPSTR lpszPathName, LPMONIKER FAR * ppmk );

#endif // !UNICODEOLE32

#endif //  _ANSIAPI_INCL
_
```

ASSERT.H (INCLUDE Sample)

```
//
+---------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       assert.h
//
//  Contents:   private definition of assert, used by NT SDK OLE2 samples
//
//  History:    8-19-94   stevebl   Created
//
//---------------------------------------------------------------------------
--

/*
 * Conditional macro definition for function calling type and variable type
 * qualifiers.
 */
#if   ( (_MSC_VER >= 800) && (_M_IX86 >= 300) )

/*
 * Definitions for MS C8-32 (386/486) compiler
 */
#define _CRTAPI1 __cdecl
#define _CRTAPI2 __cdecl

#else

/*
 * Other compilers (e.g., MIPS)
 */
#define _CRTAPI1
#define _CRTAPI2

#endif

#undef  assert

#ifdef NDEBUG

#define assert(exp)     ((void)0)

#else

#ifdef __cplusplus
extern "C" {
#endif
void _CRTAPI1
PopUpAssert(
    void * szFile,
    int iLine,
    void * szMessage);
#ifdef __cplusplus
}
#endif


#define assert(exp) (void)( (exp) || (PopUpAssert(__FILE__, __LINE__, #exp), 
0) )
#define _assert(exp, file, line) PopUpAssert(file, line, exp)


#endif  /* NDEBUG */
```

BTTNCUR.H   (INCLUDE Sample)

```
/*
 * BTTNCUR.H
 * Buttons & Cursors Version 1.1, March 1993
 *
 * Public include file for the Button Images and Cursor DLL, including
 * structures, definitions, and function prototypes.
 *
 * Copyright (c)1992-1993 Microsoft Corporation, All Rights Reserved,
 * as applied to redistribution of this source code in source form
 * License is granted to use of compiled code in shipped binaries.
 */


#ifndef _BTTNCUR_H_
#define _BTTNCUR_H_

#ifdef __cplusplus
extern "C"
     {
#endif


//Standard image bitmap

//WARNING:  Obsolete.  Use the return from UIToolDisplayData
#define IDB_STANDARDIMAGES              400

//New values for display types
#define IDB_STANDARDIMAGESMIN           400
#define IDB_STANDARDIMAGES96            400
#define IDB_STANDARDIMAGES72            401
#define IDB_STANDARDIMAGES120           402



//Image indices inside the standard bitmap.
#define TOOLIMAGE_MIN                   0
#define TOOLIMAGE_EDITCUT               0
#define TOOLIMAGE_EDITCOPY              1
#define TOOLIMAGE_EDITPASTE             2
#define TOOLIMAGE_FILENEW               3
#define TOOLIMAGE_FILEOPEN              4
#define TOOLIMAGE_FILESAVE              5
#define TOOLIMAGE_FILEPRINT             6
#define TOOLIMAGE_HELP                  7
#define TOOLIMAGE_HELPCONTEXT           8
#define TOOLIMAGE_MAX                   8


//Additional Standard Cursors as defined in the UI Design Guide.
#define IDC_NEWUICURSORMIN              500
#define IDC_RIGHTARROW                  500
```

```
#define IDC_CONTEXTHELP              501
#define IDC_MAGNIFY                  502
#define IDC_NODROP                   503
#define IDC_TABLETOP                 504
#define IDC_HSIZEBAR                 505
#define IDC_VSIZEBAR                 506
#define IDC_HSPLITBAR                507
#define IDC_VSPLITBAR                508
#define IDC_SMALLARROWS              509
#define IDC_LARGEARROWS              510
#define IDC_HARROWS                  511
#define IDC_VARROWS                  512
#define IDC_NESWARROWS               513
#define IDC_NWSEARROWS               514
#define IDC_NEWUICURSORMAX           514



//Standard sizes for toolbar buttons and bitmaps on display types

//WARNING:  These are obsolete for version 1.0 compatibility/
#define TOOLBUTTON_STDWIDTH          24
#define TOOLBUTTON_STDHEIGHT         22
#define TOOLBUTTON_STDIMAGEWIDTH     16
#define TOOLBUTTON_STDIMAGEHEIGHT    15

/*
 * Applications can call UIToolDisplayData to get the particular
 * values to use for the current display instead of using these values
 * directly.  However, if the application has the aspect ratio already
 * then these are available for them.
 */

//Sizes for 72 DPI (EGA)
#define TOOLBUTTON_STD72WIDTH        24
#define TOOLBUTTON_STD72HEIGHT       16
#define TOOLBUTTON_STD72IMAGEWIDTH   16
#define TOOLBUTTON_STD72IMAGEHEIGHT  11

//Sizes for 96 DPI (VGA)
#define TOOLBUTTON_STD96WIDTH        24
#define TOOLBUTTON_STD96HEIGHT       22
#define TOOLBUTTON_STD96IMAGEWIDTH   16
#define TOOLBUTTON_STD96IMAGEHEIGHT  15

//Sizes for 120 DPI (8514/a)
#define TOOLBUTTON_STD120WIDTH       32
#define TOOLBUTTON_STD120HEIGHT      31
#define TOOLBUTTON_STD120IMAGEWIDTH  24
#define TOOLBUTTON_STD120IMAGEHEIGHT 23


//Sizes of a standard button bar depending on the display
#define CYBUTTONBAR72                23
#define CYBUTTONBAR96                29
```

```
#define CYBUTTONBAR120                 38



/*
 * The low-word of the state contains the display state where each
 * value is mutually exclusive and contains one or more grouping bits.
 * Each group represents buttons that share some sub-state in common.
 *
 * The high-order byte controls which colors in the source bitmap,
 * black, white, gray, and dark gray, are to be converted into the
 * system colors COLOR_BTNTEXT, COLOR_HILIGHT, COLOR_BTNFACE, and
 * COLOR_BTNSHADOW.  Any or all of these bits may be set to allow
 * the application control over specific colors.
 *
 * The actual state values are split into a command group and an
 * attribute group.  Up, mouse down, and disabled states are identical,
 * but only attributes can have down, down disabled, and indeterminate
 * states.
 *
 * BUTTONGROUP_BLANK is defined so an application can draw only the button
 * without an image in the up, down, mouse down, or indeterminate
 * state, that is, BUTTONGROUP_BLANK is inclusive with BUTTONGROUP_DOWN
 * and BUTTONGROUP_LIGHTFACE.
 */


#define BUTTONGROUP_DOWN               0x0001
#define BUTTONGROUP_ACTIVE             0x0002
#define BUTTONGROUP_DISABLED           0x0004
#define BUTTONGROUP_LIGHTFACE          0x0008
#define BUTTONGROUP_BLANK              0x0010

//Command buttons only
#define COMMANDBUTTON_UP               (BUTTONGROUP_ACTIVE)
#define COMMANDBUTTON_MOUSEDOWN        (BUTTONGROUP_ACTIVE |
BUTTONGROUP_DOWN)
#define COMMANDBUTTON_DISABLED         (BUTTONGROUP_DISABLED)

//Attribute buttons only
#define ATTRIBUTEBUTTON_UP             (BUTTONGROUP_ACTIVE)
#define ATTRIBUTEBUTTON_MOUSEDOWN      (BUTTONGROUP_ACTIVE |
BUTTONGROUP_DOWN)
#define ATTRIBUTEBUTTON_DISABLED       (BUTTONGROUP_DISABLED)
#define ATTRIBUTEBUTTON_DOWN           (BUTTONGROUP_ACTIVE |
BUTTONGROUP_DOWN | BUTTONGROUP_LIGHTFACE)
#define ATTRIBUTEBUTTON_INDETERMINATE  (BUTTONGROUP_ACTIVE |
BUTTONGROUP_LIGHTFACE)
#define ATTRIBUTEBUTTON_DOWNDISABLED   (BUTTONGROUP_DISABLED |
BUTTONGROUP_DOWN | BUTTONGROUP_LIGHTFACE)

//Blank buttons only
#define BLANKBUTTON_UP                 (BUTTONGROUP_ACTIVE |
BUTTONGROUP_BLANK)
```

```
#define BLANKBUTTON_DOWN                (BUTTONGROUP_ACTIVE |
BUTTONGROUP_BLANK | BUTTONGROUP_DOWN | BUTTONGROUP_LIGHTFACE)
#define BLANKBUTTON_MOUSEDOWN           (BUTTONGROUP_ACTIVE |
BUTTONGROUP_BLANK | BUTTONGROUP_DOWN)
#define BLANKBUTTON_INDETERMINATE       (BUTTONGROUP_ACTIVE |
BUTTONGROUP_BLANK | BUTTONGROUP_LIGHTFACE)


/*
 * Specific bits to prevent conversions of specific colors to system
 * colors.  If an application uses this newer library and never specified
 * any bits, then they benefit from color conversion automatically.
 */
#define PRESERVE_BLACK                  0x0100
#define PRESERVE_DKGRAY                 0x0200
#define PRESERVE_LTGRAY                 0x0400
#define PRESERVE_WHITE                  0x0800

#define PRESERVE_ALL                    (PRESERVE_BLACK | PRESERVE_DKGRAY |
PRESERVE_LTGRAY | PRESERVE_WHITE)
#define PRESERVE_NONE                   0   //Backwards compatible



//Structure for UIToolConfigureForDisplay
typedef struct tagTOOLDISPLAYDATA
     {
     UINT        uDPI;       //Display driver DPI
     UINT        cyBar;      //Vertical size for a bar containing buttons.
     UINT        cxButton;   //Dimensions of a button.
     UINT        cyButton;
     UINT        cxImage;    //Dimensions of bitmap image
     UINT        cyImage;
     UINT        uIDImages;  //Standard resource ID for display-sensitive
images
     } TOOLDISPLAYDATA, FAR *LPTOOLDISPLAYDATA;



//Public functions in BTTNCUR.DLL
HCURSOR WINAPI UICursorLoad(UINT);
BOOL    WINAPI UIToolConfigureForDisplay(LPTOOLDISPLAYDATA);
BOOL    WINAPI UIToolButtonDraw(HDC, int, int, int, int, HBITMAP, int, int,
int, UINT);
BOOL    WINAPI UIToolButtonDrawTDD(HDC, int, int, int, int, HBITMAP, int,
int, int, UINT, LPTOOLDISPLAYDATA);


#ifdef __cplusplus
     }
#endif

#endif //_BTTNCUR_H_
```

GIZMOBAR.H   (INCLUDE Sample)

```
/*
 * GIZMOBAR.H
 * GizmoBar Version 1.01
 *
 * Public definitions for application that use the GizmoBar such as
 * messages, prototypes for API functions, notification codes, and
 * control styles.
 *
 * Copyright (c)1993-1994 Microsoft Corporation, All Rights Reserved
 *
 * Kraig Brockschmidt, Software Design Engineer
 * Microsoft Systems Developer Relations
 *
 * Internet  :  kraigb@microsoft.com
 * Compuserve:  >INTERNET:kraigb@microsoft.com
 */


#ifndef _GIZMOBAR_H_
#define _GIZMOBAR_H_

#include <bttncur.h>

#ifdef __cplusplus
extern "C"
    {
#endif


//Classname
#define CLASS_GIZMOBAR  TEXT("gizmobar")


//Message API Functions
HWND    WINAPI GBHwndAssociateSet(HWND, HWND);
HWND    WINAPI GBHwndAssociateGet(HWND);

BOOL    WINAPI GBGizmoAddW(HWND, UINT, UINT, UINT, UINT, UINT, LPWSTR ,
HBITMAP, UINT, UINT);
BOOL    WINAPI GBGizmoAddA(HWND, UINT, UINT, UINT, UINT, UINT, LPSTR ,
HBITMAP, UINT, UINT);

int     WINAPI GBGizmoTextGetW(HWND, UINT, LPWSTR, UINT);
int     WINAPI GBGizmoTextGetA(HWND, UINT, LPSTR, UINT);

void    WINAPI GBGizmoTextSetW(HWND, UINT, LPWSTR);
void    WINAPI GBGizmoTextSetA(HWND, UINT, LPSTR);

#ifdef UNICODE
#define GBGizmoAdd      GBGizmoAddW
#define GBGizmoTextGet  GBGizmoTextGetW
#define GBGizmoTextSet  GBGizmoTextSetW
```

```
#else
#define GBGizmoAdd      GBGizmoAddA
#define GBGizmoTextGet  GBGizmoTextGetA
#define GBGizmoTextSet  GBGizmoTextSetA
#endif

BOOL    WINAPI GBGizmoRemove(HWND, UINT);

LRESULT WINAPI GBGizmoSendMessage(HWND, UINT, UINT, WPARAM, LPARAM);

BOOL    WINAPI GBGizmoShow(HWND, UINT, BOOL);
BOOL    WINAPI GBGizmoEnable(HWND, UINT, BOOL);
BOOL    WINAPI GBGizmoCheck(HWND, UINT, BOOL);
UINT    WINAPI GBGizmoFocusSet(HWND, UINT);
BOOL    WINAPI GBGizmoExist(HWND, UINT);

int     WINAPI GBGizmoTypeGet(HWND, UINT);

DWORD   WINAPI GBGizmoDataSet(HWND, UINT, DWORD);
DWORD   WINAPI GBGizmoDataGet(HWND, UINT);
BOOL    WINAPI GBGizmoNotifySet(HWND, UINT, BOOL);
BOOL    WINAPI GBGizmoNotifyGet(HWND, UINT);

UINT    WINAPI GBGizmoIntGet(HWND, UINT, BOOL FAR *, BOOL);
void    WINAPI GBGizmoIntSet(HWND, UINT, UINT, BOOL);



//Notification codes sent via WM_COMMAND from GBHwndAssociateSet
#define GBN_ASSOCIATEGAIN              1
#define GBN_ASSOCIATELOSS              2
#define GBN_GIZMOADDED                 3
#define GBN_GIZMOREMOVED               4

//Message equivalents for functions.
#define GBM_HWNDASSOCIATESET           (WM_USER+0)
#define GBM_HWNDASSOCIATEGET           (WM_USER+1)
#define GBM_GIZMOADDW                  (WM_USER+2)
#define GBM_GIZMOADDA                  (WM_USER+3)
#define GBM_GIZMOREMOVE                (WM_USER+4)
#define GBM_GIZMOSENDMESSAGE           (WM_USER+5)
#define GBM_GIZMOSHOW                  (WM_USER+6)
#define GBM_GIZMOENABLE                (WM_USER+7)
#define GBM_GIZMOCHECK                 (WM_USER+8)
#define GBM_GIZMOFOCUSSET              (WM_USER+9)
#define GBM_GIZMOEXIST                 (WM_USER+10)
#define GBM_GIZMOTYPEGET               (WM_USER+11)
#define GBM_GIZMODATASET               (WM_USER+12)
#define GBM_GIZMODATAGET               (WM_USER+13)
#define GBM_GIZMONOTIFYSET             (WM_USER+14)
#define GBM_GIZMONOTIFYGET             (WM_USER+15)
#define GBM_GIZMOTEXTGETA              (WM_USER+16)
#define GBM_GIZMOTEXTGETW              (WM_USER+17)
#define GBM_GIZMOTEXTSETA              (WM_USER+18)
#define GBM_GIZMOTEXTSETW              (WM_USER+19)
```

```
#define GBM_GIZMOINTGET                  (WM_USER+20)
#define GBM_GIZMOINTSET                  (WM_USER+21)

#ifdef UNICODE
#define GBM_GIZMOTEXTGET GBM_GIZMOTEXTGETW
#define GBM_GIZMOTEXTSET GBM_GIZMOTEXTSETW
#define GBM_GIZMOADD GBM_GIZMOADDW
#else
#define GBM_GIZMOTEXTGET GBM_GIZMOTEXTGETA
#define GBM_GIZMOTEXTSET GBM_GIZMOTEXTSETA
#define GBM_GIZMOADD GBM_GIZMOADDA
#endif



/*
 * Structure passed in lParam of GBM_GIZMOADD that mirrors the
 * parameters to GBGizmoAdd.
 */

typedef struct
    {
    HWND        hWndParent;         //Parent window
    UINT        iType;              //Type of gizmo
    UINT        iGizmo;             //Position to create gizmo
    UINT        uID;                //Identifier of gizmo
    UINT        dx;                 //Dimensions of gizmo
    UINT        dy;
    LPSTR       pszText;            //Gizmo text
    HBITMAP     hBmp;               //Source of gizmo button image.
    UINT        iImage;             //Index of image from hBmp
    UINT        uState;             //Initial state of the gizmo.
    } CREATEGIZMOA, FAR *LPCREATEGIZMOA;
typedef struct
    {
    HWND        hWndParent;         //Parent window
    UINT        iType;              //Type of gizmo
    UINT        iGizmo;             //Position to create gizmo
    UINT        uID;                //Identifier of gizmo
    UINT        dx;                 //Dimensions of gizmo
    UINT        dy;
    LPWSTR      pszText;            //Gizmo text
    HBITMAP     hBmp;               //Source of gizmo button image.
    UINT        iImage;             //Index of image from hBmp
    UINT        uState;             //Initial state of the gizmo.
    } CREATEGIZMOW, FAR *LPCREATEGIZMOW;

#ifdef UNICODE
typedef CREATEGIZMOW    CREATEGIZMO;
typedef LPCREATEGIZMOW  LPCREATEGIZMO;
#define CBCREATEGIZMO sizeof(CREATEGIZMOW)
#else
typedef CREATEGIZMOA    CREATEGIZMO;
typedef LPCREATEGIZMOA  LPCREATEGIZMO;
#define CBCREATEGIZMO sizeof(CREATEGIZMOA)
```

```
#endif


//For GBM_GIZMOSENDMESSAGE
typedef struct
    {
    UINT        iMsg;
    WPARAM      wParam;
    LPARAM      lParam;
    } GBMSG, FAR * LPGBMSG;

#define CBGBMSG sizeof(GBMSG);

//For GBM_GIZMOGETTEXT
typedef struct
    {
    LPSTR       psz;
    UINT        cch;
    } GBGETTEXTA, FAR * LPGBGETTEXTA;
typedef struct
    {
    LPWSTR      psz;
    UINT        cch;
    } GBGETTEXTW, FAR * LPGBGETTEXTW;

#ifdef UNICODE
typedef GBGETTEXTW  GBGETTEXT;
typedef LPGBGETTEXTW  LPGBGETTEXT;
#define CBGBGETTEXT sizeof(GBGETTEXTW);
#else
typedef GBGETTEXTA  GBGETTEXT;
typedef LPGBGETTEXTA  LPGBGETTEXT;
#define CBGBGETTEXT sizeof(GBGETTEXTA);
#endif


//For GBM_GIZMOGETINT
typedef struct
    {
    BOOL        fSigned;
    BOOL        fSuccess;
    } GBGETINT, FAR * LPGBGETINT;

#define CBGBGETINT sizeof(GBGETINT);


//For GBM_GIZMOSETINT
typedef struct
    {
    UINT        uValue;
    BOOL        fSigned;
    } GBSETINT, FAR * LPGBSETINT;

#define CBGBSETINT sizeof(GBSETINT);
```

```
//Gizmo control types.  DO NOT CHANGE THESE!
#define GIZMOTYPE_EDIT                  0x0001
#define GIZMOTYPE_LISTBOX               0x0002
#define GIZMOTYPE_COMBOBOX              0x0004
#define GIZMOTYPE_BUTTONNORMAL          0x0008
#define GIZMOTYPE_TEXT                  0x0010
#define GIZMOTYPE_SEPARATOR             0x0020
#define GIZMOTYPE_BUTTONATTRIBUTEIN     0x0040
#define GIZMOTYPE_BUTTONATTRIBUTEEX     0x0080
#define GIZMOTYPE_BUTTONCOMMAND         0x0100


//Generic state flags for non-buttons based on BTTNCUR.H's groups.
#define GIZMO_NORMAL                    (BUTTONGROUP_ACTIVE)
#define GIZMO_DISABLED                  (BUTTONGROUP_DISABLED)



#ifdef __cplusplus
    }   //Match with extern "C" above.
#endif



#endif //_GIZMOBAR_H_
```

MSGFILTR.H (INCLUDE Sample)

```
/*************************************************************************
**
**    OLE 2 Utility Code
**
**    msgfiltr.h
**
**    This file contains Private definitions, structures, types, and
**    function prototypes for the OleStdMessageFilter implementation of
**    the IMessageFilter interface.
**    This file is part of the OLE 2.0 User Interface support library.
**
**    (c) Copyright Microsoft Corp. 1990 - 1992 All Rights Reserved
**
*************************************************************************/

#if !defined( _MSGFILTR_H_ )
#define _MSGFILTR_H_

#ifndef RC_INVOKED
#pragma message ("INCLUDING MSGFILTR.H from " __FILE__)
#endif  /* RC_INVOKED */

#include "ansiapi.h"

// Message Pending callback procedure
typedef BOOL (CALLBACK* MSGPENDINGPROC)(MSG FAR *);

// HandleInComingCall callback procedure
typedef DWORD (CALLBACK* HANDLEINCOMINGCALLBACKPROC)
   (
      DWORD                dwCallType,
      HTASK                htaskCaller,
      DWORD                dwTickCount,
      LPINTERFACEINFO      lpInterfaceInfo
   );

/* PUBLIC FUNCTIONS */
STDAPI_(LPMESSAGEFILTER) OleStdMsgFilter_Create(
      HWND hWndParent,
      LPOLESTR szAppName,
      MSGPENDINGPROC lpfnCallback,
      LPFNOLEUIHOOK  lpfnOleUIHook         // Busy dialog hook callback
);

STDAPI_(void) OleStdMsgFilter_SetInComingCallStatus(
      LPMESSAGEFILTER lpThis, DWORD dwInComingCallStatus);

STDAPI_(DWORD) OleStdMsgFilter_GetInComingCallStatus(
      LPMESSAGEFILTER lpThis);

STDAPI_(HANDLEINCOMINGCALLBACKPROC)
   OleStdMsgFilter_SetHandleInComingCallbackProc(
      LPMESSAGEFILTER             lpThis,
      HANDLEINCOMINGCALLBACKPROC  lpfnHandleInComingCallback);

STDAPI_(BOOL) OleStdMsgFilter_EnableBusyDialog(
      LPMESSAGEFILTER lpThis, BOOL fEnable);

STDAPI_(BOOL) OleStdMsgFilter_EnableNotRespondingDialog(
      LPMESSAGEFILTER lpThis, BOOL fEnable);

STDAPI_(HWND) OleStdMsgFilter_SetParentWindow(
      LPMESSAGEFILTER lpThis, HWND hWndParent);


#endif // _MSGFILTR_H_
_
```

OLE2UI.H   (INCLUDE Sample)

```
/*
 * OLE2UI.H
 *
 * Published definitions, structures, types, and function prototypes for the
 * OLE 2.0 User Interface support library.
 *
 * Copyright (c)1993 Microsoft Corporation, All Rights Reserved
 */

/* NOTE: All dialog and string resource ID's defined in this file are
 *    in the range:
 *          32248 - 32504   (0x7DF8 - 0x7EF8)
*/


#ifndef _OLE2UI_H_
#define _OLE2UI_H_

#if !defined(__cplusplus) && !defined( __TURBOC__)
#define NONAMELESSUNION     // use strict ANSI standard (for DVOBJ.H)
#endif

#include <windows.h>
#include <shellapi.h>
#include <ole2.h>
#include <string.h>
#include <dlgs.h>           //For fileopen dlg; standard include
#include "ansiapi.h"

/*************************************************************************
** DEBUG ASSERTION ROUTINES
*************************************************************************/

#if DBG
#include <assert.h>
#define FnAssert(lpstrExpr, lpstrMsg, lpstrFileName, iLine)     \
        (_assert(lpstrMsg ? lpstrMsg : lpstrExpr,               \
                 lpstrFileName,                                 \
                 iLine), NOERROR)
#endif //DBG

#include "olestd.h"

#ifdef __TURBOC__
#define _getcwd getcwd
#define _itoa   itoa
#define __max   max
#define _find_t find_t
#endif // __TURBOC__

#ifdef WIN32
#define _fmemset memset
```

```
#define _fmemcpy memcpy
#define _fmemcmp memcmp
#define _fstrcpy strcpy
#define _fstrncpy strncpy
#define _fstrlen strlen
#define _fstrrchr strrchr
#define _fstrtok strtok

// BUGBUG32: isspace function does not seem to work properly
#undef isspace
#define isspace(j) (j==' ' || j=='\t' || j=='\n')
#endif  // WIN32

#if !defined( EXPORT )
#ifdef WIN32
#define EXPORT
#else
#define EXPORT  __export
#endif  // WIN32
#endif  // !EXPORT

/*
 * Initialization / Uninitialization routines.  OleUIInitialize
 * must be called prior to using any functions in OLE2UI, and
OleUIUnInitialize
 * must be called before you app shuts down and when you are done using the
 * library.
 *
 * NOTE:  If you are using the DLL version of this library, these functions
 * are automatically called in the DLL's LibMain and WEP, so you should
 * not call them directly from your application.
 */

// Backward compatibility with older library
#define OleUIUninitialize OleUIUnInitialize

STDAPI_(BOOL) OleUIInitialize(HINSTANCE hInstance, HINSTANCE hPrevInst);
STDAPI_(BOOL) OleUIUninitialize(void);

#if !defined( SZCLASSICONBOX )
#define SZCLASSICONBOX  "ole2uiIBCls"
#endif

#if !defined( SZCLASSRESULTIMAGE )
#define SZCLASSRESULTIMAGE  "ole2uiRICls"
#endif

// object count, used to support DllCanUnloadNow and OleUICanUnloadNow
extern DWORD g_dwObjectCount;

STDAPI OleUICanUnloadNow(void);
STDAPI OleUILockLibrary(BOOL fLock);


//Dialog Identifiers as passed in Help messages to identify the source.
```

```
#define IDD_INSERTOBJECT        32248
#define IDD_CHANGEICON          32249
#define IDD_CONVERT             32250
#define IDD_PASTESPECIAL        32251
#define IDD_EDITLINKS           32252
#define IDD_FILEOPEN            32253
#define IDD_BUSY                32254
#define IDD_UPDATELINKS         32255
#define IDD_CANNOTUPDATELINK    32256
#define IDD_CHANGESOURCE        32257
#define IDD_INSERTFILEBROWSE    32258
#define IDD_CHANGEICONBROWSE    32259

// The following Dialogs are message dialogs used by OleUIPromptUser API
#define IDD_LINKSOURCEUNAVAILABLE   32260
#define IDD_SERVERNOTREG        32261
#define IDD_LINKTYPECHANGED     32262
#define IDD_SERVERNOTFOUND      32263
#define IDD_OUTOFMEMORY         32264

// Stringtable identifers
#define IDS_OLE2UIUNKNOWN       32300
#define IDS_OLE2UILINK          32301
#define IDS_OLE2UIOBJECT        32302
#define IDS_OLE2UIEDIT          32303
#define IDS_OLE2UICONVERT       32304
#define IDS_OLE2UIEDITLINKCMD_1VERB     32305
#define IDS_OLE2UIEDITOBJECTCMD_1VERB   32306
#define IDS_OLE2UIEDITLINKCMD_NVERB     32307
#define IDS_OLE2UIEDITOBJECTCMD_NVERB   32308
#define IDS_OLE2UIEDITNOOBJCMD  32309
// def. icon label (usu. "Document")
#define IDS_DEFICONLABEL        32310
#define IDS_OLE2UIPASTELINKEDTYPE  32311


#define IDS_FILTERS             32320
#define IDS_ICONFILTERS         32321
#define IDS_BROWSE              32322

//Resource identifiers for bitmaps
#define IDB_RESULTSEGA                  32325
#define IDB_RESULTSVGA                  32326
#define IDB_RESULTSHIRESVGA             32327


//Missing from windows.h
#ifndef PVOID
typedef VOID *PVOID;
#endif


//Hook type used in all structures.
typedef UINT (CALLBACK *LPFNOLEUIHOOK)(HWND, UINT, WPARAM, LPARAM);
```

```
//Strings for registered messages
#define SZOLEUI_MSG_HELP                "OLEUI_MSG_HELP"
#define SZOLEUI_MSG_ENDDIALOG           "OLEUI_MSG_ENDDIALOG"
#define SZOLEUI_MSG_BROWSE              "OLEUI_MSG_BROWSE"
#define SZOLEUI_MSG_CHANGEICON          "OLEUI_MSG_CHANGEICON"
#define SZOLEUI_MSG_CLOSEBUSYDIALOG     "OLEUI_MSG_CLOSEBUSYDIALOG"
#define SZOLEUI_MSG_FILEOKSTRING        "OLEUI_MSG_FILEOKSTRING"

//Standard error definitions
#define OLEUI_FALSE                     0
#define OLEUI_SUCCESS                   1     //No error, same as OLEUI_OK
#define OLEUI_OK                        1     //OK button pressed
#define OLEUI_CANCEL                    2     //Cancel button pressed

#define OLEUI_ERR_STANDARDMIN           100
#define OLEUI_ERR_STRUCTURENULL         101   //Standard field validation
#define OLEUI_ERR_STRUCTUREINVALID      102
#define OLEUI_ERR_CBSTRUCTINCORRECT     103
#define OLEUI_ERR_HWNDOWNERINVALID      104
#define OLEUI_ERR_LPSZCAPTIONINVALID    105
#define OLEUI_ERR_LPFNHOOKINVALID       106
#define OLEUI_ERR_HINSTANCEINVALID      107
#define OLEUI_ERR_LPSZTEMPLATEINVALID   108
#define OLEUI_ERR_HRESOURCEINVALID      109

#define OLEUI_ERR_FINDTEMPLATEFAILURE   110   //Initialization errors
#define OLEUI_ERR_LOADTEMPLATEFAILURE   111
#define OLEUI_ERR_DIALOGFAILURE         112
#define OLEUI_ERR_LOCALMEMALLOC         113
#define OLEUI_ERR_GLOBALMEMALLOC        114
#define OLEUI_ERR_LOADSTRING            115

#define OLEUI_ERR_STANDARDMAX           116   //Start here for specific
errors.



//Help Button Identifier
#define ID_OLEUIHELP                    99

// Help button for fileopen.dlg  (need this for resizing) 1038 is pshHelp
#define IDHELP  1038

// Static text control (use this instead of -1 so things work correctly for
// localization
#define  ID_STATIC                      98

//Maximum key size we read from the RegDB.
#define OLEUI_CCHKEYMAX                 256  // make any changes to this in
geticon.c too

//Maximum verb length and length of Object menu
#define OLEUI_CCHVERBMAX                32
#define OLEUI_OBJECTMENUMAX             256
```

```
//Maximum MS-DOS pathname.
#define OLEUI_CCHPATHMAX                256 // make any changes to this in
geticon.c too
#define OLEUI_CCHFILEMAX                13

//Icon label length
#define OLEUI_CCHLABELMAX               40  // make any changes to this in
geticon.c too

//Length of the CLSID string
#define OLEUI_CCHCLSIDSTRING            39


/*
 * What follows here are first function prototypes for general utility
 * functions, then sections laid out by dialog.  Each dialog section
 * defines the dialog structure, the API prototype, flags for the dwFlags
 * field, the dialog-specific error values, and dialog control IDs (for
 * hooks and custom templates.
 */


//Miscellaneous utility functions.
STDAPI_(BOOL) OleUIAddVerbMenu(LPOLEOBJECT lpOleObj,
                      LPOLESTR lpszShortType,
                      HMENU hMenu,
                      UINT uPos,
                      UINT uIDVerbMin,
                      UINT uIDVerbMax,
                      BOOL bAddConvert,
                      UINT idConvert,
                      HMENU FAR *lphMenu);

//Metafile utility functions
WINOLEAPI_(HGLOBAL) OleUIMetafilePictFromIconAndLabel(HICON, LPOLESTR,
LPOLESTR, UINT);
STDAPI_(void)    OleUIMetafilePictIconFree(HGLOBAL);
STDAPI_(BOOL)    OleUIMetafilePictIconDraw(HDC, LPRECT, HGLOBAL, BOOL);
STDAPI_(UINT)    OleUIMetafilePictExtractLabel(HGLOBAL, LPOLESTR, UINT,
LPDWORD);
STDAPI_(HICON)   OleUIMetafilePictExtractIcon(HGLOBAL);
STDAPI_(BOOL)    OleUIMetafilePictExtractIconSource(HGLOBAL,LPOLESTR,UINT
FAR *);




/*************************************************************************
** INSERT OBJECT DIALOG
*************************************************************************/


typedef struct tagOLEUIINSERTOBJECT
```

```
   {
   //These IN fields are standard across all OLEUI dialog functions.
   DWORD           cbStruct;         //Structure Size
   DWORD           dwFlags;          //IN-OUT:  Flags
   HWND            hWndOwner;        //Owning window
   LPCOLESTR       lpszCaption;      //Dialog caption bar contents
   LPFNOLEUIHOOK   lpfnHook;         //Hook callback
   LPARAM          lCustData;        //Custom data to pass to hook
   HINSTANCE       hInstance;        //Instance for customized template name
   LPCOLESTR       lpszTemplate;     //Customized template name
   HRSRC           hResource;        //Customized template handle

   //Specifics for OLEUIINSERTOBJECT.  All are IN-OUT unless otherwise spec.
   CLSID           clsid;            //Return space for class ID
   LPOLESTR        lpszFile;         //Filename for inserts or links
   UINT            cchFile;          //Size of lpszFile buffer:
OLEUI_CCHPATHMAX
   UINT            cClsidExclude;    //IN only:  CLSIDs in lpClsidExclude
   LPCLSID         lpClsidExclude;   //List of CLSIDs to exclude from
listing.

   //Specific to create objects if flags say so
   IID             iid;              //Requested interface on creation.
   DWORD           oleRender;        //Rendering option
   LPFORMATETC     lpFormatEtc;      //Desired format
   LPOLECLIENTSITE lpIOleClientSite; //Site to be use for the object.
   LPSTORAGE       lpIStorage;       //Storage used for the object
   LPVOID FAR     *ppvObj;           //Where the object is returned.
   SCODE           sc;               //Result of creation calls.
   HGLOBAL         hMetaPict;        //OUT: METAFILEPICT containing iconic
aspect.
                              //IFF we couldn't stuff it in the cache.
   } OLEUIINSERTOBJECT, *POLEUIINSERTOBJECT, FAR *LPOLEUIINSERTOBJECT;

//API prototype
STDAPI_(UINT) OleUIInsertObject(LPOLEUIINSERTOBJECT);


//Insert Object flags
#define IOF_SHOWHELP                0x00000001L
#define IOF_SELECTCREATENEW         0x00000002L
#define IOF_SELECTCREATEFROMFILE    0x00000004L
#define IOF_CHECKLINK               0x00000008L
#define IOF_CHECKDISPLAYASICON      0x00000010L
#define IOF_CREATENEWOBJECT         0x00000020L
#define IOF_CREATEFILEOBJECT        0x00000040L
#define IOF_CREATELINKOBJECT        0x00000080L
#define IOF_DISABLELINK             0x00000100L
#define IOF_VERIFYSERVERSEXIST      0x00000200L
#define IOF_DISABLEDISPLAYASICON    0x00000400L


//Insert Object specific error codes
#define OLEUI_IOERR_LPSZFILEINVALID         (OLEUI_ERR_STANDARDMAX+0)
#define OLEUI_IOERR_LPSZLABELINVALID        (OLEUI_ERR_STANDARDMAX+1)
```

```
#define OLEUI_IOERR_HICONINVALID            (OLEUI_ERR_STANDARDMAX+2)
#define OLEUI_IOERR_LPFORMATETCINVALID      (OLEUI_ERR_STANDARDMAX+3)
#define OLEUI_IOERR_PPVOBJINVALID           (OLEUI_ERR_STANDARDMAX+4)
#define OLEUI_IOERR_LPIOLECLIENTSITEINVALID (OLEUI_ERR_STANDARDMAX+5)
#define OLEUI_IOERR_LPISTORAGEINVALID       (OLEUI_ERR_STANDARDMAX+6)
#define OLEUI_IOERR_SCODEHASERROR           (OLEUI_ERR_STANDARDMAX+7)
#define OLEUI_IOERR_LPCLSIDEXCLUDEINVALID   (OLEUI_ERR_STANDARDMAX+8)
#define OLEUI_IOERR_CCHFILEINVALID          (OLEUI_ERR_STANDARDMAX+9)


//Insert Object Dialog identifiers
#define ID_IO_CREATENEW                 2100
#define ID_IO_CREATEFROMFILE            2101
#define ID_IO_LINKFILE                  2102
#define ID_IO_OBJECTTYPELIST            2103
#define ID_IO_DISPLAYASICON             2104
#define ID_IO_CHANGEICON                2105
#define ID_IO_FILE                      2106
#define ID_IO_FILEDISPLAY               2107
#define ID_IO_RESULTIMAGE               2108
#define ID_IO_RESULTTEXT                2109
#define ID_IO_ICONDISPLAY               2110
#define ID_IO_OBJECTTYPETEXT            2111
#define ID_IO_FILETEXT                  2112
#define ID_IO_FILETYPE                  2113

// Strings in OLE2UI resources
#define IDS_IORESULTNEW                 32400
#define IDS_IORESULTNEWICON             32401
#define IDS_IORESULTFROMFILE1           32402
#define IDS_IORESULTFROMFILE2           32403
#define IDS_IORESULTFROMFILEICON2       32404
#define IDS_IORESULTLINKFILE1           32405
#define IDS_IORESULTLINKFILE2           32406
#define IDS_IORESULTLINKFILEICON1       32407
#define IDS_IORESULTLINKFILEICON2       32408

/*************************************************************************
** PASTE SPECIAL DIALOG
*************************************************************************/

// Maximum number of link types
#define     PS_MAXLINKTYPES  8

//NOTE: OLEUIPASTEENTRY and OLEUIPASTEFLAG structs are defined in OLESTD.H

typedef struct tagOLEUIPASTESPECIAL
   {
   //These IN fields are standard across all OLEUI dialog functions.
   DWORD           cbStruct;       //Structure Size
   DWORD           dwFlags;        //IN-OUT:  Flags
   HWND            hWndOwner;      //Owning window
   LPCOLESTR       lpszCaption;    //Dialog caption bar contents
   LPFNOLEUIHOOK   lpfnHook;       //Hook callback
   LPARAM          lCustData;      //Custom data to pass to hook
   HINSTANCE       hInstance;      //Instance for customized template name
   LPCOLESTR       lpszTemplate;   //Customized template name
   HRSRC           hResource;      //Customized template handle

   //Specifics for OLEUIPASTESPECIAL.

   //IN  fields
   LPDATAOBJECT    lpSrcDataObj;       //Source IDataObject* (on the
                           // clipboard) for data to paste

   LPOLEUIPASTEENTRY arrPasteEntries;  //OLEUIPASTEENTRY array which
                           // specifies acceptable formats. See
                           // OLEUIPASTEENTRY for more info.
   int             cPasteEntries;      //No. of OLEUIPASTEENTRY array
entries

   UINT        FAR *arrLinkTypes;      //List of link types that are
                           // acceptable. Link types are referred
                           // to using OLEUIPASTEFLAGS in
                           // arrPasteEntries
   int             cLinkTypes;         //Number of link types
   UINT            cClsidExclude;      //Number of CLSIDs in lpClsidExclude
   LPCLSID         lpClsidExclude;     //List of CLSIDs to exclude from
list.

   //OUT fields
   int             nSelectedIndex;     //Index of arrPasteEntries[] that the
                           // user selected
   BOOL            fLink;              //Indicates if Paste or Paste Link
was
                           // selected by the user
   HGLOBAL         hMetaPict;          //Handle to Metafile containing icon
                           // and icon title selected by the user
                           // Use the Metafile utility functions
                           // defined in this header to
                           // manipulate hMetaPict
   SIZEL           sizel;              // size of object/link in its source
                           //  if the display aspect chosen by
                           //  the user matches the aspect
                           //  displayed in the source. if
                           //  different aspect is chosen then
                           //  sizel.cx=sizel.cy=0 is returned.
                           //  sizel displayed in source is
                           //  retrieved from the
                           //  ObjectDescriptor if fLink is FALSE
                           //  LinkSrcDescriptor if fLink is TRUE
   } OLEUIPASTESPECIAL, *POLEUIPASTESPECIAL, FAR *LPOLEUIPASTESPECIAL;


//API to bring up PasteSpecial dialog
STDAPI_(UINT) OleUIPasteSpecial(LPOLEUIPASTESPECIAL);


//Paste Special flags
// Show Help button. IN flag.
```

```
#define PSF_SHOWHELP                0x00000001L

//Select Paste radio button at dialog startup. This is the default if
// PSF_SELECTPASTE or PSF_SELECTPASTELINK are not specified. Also specifies
// state of button on dialog termination. IN/OUT flag.
#define PSF_SELECTPASTE             0x00000002L

//Select PasteLink radio button at dialog startup. Also specifies state of
// button on dialog termination. IN/OUT flag.
#define PSF_SELECTPASTELINK         0x00000004L

//Specfies if DisplayAsIcon button was checked on dialog termination. OUT
flag
#define PSF_CHECKDISPLAYASICON      0x00000008L
#define PSF_DISABLEDISPLAYASICON    0x00000010L


//Paste Special specific error codes
#define OLEUI_IOERR_SRCDATAOBJECTINVALID      (OLEUI_ERR_STANDARDMAX+0)
#define OLEUI_IOERR_ARRPASTEENTRIESINVALID    (OLEUI_ERR_STANDARDMAX+1)
#define OLEUI_IOERR_ARRLINKTYPESINVALID       (OLEUI_ERR_STANDARDMAX+2)
#define OLEUI_PSERR_CLIPBOARDCHANGED          (OLEUI_ERR_STANDARDMAX+3)

//Paste Special Dialog identifiers
#define ID_PS_PASTE                    500
#define ID_PS_PASTELINK                501
#define ID_PS_SOURCETEXT               502
#define ID_PS_PASTELIST                503
#define ID_PS_PASTELINKLIST            504
#define ID_PS_DISPLAYLIST              505
#define ID_PS_DISPLAYASICON            506
#define ID_PS_ICONDISPLAY              507
#define ID_PS_CHANGEICON               508
#define ID_PS_RESULTIMAGE              509
#define ID_PS_RESULTTEXT               510
#define ID_PS_RESULTGROUP              511
#define ID_PS_STXSOURCE                512
#define ID_PS_STXAS                    513

// Paste Special String IDs
#define IDS_PSPASTEDATA                32410
#define IDS_PSPASTEOBJECT              32411
#define IDS_PSPASTEOBJECTASICON        32412
#define IDS_PSPASTELINKDATA            32413
#define IDS_PSPASTELINKOBJECT          32414
#define IDS_PSPASTELINKOBJECTASICON    32415
#define IDS_PSNONOLE                   32416
#define IDS_PSUNKNOWNTYPE              32417
#define IDS_PSUNKNOWNSRC               32418
#define IDS_PSUNKNOWNAPP               32419


/*************************************************************************
** EDIT LINKS DIALOG
*************************************************************************/
```

```
/* IOleUILinkContainer Interface
** -----------------------------
**    This interface must be implemented by container applications that
**    want to use the EditLinks dialog. the EditLinks dialog calls back
**    to the container app to perform the OLE functions to manipulate
**    the links within the container.
*/

#define LPOLEUILINKCONTAINER     IOleUILinkContainer FAR*

#undef  INTERFACE
#define INTERFACE   IOleUILinkContainer

DECLARE_INTERFACE_(IOleUILinkContainer, IUnknown)
{
   //*** IUnknown methods ***/
   STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
   STDMETHOD_(ULONG,AddRef) (THIS) PURE;
   STDMETHOD_(ULONG,Release) (THIS) PURE;

   STDMETHOD_(DWORD,GetNextLink) (THIS_ DWORD dwLink) PURE;
   STDMETHOD(SetLinkUpdateOptions) (THIS_ DWORD dwLink, DWORD dwUpdateOpt)
PURE;
   STDMETHOD(GetLinkUpdateOptions) (THIS_ DWORD dwLink, DWORD FAR*
lpdwUpdateOpt) PURE;
   STDMETHOD(SetLinkSource) (THIS_
         DWORD       dwLink,
         LPOLESTR    lpszDisplayName,
         ULONG       lenFileName,
         ULONG FAR*  pchEaten,
         BOOL        fValidateSource) PURE;
   STDMETHOD(GetLinkSource) (THIS_
         DWORD       dwLink,
         LPOLESTR FAR*  lplpszDisplayName,
         ULONG FAR*  lplenFileName,
         LPOLESTR FAR*  lplpszFullLinkType,
         LPOLESTR FAR*  lplpszShortLinkType,
         BOOL FAR*   lpfSourceAvailable,
         BOOL FAR*   lpfIsSelected) PURE;
   STDMETHOD(OpenLinkSource) (THIS_ DWORD dwLink) PURE;
   STDMETHOD(UpdateLink) (THIS_
         DWORD dwLink,
         BOOL fErrorMessage,
         BOOL fErrorAction) PURE;
   STDMETHOD(CancelLink) (THIS_ DWORD dwLink) PURE;
};


typedef struct tagOLEUIEDITLINKS
   {
   //These IN fields are standard across all OLEUI dialog functions.
   DWORD           cbStruct;       //Structure Size
   DWORD           dwFlags;        //IN-OUT:  Flags
   HWND            hWndOwner;      //Owning window
   LPCOLESTR       lpszCaption;    //Dialog caption bar contents
   LPFNOLEUIHOOK   lpfnHook;       //Hook callback
   LPARAM          lCustData;      //Custom data to pass to hook
   HINSTANCE       hInstance;      //Instance for customized template name
   LPCOLESTR       lpszTemplate;   //Customized template name
   HRSRC           hResource;      //Customized template handle

   //Specifics for OLEUI<STRUCT>.  All are IN-OUT unless otherwise spec.

   LPOLEUILINKCONTAINER lpOleUILinkContainer;  //IN: Interface to manipulate
                                    //links in the container
   } OLEUIEDITLINKS, *POLEUIEDITLINKS, FAR *LPOLEUIEDITLINKS;


//API Prototype
STDAPI_(UINT) OleUIEditLinks(LPOLEUIEDITLINKS);


// Edit Links flags
#define ELF_SHOWHELP                    0x00000001L
#define ELF_DISABLEUPDATENOW            0x00000002L
#define ELF_DISABLEOPENSOURCE           0x00000004L
#define ELF_DISABLECHANGESOURCE         0x00000008L
#define ELF_DISABLECANCELLINK           0x00000010L

// Edit Links Dialog identifiers
#define ID_EL_CHANGESOURCE                 201
#define ID_EL_AUTOMATIC                    202
#define ID_EL_CLOSE                        208
#define ID_EL_CANCELLINK                   209
#define ID_EL_UPDATENOW                    210
#define ID_EL_OPENSOURCE                   211
#define ID_EL_MANUAL                       212
#define ID_EL_LINKSOURCE                   216
#define ID_EL_LINKTYPE                     217
#define ID_EL_UPDATE                       218
#define ID_EL_NULL                         -1
#define ID_EL_LINKSLISTBOX                 206
#define ID_EL_COL1                         220
#define ID_EL_COL2                         221
#define ID_EL_COL3                         222



/*************************************************************************
** CHANGE ICON DIALOG
*************************************************************************/

typedef struct tagOLEUICHANGEICON
   {
   //These IN fields are standard across all OLEUI dialog functions.
   DWORD           cbStruct;       //Structure Size
   DWORD           dwFlags;        //IN-OUT:  Flags
   HWND            hWndOwner;      //Owning window
   LPCOLESTR       lpszCaption;    //Dialog caption bar contents
   LPFNOLEUIHOOK   lpfnHook;       //Hook callback
   LPARAM          lCustData;      //Custom data to pass to hook
   HINSTANCE       hInstance;      //Instance for customized template name
   LPCOLESTR       lpszTemplate;   //Customized template name
   HRSRC           hResource;      //Customized template handle

   //Specifics for OLEUICHANGEICON.  All are IN-OUT unless otherwise spec.
   HGLOBAL         hMetaPict;      //Current and final image.  Source of the
                           //icon is embedded in the metafile itself.
   CLSID           clsid;          //IN only: class used to get Default icon
   OLECHAR         szIconExe[OLEUI_CCHPATHMAX];
   int             cchIconExe;
   } OLEUICHANGEICON, *POLEUICHANGEICON, FAR *LPOLEUICHANGEICON;


//API prototype
STDAPI_(UINT) OleUIChangeIcon(LPOLEUICHANGEICON);


//Change Icon flags
#define CIF_SHOWHELP                0x00000001L
#define CIF_SELECTCURRENT           0x00000002L
#define CIF_SELECTDEFAULT           0x00000004L
#define CIF_SELECTFROMFILE          0x00000008L
#define CIF_USEICONEXE              0x0000000aL


//Change Icon specific error codes
#define OLEUI_CIERR_MUSTHAVECLSID           (OLEUI_ERR_STANDARDMAX+0)
#define OLEUI_CIERR_MUSTHAVECURRENTMETAFILE (OLEUI_ERR_STANDARDMAX+1)
#define OLEUI_CIERR_SZICONEXEINVALID        (OLEUI_ERR_STANDARDMAX+2)


//Change Icon Dialog identifiers
#define ID_GROUP                    120
#define ID_CURRENT                  121
#define ID_CURRENTICON              122
#define ID_DEFAULT                  123
#define ID_DEFAULTICON              124
#define ID_FROMFILE                 125
#define ID_FROMFILEEDIT             126
#define ID_ICONLIST                 127
#define ID_LABEL                    128
#define ID_LABELEDIT                129
#define ID_BROWSE                   130
#define ID_RESULTICON               132
#define ID_RESULTLABEL              133

// Stringtable defines for Change Icon
#define IDS_CINOICONSINFILE         32430
#define IDS_CIINVALIDFILE           32431
#define IDS_CIFILEACCESS            32432
#define IDS_CIFILESHARE             32433
```

```
#define IDS_CIFILEOPENFAIL          32434



/*************************************************************************
** CONVERT DIALOG
*************************************************************************/

typedef struct tagOLEUICONVERT
   {
   //These IN fields are standard across all OLEUI dialog functions.
   DWORD            cbStruct;         //Structure Size
   DWORD            dwFlags;          //IN-OUT:  Flags
   HWND             hWndOwner;        //Owning window
   LPCOLESTR        lpszCaption;      //Dialog caption bar contents
   LPFNOLEUIHOOK    lpfnHook;         //Hook callback
   LPARAM           lCustData;        //Custom data to pass to hook
   HINSTANCE        hInstance;        //Instance for customized template name
   LPCOLESTR        lpszTemplate;     //Customized template name
   HRSRC            hResource;        //Customized template handle

   //Specifics for OLEUICONVERT.  All are IN-OUT unless otherwise spec.
   CLSID            clsid;            //Class ID sent in to dialog: IN only
   CLSID            clsidConvertDefault;  //Class ID to use as convert
default: IN only
   CLSID            clsidActivateDefault;  //Class ID to use as activate
default: IN only

   CLSID            clsidNew;         //Selected Class ID: OUT only
   DWORD            dvAspect;         //IN-OUT, either DVASPECT_CONTENT or
                             //DVASPECT_ICON
   WORD             wFormat;          //Original data format
   BOOL             fIsLinkedObject;  //IN only; true if object is linked
   HGLOBAL          hMetaPict;        //IN-OUT: METAFILEPICT containing
iconic aspect.
   LPOLESTR         lpszUserType;     //IN-OUT: user type name of original
class.
                             //  We'll do lookup if it's NULL.
                             //  This gets freed on exit.
   BOOL             fObjectsIconChanged;  // OUT; TRUE if ChangeIcon was
called (and not cancelled)
   LPOLESTR         lpszDefLabel;     //IN-OUT: default label to use for
icon.
                             //  if NULL, the short user type name
                             //  will be used. if the object is a
                             //  link, the caller should pass the
                             //  DisplayName of the link source
                             //  This gets freed on exit.

   UINT             cClsidExclude;    //IN: No. of CLSIDs in lpClsidExclude
   LPCLSID          lpClsidExclude;   //IN: List of CLSIDs to exclude from
list
   } OLEUICONVERT, *POLEUICONVERT, FAR *LPOLEUICONVERT;
```

```
//API prototype
STDAPI_(UINT) OleUIConvert(LPOLEUICONVERT);

// Determine if there is at least one class that can Convert or ActivateAs
// the given clsid.
STDAPI_(BOOL) OleUICanConvertOrActivateAs(
      REFCLSID    rClsid,
      BOOL        fIsLinkedObject,
      WORD        wFormat
);

//Convert Dialog flags

// IN only: Shows "HELP" button
#define CF_SHOWHELPBUTTON          0x00000001L

// IN only: lets you set the convert default object - the one that is
// selected as default in the convert listbox.
#define CF_SETCONVERTDEFAULT       0x00000002L


// IN only: lets you set the activate default object - the one that is
// selected as default in the activate listbox.

#define CF_SETACTIVATEDEFAULT       0x00000004L


// IN/OUT: Selects the "Convert To" radio button, is set on exit if
// this button was selected
#define CF_SELECTCONVERTTO         0x00000008L

// IN/OUT: Selects the "Activate As" radio button, is set on exit if
// this button was selected
#define CF_SELECTACTIVATEAS        0x00000010L
#define CF_DISABLEDISPLAYASICON    0x00000020L
#define CF_DISABLEACTIVATEAS       0x00000040L


//Convert specific error codes
#define OLEUI_CTERR_CLASSIDINVALID      (OLEUI_ERR_STANDARDMAX+1)
#define OLEUI_CTERR_DVASPECTINVALID     (OLEUI_ERR_STANDARDMAX+2)
#define OLEUI_CTERR_CBFORMATINVALID     (OLEUI_ERR_STANDARDMAX+3)
#define OLEUI_CTERR_HMETAPICTINVALID    (OLEUI_ERR_STANDARDMAX+4)
#define OLEUI_CTERR_STRINGINVALID       (OLEUI_ERR_STANDARDMAX+5)


//Convert Dialog identifiers
#define IDCV_OBJECTTYPE             150
#define IDCV_DISPLAYASICON          152
#define IDCV_CHANGEICON             153
#define IDCV_ACTIVATELIST           154
#define IDCV_CONVERTTO              155
#define IDCV_ACTIVATEAS             156
#define IDCV_RESULTTEXT             157
#define IDCV_CONVERTLIST            158
```

```
#define IDCV_ICON                   159
#define IDCV_ICONLABEL1             160
#define IDCV_ICONLABEL2             161
#define IDCV_STXCURTYPE             162
#define IDCV_GRPRESULT              163
#define IDCV_STXCONVERTTO           164

// String IDs for Convert dialog
#define IDS_CVRESULTCONVERTLINK     32440
#define IDS_CVRESULTCONVERTTO       32441
#define IDS_CVRESULTNOCHANGE        32442
#define IDS_CVRESULTDISPLAYASICON   32443
#define IDS_CVRESULTACTIVATEAS      32444
#define IDS_CVRESULTACTIVATEDIFF    32445


/*************************************************************************
** BUSY DIALOG
*************************************************************************/

typedef struct tagOLEUIBUSY
   {
   //These IN fields are standard across all OLEUI dialog functions.
   DWORD           cbStruct;         //Structure Size
   DWORD           dwFlags;          //IN-OUT:  Flags ** NOTE ** this dialog
has no flags
   HWND            hWndOwner;        //Owning window
   LPCOLESTR       lpszCaption;      //Dialog caption bar contents
   LPFNOLEUIHOOK   lpfnHook;         //Hook callback
   LPARAM          lCustData;        //Custom data to pass to hook
   HINSTANCE       hInstance;        //Instance for customized template name
   LPCOLESTR       lpszTemplate;     //Customized template name
   HRSRC           hResource;        //Customized template handle

   //Specifics for OLEUIBUSY.
   HTASK           hTask;            //IN: HTask which is blocking
   HWND FAR *      lphWndDialog;     //IN: Dialog's HWND is placed here
   } OLEUIBUSY, *POLEUIBUSY, FAR *LPOLEUIBUSY;

//API prototype
STDAPI_(UINT) OleUIBusy(LPOLEUIBUSY);

// Flags for this dialog

// IN only: Disables "Cancel" button
#define BZ_DISABLECANCELBUTTON          0x00000001L

// IN only: Disables "Switch To..." button
#define BZ_DISABLESWITCHTOBUTTON        0x00000002L

// IN only: Disables "Retry" button
#define BZ_DISABLERETRYBUTTON           0x00000004L

// IN only: Generates a "Not Responding" dialog as opposed to the
// "Busy" dialog.  The wording in the text is slightly different, and
// the "Cancel" button is grayed out if you set this flag.
#define BZ_NOTRESPONDINGDIALOG          0x00000008L

// Busy specific error/return codes
#define OLEUI_BZERR_HTASKINVALID     (OLEUI_ERR_STANDARDMAX+0)

// SWITCHTOSELECTED is returned when user hit "switch to"
#define OLEUI_BZ_SWITCHTOSELECTED    (OLEUI_ERR_STANDARDMAX+1)

// RETRYSELECTED is returned when user hit "retry"
#define OLEUI_BZ_RETRYSELECTED       (OLEUI_ERR_STANDARDMAX+2)

// CALLUNBLOCKED is returned when call has been unblocked
#define OLEUI_BZ_CALLUNBLOCKED       (OLEUI_ERR_STANDARDMAX+3)

// Busy dialog identifiers
#define IDBZ_RETRY                      600
#define IDBZ_ICON                       601
#define IDBZ_MESSAGE1                   602
#define IDBZ_SWITCHTO                   604

// Busy dialog stringtable defines
#define IDS_BZRESULTTEXTBUSY            32447
#define IDS_BZRESULTTEXTNOTRESPONDING   32448

// Links dialog stringtable defines
#define IDS_LINK_AUTO           32450
#define IDS_LINK_MANUAL         32451
#define IDS_LINK_UNKNOWN        32452
#define IDS_LINKS               32453
#define IDS_FAILED              32454
#define IDS_CHANGESOURCE        32455
#define IDS_INVALIDSOURCE       32456
#define IDS_ERR_GETLINKSOURCE   32457
#define IDS_ERR_GETLINKUPDATEOPTIONS    32458
#define IDS_ERR_ADDSTRING       32459
#define IDS_CHANGEADDITIONALLINKS   32460
#define IDS_CLOSE               32461


/*************************************************************************
** PROMPT USER DIALOGS
*************************************************************************/
#define ID_PU_LINKS             900
#define ID_PU_TEXT              901
#define ID_PU_CONVERT           902
#define ID_PU_BROWSE            904
#define ID_PU_METER             905
#define ID_PU_PERCENT           906
#define ID_PU_STOP              907

// used for -1 ids in dialogs:
#define ID_DUMMY    999

/* inside ole2ui.c */
```

```
#ifdef __cplusplus
extern "C"
#endif
int EXPORT FAR CDECL OleUIPromptUser(WORD nTemplate, HWND hwndParent, ...);

#define UPDATELINKS_STARTDELAY  2000    // Delay before 1st link updates
                              //  to give the user a chance to
                              //  dismiss the dialog before any
                              //  links update.

STDAPI_(BOOL) OleUIUpdateLinks(
      LPOLEUILINKCONTAINER lpOleUILinkCntr,
      HWND hwndParent,
      LPOLESTR lpszTitle,
      int cLinks);


/*************************************************************************
** OLE OBJECT FEEDBACK EFFECTS
*************************************************************************/

#define OLEUI_HANDLES_USEINVERSE    0x00000001L
#define OLEUI_HANDLES_NOBORDER      0x00000002L
#define OLEUI_HANDLES_INSIDE        0x00000004L
#define OLEUI_HANDLES_OUTSIDE       0x00000008L


#define OLEUI_SHADE_FULLRECT        1
#define OLEUI_SHADE_BORDERIN        2
#define OLEUI_SHADE_BORDEROUT       3

/* objfdbk.c function prototypes */
STDAPI_(void) OleUIDrawHandles(LPRECT lpRect, HDC hdc, DWORD dwFlags, UINT
cSize, BOOL fDraw);
STDAPI_(void) OleUIDrawShading(LPRECT lpRect, HDC hdc, DWORD dwFlags, UINT
cWidth);
STDAPI_(void) OleUIShowObject(LPCRECT lprc, HDC hdc, BOOL fIsLink);


/*************************************************************************
** Hatch window definitions and prototypes                             **
*************************************************************************/
#define DEFAULT_HATCHBORDER_WIDTH   4

STDAPI_(BOOL) RegisterHatchWindowClass(HINSTANCE hInst);
STDAPI_(HWND) CreateHatchWindow(HWND hWndParent, HINSTANCE hInst);
STDAPI_(UINT) GetHatchWidth(HWND hWndHatch);
STDAPI_(void) GetHatchRect(HWND hWndHatch, LPRECT lpHatchRect);
STDAPI_(void) SetHatchRect(HWND hWndHatch, LPRECT lprcHatchRect);
STDAPI_(void) SetHatchWindowSize(
      HWND        hWndHatch,
      LPRECT      lprcIPObjRect,
      LPRECT      lprcClipRect,
      LPPOINT     lpptOffset
);
```

```
/*************************************************************************
** VERSION VERIFICATION INFORMATION
*************************************************************************/

// The following magic number is used to verify that the resources we bind
// to our EXE are the same "version" as the LIB (or DLL) file which
// contains these routines.  This is not the same as the Version information
// resource that we place in OLE2UI.RC, this is a special ID that we will
// have compiled in to our EXE.  Upon initialization of OLE2UI, we will
// look in our resources for an RCDATA called "VERIFICATION" (see
OLE2UI.RC),
// and make sure that the magic number there equals the magic number below.

#define OLEUI_VERSION_MAGIC 0x4D42

#endif  //_OLE2UI_H_
```

OLESAMPL.MAK   (INCLUDE Sample)

```
#+--------------------------------------------------------------------------
-
#
#  Microsoft Windows
#  Copyright (C) Microsoft Corporation, 1992 - 1994.
#
#  File:       olesampl.mak
#
#  Contents:   common makefile for use by the SDK OLE samples
#
#              This makefile is included in place of ntwin32.mak
#              at the top of all the OLE samples.
#
#  History:     8-23-94   stevebl   Created
#
#---------------------------------------------------------------------------
-

!include <ntwin32.mak>

INCLUDE=$(MSTOOLS)\samples\ole\include;$(INCLUDE)
LIB=$(MSTOOLS)\samples\ole\lib;$(LIB)

olelibs = $(olelibs) sdkutil.lib
olelibsmt = $(olelibsmt) sdkutil.lib
olelibsdll = $(olelibsdll) sdkutil.lib
guilibs = $(guilibs) sdkutil.lib
guilibsmt = $(guilibsmt) sdkutil.lib
guilibsdll = $(guilibsdll) sdkutil.lib
```

OLESTD.H   (INCLUDE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Standard Utilities
**
**    olestd.h
**
**    This file contains file contains data structure defintions,
**    function prototypes, constants, etc. for the common OLE 2.0
**    utilities.
**    These utilities include the following:
**          Debuging Assert/Verify macros
**          HIMETRIC conversion routines
**          reference counting debug support
**          OleStd API's for common compound-document app support
**
**    (c) Copyright Microsoft Corp. 1990 - 1992 All Rights Reserved
**
*************************************************************************/

#if !defined( _OLESTD_H_ )
#define _OLESTD_H_

#if defined( __TURBOC__ ) || defined( WIN32 )
#define _based(a)
#endif

#ifndef RC_INVOKED
#include <dos.h>        // needed for filetime
#endif  /* RC_INVOKED */

#include <commdlg.h>    // needed for LPPRINTDLG
#include <shellapi.h>   // needed for HKEY
#include "ansiapi.h"

// String table defines...
#define  IDS_OLESTDNOCREATEFILE   700
#define  IDS_OLESTDNOOPENFILE     701
#define  IDS_OLESTDDISKFULL       702


/*
 * Some C interface declaration stuff
 */

#if ! defined(__cplusplus)
typedef struct tagINTERFACEIMPL {
      IUnknownVtbl FAR*       lpVtbl;
      LPVOID                  lpBack;
      int                     cRef;   // interface specific ref count.
} INTERFACEIMPL, FAR* LPINTERFACEIMPL;

#define INIT_INTERFACEIMPL(lpIFace, pVtbl, pBack)   \
      ((lpIFace)->lpVtbl = pVtbl, \
         ((LPINTERFACEIMPL)(lpIFace))->lpBack = (LPVOID)pBack,   \
         ((LPINTERFACEIMPL)(lpIFace))->cRef = 0  \
      )

#if defined( _DEBUG )
#define OleDbgQueryInterfaceMethod(lpUnk)   \
      ((lpUnk) != NULL ? ((LPINTERFACEIMPL)(lpUnk))->cRef++ : 0)
#define OleDbgAddRefMethod(lpThis, iface)   \
      ((LPINTERFACEIMPL)(lpThis))->cRef++

#if _DEBUGLEVEL >= 2
#define OleDbgReleaseMethod(lpThis, iface) \
      (--((LPINTERFACEIMPL)(lpThis))->cRef == 0 ? \
         OleDbgOut("\t" iface "* RELEASED (cRef == 0)\r\n"),1 : \
          (((LPINTERFACEIMPL)(lpThis))->cRef < 0) ? \
            ( \
               DebugBreak(), \
               OleDbgOut(  \
                  "\tERROR: " iface "* RELEASED TOO MANY TIMES\r\n") \
            ),1 : \
            1)

#else       // if _DEBUGLEVEL < 2
#define OleDbgReleaseMethod(lpThis, iface) \
      (--((LPINTERFACEIMPL)(lpThis))->cRef == 0 ? \
         1 : \
          (((LPINTERFACEIMPL)(lpThis))->cRef < 0) ? \
            ( \
               OleDbgOut(  \
                  "\tERROR: " iface "* RELEASED TOO MANY TIMES\r\n") \
      ),1 : \
            1)

#endif      // if _DEBUGLEVEL < 2

#else       // ! defined (_DEBUG)

#define OleDbgQueryInterfaceMethod(lpUnk)
#define OleDbgAddRefMethod(lpThis, iface)
#define OleDbgReleaseMethod(lpThis, iface)

#endif      // if defined( _DEBUG )

#endif      // ! defined(__cplusplus)

/*
 * Some docfiles stuff
 */

#define STGM_DFRALL (STGM_READWRITE | STGM_TRANSACTED |
STGM_SHARE_DENY_WRITE)
#define STGM_DFALL (STGM_READWRITE | STGM_TRANSACTED | STGM_SHARE_EXCLUSIVE)
#define STGM_SALL (STGM_READWRITE | STGM_SHARE_EXCLUSIVE)
```

```
/*
 * Some moniker stuff
 */

// Delimeter used to separate ItemMoniker pieces of a composite moniker
#if defined( _MAC )
#define OLESTDDELIM OLESTR(":")
#else
#define OLESTDDELIM OLESTR("\\")
#endif

/*
 * Some Concurrency stuff
 */

/* standard Delay (in msec) to wait before retrying an LRPC call.
**    this value is returned from IMessageFilter::RetryRejectedCall
*/
#define OLESTDRETRYDELAY    (DWORD)5000

/* Cancel the pending outgoing LRPC call.
**    this value is returned from IMessageFilter::RetryRejectedCall
*/
#define OLESTDCANCELRETRY   (DWORD)-1

/*
 * Some Icon support stuff.
 *
 * The following API's are now OBSOLETE because equivalent API's have been
 * added to the OLE2.DLL library
 *      GetIconOfFile       superceeded by OleGetIconOfFile
 *      GetIconOfClass      superceeded by OleGetIconOfClass
 *      OleUIMetafilePictFromIconAndLabel
 *                          superceeded by OleMetafilePictFromIconAndLabel
 *
 * The following macros are defined for backward compatibility with previous
 * versions of the OLE2UI library. It is recommended that the new Ole* API's
 * should be used instead.
 */
#define GetIconOfFile(hInst, lpszFileName, fUseFileAsLabel) \
   OleGetIconOfFile(lpszFileName, fUseFileAsLabel)

#define GetIconOfClass(hInst, rclsid, lpszLabel, fUseTypeAsLabel) \
   OleGetIconOfClass(rclsid, lpszLabel, fUseTypeAsLabel)

#define
OleUIMetafilePictFromIconAndLabel(hIcon,pszLabel,pszSourceFile,iIcon)\
   OleMetafilePictFromIconAndLabel(hIcon, pszLabel, pszSourceFile, iIcon)


/*
 * Some Clipboard Copy/Paste & Drag/Drop support stuff
 */

//Macro to set all FormatEtc fields
```

```
#define SETFORMATETC(fe, cf, asp, td, med, li)   \
   ((fe).cfFormat=cf, \
    (fe).dwAspect=asp, \
    (fe).ptd=td, \
    (fe).tymed=med, \
    (fe).lindex=li)

//Macro to set interesting FormatEtc fields defaulting the others.
#define SETDEFAULTFORMATETC(fe, cf, med)  \
   ((fe).cfFormat=cf, \
    (fe).dwAspect=DVASPECT_CONTENT, \
    (fe).ptd=NULL, \
    (fe).tymed=med, \
    (fe).lindex=-1)

// Macro to test if two FormatEtc structures are an exact match
#define IsEqualFORMATETC(fe1, fe2)  \
   (OleStdCompareFormatEtc(&(fe1), &(fe2))==0)

// Clipboard format strings
#define CF_EMBEDSOURCE      TEXT("Embed Source")
#define CF_EMBEDDEDOBJECT   TEXT("Embedded Object")
#define CF_LINKSOURCE       TEXT("Link Source")
#define CF_CUSTOMLINKSOURCE TEXT("Custom Link Source")
#define CF_OBJECTDESCRIPTOR TEXT("Object Descriptor")
#define CF_LINKSRCDESCRIPTOR TEXT("Link Source Descriptor")
#define CF_OWNERLINK        TEXT("OwnerLink")
#define CF_FILENAME         TEXT("FileName")

#define OleStdQueryOleObjectData(lpformatetc)   \
   (((lpformatetc)->tymed & TYMED_ISTORAGE) ?    \
         NOERROR : ResultFromScode(DV_E_FORMATETC))

#define OleStdQueryLinkSourceData(lpformatetc)   \
   (((lpformatetc)->tymed & TYMED_ISTREAM) ?    \
         NOERROR : ResultFromScode(DV_E_FORMATETC))

#define OleStdQueryObjectDescriptorData(lpformatetc)    \
   (((lpformatetc)->tymed & TYMED_HGLOBAL) ?    \
         NOERROR : ResultFromScode(DV_E_FORMATETC))

#define OleStdQueryFormatMedium(lpformatetc, tymd)  \
   (((lpformatetc)->tymed & tymd) ?    \
         NOERROR : ResultFromScode(DV_E_FORMATETC))

// Make an independent copy of a MetafilePict
#define OleStdCopyMetafilePict(hpictin, phpictout)  \
   (*(phpictout) = OleDuplicateData(hpictin,CF_METAFILEPICT,GHND|
GMEM_SHARE))


// REVIEW: these need to be added to OLE2.H
#if !defined( DD_DEFSCROLLINTERVAL )
#define DD_DEFSCROLLINTERVAL    50
#endif
```

```
#if !defined( DD_DEFDRAGDELAY )
#define DD_DEFDRAGDELAY         200
#endif

#if !defined( DD_DEFDRAGMINDIST )
#define DD_DEFDRAGMINDIST       2
#endif


/* OleStdGetDropEffect
** -------------------
**
** Convert a keyboard state into a DROPEFFECT.
**
** returns the DROPEFFECT value derived from the key state.
**    the following is the standard interpretation:
**          no modifier -- Default Drop     (NULL is returned)
**          CTRL        -- DROPEFFECT_COPY
**          SHIFT       -- DROPEFFECT_MOVE
**          CTRL-SHIFT  -- DROPEFFECT_LINK
**
**    Default Drop: this depends on the type of the target application.
**    this is re-interpretable by each target application. a typical
**    interpretation is if the drag is local to the same document
**    (which is source of the drag) then a MOVE operation is
**    performed. if the drag is not local, then a COPY operation is
**    performed.
*/
#define OleStdGetDropEffect(grfKeyState)    \
   ( (grfKeyState & MK_CONTROL) ?          \
      ( (grfKeyState & MK_SHIFT) ? DROPEFFECT_LINK : DROPEFFECT_COPY ) :  \
      ( (grfKeyState & MK_SHIFT) ? DROPEFFECT_MOVE : 0 ) )


/* The OLEUIPASTEFLAG enumeration is used by the OLEUIPASTEENTRY structure.
 *
 * OLEUIPASTE_ENABLEICON    If the container does not specify this flag for
the entry in the
 *   OLEUIPASTEENTRY array passed as input to OleUIPasteSpecial, the
DisplayAsIcon button will be
 *   unchecked and disabled when the the user selects the format that
corresponds to the entry.
 *
 * OLEUIPASTE_PASTEONLY     Indicates that the entry in the OLEUIPASTEENTRY
array is valid for pasting only.
 * OLEUIPASTE_PASTE         Indicates that the entry in the OLEUIPASTEENTRY
array is valid for pasting. It
 *   may also be valid for linking if any of the following linking flags are
specified.
 *
 * If the entry in the OLEUIPASTEENTRY array is valid for linking, the
following flags indicate which link
 * types are acceptable by OR'ing together the appropriate
OLEUIPASTE_LINKTYPE<#> values.
```

```
 * These values correspond as follows to the array of link types passed to
OleUIPasteSpecial:
 *   OLEUIPASTE_LINKTYPE1=arrLinkTypes[0]
 *   OLEUIPASTE_LINKTYPE2=arrLinkTypes[1]
 *   OLEUIPASTE_LINKTYPE3=arrLinkTypes[2]
 *   OLEUIPASTE_LINKTYPE4=arrLinkTypes[3]
 *   OLEUIPASTE_LINKTYPE5=arrLinkTypes[4]
 *   OLEUIPASTE_LINKTYPE6=arrLinkTypes[5]
 *   OLEUIPASTE_LINKTYPE7=arrLinkTypes[6]
 *  OLEUIPASTE_LINKTYPE8=arrLinkTypes[7]
 *
 * where,
 *   UINT arrLinkTypes[8] is an array of registered clipboard formats for
linking. A maximium of 8 link
 *   types are allowed.
 */

typedef enum tagOLEUIPASTEFLAG
{
   OLEUIPASTE_ENABLEICON    = 2048,     // enable display as icon
   OLEUIPASTE_PASTEONLY     = 0,
   OLEUIPASTE_PASTE         = 512,
   OLEUIPASTE_LINKANYTYPE   = 1024,
   OLEUIPASTE_LINKTYPE1     = 1,
   OLEUIPASTE_LINKTYPE2     = 2,
   OLEUIPASTE_LINKTYPE3     = 4,
   OLEUIPASTE_LINKTYPE4     = 8,
   OLEUIPASTE_LINKTYPE5     = 16,
   OLEUIPASTE_LINKTYPE6     = 32,
   OLEUIPASTE_LINKTYPE7     = 64,
   OLEUIPASTE_LINKTYPE8     = 128
} OLEUIPASTEFLAG;

/*
 * PasteEntry structure
 * --------------------
 * An array of OLEUIPASTEENTRY entries is specified for the PasteSpecial
dialog
 * box. Each entry includes a FORMATETC which specifies the formats that are
 * acceptable, a string that is to represent the format in the  dialog's
list
 * box, a string to customize the result text of the dialog and a set of
flags
 * from the OLEUIPASTEFLAG enumeration.  The flags indicate if the entry is
 * valid for pasting only, linking only or both pasting and linking. If the
 * entry is valid for linking, the flags indicate which link types are
 * acceptable by OR'ing together the appropriate OLEUIPASTE_LINKTYPE<#>
values.
 * These values correspond to the array of link types as follows:
 *   OLEUIPASTE_LINKTYPE1=arrLinkTypes[0]
 *   OLEUIPASTE_LINKTYPE2=arrLinkTypes[1]
 *   OLEUIPASTE_LINKTYPE3=arrLinkTypes[2]
 *   OLEUIPASTE_LINKTYPE4=arrLinkTypes[3]
 *   OLEUIPASTE_LINKTYPE5=arrLinkTypes[4]
 *   OLEUIPASTE_LINKTYPE6=arrLinkTypes[5]
 *   OLEUIPASTE_LINKTYPE7=arrLinkTypes[6]
 *   OLEUIPASTE_LINKTYPE8=arrLinkTypes[7]
 *   UINT arrLinkTypes[8]; is an array of registered clipboard formats
 *                        for linking. A maximium of 8 link types are
allowed.
 */

typedef struct tagOLEUIPASTEENTRYA
{
   FORMATETC        fmtetc;            // Format that is acceptable. The
paste
                              //   dialog checks if this format is
                              //   offered by the object on the
                              //   clipboard and if so offers it for
                              //   selection to the user.
   LPCSTR           lpstrFormatName;   // String that represents the format
to the user. Any %s
                              //   in this string is replaced by the
FullUserTypeName
                              //   of the object on the clipboard and the
resulting string
                              //   is placed in the list box of the dialog.
Atmost
                              //   one %s is allowed. The presence or
absence of %s indicates
                              //   if the result text is to indicate that
data is
                              //   being pasted or that an object that can
be activated by
                              //   an application is being pasted. If %s is
                              //   present, the result-text says that an
object is being pasted.
                              //   Otherwise it says that data is being
pasted.
   LPCSTR           lpstrResultText;   // String to customize the result
text of the dialog when
                              //  the user selects the format correspoding
to this
                              //  entry. Any %s in this string is replaced
by the the application
                              //  name or FullUserTypeName of the object on
                              //  the clipboard. Atmost one %s is allowed.
   DWORD            dwFlags;           // Values from OLEUIPASTEFLAG enum
   DWORD            dwScratchSpace;    // Scratch space available to be used
                              //   by routines which loop through an
                              //   IEnumFORMATETC* to mark if the
                              //   PasteEntry format is available.
                              //   this field CAN be left uninitialized.
} OLEUIPASTEENTRYA, *POLEUIPASTEENTRYA, FAR *LPOLEUIPASTEENTRYA;
typedef struct tagOLEUIPASTEENTRYW
{
   FORMATETC        fmtetc;            // Format that is acceptable. The
paste
                              //   dialog checks if this format is
                              //   offered by the object on the
                              //   clipboard and if so offers it for
                              //   selection to the user.
   LPCWSTR           lpstrFormatName;   // String that represents the format
to the user. Any %s
                              //   in this string is replaced by the
FullUserTypeName
                              //   of the object on the clipboard and the
resulting string
                              //   is placed in the list box of the dialog.
Atmost
                              //   one %s is allowed. The presence or
absence of %s indicates
                              //   if the result text is to indicate that
data is
                              //   being pasted or that an object that can
be activated by
                              //   an application is being pasted. If %s is
                              //   present, the result-text says that an
object is being pasted.
                              //   Otherwise it says that data is being
pasted.
   LPCWSTR           lpstrResultText;   // String to customize the result
text of the dialog when
                              //  the user selects the format correspoding
to this
                              //  entry. Any %s in this string is replaced
by the the application
                              //  name or FullUserTypeName of the object on
                              //  the clipboard. Atmost one %s is allowed.
   DWORD            dwFlags;           // Values from OLEUIPASTEFLAG enum
   DWORD            dwScratchSpace;    // Scratch space available to be used
                              //   by routines which loop through an
                              //   IEnumFORMATETC* to mark if the
                              //   PasteEntry format is available.
                              //   this field CAN be left uninitialized.
} OLEUIPASTEENTRYW, *POLEUIPASTEENTRYW, FAR *LPOLEUIPASTEENTRYW;
#ifdef UNICODE
typedef OLEUIPASTEENTRYW OLEUIPASTEENTRY;
typedef POLEUIPASTEENTRYW POLEUIPASTEENTRY;
typedef LPOLEUIPASTEENTRYW LPOLEUIPASTEENTRY;
#else
typedef OLEUIPASTEENTRYA OLEUIPASTEENTRY;
typedef POLEUIPASTEENTRYA POLEUIPASTEENTRY;
typedef LPOLEUIPASTEENTRYA LPOLEUIPASTEENTRY;
#endif

#define OLESTDDROP_NONE         0
#define OLESTDDROP_DEFAULT      1
#define OLESTDDROP_NONDEFAULT   2


/*
 * Some misc stuff
 */
```

```
#define EMBEDDINGFLAG "Embedding"     // Cmd line switch for launching a
srvr

#define HIMETRIC_PER_INCH   2540      // number HIMETRIC units per inch
#define PTS_PER_INCH        72        // number points (font size) per inch

#define MAP_PIX_TO_LOGHIM(x,ppli)   MulDiv(HIMETRIC_PER_INCH, (x), (ppli))
#define MAP_LOGHIM_TO_PIX(x,ppli)   MulDiv((ppli), (x), HIMETRIC_PER_INCH)

// Returns TRUE if all fields of the two Rect's are equal, else FALSE.
#define AreRectsEqual(lprc1, lprc2)     \
   (((lprc1->top == lprc2->top) &&     \
     (lprc1->left == lprc2->left) &&   \
     (lprc1->right == lprc2->right) && \
     (lprc1->bottom == lprc2->bottom)) ? TRUE : FALSE)

#define LSTRCPYN(lpdst, lpsrc, cch) \
(\
   (lpdst)[(cch)-1] = '\0', \
   ((cch)>1 ? _fstrncpy(lpdst, lpsrc, (cch)-1) : 0)\
)



/****** DEBUG Stuff *****************************************************/

#ifdef _DEBUG

#if !defined( _DBGTRACE )
#define _DEBUGLEVEL 2
#else
#define _DEBUGLEVEL _DBGTRACE
#endif


#if defined( NOASSERT )

#define OLEDBGASSERTDATA
#define OleDbgAssert(a)
#define OleDbgAssertSz(a, b)
#define OleDbgVerify(a)
#define OleDbgVerifySz(a, b)

#else   // ! NOASSERT

#define OLEDBGASSERTDATA    \
      static char _based(_segname("_CODE")) _szAssertFile[]= __FILE__;

#define OleDbgAssert(a) \
      (!(a) ? FnAssert(#a, NULL, _szAssertFile, __LINE__) : (HRESULT)1)

#define OleDbgAssertSz(a, b)    \
      (!(a) ? FnAssert(#a, b, _szAssertFile, __LINE__) : (HRESULT)1)

#define OleDbgVerify(a) \
      OleDbgAssert(a)

#define OleDbgVerifySz(a, b)     \
      OleDbgAssertSz(a, b)

#endif  // ! NOASSERT


#define OLEDBGDATA_MAIN(szPrefix)    \
      char near g_szDbgPrefix[] = szPrefix;     \
      OLEDBGASSERTDATA
#define OLEDBGDATA  \
      extern char near g_szDbgPrefix[];     \
      OLEDBGASSERTDATA

#define OLEDBG_BEGIN(lpsz) \
      OleDbgPrintAlways(g_szDbgPrefix,lpsz,1);

#define OLEDBG_END  \
      OleDbgPrintAlways(g_szDbgPrefix,"End\r\n",-1);

#define OleDbgOut(lpsz) \
      OleDbgPrintAlways(g_szDbgPrefix,lpsz,0)

#define OleDbgOutNoPrefix(lpsz) \
      OleDbgPrintAlways("",lpsz,0)

#define OleDbgOutRefCnt(lpsz,lpObj,refcnt)       \
      OleDbgPrintRefCntAlways(g_szDbgPrefix,lpsz,lpObj,(ULONG)refcnt)

#define OleDbgOutRect(lpsz,lpRect)       \
      OleDbgPrintRectAlways(g_szDbgPrefix,lpsz,lpRect)

#define OleDbgOutHResult(lpsz,hr)    \
      OleDbgPrintScodeAlways(g_szDbgPrefix,lpsz,GetScode(hr))

#define OleDbgOutScode(lpsz,sc) \
      OleDbgPrintScodeAlways(g_szDbgPrefix,lpsz,sc)

#define OleDbgOut1(lpsz)     \
      OleDbgPrint(1,g_szDbgPrefix,lpsz,0)

#define OleDbgOutNoPrefix1(lpsz)     \
      OleDbgPrint(1,"",lpsz,0)

#define OLEDBG_BEGIN1(lpsz)     \
      OleDbgPrint(1,g_szDbgPrefix,lpsz,1);

#define OLEDBG_END1 \
      OleDbgPrint(1,g_szDbgPrefix,"End\r\n",-1);

#define OleDbgOutRefCnt1(lpsz,lpObj,refcnt)      \
      OleDbgPrintRefCnt(1,g_szDbgPrefix,lpsz,lpObj,(ULONG)refcnt)

#define OleDbgOutRect1(lpsz,lpRect)      \
      OleDbgPrintRect(1,g_szDbgPrefix,lpsz,lpRect)

#define OleDbgOut2(lpsz)    \
      OleDbgPrint(2,g_szDbgPrefix,lpsz,0)

#define OleDbgOutNoPrefix2(lpsz)    \
      OleDbgPrint(2,"",lpsz,0)

#define OLEDBG_BEGIN2(lpsz)    \
      OleDbgPrint(2,g_szDbgPrefix,lpsz,1);

#define OLEDBG_END2 \
      OleDbgPrint(2,g_szDbgPrefix,"End\r\n",-1);

#define OleDbgOutRefCnt2(lpsz,lpObj,refcnt)     \
      OleDbgPrintRefCnt(2,g_szDbgPrefix,lpsz,lpObj,(ULONG)refcnt)

#define OleDbgOutRect2(lpsz,lpRect)     \
      OleDbgPrintRect(2,g_szDbgPrefix,lpsz,lpRect)

#define OleDbgOut3(lpsz)    \
      OleDbgPrint(3,g_szDbgPrefix,lpsz,0)

#define OleDbgOutNoPrefix3(lpsz)    \
      OleDbgPrint(3,"",lpsz,0)

#define OLEDBG_BEGIN3(lpsz)    \
      OleDbgPrint(3,g_szDbgPrefix,lpsz,1);

#define OLEDBG_END3 \
      OleDbgPrint(3,g_szDbgPrefix,"End\r\n",-1);

#define OleDbgOutRefCnt3(lpsz,lpObj,refcnt)     \
      OleDbgPrintRefCnt(3,g_szDbgPrefix,lpsz,lpObj,(ULONG)refcnt)

#define OleDbgOutRect3(lpsz,lpRect)     \
      OleDbgPrintRect(3,g_szDbgPrefix,lpsz,lpRect)

#define OleDbgOut4(lpsz)    \
      OleDbgPrint(4,g_szDbgPrefix,lpsz,0)

#define OleDbgOutNoPrefix4(lpsz)    \
      OleDbgPrint(4,"",lpsz,0)

#define OLEDBG_BEGIN4(lpsz)    \
      OleDbgPrint(4,g_szDbgPrefix,lpsz,1);

#define OLEDBG_END4 \
      OleDbgPrint(4,g_szDbgPrefix,"End\r\n",-1);

#define OleDbgOutRefCnt4(lpsz,lpObj,refcnt)     \
      OleDbgPrintRefCnt(4,g_szDbgPrefix,lpsz,lpObj,(ULONG)refcnt)

#define OleDbgOutRect4(lpsz,lpRect)     \
      OleDbgPrintRect(4,g_szDbgPrefix,lpsz,lpRect)
```

```
#else   //  !_DEBUG

#define OLEDBGDATA_MAIN(szPrefix)
#define OLEDBGDATA
#define OleDbgAssert(a)
#define OleDbgAssertSz(a, b)
#define OleDbgVerify(a)         (a)
#define OleDbgVerifySz(a, b)    (a)
#define OleDbgOutHResult(lpsz,hr)
#define OleDbgOutScode(lpsz,sc)
#define OLEDBG_BEGIN(lpsz)
#define OLEDBG_END
#define OleDbgOut(lpsz)
#define OleDbgOut1(lpsz)
#define OleDbgOut2(lpsz)
#define OleDbgOut3(lpsz)
#define OleDbgOut4(lpsz)
#define OleDbgOutNoPrefix(lpsz)
#define OleDbgOutNoPrefix1(lpsz)
#define OleDbgOutNoPrefix2(lpsz)
#define OleDbgOutNoPrefix3(lpsz)
#define OleDbgOutNoPrefix4(lpsz)
#define OLEDBG_BEGIN1(lpsz)
#define OLEDBG_BEGIN2(lpsz)
#define OLEDBG_BEGIN3(lpsz)
#define OLEDBG_BEGIN4(lpsz)
#define OLEDBG_END1
#define OLEDBG_END2
#define OLEDBG_END3
#define OLEDBG_END4
#define OleDbgOutRefCnt(lpsz,lpObj,refcnt)
#define OleDbgOutRefCnt1(lpsz,lpObj,refcnt)
#define OleDbgOutRefCnt2(lpsz,lpObj,refcnt)
#define OleDbgOutRefCnt3(lpsz,lpObj,refcnt)
#define OleDbgOutRefCnt4(lpsz,lpObj,refcnt)
#define OleDbgOutRect(lpsz,lpRect)
#define OleDbgOutRect1(lpsz,lpRect)
#define OleDbgOutRect2(lpsz,lpRect)
#define OleDbgOutRect3(lpsz,lpRect)
#define OleDbgOutRect4(lpsz,lpRect)

#endif  //  _DEBUG


/*************************************************************************
** Function prototypes
*************************************************************************/


//OLESTD.C
STDAPI_(int) SetDCToAnisotropic(HDC hDC, LPRECT lprcPhysical, LPRECT
lprcLogical, LPRECT lprcWindowOld, LPRECT lprcViewportOld);
STDAPI_(int) SetDCToDrawInHimetricRect(HDC, LPRECT, LPRECT, LPRECT, LPRECT);
STDAPI_(int) ResetOrigDC(HDC, int, LPRECT, LPRECT);
STDAPI_(int)        XformWidthInHimetricToPixels(HDC, int);
STDAPI_(int)        XformWidthInPixelsToHimetric(HDC, int);
STDAPI_(int)        XformHeightInHimetricToPixels(HDC, int);
STDAPI_(int)        XformHeightInPixelsToHimetric(HDC, int);

STDAPI_(void) XformRectInPixelsToHimetric(HDC, LPRECT, LPRECT);
STDAPI_(void) XformRectInHimetricToPixels(HDC, LPRECT, LPRECT);
STDAPI_(void) XformSizeInPixelsToHimetric(HDC, LPSIZEL, LPSIZEL);
STDAPI_(void) XformSizeInHimetricToPixels(HDC, LPSIZEL, LPSIZEL);
STDAPI_(int) XformWidthInHimetricToPixels(HDC, int);
STDAPI_(int) XformWidthInPixelsToHimetric(HDC, int);
STDAPI_(int) XformHeightInHimetricToPixels(HDC, int);
STDAPI_(int) XformHeightInPixelsToHimetric(HDC, int);

STDAPI_(void) ParseCmdLine(LPSTR, BOOL FAR *, LPSTR);

STDAPI_(BOOL) OleStdIsOleLink(LPUNKNOWN lpUnk);
STDAPI_(LPUNKNOWN) OleStdQueryInterface(LPUNKNOWN lpUnk, REFIID riid);
STDAPI_(LPSTORAGE) OleStdCreateRootStorage(LPOLESTR lpszStgName, DWORD
grfMode);
STDAPI_(LPSTORAGE) OleStdOpenRootStorage(LPOLESTR lpszStgName, DWORD
grfMode);
STDAPI_(LPSTORAGE) OleStdOpenOrCreateRootStorage(LPOLESTR lpszStgName, DWORD
grfMode);
STDAPI_(LPSTORAGE) OleStdCreateChildStorage(LPSTORAGE lpStg, LPOLESTR
lpszStgName);
STDAPI_(LPSTORAGE) OleStdOpenChildStorage(LPSTORAGE lpStg, LPOLESTR
lpszStgName, DWORD grfMode);
STDAPI_(BOOL) OleStdCommitStorage(LPSTORAGE lpStg);
STDAPI OleStdDestroyAllElements(LPSTORAGE lpStg);

STDAPI_(LPSTORAGE) OleStdCreateStorageOnHGlobal(
      HANDLE hGlobal,
      BOOL fDeleteOnRelease,
      DWORD dwgrfMode
);
STDAPI_(LPSTORAGE) OleStdCreateTempStorage(BOOL fUseMemory, DWORD grfMode);
STDAPI OleStdDoConvert(LPSTORAGE lpStg, REFCLSID rClsidNew);
STDAPI_(BOOL) OleStdGetTreatAsFmtUserType(
      REFCLSID        rClsidApp,
      LPSTORAGE       lpStg,
      CLSID FAR*      lpclsid,
      CLIPFORMAT FAR* lpcfFmt,
      LPOLESTR FAR*      lplpszType
);
STDAPI OleStdDoTreatAsClass(LPOLESTR lpszUserType, REFCLSID rclsid, REFCLSID
rclsidNew);
STDAPI_(BOOL) OleStdSetupAdvises(LPOLEOBJECT lpOleObject, DWORD
dwDrawAspect,
               LPOLESTR lpszContainerApp, LPOLESTR lpszContainerObj,
               LPADVISESINK lpAdviseSink, BOOL fCreate);
STDAPI OleStdSwitchDisplayAspect(
      LPOLEOBJECT             lpOleObj,
      LPDWORD                 lpdwCurAspect,
      DWORD                  dwNewAspect,
      HGLOBAL                hMetaPict,
      BOOL                   fDeleteOldAspect,
      BOOL                   fSetupViewAdvise,
      LPADVISESINK           lpAdviseSink,
      BOOL FAR*              lpfMustUpdate
);
STDAPI OleStdSetIconInCache(LPOLEOBJECT lpOleObj, HGLOBAL hMetaPict);
STDAPI_(HGLOBAL) OleStdGetData(
      LPDATAOBJECT       lpDataObj,
      CLIPFORMAT         cfFormat,
      DVTARGETDEVICE FAR* lpTargetDevice,
      DWORD              dwAspect,
      LPSTGMEDIUM        lpMedium
);
STDAPI_(void) OleStdMarkPasteEntryList(
      LPDATAOBJECT       lpSrcDataObj,
      LPOLEUIPASTEENTRY  lpPriorityList,
      int                cEntries
);
STDAPI_(int) OleStdGetPriorityClipboardFormat(
      LPDATAOBJECT       lpSrcDataObj,
      LPOLEUIPASTEENTRY  lpPriorityList,
      int                cEntries
);
STDAPI_(BOOL) OleStdIsDuplicateFormat(
      LPFORMATETC        lpFmtEtc,
      LPFORMATETC        arrFmtEtc,
      int                nFmtEtc
);
STDAPI_(void) OleStdRegisterAsRunning(LPUNKNOWN lpUnk, LPMONIKER lpmkFull,
DWORD FAR* lpdwRegister);
STDAPI_(void) OleStdRevokeAsRunning(DWORD FAR* lpdwRegister);
STDAPI_(void) OleStdNoteFileChangeTime(LPOLESTR lpszFileName, DWORD
dwRegister);
STDAPI_(void) OleStdNoteObjectChangeTime(DWORD dwRegister);
STDAPI OleStdGetOleObjectData(
      LPPERSISTSTORAGE   lpPStg,
      LPFORMATETC        lpformatetc,
      LPSTGMEDIUM        lpMedium,
      BOOL               fUseMemory
);
STDAPI OleStdGetLinkSourceData(
      LPMONIKER          lpmk,
      LPCLSID            lpClsID,
      LPFORMATETC        lpformatetc,
      LPSTGMEDIUM        lpMedium
);
STDAPI_(HGLOBAL) OleStdGetObjectDescriptorData(
      CLSID              clsid,
      DWORD              dwAspect,
      SIZEL              sizel,
      POINTL             pointl,
      DWORD              dwStatus,
      LPOLESTR           lpszFullUserTypeName,
      LPOLESTR            lpszSrcOfCopy
);
STDAPI_(HGLOBAL) OleStdGetObjectDescriptorDataFromOleObject(
      LPOLEOBJECT         lpOleObj,
      LPOLESTR            lpszSrcOfCopy,
      DWORD               dwAspect,
      POINTL              pointl,
      LPSIZEL             lpSizelHim
);
STDAPI_(HGLOBAL) OleStdFillObjectDescriptorFromData(
      LPDATAOBJECT       lpDataObject,
      LPSTGMEDIUM        lpmedium,
      CLIPFORMAT FAR*    lpcfFmt
);
STDAPI_(HANDLE) OleStdGetMetafilePictFromOleObject(
      LPOLEOBJECT         lpOleObj,
      DWORD               dwDrawAspect,
      LPSIZEL             lpSizelHim,
      DVTARGETDEVICE FAR* ptd
);

STDAPI_(void) OleStdCreateTempFileMoniker(LPOLESTR lpszPrefixString, UINT
FAR* lpuUnique, LPOLESTR lpszName, LPMONIKER FAR* lplpmk);
STDAPI_(LPMONIKER) OleStdGetFirstMoniker(LPMONIKER lpmk);
STDAPI_(ULONG) OleStdGetLenFilePrefixOfMoniker(LPMONIKER lpmk);
STDAPI OleStdMkParseDisplayName(
      REFCLSID        rClsid,
      LPBC            lpbc,
      LPOLESTR        lpszUserName,
      ULONG FAR*      lpchEaten,
      LPMONIKER FAR*  lplpmk
);
STDAPI_(LPVOID) OleStdMalloc(ULONG ulSize);
STDAPI_(LPVOID) OleStdRealloc(LPVOID pmem, ULONG ulSize);
STDAPI_(void) OleStdFree(LPVOID pmem);
STDAPI_(ULONG) OleStdGetSize(LPVOID pmem);
STDAPI_(void) OleStdFreeString(LPOLESTR lpsz, LPMALLOC lpMalloc);
STDAPI_(LPOLESTR) OleStdCopyString(LPOLESTR lpszSrc, LPMALLOC lpMalloc);
STDAPI_(ULONG) OleStdGetItemToken(LPOLESTR lpszSrc, LPOLESTR lpszDst,int
nMaxChars);

STDAPI_(UINT)     OleStdIconLabelTextOut(HDC        hDC,
                                HFONT      hFont,
                                int        nXStart,
                                int        nYStart,
                                UINT       fuOptions,
                                RECT FAR * lpRect,
                                LPOLESTR   lpszString,
                                UINT       cchString,
                                int FAR *  lpDX);

// registration database query functions
STDAPI_(UINT)     OleStdGetAuxUserType(REFCLSID rclsid,
                              WORD   wAuxUserType,
                              LPOLESTR  lpszAuxUserType,
                             int    cch,
                             HKEY   hKey);

STDAPI_(UINT)     OleStdGetUserTypeOfClass(REFCLSID rclsid,
                                 LPOLESTR lpszUserType,
                                 UINT cch,
                                 HKEY hKey);

STDAPI_(BOOL) OleStdGetMiscStatusOfClass(REFCLSID, HKEY, DWORD FAR *);
STDAPI_(CLIPFORMAT) OleStdGetDefaultFileFormatOfClass(
      REFCLSID        rclsid,
      HKEY            hKey
);

STDAPI_(void) OleStdInitVtbl(LPVOID lpVtbl, UINT nSizeOfVtbl);
STDMETHODIMP OleStdNullMethod(LPUNKNOWN lpThis);
STDAPI_(BOOL) OleStdCheckVtbl(LPVOID lpVtbl, UINT nSizeOfVtbl, LPOLESTR
lpszIface);
STDAPI_(ULONG) OleStdVerifyRelease(LPUNKNOWN lpUnk, LPOLESTR lpszMsg);
STDAPI_(ULONG) OleStdRelease(LPUNKNOWN lpUnk);

STDAPI_(HDC) OleStdCreateDC(DVTARGETDEVICE FAR* ptd);
STDAPI_(HDC) OleStdCreateIC(DVTARGETDEVICE FAR* ptd);
STDAPI_(DVTARGETDEVICE FAR*) OleStdCreateTargetDevice(LPPRINTDLG
lpPrintDlg);
STDAPI_(BOOL) OleStdDeleteTargetDevice(DVTARGETDEVICE FAR* ptd);
STDAPI_(DVTARGETDEVICE FAR*) OleStdCopyTargetDevice(DVTARGETDEVICE FAR*
ptdSrc);
STDAPI_(BOOL) OleStdCopyFormatEtc(LPFORMATETC petcDest, LPFORMATETC
petcSrc);
STDAPI_(int) OleStdCompareFormatEtc(FORMATETC FAR* pFetcLeft, FORMATETC FAR*
pFetcRight);
STDAPI_(BOOL) OleStdCompareTargetDevice
   (DVTARGETDEVICE FAR* ptdLeft, DVTARGETDEVICE FAR* ptdRight);


STDAPI_(void) OleDbgPrint(
      int     nDbgLvl,
      LPSTR   lpszPrefix,
      LPSTR   lpszMsg,
      int     nIndent
);
STDAPI_(void) OleDbgPrintAlways(LPSTR lpszPrefix, LPSTR lpszMsg, int
nIndent);
STDAPI_(void) OleDbgSetDbgLevel(int nDbgLvl);
STDAPI_(int) OleDbgGetDbgLevel( void );
STDAPI_(void) OleDbgIndent(int n);
STDAPI_(void) OleDbgPrintRefCnt(
      int         nDbgLvl,
      LPSTR       lpszPrefix,
      LPSTR       lpszMsg,
      LPVOID      lpObj,
      ULONG       refcnt
);
STDAPI_(void) OleDbgPrintRefCntAlways(
      LPSTR       lpszPrefix,
      LPSTR       lpszMsg,
      LPVOID      lpObj,
      ULONG       refcnt
);
STDAPI_(void) OleDbgPrintRect(
      int         nDbgLvl,
      LPSTR       lpszPrefix,
      LPSTR       lpszMsg,
      LPRECT      lpRect
);
STDAPI_(void) OleDbgPrintRectAlways(
      LPSTR       lpszPrefix,
      LPSTR       lpszMsg,
      LPRECT      lpRect
);
STDAPI_(void) OleDbgPrintScodeAlways(LPSTR lpszPrefix, LPSTR lpszMsg, SCODE
sc);

// debug implementation of the IMalloc interface.
STDAPI OleStdCreateDbAlloc(ULONG reserved, IMalloc FAR* FAR* ppmalloc);


STDAPI_(LPENUMFORMATETC)
  OleStdEnumFmtEtc_Create(ULONG nCount, LPFORMATETC lpEtc);

STDAPI_(LPENUMSTATDATA)
  OleStdEnumStatData_Create(ULONG nCount, LPSTATDATA lpStat);

STDAPI_(BOOL)
  OleStdCopyStatData(LPSTATDATA pDest, LPSTATDATA pSrc);

STDAPI_(HPALETTE)
  OleStdCreateStandardPalette(void);

#if defined( OBSOLETE )

/*************************************************************************
** The following API's have been converted into macros:
**          OleStdQueryOleObjectData
**          OleStdQueryLinkSourceData
**          OleStdQueryObjectDescriptorData
**          OleStdQueryFormatMedium
**          OleStdCopyMetafilePict
**          AreRectsEqual
**          OleStdGetDropEffect
**
**    These macros are defined above
*************************************************************************/
STDAPI_(BOOL) AreRectsEqual(LPRECT lprc1, LPRECT lprc2);
STDAPI_(BOOL) OleStdCopyMetafilePict(HANDLE hpictin, HANDLE FAR* phpictout);
STDAPI OleStdQueryOleObjectData(LPFORMATETC lpformatetc);
STDAPI OleStdQueryLinkSourceData(LPFORMATETC lpformatetc);
STDAPI OleStdQueryObjectDescriptorData(LPFORMATETC lpformatetc);
STDAPI OleStdQueryFormatMedium(LPFORMATETC lpformatetc, TYMED tymed);
```

```
STDAPI_(DWORD) OleStdGetDropEffect ( DWORD grfKeyState );
#endif  // OBSOLETE


#endif // _OLESTD_H_
```

STABLIZE.H (INCLUDE Sample)

```
//
+---------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       stablize.h
//
//  Contents:   Stabilization Classes used to stabilize objects during
//              re-entrant calls.
//
//  Classes:    CSafeRefCount
//              CStabilize
//
//  History:    8-26-94   stevebl   Modified from code written by AlexGo
//
//--------------------------------------------------------------------------
--

#ifndef __STABLIZE__
#define __STABLIZE__

//+-------------------------------------------------------------------------
//
//  Class:      CSafeRefCount
//
//  Purpose:    A concrete class for objects to inherit from.
//              CSafeRefCount will keep track of reference counts,
//              nesting counts, and zombie states, allowing objects
//              to easily manage the liveness of their memory images.
//
//  Interface:
//
//  History:    dd-mmm-yy Author    Comment
//              01-Aug-94 alexgo    author
//
//  Notes:      inherits CPrivAlloc
//
//--------------------------------------------------------------------------

class CSafeRefCount
{
public:
        ULONG   SafeAddRef();
        ULONG   SafeRelease();
        ULONG   IncrementNestCount();
        ULONG   DecrementNestCount();
                CSafeRefCount();
        virtual ~CSafeRefCount();

private:
        ULONG   m_cRefs;
        ULONG   m_cNest;
        BOOL    m_fInDelete;
};

//+-------------------------------------------------------------------------
//
//  Class:      CStabilize
//
//  Purpose:    An instance of this class should be allocated on the
//              stack of every object method that makes an outgoing call.
//              The contstructor takes a pointer to the object's base
//              CSafeRefCount class.
//
//  Interface:
//
//  History:    dd-mmm-yy Author    Comment
//              01-Aug-94 alexgo    author
//
//  Notes:      The constructor will increment the nest count of the
//              object while the destructor will decrement it.
//
//--------------------------------------------------------------------------

class CStabilize
{
public:
        inline CStabilize( CSafeRefCount *pObjSafeRefCount );
        inline ~CStabilize();

private:
        CSafeRefCount * m_pObjSafeRefCount;
};

inline CStabilize::CStabilize( CSafeRefCount *pObjSafeRefCount )
{
        pObjSafeRefCount->IncrementNestCount();
        m_pObjSafeRefCount = pObjSafeRefCount;
}

inline CStabilize::~CStabilize()
{
        m_pObjSafeRefCount->DecrementNestCount();
}

#endif  // __STABLIZE__
```

INOLE2

These topics contain bug fixes and Win32 update information for the sample code included with Kraig Brockschmidt's *Inside OLE 2,* 1st ed., Microsoft Press, 1994.

MAKEFILE   (INOLE2 Sample)

```
DIRLIST  =  stastrip    \
            classlib    \
            bcdemo      \
            gbdemo      \
            chap02      \
            chap03      \
            chap04      \
            chap05      \
            chap06      \
            chap07      \
            chap08      \
            chap09      \
            chap10      \
            chap11      \
            chap12      \
            chap13      \
            chap14      \
            chap15      \
            chap16


all: lib $(DIRLIST)

INCLUDE=$(INCLUDE);$(MSTOOLS)\samples\ole\inole2\inc

$(DIRLIST):
        cd $@
        @echo *** ole\inole2\$@ *** >>$(MSTOOLS)\samples\inole2.tmp
        @nmake -a -i 1>>$(MSTOOLS)\samples\inole2.tmp 2>&1
        cd ..

lib:
        cd $@
        @echo *** ole\inole2\$@ *** >>$(MSTOOLS)\samples\inole2.tmp
        @nmake -i 1>>$(MSTOOLS)\samples\inole2.tmp 2>&1
        cd ..
```

NOTES.TXT (INOLE2 Sample)

Additional Notes Concerning the Win32 version of Inside OLE 2 samples.

These notes are intended to supplement changes described in UPDATE.TXT. That text file describes earlier changes to these samples from the printed versions in Inside OLE 2. These notes will not be as detailed as those in UPDATE.TXT and are intended to point out some of the changes in the Win32 port.

The process of porting exposed numerous minute bugs in various samples. Many of these are fixed by adding a parameter here or there to some function, or slightly modifying the behavior of a function. This file doesn't document all such changes, but will highlight some of the more important ones.

General Changes:

All the samples will compile for either Win16 or Win32 targets. Most of the differences between the two environments are isolated in the macros in inc\book1632.h. Occasionally there will be a #ifdef WIN32 or a #ifdef UNICODE in a sample where a macro wasn't practical.

In order to make a single Win16/Win32 code base, all the code compiled for Win16 is large model. Therefore you will find no "FAR" keywords anywhere, and an effort has been made to remove "LP" pointer types and simply use "P", especially for sample-specific types.

The makefile system has been overhauled to centralize all the rules and compiler options into the files inole2a.mak and inole2b.mak. All individual sample makefiles define their specific parameters and include these two files to perform the compilation. In addition, all the samples (Cosmo, Patron, CoCosmo, etc.) that used to compile into separate MDI and SDI directories now compile into just one. There was little use of having both versions co-exist, and since the same sort of provision was not made for Win16/Win32, Unicode/Ansi, and Debug/Non-Debug, the MDI/SDI provision has been removed. Running any MAKEALL.BAT will build whatever version is set by the environment variables as described in BUILD.TXT.

Chapter 2

Due to differences in structure packing and alignment, files written by a 16-bit version of Cosmo are not interchangeable with the 32-bit version and vice-versa. Cosmo will report a failure in such cases. This applies to all later chapters as well.

Chapter 4

The MALLOC sample does not allow access to shared memory under Win32. This is a restriction of OLE 32-bits as the CoGetMalloc function fails when passed MEMCTX_SHARED. All applications should use MEMCTX_TASK for all memory allocation purposes under Win32, and should also do so under Win16 for portability.

OLE 32-bit uses registry entries of InProcServer32, LocalServer32, and InProcHandler32. This doesn't affect object users, like the OBJUSER sample, but does affect the registation files. It is very important that your own applications register themselves appropriately depending on whether they are 16- or 32-bits. The new registry keys allow you to register both types of modules together.

The EKOALA sample in this chapter has the IExternalConnection interface implemented on the object. This will demonstrate how the AddConnection and RemoveConnection members of this interface are called when LRPC stubs connect to the object. More information about stubs is found in the OLE 2 SDK document REMOTING.DOC.

Chapter 5

Due to differences in structure packing and alignment as well as the differences between Win16 and Win32 DEVMODE structures, files written by a 16-bit version of Patron are not interchangeable with the 32-bit version and vice-versa. Patron will report a failure in such cases.

When the Smasher sample attempts to copy a compressed file back over it's original, it used StgOpenStorage on the original file and calls IStorage::CopyTo from the compressed file to copy back to the original file. This operation, however, doesn't overwrite the original file, and as a result the individual streams are not laid out sequentially as they are in the compressed file. In order to get the fully defragmented file back to the original filename, you need to copy the defragmented file back over the original file not using the IStorage interface at all. In order to demonstrate the difference, a new sample is added to Chapter 5 called Fragmenter, which is found in CHAP05\FRAGMENT.

Chapter 8

This chapter's Patron example (and all those in later chapters) now pay attention to the "Drag Delay" and "Drag Distance" values to "debounce" the mouse before starting a drag & drop operation. The drag delay is the amount of time the mouse has to be held down in the same position (defined as the mouse down point plus or minus the "Drag Distance" value) before drag & drop starts. This is so a brief unwanted click in the pick region doesn't start the operation right away. The drag distance is the number of pixels around the original mouse down point where the delay is meaningful, but if the mouse moves ourside this rectangle before the delay has expired, and the button is still held down, then a drag & drop operation is started.

Chapters 9 & 10

Patron in this and all later chapters now pays attention to the OLEMISC_RECOMPOSEONRESIZE flag. Cosmo in Chapter 10 sets this flag as a test for Patron. When Patron resizes and embedded object and this flag is set, it explicitly runs the server (with OleRun) and requests a new rendering (see CTenant::SizeSet in TENANT.CPP).

Note that if you attempt to use the Create From File feature in Patron's Insert Object dialog with a Cosmo object, the creation will fail. Cosmo requires the IPersistFile interface in order to support Create From File, so you need to use the Chapter 13 Cosmo for this feature to work.

Chapter 11

Polyline has the IRunnableObject interface added and has changed the handling of the editing dialog box significantly. A problem arose where Polyline was depending on the existence of its editing window for determining the size of the object. When a container opened a document with a Polyline object therein and asked Polyline for its extents, there wasn't much on which to base the answer. To correct this, Polyline implements IRunnableObject which allows it to know when the object is "running" but not active. When Polyline is now run, it creates the dialog but leaves it hidden. Only on IOleObject::DoVerb (PRIMARY) or (SHOW) will Polyline show the dialog. Closing the dialog then only hides it, and it isn't actually destroyed until the Polyline object itself is destroyed.

Chapter 12

Reconnecting a link that was already running when Patron opened a file with a linked object didn't work. The reason is that OleLoad will not attempt to reconnect an already running object on loading if OleLoad is not given an IOleClientSite pointer. Since Patron doesn't provide this pointer to OleLoad, it must call IOleLink::BindIfRunning after loading the object to perform the reconnect.

Chapter 13

General improvements have been made to Patron to handle linkings to embeddings better and more consistently. There were a number of bugs fixed in functions like IOleClientSite::GetMoniker and in moniker handling overall.

There is a version of Polyline added to this chapter to handle linking to embeddings as well. Objects that support this feature must implement IOleObject::GetMoniker and IOleObject::SetMoniker and must be registered without OLEMISC_CANTLINKINSIDE (which is set for Polyline and Cosmo prior to Chapter 13). Polyline also implements the IExternalConnection interface to provide for proper shutdown of the intermediate container that is running in linking to embedding cases.

Chapters 15 & 16

Improvements have been made to Patron and Polyline for more robust handling of inside-out objects as well as multiple active objects. True handling of many such objects really requires some of the features available in OLE Controls, but that's another whole topic...

UPDATE.TXT (INOLE2 Sample)

Updates to Inside OLE 2 Book and Sample Code, January 1994
Pre-Release Win32 versions (through Chapter 2)

This file lists describe bugs found in the sample code of the book Inside OLE 2 and corrections for those problems. Updated code is found on Compuserve, Internet, and the MSDN CD (this also contains the book text).

There are three sections to this file:

```
    Compiler Errors:    Changes to the OLE 2 SDK that caused
                        compiler errors with original Inside OLE 2
                        samples.  Also some considerations for
                        different compilers.

    Bug Fixes:          Other changes to the source code to fix
                        a few major and a number of minor problems.

    Other Topics:       List of typographical errors in the book
                        text (MSDN CD version includes most
                        corrections already, but do double-check)
                        and other topics of interest, such as Win32
                        version information.
```

NOTE: To operate properly with OLE 2.01 after updating the sources to compile it is necessary to rebuild an OLE 2.01 version of the OLE2UI library for the Inside OLE 2 samples. Build a library named "BOOKUI.DLL" according to the instructions in the SDK. With this new file replace the INOLE2\BUILD\BOOKUI.DLL file included with the book. A new version of the library is included with this revision of the samples.

NOTE: To compile Win16 versions of the code, do "SET WIN16=1" from your command prompt (or put it in AUTOEXEC.BAT). This only affects the samples that have so far been ported, but is crucial to building them properly for the Windows 3.1 target environment.

Please send additional bug reports and problems to the author at kraigb@microsft.com on Internet. CompuServe users must prefix the address with ">INTERNET:"

Section 1: Compiler Errors

The sample code of the book Inside OLE 2 was originally written for the OLE 2.00 libraries. The OLE 2.01 libraries included a few minor changes that cause compilation errors with the original book samples.

1. Errors concerning IViewObject::Draw. The OLE 2.01 SDK changed the prototype of IViewObject::Draw from using "const LPRECTL" to "LPCRECTL." This affects HCOSMO and Polyline samples of chapter 11 and Poltilne sample of chapter 16, causing them to fail compiling.

   To correct the error, replace all occurances of "const LPRECTL" in declaration of CImpIViewObject::Draw to "LPCRECTL" to match OLE 2.01. This occurs in the HCOSMO.H and IVIEWOBJ.CPP files of HCOSMO in chapter 11, and the POLYLINE.H and IVIEWOBJ.CPP files of Polyline in chapters 11 and 16. Note that there are also affected files in the INTERFAC directory, IVIEWOBJ.H and IVIEWOBJ.CPP, although these will not show in compilations.

2. Errors concerning OleUIAddVerbMenu in Patron samples of chapters 9, 12, 13, 14, and 15.

   The OLE 2.01 UI library added a parameter to this function, an added "idVerbMax" that indicates the largest number the container allows for a verb menu item identifier. This new parameter occurs immediately after the "idVerbMin" parameter.

   In the Patron directories of chapters 9, 12, 13, 14, and 15 in the files PAGE.CPP and TENANT.CPP, the parameter IDM_VERBMAX is added after the IDM_VERBMIN parameter (both are defined in resource.h). The functions affected are CTenant::AddVerbMenu (TENANT.CPP) and CPage::FQueryObjectSelected (PAGE.CPP).

3. Errors concerning OleGetObjectDescriptorFromOleObject affecting Patron samples in chapters 9, 12, 13, 14, and 15.

   The OLE 2.01 UI library added a parameter to the end of this function: an LPSIZEL that contains the extents of the object. The OLE 2.00 code appeared as follows:

```
    stm.hGlobal=OleStdGetObjectDescriptorDataFromOleObject
        (m_pIOleObject, NULL, m_fe.dwAspect, ptl);
```

   The corrections for OLE 2.01 require the declaration and initialization of a new SIZEL variable for the last parameter:

```
    //Declare new variable
    SIZEL   szl;

    ...


    //Initialze
```

```
        SETSIZEL(szl, (10*(m_rcl.right-m_rcl.left))
            , (10 * (m_rcl.bottom-m_rcl.top)));

        //Add to parameter list.
        stm.hGlobal=OleStdGetObjectDescriptorDataFromOleObject
            (m_pIOleObject, NULL, m_fe.dwAspect, ptl, &szl);
```

This correctly computes the size of the object in question and stores those extents in szl and in the object descriptor.

The changes affect the function CTenant::CopyEmbeddedObject in chapters 9, 12, 13, 14, and 15. The function is found in all the TENANT.CPP files in the PATRON directories of these chapters. Note that it also affects the function CTenant::CopyLinkedObject.in chapters 12, 13, 14, and 15,

4. Errors concerning IDS_CLOSE in Cosmo samples of chapters 10, 13, 14, and 16. The OLE2UI library of OLE 2.01 declared an IDS_CLOSE symbol that conflicts with one of the same name declared in Cosmo's RESOURCE.H file.

   To correct the problem the IDS_CLOSE in RESOURCE.H is renamed to "IDS_CLOSE2" and occurances of that symbol in COSMO.RC are changed to match. This affects these two files in chapters 10, 13, 14, and 16.

5. Errors concerning GETICON.H in Patron samples of chapters 14 and 15. This file was necessary to include in container applications that supports the Convert dialog in OLE 2.00. The OLE 2.01 SDK removed the dependency on this file, so it no longer has to be included. Therefore the #include statement in the TENANT.CPP files of Patron in chapters 14 and 15 have been commented out.

   To correct the problem, the line "#include <geticon.h>" is commented out at the top of the TENANT.CPP files of Patron in chapters 14 and 15.

6. Linker errors on Windows NT hosted development environments concerning the CLASSLIB library. A filename in the FILES.LST of CLASSLIB has an extra character causing it to appear as a long filename, which on a Windows NT system is valid but does not exist.

   The first file in FILES.LST is changed from "cstrtable.obj" to "cstrtabl.obj" to correct the problem, that is, remove the extra 'e'.

7. The function declarations for LibMain and WEP in all DLLs are slightly incorrect and have been updated. These can

ause compiler errors on non-Microsoft compilers. LibMain should return an int (not a HANDLE) and take an HINSTANCE as the first parameter (not a HANDLE either). WEP should return an int rather than void. WEP should always return a zero to provide some return value although that value is not used.

Section 2: Sample Code Bug Fixes

1. GP Fault with Patron sample during Activate As, chapters 14 and 15.

   The problem lies in the function CTenant::Close with the fReopen flag. When this function determines that there are no references to the tenant IStorage, it normally resets the internal state of the tenant to defaults, including setting the member m_pIStorage to NULL. However, if fReopen is TRUE, this NULL assignment is skipped.

   The crash occurs when m_pIStorage is left non-NULL but the storage itself has been destroyed. Thus m_pIStorage is an invalid pointer.

   To correct this problem it is necessary to insure that the m_pIStorage variable is set to NULL any time the actual reference count on it is zero

   A condition in the function CTenant::Close in TENANT.CPP of both chapters is removed to correct the problem:

   The lines:

   ```
   if (!fReopen)
       m_pIStorage=NULL;
   ```

   are replaced with just:

   ```
   m_pIStorage=NULL;
   ```

   The fReopen flag was used in an attempt to provide some optimization when performing Activate As, but is not necessary. The updated sample code has removed the flag entirely. This affects PAGE.CPP, TENANT.CPP, and TENANT.H files in Patron of chapters 14 and 15.

2. Convert To function doesn't seem to work in Patron samples of chapters 14 and 15. The problem occurs in the function CPage::FConvertObject in the PAGE.CPP file of both chapters.

The problem is that CPage::FConvertObject is releasing the page's IStorage pointer without commiting it. Since Patron uses transacted storage, this discards all changes including the conversion that just happened.

The code in error is as follows:

```
if ((CF_SELECTCONVERTTO & ct.dwFlags)
    && !IsEqualCLSID(ct.clsid, ct.clsidNew))
    {
    LPSTORAGE   pIStorage;

    //This should be the only close necessary.
    m_pTenantCur->RectGet(&rcl, FALSE);
    m_pTenantCur->StorageGet(&pIStorage);
    m_pTenantCur->Close(FALSE, FALSE);

    hr=OleStdDoConvert(pIStorage, ct.clsidNew);
    pIStorage->Release();

...
```

This code does not commit the changes to the storage affected by OleStdDoConvert before releasing pIStorage. To correct this problem, include a call to pIStorage->Commit:

```
if ((CF_SELECTCONVERTTO & ct.dwFlags)
    && !IsEqualCLSID(ct.clsid, ct.clsidNew))
    {
    LPSTORAGE   pIStorage;

    //This should be the only close necessary.
    m_pTenantCur->RectGet(&rcl, FALSE);
    m_pTenantCur->StorageGet(&pIStorage);
    m_pTenantCur->Close(FALSE);

    hr=OleStdDoConvert(pIStorage, ct.clsidNew);

    pIStorage->Commit(STGC_ONLYIFCURRENT);
    pIStorage->Release();

...
```

Note that the second parameter to CTenant::Close has also been removed reflecting Bug Fix #1.

3. Unitialized variable in CLASSLIB causes string allocation failure with non-Microsoft compilers. In the file CSTRTABL.CPP, function CStringTable::FInit, initialize the variable cchUsed to zero:

```
UINT        cchUsed=0;
```

4. The CImpIPolyline::WriteToFile function in PolyLine chapter 4 returns an incorrect value. In this function in the file IPOLYLIN.CPP, the line:

   ```
   return (m_pObj->m_fDirty) ?
   ```

   is changed to read:

   ```
   return (!m_pObj->m_fDirty) ?
   ```

5. The Polyline samples of chapters 5, 6, 11, and 16 attempt to initialize a clipboard format with an invalid string. The following line is found in the CPolyline::CPolyline function in the POLYLINE.CPP file of chapters 5, 6, 11, and 16:

   ```
   m_cf=RegisterClipboardFormat(PSZ(IDS_STORAGEFORMAT));
   ```

   This call is incorrectly made before the stringtable used by the PSZ macro is initialized. To correct the problem, the call is moved into CPolyline::FInit after the stringtable is initialized:

   ```
   ...

   if (!m_pST->FInit(IDS_POLYLINEMIN, IDS_POLYLINEMAX))
       return FALSE;

   m_cf=RegisterClipboardFormat(PSZ(IDS_STORAGEFORMAT));

   ...
   ```

   In chapter 6 it also requires movement of lines that depend on m_cf from the constructor until after the code above in CPolyline::FInit. This affects code that initializes the FORMATETC arrays elements m_rgfeGet[0] and m_rgfeSet[0].

6. The return value from CImpIOleObject::GetClientSite in the IOLEOBJ.CPP file of Cosmo, chapters 10, 13, 14, and 16 is changed to NOERROR from ResultFromScode(E_NOTIMPL).

7. Component Cosmo, Chapter 6, fails to release the Polyline's IDataObject interface in the destructor CCosmoDoc::~CCosmoDoc after calling IDataObject::DUnadvise. In DOCUMENT.CPP the lines below:

   ```
   if (SUCCEEDED(hr))
       pIDataObject->DUnadvise(m_dwConn);
   ```

   are changed to read:

```
    if (SUCCEEDED(hr))
        {
        pIDataObject->DUnadvise(m_dwConn);
        pIDataObject->Release();
        }
```

8. The Data Transfer Object of Chapter 7 does not call AddRef on IStorage and IStream objects returned from CImpIDataObject::GetData (IDATAOBJ.CPP). In this function the following lines are added after the line "*pSTM=pRen->stm;":

```
    /*
     * Must remember to AddRef any other objects
     * in the STGMEDIUM:  storages and streams.
     */
    if (TYMED_ISTORAGE==pSTM->tymed)
        pSTM->pstg->AddRef();

    if (TYMED_ISTREAM==pSTM->tymed)
        pSTM->pstm->AddRef();
```

   This will correctly insure that multiple copies of a STGMEDIUM containing a storage element are each counted such that the storage element does not become invalid prematurely.

9. The Patron samples of chapters 5 and beyond do not work correctly with some printer drivers. The problem is that Patron is not sensitive to drivers that store driver-specific information at the end of the DEVMODE structure. In making copies of the DEVMODE structure, Patron truncates that driver-specific data.

   This will cause the CreateDC and CreateIC functions to fail, which manifests in Patron as a blank document window with no visible pages. It will also cause the PrintDlg function to GP Fault inside the printer driver when that driver attempts to access driver-specific data that does not exist. The problems only appear on some printer drivers because other use no driver-specific data and are thus unaffected by the truncation.

   The correction identically affects the implementation of the CPages class of Patron in chapters 2, 5, 7, 8, 9, 12, 13, 14, and 15. CPages is defined in PAGES.H and implemented in PAGES.CPP. The functions of interest are CPages::DevModeSet, CPages::DevModeGet in all chapters, CPages::ConfigureForDevice in chapter 5, and CPages::DevReadConfig in all chapters except 5. These functions are modified to be sensitive to the variable-length nature of DEVMODE in order to work with all printer drivers, and the changes are too complex to be listed here. The listings of these functions on pages 257-259 of Inside OLE 2 are thus incorrect as well.

   NOTE: the changes render files generated by previous versions of Patron unreadable by a new version. This is, after all, a sample not meant for production work. Files may, however, be converted by running an old version of Patron from chapter 14 or 15, loading the file, and transferring all objects to a new document opened in another version of Patron from chapter 15 or 14 (opposite of the old version you are running), and saving that new document.

On a related not, printing page ranges from Patron always prints the range offset by one. Corrections to this problem have been made in the CPages::Print function of the PAGEWIN.CPP file in all chapters.

10. The Cosmo application will fail to load an object when activated if used in conjunction with the custom object handler HCOSMO from chapter 11. This will occur when either reloading an object from a save container file or reactivating the object after it has been created and deactivated once already.

    The problem is that HCOSMO imcorrectly keeps the object stream open in the object's IStorage so it can perform low-memory saves without having to open the stream again. However, the OLE 2 implementation of STORAGE.DLL only allows one process to open any one particular stream using the same IStorage pointer as a parent. Therefore when the Cosmo application receives the IStorage pointer and attempts to open the same stream, it fails. This shows up in HCOSMO's implementation of IOleObject::DoVerb where it delegates the call to the default handler. In this case the default handler passes back Cosmo's error code, which is STG_E_READFAULT.

    Since the HCOSMO handler never makes changes to the object, it will never have to save anything in a low-memory situation. Specifically, it will never have to save any changes to the storage when its IPersistStorage::Save is called with the fSameAsLoad flag set to TRUE. Since this is the only case where an IPersistStorage implementation should not attempt to allocate memory, it is totally unnecessary for HCOSMO to cache any pointers to the stream.

    To correct the problem, remove the m_pIStorage and m_pIStream members from the CFigure class defined in HCOSMO.H and FIGURE.CPP. Then remove any code that deals with them in HCOSMO's implementation of IPersistStorage found in IPERSTOR.CPP:

    a. In CImpIPersistStorage::InitNew, remove the "HRESULT hr;" local variable and the NULL check on m_pObj->m_pIStorage at the top of the function. Then remove all code between and including the lines:

    ```
    hr=pIStorage->CreateStream("CONTENTS", STGM_DIRECT
    ```

    and

    ```
    m_pObj->m_pIStorage=pIStorage;
    ```

    The remainder of the code is still important to initailize the handler's internal storage.

b. In CImpIPersistStorage::Load, remove the NULL check on m_pObj->m_pIStorage. After the call to pIStream->Read, add the line:

```
pIStream->Release();
```

Then remove the lines that hold the IStorage:

```
pIStorage->AddRef();
m_pObj->m_pIStorage=pIStorage;
```

c. In CImpIPersistStorage::Save, replace the "if (fSameAsLoad)" condition with:

```
if (fSameAsLoad)
    {
    m_pObj->m_pDefIPersistStorage->Save(pIStorage
        , fSameAsLoad);
    return NOERROR;
    }
```

Move all the remaining code in the "else" condition out of the condition completely and add the following line just before the call to WriteFmtUserTypeStg:

```
WriteClassStg(pIStorage, CLSID_Cosmo2Figure);
```

d. In CImpIPersistStorage::SaveCompleted, remove all code except for:

```
m_pObj->m_pDefIPersistStorage->SaveCompleted(pIStorage);
return NOERROR;
```

e. In CImpIPersistStorage::HandsOffStorage, remove all code except for:

```
m_pObj->m_pDefIPersistStorage->HandsOffStorage();
return NOERROR;
```

11. The "(c)" in all ABOUT.DLG files is replaced with the ANSI 169 copyright character.

12. The CClient::QueryCloseAllDocuments function in the CCLIENT.CPP file of CLASSLIB may cause a GP Fault when closing an application with one or more documents iconized. The problem may be corrected by adding the line "hPrevClose=NULL" in the code below:

```
BOOL CClient::QueryCloseAllDocuments(BOOL fClose)
    {
    ...
```

```
        for ( ; hWndT; hWndT=GetWindow(hWndT, GW_HWNDNEXT))
            {
            if (NULL!=hPrevClose)
                {
                pDoc=(LPCDocument)SendMessage(hPrevClose
                    , DOCM_PDOCUMENT, 0, 0L);
                CloseDocument(pDoc);
                hPrevClose=NULL;
                }

        ...


        //Close the last window as necessary.
        if (fClose && NULL!=hPrevClose)
            {
            pDoc=(LPCDocument)SendMessage(hPrevClose, DOCM_PDOCUMENT, 0,
0L);
            CloseDocument(pDoc);
            hPrevClose=NULL;
            }


        ...
```

13. STASTRIP, file INIT.C, contains a call to LocalAlloc where the return value is incorrectly cast immediately into a far pointer:

```
        pST->ppsz=(LPSTR *)LocalAlloc(...)
```

This is changed to first cast the return value into a near pointer then into a far pointer to insure the conversion happens properly:

```
        pST->ppsz=(LPSTR *)(char *)LocalAlloc(...)
```

14. The implementation of IEnumRECT::Next in both CHAP03\ENUMCPP and CHAP03\ENUMC samples, file IENUM.C[PP] is incorrect on two counts. First, an enumerator can accept a NULL in the last parameter to Next (called pceltFetched in the OLE 2 documentation, pdwRects in these samples) if the first parameter (celt in docs, cRects in samples) is one. Therefore the code:

```
        if (NULL==pdwRects)
            return FALSE;

        *pdwRects=0L;
```

is changed to read:

```
        if (NULL==pdwRects)
            {
            if (1L!=cRect)
                return FALSE;
            }
        else
            *pdwRects=0L;
```

Second, the value stored in pceltFetched (pdwRects) is incorrect. The line:

```
        *pdwRects=(cRectReturn-cRect);
```

is changed to:

```
        if (NULL!=pdwRects)
            *pdwRects=cRectReturn;
```

The same changes affect the IENUM.CPP files in the INTERFACE, CHAP06\POLYLINE, CHAP06\DDATAOBJ, CHAP06\EDATAOBJ, and CHAP07\DATATRAN directories as well.

15. The CStringTable in CLASSLIB file CSTRTABL.CPP will create an array of invalid string pointers in CStringTable::FInit if the call to _frealloc returns a different memory block. To avoid this problem simply delete the following lines:

```
        //Now reallocate the string memory to however much we used, plus 1
        psz=(LPSTR)_frealloc(m_pszStrings, cchUsed+1);

        if (NULL!=psz)
            m_pszStrings=psz;
```

This wastes a little memory, but is simpler than trying to recompute all the string pointers. CLASSLIB is, after all, meant to be simple for the purposes of sample code.

Section 3: Other Topics
----

1. Corrections to Inside OLE 2 printed text.

```
    Page 82-:  The exact implementations of CImpIEnumRECT::Next are
    Page 83;   incorrect according to Bug Fix #14 above.  Also, on page
    Page 92-   83 there should be no space between "book" and "1632"
    Page 93    in the "#include <book 1632.h>"

    Page 105:  The second use of QueryInterface in the Transitive
               property should read

                    "pInterface2->QueryInterface(IInterface3)"
```

The book has IInterface2 as the parameter to QueryInterface, which is incorrect.

Page 157: A space is missing between "LPLPVOID" and "ppv" in the last parameter of the declaration CKoala::QueryInterface.

Page 161: The prototype for WEP should return an int and not VOID.

Page 197: The line of code under "if (IsEqual(riid, IID_IAnimal))" should read "*ppv=(LPVOID)m_pAnimal". In the text the underscore is mistakenly a space, dash, and space.

Page 199: A space is missing between "LPLPVOID" and "ppv" in the last parameter of the declaration CKoala::QueryInterface.

Page 243: The second sentence in the second paragraph should read: "The Structured Storage definition of IStream allows streams to contain up to 2^64 (2 raised to the 64th power) addressable bytes of data." The text incorrectly reads "264" instead of "2^64."

Page 257-: The code listing of CPages::DevModeSet, CPages::DevModeGet,
Page 259 and CPages::ConfigureForDevice will change according to Bug Fix #9 above.

Page 331: The description of IDataObject::GetCanonicalFormatEtc is incorrect. For correct details, see the OLE 2 Programmer's Reference.

Page 289-: Both LibMain and WEP functions should return an int as
Page 290 described in Compile Fix #7.

Page 316: The implementation of CImpIEnumFormatEtc::Next is incorrect according to Bug fix #14 above.

Page 365: The "const LPRECTL" parameters in IViewObject::Draw should be of type "LPCRECTL" to match OLE 2.01 as described in Compiler Error #1 above.

Page 544-: In OLE 2.01 there is an additional parameter to
Page 546 OleUIAddVerbMenu called "idVerbMax" which comes immediately after "idVerbMin." This was an addition to OLE 2.01 which is not present in OLE 2.00 as described in Compiler Error #2 above.

Page 554: Under OLE 2.01 the OleStdGetObjectDescriptorFromOleObject function used in the code sample requires an additional LPSIZEL as the last parameter as described in Compiler Error #3 above.

Page 630: The right curly brace just above the "else" at the bottom of the page should be indented to align with the call to ModifyMenu above it.

Page 663-: Page 665 The bottom of page 663 mentions how a handler should hold onto open streams for low-memory save situations. This is ONLY important for elements that are entirely manipulated by the handler and never touched by the server. The handler should not cache or hold pointers to the same elements the server must access as described in Bug Fix #10 above. If a handler manipulates and changes data unaffected by the server, it must fulfill the IPersistStorage contract for only those formats. The server is responsible for all others that it modifies.

Page 668: The "const LPRECTL" parameters in IViewObject::Draw should be of type "LPCRECTL" to match OLE 2.01 as described in Compiler Error #1 above.

Page 687: There should not be "//" before the large bold "IOLEOBJ.CPP" in this code listing. The label is not part of the code.

Page 788: Same problem with OleStdGetObjectDescriptorFromOleObject as described above for page 554.

Page 814: Same problem with OleUIAddVerbMenu as described above for pages 544-546.

Page 817: Remove the second parameter to pTenantCur->Close. It should read "pTenantCur->Close(FALSE);" This matches the change necessary for Bug Fix #1 above.

Page 817: There should be a call to pIStorage->Commit(STGC_ONLYIFCURRENT) immediately after the call to OleStdDoConvert and before the call to pIStorage->Release, as described in Bug Fix #2 above.

Page 819: Remove the second parameter to pTenantCur->Close. It should read "pTenantCur->Close(FALSE);" This matches the change necessary for Bug Fix #1 above.

Page 861: The second sentence of the third paragraph under "Active vs. UI Active and Inside-Out Objects" should read: "The answer...as they are visible-end users DO NOT have to double-click..." The DO NOT is important.

2. Clarification of 64K limit on IMalloc allocations

   Inside OLE 2 states on page 127 that any single allocation done through the default shared or task allocators in OLE 2 are limited to 64K. This is true for both OLE 2.00 and OLE 2.01.

   The OLE 2 specifications and Programmer's Refernece state that "If the allocation is greater than 64K, a huge pointer is returned." This seems to contradict the actual allocators implemented in OLE 2.

   The reason is that the specs and documentation provide the standard definition of the IMalloc interface, not the specs nor the documenation for OLE 2's implementation of that interface on its standard shared and task allocators. While IMalloc is not inherently limited to 64K in the specifications, OLE 2's default implementation is limited.

   An application can implement its own allocator and provide an IMalloc interface that can allocate blocks larger than 64K, in which case that custom allocator must return a huge pointer for that allocation.

3. HCOSMO handler limitations. The sample handler HCOSMO from chapter 11 of Inside OLE 2 is meant to be used with the version of Cosmo from Chapter 10. There are no provisions for using this handler with any other version of Cosmo, specifically those in chapters 13, 14, and 16. With those later versions, you may encounter GP Faults. To bypass problems, be sure that OLE2.DLL is registered for Cosmo's InProcHandler instead of HCOSMO.DLL. The .REG files in the CHAP13, CHAP14, and CHAP16 directories will make this change. Be sure to merge those files with your registration database before using any sample from the corresponding chapter.

4. Information on Windows NT ports of Inside OLE 2 Samples

   This revision of the sample code in the following directories contain full pre-release Win32 ports of their respective samples:

   INC
   BTTNCUR
   GIZMOBAR
   STASTRIP
   CLASSLIB
   CHAP02\SKEL

    CHAP02\COSMO10
    CHAP02\COSMO
    CHAP02\PATRON

These are intended to serve as examples of how the remainder of the code would be ported. Mostly there are changes to remove "FAR PASCAL" function types, change functions like "MoveTo" to "MoveToEx" and correcting minor oversights in parameter packing and so forth. Note that several new macros and definitions have been added to the file \INC\BOOK1632.H to handle the updated sources that compile for both Win3.1 and Windows NT target environments. If you encounter a strange function or macro in the sources that you don't recognize, check in this file.

The samples above have been compiled with Visual C++ 1.1 for Windows NT and run under Windows NT version 3.1. There is, however, no guarantee for perfect compilations or execution of these pre-release samples.

Note that the .LIB files in the LIB directory and the .DLL files in the BUILD directory are Win16 versions.

These samples are currently ANSI. A final Unicode Win32 port of the sample code will be available after OLE 2 on Windows NT has shipped final. Other interim releases may be available before then, but no guarantees.

OLEANSI

Developer's Notes.

[Overview]

Welcome to the world of UNICODE! 32bit OLE2 is now a native UNICODE subsystem for Microsoft's Win32 platforms. OLE2ANSI is a .DLL that will 'thunk' existing ANSI OLE2 APIs and Interfaces in your current application to/from UNICODE.

While this sample code does make converting existing ANSI OLE2 application to UNICODE OLE2 almost painless; it does exact some overhead. If at all possible, ANSI applications should be ported to UNICODE.

OLE2ANSI is NOT part of the OLE2 subsystem, it's sample code of how a generic ANSI <--> UNICODE thunk for OLE2 could be written. Feel free to use fragments of code for you development needs.

[Using Ole2Ansi.Dll]

1. Change the "TARGET = ole2ansi" value in the file 'makefile' to an unique name to avoid conflicts. Any references to "OLE2ANSI" in this 'readme.txt' file actually refers to your unique target name.

2. Build the Ole2Ansi sample code with the provided 'makefile'.

3. Add -DOLE2ANSI to your application's compiler option list.

4. Add OLE2ANSI.LIB to your application's link library list. OLE2ANSI.LIB must be placed before OLE32.LIB and OLEAUT32.LIB.

5. Recompile/link your application.

[Suggestions for performance enhancements]

1. Use UNICODE strings where possible to avoid some (or all) wrapped API's and interfaces.

2. Manually wrap interfaces via Ole2AnsiWFromA and retain for the life of the ANSI object. Pass the wrapped interface (which are now UNICODE) directly to OLE2's APIs and methods.

3. Merge the Ole2Ansi source in your application. Note that you will need to rename some of the OLE2 APIs in your source code. For example, if your application calls CLSIDFromString(), it will need to be changed to CLSIDFromStringA().

[Release Notes]

1. Ole2Ansi will now detect bad applications that do the following and recover.
CreateObject(&pObj); pObj->QI(IID_IUnknown, &pUnk);
pUnk->Release(); pUnk->Release();

A message will be displayed to the debugger.

2. Added two new API's to permit convertion of Interfaces to/from ANSI/UNICODE:
HRESULT Ole2AnsiAFromW(REFIID iid, LPUNKNOWN pWide, LPUNKNOWN * pANSI);
HRESULT Ole2AnsiWFromA(REFIID iid, LPUNKNOWN pANSI, LPUNKNOWN * pWide);

3. Must set HKEY_CURRENT_USER\Software\Microsoft\Ole2Ansi\Trace to DWORD Hex FFFF to enable Ole2Ansi tracing.

MAKEFILE (OLEANSI Sample)

```
!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

#
#  This makefile will build the library Ole2Ansi.Dll.
#
#
#  To Build:
#
#    Make sure the environment variables Include and Lib are defined and
#    a C++ compiler is in the current path.
#
#    Then just type "nmake"
#
#

#Add '#' to the next line for 'noisy' operation
!CMDSWITCHES +s

#
#Compiler flags
#Use "SET NODEBUG=1" to compile non-debug version.
#

!IFDEF NODEBUG
DEFS = -DMAKEDLL -DWIN32 -D_WIN32
!ELSE
DEFS = -DMAKEDLL -DDBG -DWIN32 -D_WIN32
!ENDIF

.SUFFIXES: .h .obj .exe .cpp .rc .def

TARGET  = ole2ansi

goal:   $(TARGET).dll

clean:
    del *.obj
    del *.exp
    del *.lib
    del *.dll
    del *.pch

#  String pooling ok for Ole2Ansi.
cflags = -Gf $(cflags)

PCHFLAGS= -Yuole2ansi.h -Fpole2ansi.pch

INCLS    = ole2ansi.h

PRECOMPOBJ = ole2ansi.obj

OBJS1    = AnsiComp.obj WideComp.obj AnsiStor.obj WideStor.obj
```

```
OBJS2    = AnsiDvOb.obj WideDvOb.obj AnsiMoni.obj WideMoni.obj
OBJS3    = AnsiOle1.obj WideOle1.obj AnsiOle2.obj WideOle2.obj AnsiOle3.obj
OBJS4    = AnsiCtl.obj WideCtl.obj ConvCtl.obj WrapCtl.obj
OBJS5    = AnsiDisp.obj WideDisp.obj Thunk.obj Globals.obj
OBJS6    = AnsiOA.obj Convert.obj Convert2.obj Wrapper.obj Trace.obj
Debug.obj
OBJS     = $(OBJS1) $(OBJS2) $(OBJS3) $(OBJS4) $(OBJS5) $(OBJS6)

LIBS     = ole32.lib oleaut32.lib kernel32.lib user32.lib uuid.lib
advapi32.lib $(libcdll)

#####

ole2ansi.pch: ole2ansi.h ole2ansi.cpp
    echo +++++++++
    echo Precompiling ole2ansi.h
    $(cc) $(cflags) $(cdebug) $(DEFS) -Ycole2ansi.h -Fpole2ansi.pch
-Foole2ansi ole2ansi.cpp

.cpp.obj:
    echo ++++++++++
    echo Compiling $*.cpp
    $(cc) $(cflags) $(cdebug) $(DEFS) $(PCHFLAGS) $*.cpp

.c.obj:
    echo ++++++++++
    echo Compiling $*.c
    $(cc) $(cflags) $(cdebug) $(DEFS) $*.c

#Build import library
$(TARGET).lib: ole2ansi.pch $(OBJS) ole2ansi.def
    echo ++++++++++
    echo Building Import Library
!IF "$(CPU)" != "i386"
    @copy ole2ansi.rsc ole2ansi.def
!ENDIF
    $(implib) -machine:$(CPU)   \
    -def:ole2ansi.def           \
    $(OBJS)                     \
    $(PRECOMPOBJ)               \
    -out:$(TARGET).lib
    if not exist ..\lib mkdir ..\lib
    copy $(TARGET).lib ..\lib

#Build a linker response file depending on build flags
$(TARGET).dll: ole2ansi.pch $(TARGET).lib $(OBJS) ole2ansi.def
    echo ++++++++++
    echo Linking $@
    $(link) $(ldebug) $(dlllflags) \
        /Out:$(TARGET).dll      \
        $(OBJS)                 \
        $(PRECOMPOBJ)           \
        $(OLELIBS)              \
        $(TARGET).exp           \
        $(LIBS)
```

```
if not exist ..\bin mkdir ..\bin
copy $(TARGET).dll ..\bin
```

ANSICOMP.H (OLEANSI Sample)

```
//
+---------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       AnsiComp.cpp
//
//  Contents:   ANSI Wrappers for Unicode CompObj Interfaces and APIs.
//
//  Classes:    CEnumStringA - ANSI wrapper object for IEnumString.
//
//  History:    01-Nov-93   v-kentc     Created.
//
//---------------------------------------------------------------------------
-


interface IStreamA;
typedef IStreamA * LPSTREAMA;

typedef char FAR* BSTRA;


/*---------------------------------------------------------------------*/
/*                            IEnumStringA                             */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IEnumStringA

DECLARE_INTERFACE_(IEnumStringA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IEnumString methods ***
     STDMETHOD(Next) (THIS_ ULONG celt,
                           char * * rgelt,
                           ULONG * pceltFetched) PURE;
     STDMETHOD(Skip) (THIS_ ULONG celt) PURE;
     STDMETHOD(Reset) (THIS) PURE;
     STDMETHOD(Clone) (THIS_ IEnumStringA * * ppenm) PURE;
};
typedef IEnumStringA * LPENUMSTRINGA;


/*---------------------------------------------------------------------*/
/*                             IMarshalA                               */
/*---------------------------------------------------------------------*/
```

```
#undef  INTERFACE
#define INTERFACE   IMarshalA

DECLARE_INTERFACE_(IMarshalA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IMarshal methods ***
     STDMETHOD(GetUnmarshalClass)(THIS_ REFIID riid, LPVOID pv,
                 DWORD dwDestContext, LPVOID pvDestContext,
                 DWORD mshlflags, LPCLSID pCid) PURE;
     STDMETHOD(GetMarshalSizeMax)(THIS_ REFIID riid, LPVOID pv,
                 DWORD dwDestContext, LPVOID pvDestContext,
                 DWORD mshlflags, LPDWORD pSize) PURE;
     STDMETHOD(MarshalInterface)(THIS_ LPSTREAMA pStm, REFIID riid,
                 LPVOID pv, DWORD dwDestContext, LPVOID pvDestContext,
                 DWORD mshlflags) PURE;
     STDMETHOD(UnmarshalInterface)(THIS_ LPSTREAMA pStm, REFIID riid,
                 LPVOID FAR* ppv) PURE;
     STDMETHOD(ReleaseMarshalData)(THIS_ LPSTREAMA pStm) PURE;
     STDMETHOD(DisconnectObject)(THIS_ DWORD dwReserved) PURE;
};
typedef IMarshalA * LPMARSHALA;



/*----------------------------------------------------------------------*/
/*                           IStdMarshalInfoA
*/
/*----------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IStdMarshalInfoA

DECLARE_INTERFACE_(IStdMarshalInfoA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IStdMarshalInfo methods ***
     STDMETHOD(GetClassForHandler)(THIS_ DWORD dwMemctx, LPVOID pvMemctx,
                 LPCLSID pclsid) PURE;
};
typedef IStdMarshalInfoA * LPSTDMARSHALINFOA;


#ifndef NOERRORINFO
/*----------------------------------------------------------------------*/
/*                    IErrorInfoA/ICreateErrorInfoA                     */
/*----------------------------------------------------------------------*/
```

```
#undef  INTERFACE
#define INTERFACE   IErrorInfoA

DECLARE_INTERFACE_(IErrorInfoA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IErrorInfo methods ***
     STDMETHOD(GetGUID)(THIS_ GUID *pguid) PURE;
     STDMETHOD(GetSource)(THIS_ BSTRA *pbstrSource) PURE;
     STDMETHOD(GetDescription)(THIS_ BSTRA *pbstrDescription) PURE;
     STDMETHOD(GetHelpFile)(THIS_ BSTRA *pbstrHelpFile) PURE;
     STDMETHOD(GetHelpContext)(THIS_ DWORD *pdwHelpContext) PURE;

};
typedef IErrorInfoA * LPERRORINFOA;


#undef  INTERFACE
#define INTERFACE  ICreateErrorInfoA

DECLARE_INTERFACE_(ICreateErrorInfoA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** ICreateErrorInfo methods ***
     STDMETHOD(SetGUID)(THIS_ REFGUID rguid) PURE;
     STDMETHOD(SetSource)(THIS_ LPSTR szSource) PURE;
     STDMETHOD(SetDescription)(THIS_ LPSTR szDescription) PURE;
     STDMETHOD(SetHelpFile)(THIS_ LPSTR szHelpFile) PURE;
     STDMETHOD(SetHelpContext)(THIS_ DWORD dwHelpContext) PURE;
};
typedef ICreateErrorInfoA * LPCREATEERRORINFOA;

#endif //!NOERRORINFO


//
+---------------------------------------------------------------------------
//
//  Class:      CUnknownA
//
//  Synopsis:   Class definition of IUnknownA
//
//----------------------------------------------------------------------------
-
class CUnknownA : public CInterface
```

```
{
public:
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface)(REFIID riid, LPVOID * ppvObj);
     STDMETHOD_(unsigned long, AddRef)(void);
     STDMETHOD_(unsigned long, Release)(void);

     CUnknownA(LPUNKNOWN punk, IDINTERFACE idRef = ID_NULL);
     void AddInterface(IDINTERFACE idInterface, LPUNKNOWN pUnk);
     long FreeInterface(IDINTERFACE idInterface);

public:
     UINT        m_idPrimary;
     LPUNKNOWN   m_pInterfaces[ID_SIZE];
};


//
+--------------------------------------------------------------------------
//
//  Class:      CEnumStringA
//
//  Synopsis:   Class definition of IEnumStringA
//
//--------------------------------------------------------------------------
-
class CEnumStringA : CAnsiInterface
{
public:
     // *** IEnumStringA methods ***
     STDMETHOD(Next) (ULONG celt,
                                char * * rgelt,
                                ULONG FAR* pceltFetched);
     STDMETHOD(Skip) (ULONG celt);
     STDMETHOD(Reset) (VOID);
     STDMETHOD(Clone) (IEnumStringA * * ppenm);

     inline CEnumStringA(LPUNKNOWN pUnk, IEnumString * pObj) :
                 CAnsiInterface(ID_IEnumString, pUnk, (LPUNKNOWN)pObj) {};

     inline IEnumString * GetWide() const
                 { return (IEnumString *)m_pObj; };
};


//
+--------------------------------------------------------------------------
//
//  Class:      CClassFactoryA
//
//  Synopsis:   Class definition of IClassFactoryA
//
//--------------------------------------------------------------------------
-
class CClassFactoryA : CAnsiInterface
```

```
{
public:
     // *** IClassFactory methods ***
     STDMETHOD(CreateInstance) (LPUNKNOWN pUnkOuter,
                         REFIID riid,
                         LPVOID FAR* ppvObject);
     STDMETHOD(LockServer) (BOOL fLock);

     inline CClassFactoryA(LPUNKNOWN pUnk, IClassFactory * pObj) :
                 CAnsiInterface(ID_IClassFactory, pUnk, (LPUNKNOWN)pObj) {};

     inline IClassFactory * GetWide() const
                 { return (IClassFactory *)m_pObj; };
};

typedef IClassFactory * LPCLASSFACTORYA;


//
+----------------------------------------------------------------------------
//
//  Class:      CMarshalA
//
//  Synopsis:   Class definition of IMarshalA
//
//-----------------------------------------------------------------------------
-
class CMarshalA : CAnsiInterface
{
public:
     // *** IMarshal methods ***
     STDMETHOD(GetUnmarshalClass)(REFIID riid, LPVOID pv,
                 DWORD dwDestContext, LPVOID pvDestContext,
                 DWORD mshlflags, LPCLSID pCid);
     STDMETHOD(GetMarshalSizeMax)(REFIID riid, LPVOID pv,
                 DWORD dwDestContext, LPVOID pvDestContext,
                 DWORD mshlflags, LPDWORD pSize);
     STDMETHOD(MarshalInterface)(LPSTREAMA pStm, REFIID riid,
                 LPVOID pv, DWORD dwDestContext, LPVOID pvDestContext,
                 DWORD mshlflags);
     STDMETHOD(UnmarshalInterface)(LPSTREAMA pStm, REFIID riid,

                 LPVOID FAR* ppv);
     STDMETHOD(ReleaseMarshalData)(LPSTREAMA pStm);
     STDMETHOD(DisconnectObject)(DWORD dwReserved);

     inline CMarshalA(LPUNKNOWN pUnk, IMarshal * pObj) :
                 CAnsiInterface(ID_IMarshal, pUnk, (LPUNKNOWN)pObj) {};

     inline IMarshal * GetWide() const
                 { return (IMarshal *)m_pObj; };
};

//
+----------------------------------------------------------------------------
```

```
//
//  Class:      CStdMarshalInfoA
//
//  Synopsis:   Class definition of IStdMarshalInfoA
//
//--------------------------------------------------------------------------
-
class CStdMarshalInfoA : CAnsiInterface
{
public:
     // *** IStdMarshalInfo methods ***
     STDMETHOD(GetClassForHandler)(DWORD dwMemctx, LPVOID pvMemctx,
                 LPCLSID pclsid);

     inline CStdMarshalInfoA(LPUNKNOWN pUnk, IStdMarshalInfo * pObj) :
                 CAnsiInterface(ID_IStdMarshalInfo, pUnk, (LPUNKNOWN)pObj)
{};

     inline IStdMarshalInfo * GetWide() const
                 { return (IStdMarshalInfo *)m_pObj; };
};
```

ANSICOMP.CPP (OLEANSI Sample)

```
//
+---------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       ansicomp.cpp
//
//  Contents:   ANSI Wrappers for Unicode CompObj Interfaces and APIs.
//
//  Classes:    CUnknownA      - ANSI wrapper object for IUnknown.
//              CEnumStringA   - ANSI wrapper object for IEnumString.
//              CClassFactoryA - ANSI wrapper object for IClassFactory.
//
//  Functions:  CLSIDFromProgIDA
//              CLSIDFromProgIDA
//              CLSIDFromStringA
//              CoLoadLibraryA
//              CoCreateInstanceA
//              CoGetClassObjectA
//              IIDFromStringA
//              ProgIDFromCLSIDA
//              CoRegisterClassObjectA
//              StringFromCLSIDA
//              StringFromGUID2A
//              StringFromIIDA
//
//  History:    01-Nov-93   v-kentc     Created.
//
//---------------------------------------------------------------------------
-

#include "Ole2Ansi.h"
```

IUnknownA Implementation   (OLEANSI Sample)

```
//
*********************************************************************
//
//                    IUnknownA Implementation
//
//
*********************************************************************


//
+--------------------------------------------------------------------
//
//  Member:     CUnknownA::CUnknownA, public
//
//  Synopsis:   Constructor.
//
//  Arguments:  [pestr] -- Unicode CUnknown object.
//
//  Returns:    OLE2 result code.
//
//---------------------------------------------------------------------
-
CUnknownA::CUnknownA(IUnknown * pWide, IDINTERFACE idRef) :
           CInterface(ID_IUnknown | ID_ANSIINTERFACE, NULL,
(LPUNKNOWN)pWide)
{
     m_idPrimary    = idRef;

     memset(m_pInterfaces, 0, sizeof(m_pInterfaces));
     TraceAddRef("CUnknownA::CUnknownA", pWide, pWide->AddRef());
}


void CUnknownA::AddInterface(IDINTERFACE idInterface, LPUNKNOWN pUnk)
{
     m_pInterfaces[idInterface] = pUnk;
}


long CUnknownA::FreeInterface(IDINTERFACE idInterface)
{
     WrapDeleteWrapper(m_pInterfaces[idInterface]);
     m_pInterfaces[idInterface] = NULL;
     TraceRelease("CUnknownA::FreeInterface", this, Release());
     return 0;
}


//
+--------------------------------------------------------------------
//
//  Member:     CUnknownA::QueryInterface, public
```

```
//
//  Synopsis:   IUnknown::QueryInterface method implementation.
//
//  Arguments:  [riid] -- Interface ID.
//              [ppv]  -- Pointer to query results.
//
//  Returns:    OLE2 result code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CUnknownA::QueryInterface(REFIID riid, void * * ppv)
{
     TraceMethodEnter("CUnknownA::QueryInterface", this);

     IDINTERFACE      idRef;
     LPUNKNOWN pUnk;
     HRESULT   hResult;


     *ppv = NULL;

     if (riid == IID_IUnknown)
     {
           *ppv = this;
           TraceAddRef("CUnknownA::QueryInterface", this, AddRef());
           return ResultFromScode(S_OK);
     }

     idRef = WrapTranslateIID(riid);

     //
     //  If the id is NULL then this is not a thunked interface.
     //
     if (idRef == ID_NULL)
     {
           hResult = m_pObj->QueryInterface(riid, ppv);

           if (SUCCEEDED(hResult))
                 TraceWarning("Interface not handled");

           return hResult;
     }

     //
     //  If the requested interface has already been wrapped then use the
     //  original wrapped object.
     //
     if (m_pInterfaces[idRef])
     {
           *ppv = m_pInterfaces[idRef];
           TraceAddRef("CUnknownA::QueryInterface", m_pInterfaces[idRef],
m_pInterfaces[idRef]->AddRef());
           return ResultFromScode(S_OK);
     }
```

```
    //
    //  Query the Unicode interface and then wrap it.
    //
    hResult = m_pObj->QueryInterface(riid, (LPVOID *)&pUnk);
    if (FAILED(hResult))
          return hResult;

    *ppv = m_pInterfaces[idRef] = WrapAnyAFromW(this, idRef, pUnk);
    if (m_pInterfaces[idRef] == NULL)
          return ResultFromScode(E_OUTOFMEMORY);

    TraceAddRef("CUnknownA::QueryInterface", this, AddRef());

    return ResultFromScode(S_OK);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CUnknownA::AddRef, public
//
//  Synopsis:   IUnknown::AddRef method implementation.
//
//  Returns:    Current reference count.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP_(unsigned long) CUnknownA::AddRef()
{
    TraceMethodEnter("CUnknownA::AddRef", this);


    LONG lRet;


    lRet = InterlockedIncrement(&m_refs);

    TraceAddRef("CUnknownA::AddRef", this, m_refs);

    m_pObj->AddRef();

    return lRet;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CUnknownA::Release, public
//
//  Synopsis:   IUnknown::Release method implementation.
//
//  Returns:    Current reference count.
//
```

```
//--------------------------------------------------------------------------
-
STDMETHODIMP_(unsigned long) CUnknownA::Release()
{
     TraceMethodEnter("CUnknownA::Release", this);

     LONG lRet;


     lRet = InterlockedDecrement(&m_refs);

     TraceRelease("CUnknownA::Release", this, m_refs);

     if (m_pInterfaces[m_idPrimary] != NULL)
     {
           if (m_refs < ((CUnknownA *)m_pInterfaces[m_idPrimary])->m_refs)
           {
                 ((CUnknownA *)m_pInterfaces[m_idPrimary])->m_refs--;
                 if (((CUnknownA *)m_pInterfaces[m_idPrimary])->m_refs == 0)
                 {
                       ((CUnknownA *)m_pInterfaces[m_idPrimary])->m_pObj-
>Release();
                       WrapDeleteWrapper(m_pInterfaces[m_idPrimary]);
                       m_pInterfaces[m_idPrimary] = NULL;
                 }

                 TraceWarning("Had to adjust primary interface.");
           }
     }

     if (lRet == 0)
     {
           for (UINT i = 0; i < ID_SIZE; i++)
           {
                 if (m_pInterfaces[i])
                 {
                       TraceWarning("Interfaces not properly released.");
                       if (i != m_idPrimary)
                             ((CUnknownA *)m_pInterfaces[i])->m_pObj-
>Release();
                       WrapDeleteWrapper(m_pInterfaces[i]);
                 }
           }

           ++m_refs;
           TraceRelease("CUnknownA::Release", m_pObj, m_pObj->Release());
           --m_refs;

           //
           //  The thunked routine CreateDispTypeInfoA creates an UNICODE
image
           //  of the InterfaceData structure which must be kept around
until
           //  the TypeInfo is released.   So if we got it, time to free it.
           //
```

```
            if (m_pInterfaceData)
                ConvertInterfaceDataFree(m_pInterfaceData);

            WrapDeleteWrapper(this);

            return 0;
        }

        m_pObj->Release();

        return lRet;
}
```

IEnumStringA Implementation (OLEANSI Sample)

```
//
**********************************************************************
//
//                    IEnumStringA Implementation
//
//
**********************************************************************

//
+---------------------------------------------------------------------
//
//  Member:     CEnumStringA::Next, public
//
//  Synopsis:   Thunks Next to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------
-
STDMETHODIMP CEnumStringA::Next(
           ULONG celt,
           LPSTR * rgelt,
           ULONG * pceltFetched)
{
     TraceNotify("CEnumStringA::Next");

     ULONG   celtFetched;
     HRESULT hReturn;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IEnumString, Next));

     if (pceltFetched == NULL)
           pceltFetched = &celtFetched;

     hReturn = GetWide()->Next(celt, (OLECHAR * *)rgelt, pceltFetched);
     if (FAILED(hReturn))
           return (hReturn);

     hResult = ConvertStringArrayToA(rgelt, *pceltFetched);
     if (FAILED(hResult))
           return (hResult);

     return hReturn;
}


//
+---------------------------------------------------------------------
//
//  Member:     CEnumStringA::Skip, public
```

```
//
//  Synopsis:   Thunks Skip to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CEnumStringA::Skip(ULONG celt)
{
     TraceNotify("CEnumStringA::Skip");

     _DebugHook(GetWide(), MEMBER_PTR(IEnumString, Skip));

     return GetWide()->Skip(celt);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CEnumStringA::Reset, public
//
//  Synopsis:   Thunks Reset to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CEnumStringA::Reset(VOID)
{
     TraceNotify("CEnumStringA::Reset");

     _DebugHook(GetWide(), MEMBER_PTR(IEnumString, Reset));

     return GetWide()->Reset();
}


//
+---------------------------------------------------------------------------
//
//  Member:     CEnumStringA::Clone, public
//
//  Synopsis:   Thunks Clone to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CEnumStringA::Clone(IEnumStringA * * ppenmA)
{
     TraceNotify("CEnumStringA::Clone");

     IEnumString * penm;
```

```
    _DebugHook(GetWide(), MEMBER_PTR(IEnumString, Clone));

    *ppenmA = NULL;

    HRESULT hResult = GetWide()->Clone(&penm);
    if (FAILED(hResult))
        return (hResult);

    hResult = WrapIEnumStringAFromW(penm, ppenmA);

    if (penm)
        penm->Release();

    return hResult;
}
```

IClassFactoryA Implementation (OLEANSI Sample)

```
//
****************************************************************************
//
//                   IClassFactoryA Implementation
//
//
****************************************************************************

//
+---------------------------------------------------------------------------
//
//  Member:     CClassFactoryA::CreateInstance, public
//
//  Synopsis:   Thunks Skip to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CClassFactoryA::CreateInstance(LPUNKNOWN pUnkOuterA,
             REFIID riid, LPVOID FAR* ppvObject)
{
     TraceNotify("CClassFactoryA::CreateInstance");

     LPUNKNOWN pUnk;
     IDINTERFACE      idRef;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IClassFactory, CreateInstance));

     hResult = GetWide()->CreateInstance(pUnkOuterA, riid, (LPVOID *)&pUnk);
     if (FAILED(hResult))
           return hResult;

     //
     //  Convert the 16 bytes GUID into an internal integer for speed.
     //
     idRef = WrapTranslateIID(riid);

     hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)ppvObject);

     pUnk->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CClassFactoryA::LockServer, public
```

```
//
//  Synopsis:   Thunks Reset to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CClassFactoryA::LockServer(BOOL fLock)
{
     TraceNotify("CClassFactoryA::LockServer");

     _DebugHook(GetWide(), MEMBER_PTR(IClassFactory, LockServer));

     return GetWide()->LockServer(fLock);
}
```

IMarshalA Implementation   (OLEANSI Sample)

```
//
***************************************************************************
//
//                   IMarshalA Implementation
//
//
***************************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     CMarshalA::GetUnmarshalClass, public
//
//  Synopsis:   Thunks GetUnmarshalClass to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CMarshalA::GetUnmarshalClass(REFIID riid, LPVOID pv,
                 DWORD dwDestContext, LPVOID pvDestContext,
                 DWORD mshlflags, LPCLSID pCid)
{
     TraceNotify("CMarshalA::GetUnmarshalClass");

     _DebugHook(GetWide(), MEMBER_PTR(IMarshal, GetUnmarshalClass));

     return GetWide()->GetUnmarshalClass(riid, pv, dwDestContext,
pvDestContext,
                 mshlflags, pCid);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CMarshalA::GetMarshalSizeMax, public
//
//  Synopsis:   Thunks GetMarshalSizeMax to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CMarshalA::GetMarshalSizeMax(REFIID riid, LPVOID pv,
                 DWORD dwDestContext, LPVOID pvDestContext,
                 DWORD mshlflags, LPDWORD pSize)
{
     TraceNotify("CMarshalA::GetMarshalSizeMax");

     _DebugHook(GetWide(), MEMBER_PTR(IMarshal, GetMarshalSizeMax));
     return GetWide()->GetMarshalSizeMax(riid, pv, dwDestContext,
pvDestContext,
                 mshlflags, pSize);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CMarshalA::MarshalInterface, public
//
//  Synopsis:   Thunks MarshalInterface to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CMarshalA::MarshalInterface(LPSTREAMA pStmA, REFIID riid,
                 LPVOID pv, DWORD dwDestContext, LPVOID pvDestContext,
                 DWORD mshlflags)
{
     TraceNotify("CMarshalA::MarshalInterface");

     LPSTREAM pStm;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IMarshal, MarshalInterface));

     hResult = WrapIStreamWFromA(pStmA, &pStm);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->MarshalInterface(pStm, riid, pv, dwDestContext,
                 pvDestContext, mshlflags);

     if (pStm)
           pStm->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CMarshalA::UnmarshalInterface, public
//
//  Synopsis:   Thunks UnmarshalInterface to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CMarshalA::UnmarshalInterface(LPSTREAMA pStmA, REFIID riid,
            LPVOID FAR* ppv)
{
     TraceNotify("CMarshalA::UnmarshalInterface");

     LPSTREAM pStm;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IMarshal, UnmarshalInterface));

     hResult = WrapIStreamWFromA(pStmA, &pStm);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->UnmarshalInterface(pStm, riid, ppv);

     if (pStm)
           pStm->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CMarshalA::ReleaseMarshalData, public
//
//  Synopsis:   Thunks ReleaseMarshalData to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CMarshalA::ReleaseMarshalData(LPSTREAMA pStmA)
{
     TraceNotify("CMarshalA::ReleaseMarshalData");

     LPSTREAM pStm;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IMarshal, ReleaseMarshalData));

     hResult = WrapIStreamWFromA(pStmA, &pStm);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->ReleaseMarshalData(pStm);

     if (pStm)
           pStm->Release();

     return hResult;
```

```
}


//
+---------------------------------------------------------------------------
//
//  Member:     CMarshalA::DisconnectObject, public
//
//  Synopsis:   Thunks DisconnectObject to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CMarshalA::DisconnectObject(DWORD dwReserved)
{
     TraceNotify("CMarshalA::DisconnectObject");

     _DebugHook(GetWide(), MEMBER_PTR(IMarshal, DisconnectObject));

     return GetWide()->DisconnectObject(dwReserved);
}
```

CompObj API Thunks. (OLEANSI Sample)

```
//
***************************************************************************
//
//                          CompObj API Thunks.
//
//
***************************************************************************


//
+---------------------------------------------------------------------------
//
//  Routine:    CLSIDFromProgIDA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI CLSIDFromProgIDA(LPCSTR lpszProgIDA, LPCLSID lpclsid)
{
     TraceSTDAPIEnter("CLSIDFromProgIDA");

     WCHAR  szProgID[MAX_STRING];


     ConvertStringToW(lpszProgIDA, szProgID);

     return CLSIDFromProgID(szProgID, lpclsid);
}


//
+---------------------------------------------------------------------------
//
//  Routine:    CLSIDFromStringA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI CLSIDFromStringA(LPSTR lpszA, LPCLSID pclsid)
{
     TraceSTDAPIEnter("CLSIDFromStringA");

     WCHAR sz[MAX_STRING];


     ConvertStringToW(lpszA, sz);
     return CLSIDFromString(sz, pclsid);
}


//
+---------------------------------------------------------------------------
//
//  Routine:    CoCreateInstanceA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI CoCreateInstanceA(REFCLSID rclsid, LPUNKNOWN pUnkOuterA,
                             DWORD dwClsContext, REFIID riid, LPVOID FAR*
ppv)
{
     TraceSTDAPIEnter("CoCreateInstanceA");

     LPUNKNOWN pUnk;
     IDINTERFACE      idRef;
     HRESULT hResult;


     hResult = CoCreateInstance(rclsid, pUnkOuterA, dwClsContext, riid,
(LPVOID *)&pUnk);
     if (FAILED(hResult))
          goto Error;

     //
     //  Convert the 16 bytes GUID into an internal integer for speed.
     //
     idRef = WrapTranslateIID(riid);

     hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)ppv);

     if (pUnk)
          pUnk->Release();

Error:
     TraceSTDAPIExit("CoCreateInstanceA", hResult);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    CoGetClassObjectA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
```

```
//
//  Returns:    See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI CoGetClassObjectA(REFCLSID rclsid, DWORD dwClsContext, LPVOID
pvReserved,
           REFIID riid, LPVOID FAR* ppv)
{
     TraceSTDAPIEnter("CoGetClassObjectA");

     LPUNKNOWN pUnk;
     IDINTERFACE      idRef;
     HRESULT hResult;

     hResult =  CoGetClassObject(rclsid, dwClsContext, pvReserved, riid,
(LPVOID *)&pUnk);
     if (FAILED(hResult))
           return hResult;

     //
     //  Convert the 16 bytes GUID into an internal integer for speed.
     //
     idRef = WrapTranslateIID(riid);

     hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)ppv);

     if (pUnk)
           pUnk->Release();

     return hResult;
}



//
+---------------------------------------------------------------------------
//
//  Routine:    CoRegisterClassObjectA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI CoRegisterClassObjectA(REFCLSID rclsid, LPUNKNOWN pUnk,
           DWORD dwClsContext, DWORD flags, LPDWORD lpdwRegister)
{
     TraceSTDAPIEnter("CoRegisterClassObjectA");

     LPUNKNOWN lpCF;
     HRESULT hResult;
     hResult = WrapIUnknownWFromA(pUnk, &lpCF);
     if (FAILED(hResult))
           return hResult;

     hResult = CoRegisterClassObject(rclsid, lpCF, dwClsContext, flags,
                 lpdwRegister);

     if (lpCF)
           lpCF->Release();

     TraceSTDAPIExit("CoRegisterClassObjectA", hResult);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    CoRevokeClassObjectA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI CoRevokeClassObjectA(DWORD dwRegister)
{
     TraceSTDAPIEnter("CoRevokeClassObjectA");

     HRESULT hResult;


     hResult = CoRevokeClassObject(dwRegister);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    CoLoadLibraryA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI_(HINSTANCE) CoLoadLibraryA(LPSTR pszLibNameA, BOOL bAutoFree)
{
     TraceSTDAPIEnter("CoLoadLibraryA");
     WCHAR szLibName[MAX_STRING];


     ConvertStringToW(pszLibNameA, szLibName);

     return CoLoadLibrary(szLibName, bAutoFree);
}


//
+--------------------------------------------------------------------------
//
//  Routine:    CoLockObjectExternalA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
STDAPI CoLockObjectExternalA(LPUNKNOWN pUnkA, BOOL fLock,
           BOOL fLastUnlockReleases)
{
     TraceSTDAPIEnter("CoLockObjectExternalA");

     LPUNKNOWN pUnk;
     HRESULT hResult;


     hResult = WrapIUnknownWFromA(pUnkA, &pUnk);
     if (FAILED(hResult))
           return hResult;

     hResult = CoLockObjectExternal(pUnk, fLock, fLastUnlockReleases);

     if (pUnk)
           pUnk->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Routine:    CoDisconnectObjectA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
STDAPI CoDisconnectObjectA(LPUNKNOWN pUnkA, DWORD dwReserved)
{
     TraceSTDAPIEnter("CoDisconnectObjectA");

     LPUNKNOWN pUnk;
     HRESULT hResult;


     hResult = WrapIUnknownWFromA(pUnkA, &pUnk);
     if (FAILED(hResult))
           return hResult;

     hResult = CoDisconnectObject(pUnk, dwReserved);

     if (pUnk)
           pUnk->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Routine:    IIDFromStringA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
STDAPI IIDFromStringA(LPSTR lpszA, LPIID lpiid)
{
     TraceSTDAPIEnter("IIDFromStringA");

     WCHAR sz[MAX_STRING];


     ConvertStringToW(lpszA, sz);

     return IIDFromString(sz, lpiid);
}


//
+--------------------------------------------------------------------------
//
//  Routine:    ProgIDFromCLSIDA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
```

```
STDAPI ProgIDFromCLSIDA(REFCLSID clsid, LPSTR * lplpszProgIDA)
{
     TraceSTDAPIEnter("ProgIDFromCLSIDA");

     LPWSTR lpszProgID;
     HRESULT hResult;


     if (lplpszProgIDA == NULL)
           return ResultFromScode(E_INVALIDARG);

     hResult = ProgIDFromCLSID(clsid, &lpszProgID);
     if (FAILED(hResult))
           return (hResult);

     hResult = ConvertStringToA(lpszProgID, lplpszProgIDA);

     ConvertStringFree(lpszProgID);

     return (hResult);
}


//
+---------------------------------------------------------------------------
//
//  Routine:    StringFromCLSIDA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI StringFromCLSIDA(REFCLSID rclsid, LPSTR * lplpszA)
{
     TraceSTDAPIEnter("StringFromCLSIDA");

     LPWSTR lpsz;
     HRESULT hResult;


     hResult = StringFromCLSID(rclsid, &lpsz);
     if (FAILED(hResult))
           return (hResult);

     hResult = ConvertStringToA(lpsz, lplpszA);

     ConvertStringFree(lpsz);

     return (hResult);
}
```

```
//
+--------------------------------------------------------------------------
//
//  Routine:    StringFromGUID2A
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
STDAPI_(int) StringFromGUID2A(REFGUID rguid, LPSTR szGuid, int cbMax)
{
     TraceSTDAPIEnter("StringFromGUID2A");

   int cchRet;

#ifndef CCH_SZGUID0
// char count of a guid in ansi/unicode form (including trailing null)
#define CCH_SZGUID0 39
#endif

   OLECHAR szGuidW[CCH_SZGUID0];

   cchRet = StringFromGUID2(rguid, szGuidW, CCH_SZGUID0);

   // convert szGuidW from unicode to Ansi.  Can't just convert in place.
   WideCharToMultiByte(CP_ACP,
                    0,
                    szGuidW,
                    CCH_SZGUID0,
                    szGuid,
                    cbMax,
                    NULL,
                    NULL);
   return cchRet;
}



//
+--------------------------------------------------------------------------
//
//  Routine:    StringFromIIDA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
STDAPI StringFromIIDA(REFIID rclsid, LPSTR * lplpszA)
{
     TraceSTDAPIEnter("StringFromIIDA");
     LPWSTR lpsz;
     HRESULT hResult;

     hResult = StringFromIID(rclsid, &lpsz);
     if (FAILED(hResult))
           return (hResult);

     hResult = ConvertStringToA(lpsz, lplpszA);

     ConvertStringFree(lpsz);

     return (hResult);
}
```

IStdMarshalInfoA Implementation (OLEANSI Sample)

```
//
****************************************************************************
//
//                   IStdMarshalInfoA Implementation
//
//
****************************************************************************

//
+---------------------------------------------------------------------------
//
//  Member:     CStdMarshalInfoA::GetClassForHandler, public
//
//  Synopsis:   Thunks GetClassForHandler to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CStdMarshalInfoA::GetClassForHandler(DWORD dwMemctx,
                 LPVOID pvMemctx, LPCLSID pclsid)
{
     TraceMethodEnter("CStdMarshalInfoA::GetClassForHandler", this);

     _DebugHook(GetWide(), MEMBER_PTR(IStdMarshalInfo, GetClassForHandler));

     return GetWide()->GetClassForHandler(dwMemctx, pvMemctx, pclsid);
}
```

ANSICTL.H (OLEANSI Sample)

```
//
+---------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       AnsiCtl.h
//
//  Contents:   ANSI Wrappers for OLE Control Interfaces and APIs.
//
//  History:    27-May-94   johnels     Created.
//
//---------------------------------------------------------------------------
-


typedef interface IOleControlA FAR* LPOLECONTROLA;
typedef interface IOleControlSiteA FAR* LPOLECONTROLSITEA;
typedef interface ISimpleFrameSiteA FAR* LPSIMPLEFRAMESITEA;
typedef interface IPersistStreamInitA FAR* LPPERSISTSTREAMINITA;
typedef interface IPropertyNotifySinkA FAR* LPPROPERTYNOTIFYSINKA;
typedef interface IProvideClassInfoA FAR* LPPROVIDECLASSINFOA;
typedef interface IConnectionPointContainerA FAR*
LPCONNECTIONPOINTCONTAINERA;
typedef interface IEnumConnectionPointsA FAR* LPENUMCONNECTIONPOINTSA;
typedef interface IConnectionPointA FAR* LPCONNECTIONPOINTA;
typedef interface IEnumConnectionsA FAR* LPENUMCONNECTIONSA;
typedef struct tagCONNECTDATAA FAR* LPCONNECTDATAA;
typedef interface IClassFactory2A FAR* LPCLASSFACTORY2A;
typedef interface ISpecifyPropertyPagesA FAR* LPSPECIFYPROPERTYPAGESA;
typedef interface IPerPropertyBrowsingA FAR* LPPERPROPERTYBROWSINGA;
typedef interface IPropertyPageSiteA FAR* LPPROPERTYPAGESITEA;
typedef struct tagPROPPAGEINFOA FAR* LPPROPPAGEINFOA;
typedef interface IPropertyPageA FAR* LPPROPERTYPAGEA;
typedef interface IPropertyPage2A FAR* LPPROPERTYPAGE2A;
typedef struct tagOCPFIPARAMSA FAR* LPOCPFIPARAMSA;
typedef struct tagFONTDESCA FAR* LPFONTDESCA;
typedef interface IFontA FAR* LPFONTA;
typedef interface IFontDispA FAR* LPFONTDISPA;
typedef interface IPictureA FAR* LPPICTUREA;
typedef interface IPictureDispA FAR* LPPICTUREDISPA;


////////////////////////////////////////////////////////////////////////////
//
// IPropertyNotifySinkA interface

#undef  INTERFACE
#define INTERFACE IPropertyNotifySinkA

DECLARE_INTERFACE_(IPropertyNotifySinkA, IUnknown)
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef)(THIS) PURE;
     STDMETHOD_(ULONG,Release)(THIS) PURE;

     // IPropertyNotifySinkA methods
     STDMETHOD(OnChanged)(THIS_ DISPID dispid) PURE;
     STDMETHOD(OnRequestEdit)(THIS_ DISPID dispid) PURE;
};


/////////////////////////////////////////////////////////////////////////////
//
// IProvideClassInfoA interface

#undef  INTERFACE
#define INTERFACE IProvideClassInfoA

DECLARE_INTERFACE_(IProvideClassInfoA, IUnknown)
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef)(THIS) PURE;
     STDMETHOD_(ULONG,Release)(THIS) PURE;

     // IProvideClassInfoA methods
     STDMETHOD(GetClassInfo)(THIS_ LPTYPEINFOA FAR* ppTI) PURE;
};


/////////////////////////////////////////////////////////////////////////////
//
// IConnectionPointContainerA interface

#undef  INTERFACE
#define INTERFACE IConnectionPointContainerA

DECLARE_INTERFACE_(IConnectionPointContainerA, IUnknown)
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef)(THIS) PURE;
     STDMETHOD_(ULONG,Release)(THIS) PURE;

     // IConnectionPointContainerA methods
     STDMETHOD(EnumConnectionPoints)(
                    THIS_ LPENUMCONNECTIONPOINTSA FAR* ppEnum) PURE;
     STDMETHOD(FindConnectionPoint)(
                    THIS_ REFIID iid, LPCONNECTIONPOINTA FAR* ppCP) PURE;
};


/////////////////////////////////////////////////////////////////////////////
//
// IEnumConnectionPointsA interface
```

```
#undef  INTERFACE
#define INTERFACE IEnumConnectionPointsA

DECLARE_INTERFACE_(IEnumConnectionPointsA, IUnknown)
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef)(THIS) PURE;
     STDMETHOD_(ULONG,Release)(THIS) PURE;

     // IEnumConnectionPointsA methods
     STDMETHOD(Next)(
                       THIS_ ULONG cConnections,
                       LPCONNECTIONPOINTA FAR* rgpcn, ULONG FAR* lpcFetched)
PURE;
     STDMETHOD(Skip)(THIS_ ULONG cConnections) PURE;
     STDMETHOD(Reset)(THIS) PURE;
     STDMETHOD(Clone)(THIS_ LPENUMCONNECTIONPOINTSA FAR* ppEnum) PURE;
};

////////////////////////////////////////////////////////////////////////////
//
// IConnectionPointA interface

#undef  INTERFACE
#define INTERFACE IConnectionPointA

DECLARE_INTERFACE_(IConnectionPointA, IUnknown)
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef)(THIS) PURE;
     STDMETHOD_(ULONG,Release)(THIS) PURE;

     // IConnectionPointA methods
     STDMETHOD(GetConnectionInterface)(THIS_ IID FAR* pIID) PURE;
     STDMETHOD(GetConnectionPointContainer)(
                       THIS_ LPCONNECTIONPOINTCONTAINERA FAR* ppCPC) PURE;
     STDMETHOD(Advise)(THIS_ LPUNKNOWN pUnkSink, DWORD FAR* pdwCookie) PURE;
     STDMETHOD(Unadvise)(THIS_ DWORD dwCookie) PURE;
     STDMETHOD(EnumConnections)(THIS_ LPENUMCONNECTIONSA FAR* ppEnum) PURE;
};


////////////////////////////////////////////////////////////////////////////
//
// CONNECTDATAA structure

typedef struct tagCONNECTDATAA
{
     LPUNKNOWN pUnk;
     DWORD dwCookie;
} CONNECTDATAA;
```

```
////////////////////////////////////////////////////////////////////////////
//
// IEnumConnectionsA interface

#undef  INTERFACE
#define INTERFACE IEnumConnectionsA

DECLARE_INTERFACE_(IEnumConnectionsA, IUnknown)
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef)(THIS) PURE;
     STDMETHOD_(ULONG,Release)(THIS) PURE;

     // IEnumConnectionsA methods
     STDMETHOD(Next)(
                       THIS_ ULONG cConnections,
                       LPCONNECTDATAA rgcd, ULONG FAR* lpcFetched) PURE;
     STDMETHOD(Skip)(THIS_ ULONG cConnections) PURE;
     STDMETHOD(Reset)(THIS) PURE;
     STDMETHOD(Clone)(THIS_ LPENUMCONNECTIONSA FAR* ppecn) PURE;
};


////////////////////////////////////////////////////////////////////////////
//
// IOleControlA interface

#undef  INTERFACE
#define INTERFACE IOleControlA

DECLARE_INTERFACE_(IOleControlA, IUnknown)
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef)(THIS) PURE;
     STDMETHOD_(ULONG,Release)(THIS) PURE;

     // IOleControlA methods
     STDMETHOD(GetControlInfo)(THIS_ LPCONTROLINFO pCI) PURE;
     STDMETHOD(OnMnemonic)(THIS_ LPMSG pMsg) PURE;
     STDMETHOD(OnAmbientPropertyChange)(THIS_ DISPID dispid) PURE;
     STDMETHOD(FreezeEvents)(THIS_ BOOL bFreeze) PURE;
};


////////////////////////////////////////////////////////////////////////////
//
// IOleControlSiteA interface

#undef  INTERFACE
#define INTERFACE IOleControlSiteA

DECLARE_INTERFACE_(IOleControlSiteA, IUnknown)
```

```
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef)(THIS) PURE;
     STDMETHOD_(ULONG,Release)(THIS) PURE;

     // IOleControlSiteA methods
     STDMETHOD(OnControlInfoChanged)(THIS) PURE;
     STDMETHOD(LockInPlaceActive)(THIS_ BOOL fLock) PURE;
     STDMETHOD(GetExtendedControl)(THIS_ LPDISPATCHA FAR* ppDisp) PURE;
     STDMETHOD(TransformCoords)(THIS_ POINTL FAR* lpptlHimetric,
           POINTF FAR* lpptfContainer, DWORD flags) PURE;
     STDMETHOD(TranslateAccelerator)(THIS_ LPMSG lpMsg, DWORD grfModifiers)
PURE;
     STDMETHOD(OnFocus)(THIS_ BOOL fGotFocus) PURE;
     STDMETHOD(ShowPropertyFrame)(THIS) PURE;
};

#define XFORMCOORDS_POSITION            0x1
#define XFORMCOORDS_SIZE                0x2
#define XFORMCOORDS_HIMETRICTOCONTAINER 0x4
#define XFORMCOORDS_CONTAINERTOHIMETRIC 0x8


/////////////////////////////////////////////////////////////////////////////
//
// ISimpleFrameSiteA interface

#undef  INTERFACE
#define INTERFACE ISimpleFrameSiteA

DECLARE_INTERFACE_(ISimpleFrameSiteA, IUnknown)
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef)(THIS) PURE;
     STDMETHOD_(ULONG,Release)(THIS) PURE;

     // ISimpleFrameSite methods
     STDMETHOD(PreMessageFilter)(HWND hwnd, UINT msg, WPARAM wp, LPARAM lp,
           LRESULT FAR* lplResult, DWORD FAR * lpdwCookie) PURE;
     STDMETHOD(PostMessageFilter)(HWND hwnd, UINT msg, WPARAM wp, LPARAM lp,
           LRESULT FAR* lplResult, DWORD dwCookie) PURE;
};


/////////////////////////////////////////////////////////////////////////////
//
// IPersistStreamInitA interface

#undef  INTERFACE
#define INTERFACE IPersistStreamInitA

DECLARE_INTERFACE_(IPersistStreamInitA, IPersist)
```

```
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef)(THIS) PURE;
     STDMETHOD_(ULONG,Release)(THIS) PURE;

     // IPersistA methods
     STDMETHOD(GetClassID)(THIS_ LPCLSID lpClassID) PURE;

     // IPersistStreamInitA methods
     STDMETHOD(IsDirty)(THIS) PURE;
     STDMETHOD(Load)(THIS_ LPSTREAMA pStm) PURE;
     STDMETHOD(Save)(THIS_ LPSTREAMA pStm, BOOL fClearDirty) PURE;
     STDMETHOD(GetSizeMax)(THIS_ ULARGE_INTEGER FAR* pcbSize) PURE;
     STDMETHOD(InitNew)(THIS) PURE;
};


/////////////////////////////////////////////////////////////////////////////
//
// IClassFactory2A interface

#undef  INTERFACE
#define INTERFACE IClassFactory2A

DECLARE_INTERFACE_(IClassFactory2A, IClassFactory)
{
     //  IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef)(THIS) PURE;
     STDMETHOD_(ULONG,Release)(THIS) PURE;

     // IClassFactoryA methods
     STDMETHOD(CreateInstance)(THIS_ LPUNKNOWN pUnkOuter, REFIID riid,
                 LPVOID FAR* ppvObject) PURE;
     STDMETHOD(LockServer)(THIS_ BOOL fLock) PURE;

     //  IClassFactory2A methods
     STDMETHOD(GetLicInfo)(THIS_ LPLICINFO pLicInfo) PURE;
     STDMETHOD(RequestLicKey)(THIS_ DWORD dwReserved, BSTRA FAR* pbstrKey)
PURE;
     STDMETHOD(CreateInstanceLic)(THIS_ LPUNKNOWN pUnkOuter,
                 LPUNKNOWN pUnkReserved, REFIID riid, BSTRA bstrKey,
                 LPVOID FAR* ppvObject) PURE;
};


/////////////////////////////////////////////////////////////////////////////
//
// ISpecifyPropertyPagesA interface

#undef  INTERFACE
#define INTERFACE ISpecifyPropertyPagesA

DECLARE_INTERFACE_(ISpecifyPropertyPagesA, IUnknown)
```

```
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef)(THIS) PURE;
     STDMETHOD_(ULONG,Release)(THIS) PURE;

     // ISpecifyPropertyPagesA interface
     STDMETHOD(GetPages)(THIS_ CAUUID FAR* pPages) PURE;
};


/////////////////////////////////////////////////////////////////////////////
//
// CALPSTR structure

typedef struct tagCALPSTR
{
     ULONG cElems;
     LPSTR FAR* pElems;

} CALPSTR;


/////////////////////////////////////////////////////////////////////////////
//
// IPerPropertyBrowsingA interface

#undef  INTERFACE
#define INTERFACE IPerPropertyBrowsingA

DECLARE_INTERFACE_(IPerPropertyBrowsingA, IUnknown)
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef)(THIS) PURE;
     STDMETHOD_(ULONG,Release)(THIS) PURE;

     // IPerPropertyBrowsingA interface
     STDMETHOD(GetDisplayString)(THIS_ DISPID dispid, BSTRA FAR* lpbstr)
PURE;
     STDMETHOD(MapPropertyToPage)(THIS_ DISPID dispid, LPCLSID lpclsid)
PURE;
     STDMETHOD(GetPredefinedStrings)(THIS_ DISPID dispid,
           CALPSTR FAR* lpcaStringsOut, CADWORD FAR* lpcaCookiesOut) PURE;
     STDMETHOD(GetPredefinedValue)(THIS_ DISPID dispid, DWORD dwCookie,
           VARIANTA FAR* pvarOut) PURE;
};


/////////////////////////////////////////////////////////////////////////////
//
// IPropertyPageSiteA interface

#undef  INTERFACE
#define INTERFACE IPropertyPageSiteA
```

```
DECLARE_INTERFACE_(IPropertyPageSiteA, IUnknown)
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef)(THIS) PURE;
     STDMETHOD_(ULONG,Release)(THIS) PURE;

     // IPropertyPageSiteA methods
     STDMETHOD(OnStatusChange)(THIS_ DWORD flags) PURE;
     STDMETHOD(GetLocaleID)(THIS_ LCID FAR* pLocaleID) PURE;
     STDMETHOD(GetPageContainer)(THIS_ LPUNKNOWN FAR* ppUnk) PURE;
     STDMETHOD(TranslateAccelerator)(THIS_ LPMSG lpMsg) PURE;
};


/////////////////////////////////////////////////////////////////////////////
//
// PROPPAGEINFOA structure

typedef struct tagPROPPAGEINFOA
{
     size_t cb;
     LPSTR pszTitle;
     SIZE size;
     LPSTR pszDocString;
     LPSTR pszHelpFile;
     DWORD dwHelpContext;

} PROPPAGEINFOA;


/////////////////////////////////////////////////////////////////////////////
//
// IPropertyPageA interface

#undef  INTERFACE
#define INTERFACE IPropertyPageA

DECLARE_INTERFACE_(IPropertyPageA, IUnknown)
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef)(THIS) PURE;
     STDMETHOD_(ULONG,Release)(THIS) PURE;

     // IPropertyPageA methods
     STDMETHOD(SetPageSite)(THIS_ LPPROPERTYPAGESITEA pPageSite) PURE;
     STDMETHOD(Activate)(THIS_ HWND hwndParent, LPCRECT rect,
                    BOOL bModal) PURE;
     STDMETHOD(Deactivate)(THIS) PURE;
     STDMETHOD(GetPageInfo)(THIS_ LPPROPPAGEINFOA pPageInfo) PURE;
     STDMETHOD(SetObjects)(THIS_ ULONG cObjects, LPUNKNOWN FAR* ppunk) PURE;
     STDMETHOD(Show)(THIS_ UINT nCmdShow) PURE;
     STDMETHOD(Move)(LPCRECT prect) PURE;
     STDMETHOD(IsPageDirty)(THIS) PURE;
     STDMETHOD(Apply)(THIS) PURE;
     STDMETHOD(Help)(THIS_ LPCSTR lpszHelpDir) PURE;
     STDMETHOD(TranslateAccelerator)(THIS_ LPMSG lpMsg) PURE;
};


//////////////////////////////////////////////////////////////////////////////
//
// IPropertyPage2A interface

#undef  INTERFACE
#define INTERFACE IPropertyPage2A

DECLARE_INTERFACE_(IPropertyPage2A, IPropertyPageA)
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef)(THIS) PURE;
     STDMETHOD_(ULONG,Release)(THIS) PURE;

     // IPropertyPageA methods
     STDMETHOD(SetPageSite)(THIS_ LPPROPERTYPAGESITEA pPageSite) PURE;
     STDMETHOD(Activate)(THIS_ HWND hwndParent, LPCRECT rect,
                       BOOL bModal) PURE;
     STDMETHOD(Deactivate)(THIS) PURE;
     STDMETHOD(GetPageInfo)(THIS_ LPPROPPAGEINFOA pPageInfo) PURE;
     STDMETHOD(SetObjects)(THIS_ ULONG cObjects, LPUNKNOWN FAR* ppunk) PURE;
     STDMETHOD(Show)(THIS_ UINT nCmdShow) PURE;
     STDMETHOD(Move)(LPCRECT prect) PURE;
     STDMETHOD(IsPageDirty)(THIS) PURE;
     STDMETHOD(Apply)(THIS) PURE;
     STDMETHOD(Help)(THIS_ LPCSTR lpszHelpDir) PURE;
     STDMETHOD(TranslateAccelerator)(THIS_ LPMSG lpMsg) PURE;

     // IPropertyPage2A methods
     STDMETHOD(EditProperty)(THIS_ DISPID dispid) PURE;
};


//////////////////////////////////////////////////////////////////////////////
/
// OCPFIPARAMSA structure (parameters for OleCreatePropertyFrameIndirectA)

typedef struct tagOCPFIPARAMSA
{
     ULONG cbStructSize;
     HWND hWndOwner;
     int x;
     int y;
     LPSTR lpszCaption;
     ULONG cObjects;
     LPUNKNOWN FAR* lplpUnk;
     ULONG cPages;
     CLSID FAR* lpPages;
     LCID lcid;
     DISPID dispidInitialProperty;

} OCPFIPARAMSA;


////////////////////////////////////////////////////////////////////////////
/
// IFontA interface

#undef INTERFACE
#define INTERFACE IFontA

DECLARE_INTERFACE_(IFontA, IUnknown)
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG, AddRef)(THIS) PURE;
     STDMETHOD_(ULONG, Release)(THIS) PURE;

     // IFontA methods
     STDMETHOD(get_Name)(THIS_ BSTRA FAR* pname) PURE;
     STDMETHOD(put_Name)(THIS_ BSTRA name) PURE;
     STDMETHOD(get_Size)(THIS_ CY FAR* psize) PURE;
     STDMETHOD(put_Size)(THIS_ CY size) PURE;
     STDMETHOD(get_Bold)(THIS_ BOOL FAR* pbold) PURE;
     STDMETHOD(put_Bold)(THIS_ BOOL bold) PURE;
     STDMETHOD(get_Italic)(THIS_ BOOL FAR* pitalic) PURE;
     STDMETHOD(put_Italic)(THIS_ BOOL italic) PURE;
     STDMETHOD(get_Underline)(THIS_ BOOL FAR* punderline) PURE;
     STDMETHOD(put_Underline)(THIS_ BOOL underline) PURE;
     STDMETHOD(get_Strikethrough)(THIS_ BOOL FAR* pstrikethrough) PURE;
     STDMETHOD(put_Strikethrough)(THIS_ BOOL strikethrough) PURE;
     STDMETHOD(get_Weight)(THIS_ short FAR* pweight) PURE;
     STDMETHOD(put_Weight)(THIS_ short weight) PURE;
     STDMETHOD(get_Charset)(THIS_ short FAR* pcharset) PURE;
     STDMETHOD(put_Charset)(THIS_ short charset) PURE;
     STDMETHOD(get_hFont)(THIS_ HFONT FAR* phfont) PURE;
     STDMETHOD(Clone)(THIS_ IFontA FAR* FAR* lplpfont) PURE;
     STDMETHOD(IsEqual)(THIS_ IFontA FAR * lpFontOther) PURE;
     STDMETHOD(SetRatio)(THIS_ long cyLogical, long cyHimetric) PURE;
     STDMETHOD(QueryTextMetrics)(THIS_ LPTEXTMETRIC lptm) PURE;
     STDMETHOD(AddRefHfont)(THIS_ HFONT hfont) PURE;
     STDMETHOD(ReleaseHfont)(THIS_ HFONT hfont) PURE;
};


////////////////////////////////////////////////////////////////////////////
/
// IFontDispA interface

#undef INTERFACE
#define INTERFACE IFontDispA
```

```
DECLARE_INTERFACE_(IFontDispA, IDispatchA)
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG, AddRef)(THIS) PURE;
     STDMETHOD_(ULONG, Release)(THIS) PURE;

     // IDispatch methods
     STDMETHOD(GetTypeInfoCount)(THIS_ UINT FAR* pctinfo) PURE;

     STDMETHOD(GetTypeInfo)(
       THIS_
       UINT itinfo,
       LCID lcid,
       ITypeInfoA FAR* FAR* pptinfo) PURE;

     STDMETHOD(GetIDsOfNames)(
       THIS_
       REFIID riid,
       LPSTR FAR* rgszNames,
       UINT cNames,
       LCID lcid,
       DISPID FAR* rgdispid) PURE;

     STDMETHOD(Invoke)(
       THIS_
       DISPID dispidMember,
       REFIID riid,
       LCID lcid,
       WORD wFlags,
       DISPPARAMSA FAR* pdispparams,
       VARIANTA FAR* pvarResult,
       EXCEPINFOA FAR* pexcepinfo,
       UINT FAR* puArgErr) PURE;
};


////////////////////////////////////////////////////////////////////////////
/
// FONTDESCA structure

typedef struct tagFONTDESCA
{
     UINT  cbSizeofstruct;
     LPSTR lpstrName;
     CY    cySize;
     SHORT sWeight;
     SHORT sCharset;
     BOOL  fItalic;
     BOOL  fUnderline;
     BOOL  fStrikethrough;
} FONTDESCA;
```

```
////////////////////////////////////////////////////////////////////////////
/
// IPictureA interface

DECLARE_INTERFACE_(IPictureA, IUnknown)
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG, AddRef)(THIS) PURE;
     STDMETHOD_(ULONG, Release)(THIS) PURE;

     // IPictureA methods
     STDMETHOD(get_Handle)(THIS_ OLE_HANDLE FAR* phandle) PURE;
     STDMETHOD(get_hPal)(THIS_ OLE_HANDLE FAR* phpal) PURE;
     STDMETHOD(get_Type)(THIS_ short FAR* ptype) PURE;
     STDMETHOD(get_Width)(THIS_ OLE_XSIZE_HIMETRIC FAR* pwidth) PURE;
     STDMETHOD(get_Height)(THIS_ OLE_YSIZE_HIMETRIC FAR* pheight) PURE;
     STDMETHOD(Render)(THIS_ HDC hdc, long x, long y, long cx, long cy,
           OLE_XPOS_HIMETRIC xSrc, OLE_YPOS_HIMETRIC ySrc,
           OLE_XSIZE_HIMETRIC cxSrc, OLE_YSIZE_HIMETRIC cySrc,
           LPRECT lprcWBounds) PURE;
     STDMETHOD(set_hPal)(THIS_ OLE_HANDLE hpal) PURE;
     STDMETHOD(get_CurDC)(THIS_ HDC FAR * phdcOut) PURE;
     STDMETHOD(SelectPicture)(THIS_
           HDC hdcIn, HDC FAR * phdcOut, OLE_HANDLE FAR * phbmpOut) PURE;
     STDMETHOD(get_KeepOriginalFormat)(THIS_ BOOL * pfkeep) PURE;
     STDMETHOD(put_KeepOriginalFormat)(THIS_ BOOL fkeep) PURE;
     STDMETHOD(PictureChanged)(THIS) PURE;
     STDMETHOD(SaveAsFile)(THIS_ LPSTREAMA lpstream, BOOL fSaveMemCopy,
           LONG FAR * lpcbSize) PURE;
     STDMETHOD(get_Attributes)(THIS_ DWORD FAR * lpdwAttr) PURE;
};


////////////////////////////////////////////////////////////////////////////
/
// IPictureDispA interface

DECLARE_INTERFACE_(IPictureDispA, IDispatchA)
{
     // IUnknown methods
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR* ppvObj) PURE;
     STDMETHOD_(ULONG, AddRef)(THIS) PURE;
     STDMETHOD_(ULONG, Release)(THIS) PURE;

     // IDispatchA methods
     STDMETHOD(GetTypeInfoCount)(THIS_ UINT FAR* pctinfo) PURE;

     STDMETHOD(GetTypeInfo)(
       THIS_
       UINT itinfo,
       LCID lcid,
       ITypeInfoA FAR* FAR* pptinfo) PURE;

     STDMETHOD(GetIDsOfNames)(
       THIS_
       REFIID riid,
       LPSTR FAR* rgszNames,
       UINT cNames,
       LCID lcid,
       DISPID FAR* rgdispid) PURE;

     STDMETHOD(Invoke)(
       THIS_
       DISPID dispidMember,
       REFIID riid,
       LCID lcid,
       WORD wFlags,
       DISPPARAMSA FAR* pdispparams,
       VARIANTA FAR* pvarResult,
       EXCEPINFOA FAR* pexcepinfo,
       UINT FAR* puArgErr) PURE;
};


////////////////////////////////////////////////////////////////////////////
/
// Property sheet APIs

STDAPI OleCreatePropertyFrameA(HWND hwndOwner, UINT x, UINT y,
     LPCSTR lpszCaption, ULONG cObjects, LPUNKNOWN FAR* ppUnk,
     ULONG cPages, LPCLSID pPageClsID, LCID lcid,
     DWORD dwReserved, LPVOID pvReserved);

STDAPI OleCreatePropertyFrameIndirectA(LPOCPFIPARAMSA lpParams);


////////////////////////////////////////////////////////////////////////////
/
// Font APIs

STDAPI OleCreateFontIndirectA(LPFONTDESCA lpfd,
           REFIID riid, LPVOID FAR* lplpvObj);


////////////////////////////////////////////////////////////////////////////
/
// CPropertyNotifySinkA class

class CPropertyNotifySinkA : CAnsiInterface
{
public:
     // IPropertyNotifySinkA methods
     STDMETHOD(OnChanged)(DISPID dispid);
     STDMETHOD(OnRequestEdit)(DISPID dispid);

     inline CPropertyNotifySinkA(LPUNKNOWN pUnk, IPropertyNotifySink* pWide)
:
```

```
                 CAnsiInterface(ID_IPropertyNotifySink, pUnk,
(LPUNKNOWN)pWide) {};

     inline IPropertyNotifySink* GetWide() const
                 { return (IPropertyNotifySink*)m_pObj; };
};


////////////////////////////////////////////////////////////////////////////
/
// CProvideClassInfoA class

class CProvideClassInfoA : CAnsiInterface
{
public:
     // IProvideClassInfoA methods
     STDMETHOD(GetClassInfo)(LPTYPEINFOA FAR* ppTI);

     inline CProvideClassInfoA(LPUNKNOWN pUnk, IProvideClassInfo* pWide) :
                 CAnsiInterface(ID_IProvideClassInfo, pUnk,
(LPUNKNOWN)pWide) {};

     inline IProvideClassInfo* GetWide() const
                 { return (IProvideClassInfo*)m_pObj; };
};


////////////////////////////////////////////////////////////////////////////
/
// CConnectionPointContainerA class

class CConnectionPointContainerA : CAnsiInterface
{
public:
     // IConnectionPointContainerA methods
     STDMETHOD(EnumConnectionPoints)(
                       LPENUMCONNECTIONPOINTSA FAR* ppEnum);
     STDMETHOD(FindConnectionPoint)(
                       REFIID iid, LPCONNECTIONPOINTA FAR* ppCP);

     inline CConnectionPointContainerA(LPUNKNOWN pUnk,
IConnectionPointContainer* pWide) :
                 CAnsiInterface(ID_IConnectionPointContainer, pUnk,
(LPUNKNOWN)pWide) {};

     inline IConnectionPointContainer* GetWide() const
                 { return (IConnectionPointContainer*)m_pObj; };
};


////////////////////////////////////////////////////////////////////////////
/
// CEnumConnectionPointsA class

class CEnumConnectionPointsA : CAnsiInterface
```

```
{
public:
     // IEnumConnectionPointsA methods
     STDMETHOD(Next)(
                       ULONG cConnections,
                       LPCONNECTIONPOINTA FAR* rgpcn, ULONG FAR*
lpcFetched);
     STDMETHOD(Skip)(ULONG cConnections);
     STDMETHOD(Reset)();
     STDMETHOD(Clone)(LPENUMCONNECTIONPOINTSA FAR* ppEnum);

     inline CEnumConnectionPointsA(LPUNKNOWN pUnk, IEnumConnectionPoints*
pWide) :
                 CAnsiInterface(ID_IEnumConnectionPoints, pUnk,
(LPUNKNOWN)pWide) {};

     inline IEnumConnectionPoints* GetWide() const
                 { return (IEnumConnectionPoints*)m_pObj; };
};


///////////////////////////////////////////////////////////////////////////
/
// CConnectionPointA class

class CConnectionPointA : CAnsiInterface
{
public:
     // IConnectionPointA methods
     STDMETHOD(GetConnectionInterface)(IID FAR* pIID);
     STDMETHOD(GetConnectionPointContainer)(
                       LPCONNECTIONPOINTCONTAINERA FAR* ppCPC);
     STDMETHOD(Advise)(LPUNKNOWN pUnkSink, DWORD FAR* pdwCookie);
     STDMETHOD(Unadvise)(DWORD dwCookie);
     STDMETHOD(EnumConnections)(LPENUMCONNECTIONSA FAR* ppEnum);

     inline CConnectionPointA(LPUNKNOWN pUnk, IConnectionPoint* pWide) :
                 CAnsiInterface(ID_IConnectionPoint, pUnk, (LPUNKNOWN)pWide)
{};

     inline IConnectionPoint* GetWide() const
                 { return (IConnectionPoint*)m_pObj; };

};


///////////////////////////////////////////////////////////////////////////
/
// CEnumConnectionsA class

class CEnumConnectionsA : CAnsiInterface
{
public:
     // IEnumConnectionsA methods
     STDMETHOD(Next)(
                       ULONG cConnections,
                       LPCONNECTDATAA rgcd, ULONG FAR* lpcFetched);
     STDMETHOD(Skip)(ULONG cConnections);
     STDMETHOD(Reset)();
     STDMETHOD(Clone)(LPENUMCONNECTIONSA FAR* ppecn);

     inline CEnumConnectionsA(LPUNKNOWN pUnk, IEnumConnections* pWide) :
                 CAnsiInterface(ID_IEnumConnections, pUnk, (LPUNKNOWN)pWide)
{};

     inline IEnumConnections* GetWide() const
                 { return (IEnumConnections*)m_pObj; };
};


/////////////////////////////////////////////////////////////////////////////
/
// COleControlA class

class COleControlA : CAnsiInterface
{
public:
     // IOleControlA methods
     STDMETHOD(GetControlInfo)(LPCONTROLINFO pCI);
     STDMETHOD(OnMnemonic)(LPMSG pMsg);
     STDMETHOD(OnAmbientPropertyChange)(DISPID dispid);
     STDMETHOD(FreezeEvents)(BOOL bFreeze);

     inline COleControlA(LPUNKNOWN pUnk, IOleControl* pWide) :
                 CAnsiInterface(ID_IOleControl, pUnk, (LPUNKNOWN)pWide) {};

     inline IOleControl* GetWide() const
                 { return (IOleControl*)m_pObj; };
};


/////////////////////////////////////////////////////////////////////////////
/
// COleControlSiteA class

class COleControlSiteA : CAnsiInterface
{
public:
     // IOleControlSiteA methods
     STDMETHOD(OnControlInfoChanged)();
     STDMETHOD(LockInPlaceActive)(BOOL fLock);
     STDMETHOD(GetExtendedControl)(LPDISPATCHA FAR* ppDisp);
     STDMETHOD(TransformCoords)(POINTL FAR* lpptlHimetric,
           POINTF FAR* lpptfContainer, DWORD flags);
     STDMETHOD(TranslateAccelerator)(LPMSG lpMsg, DWORD grfModifiers);
     STDMETHOD(OnFocus)(BOOL fGotFocus);
     STDMETHOD(ShowPropertyFrame)();

     inline COleControlSiteA(LPUNKNOWN pUnk, IOleControlSite* pWide) :
```

```
                CAnsiInterface(ID_IOleControlSite, pUnk, (LPUNKNOWN)pWide)
{};

     inline IOleControlSite* GetWide() const
                { return (IOleControlSite*)m_pObj; };
};


///////////////////////////////////////////////////////////////////////////////
/
// CSimpleFrameSiteA class

class CSimpleFrameSiteA : CAnsiInterface
{
public:
     // ISimpleFrameSite methods
     STDMETHOD(PreMessageFilter)(HWND hwnd, UINT msg, WPARAM wp, LPARAM lp,
           LRESULT FAR* lplResult, DWORD FAR * lpdwCookie);
     STDMETHOD(PostMessageFilter)(HWND hwnd, UINT msg, WPARAM wp, LPARAM lp,
           LRESULT FAR* lplResult, DWORD dwCookie);

     inline CSimpleFrameSiteA(LPUNKNOWN pUnk, ISimpleFrameSite* pWide) :
                CAnsiInterface(ID_ISimpleFrameSite, pUnk, (LPUNKNOWN)pWide)
{};

     inline ISimpleFrameSite* GetWide() const
                { return (ISimpleFrameSite*)m_pObj; };
};


///////////////////////////////////////////////////////////////////////////////
/
// CPersistStreamInitA class

class CPersistStreamInitA : CAnsiInterface
{
public:
     // IPersistA methods
     STDMETHOD(GetClassID)(LPCLSID lpClassID);

     // IPersistStreamInitA methods
     STDMETHOD(IsDirty)();
     STDMETHOD(Load)(LPSTREAMA pStm);
     STDMETHOD(Save)(LPSTREAMA pStm, BOOL fClearDirty);
     STDMETHOD(GetSizeMax)(ULARGE_INTEGER FAR* pcbSize);
     STDMETHOD(InitNew)();

     inline CPersistStreamInitA(LPUNKNOWN pUnk, IPersistStreamInit* pWide) :
                CAnsiInterface(ID_IPersistStreamInit, pUnk,
(LPUNKNOWN)pWide) {};

     inline IPersistStreamInit* GetWide() const
                { return (IPersistStreamInit*)m_pObj; };
};
```

```
/////////////////////////////////////////////////////////////////////////////
/
// CClassFactory2A class

class CClassFactory2A : CAnsiInterface
{
public:
     //  IClassFactoryA methods
     STDMETHOD(CreateInstance) (LPUNKNOWN pUnkOuter,
                         REFIID riid, LPVOID FAR* ppvObject);
     STDMETHOD(LockServer) (BOOL fLock);

     //  IClassFactory2A methods
     STDMETHOD(GetLicInfo)(LPLICINFO pLicInfo);
     STDMETHOD(RequestLicKey)(DWORD dwReserved, BSTRA FAR* pbstrKey);
     STDMETHOD(CreateInstanceLic)(LPUNKNOWN pUnkOuter,
                 LPUNKNOWN pUnkReserved, REFIID riid, BSTRA bstrKey,
                 LPVOID FAR* ppvObject);

     inline CClassFactory2A(LPUNKNOWN pUnk, IClassFactory2* pWide) :
                 CAnsiInterface(ID_IClassFactory2, pUnk, (LPUNKNOWN)pWide)
{};

     inline IClassFactory2* GetWide() const
                 { return (IClassFactory2*)m_pObj; };
};


/////////////////////////////////////////////////////////////////////////////
/
// CSpecifyPropertyPagesA class

class CSpecifyPropertyPagesA : CAnsiInterface
{
public:
     // ISpecifyPropertyPagesA interface
     STDMETHOD(GetPages)(CAUUID FAR* pPages);

     inline CSpecifyPropertyPagesA(LPUNKNOWN pUnk, ISpecifyPropertyPages*
pWide) :
                 CAnsiInterface(ID_ISpecifyPropertyPages, pUnk,
(LPUNKNOWN)pWide) {};

     inline ISpecifyPropertyPages* GetWide() const
                 { return (ISpecifyPropertyPages*)m_pObj; };
};


/////////////////////////////////////////////////////////////////////////////
/
// CPerPropertyBrowsingA class

class CPerPropertyBrowsingA : CAnsiInterface
```

```
{
public:
     // IPerPropertyBrowsingA interface
     STDMETHOD(GetDisplayString)(DISPID dispid, BSTRA FAR* lpbstr);
     STDMETHOD(MapPropertyToPage)(DISPID dispid, LPCLSID lpclsid);
     STDMETHOD(GetPredefinedStrings)(DISPID dispid,
           CALPSTR FAR* lpcaStringsOut, CADWORD FAR* lpcaCookiesOut);
     STDMETHOD(GetPredefinedValue)(DISPID dispid, DWORD dwCookie,
           VARIANTA FAR* pvarOut);

     inline CPerPropertyBrowsingA(LPUNKNOWN pUnk, IPerPropertyBrowsing*
pWide) :
                 CAnsiInterface(ID_IPerPropertyBrowsing, pUnk,
(LPUNKNOWN)pWide) {};

     inline IPerPropertyBrowsing* GetWide() const
                 { return (IPerPropertyBrowsing*)m_pObj; };
};


/////////////////////////////////////////////////////////////////////////////
/
// CPropertyPageSiteA class

class CPropertyPageSiteA : CAnsiInterface
{
public:
     // IPropertyPageSiteA methods
     STDMETHOD(OnStatusChange)(DWORD flags);
     STDMETHOD(GetLocaleID)(LCID FAR* pLocaleID);
     STDMETHOD(GetPageContainer)(LPUNKNOWN FAR* ppUnk);
     STDMETHOD(TranslateAccelerator)(LPMSG lpMsg);

     inline CPropertyPageSiteA(LPUNKNOWN pUnk, IPropertyPageSite* pWide) :
                 CAnsiInterface(ID_IPropertyPageSite, pUnk,
(LPUNKNOWN)pWide) {};

     inline IPropertyPageSite* GetWide() const
                 { return (IPropertyPageSite*)m_pObj; };
};


/////////////////////////////////////////////////////////////////////////////
/
// CPropertyPage2A class

class CPropertyPage2A : CAnsiInterface
{
public:
     // IPropertyPageA methods
     STDMETHOD(SetPageSite)(LPPROPERTYPAGESITEA pPageSite);
     STDMETHOD(Activate)(HWND hwndParent, LPCRECT rect,
                       BOOL bModal);
     STDMETHOD(Deactivate)();
     STDMETHOD(GetPageInfo)(LPPROPPAGEINFOA pPageInfo);
     STDMETHOD(SetObjects)(ULONG cObjects, LPUNKNOWN FAR* ppunk);
     STDMETHOD(Show)(UINT nCmdShow);
     STDMETHOD(Move)(LPCRECT prect);
     STDMETHOD(IsPageDirty)();
     STDMETHOD(Apply)();
     STDMETHOD(Help)(LPCSTR lpszHelpDir);
     STDMETHOD(TranslateAccelerator)(LPMSG lpMsg);

     // IPropertyPage2A methods
     STDMETHOD(EditProperty)(DISPID dispid);

     inline CPropertyPage2A(LPUNKNOWN pUnk, IPropertyPage2* pWide) :
                 CAnsiInterface(ID_IPropertyPage2, pUnk, (LPUNKNOWN)pWide)
{};

     inline IPropertyPage2* GetWide() const
                 { return (IPropertyPage2*)m_pObj; };
};


////////////////////////////////////////////////////////////////////////////
/
// CFontA class

class CFontA : CAnsiInterface
{
public:
     // IFontA methods
     STDMETHOD(get_Name)(BSTRA FAR* pname);
     STDMETHOD(put_Name)(BSTRA name);
     STDMETHOD(get_Size)(CY FAR* psize);
     STDMETHOD(put_Size)(CY size);
     STDMETHOD(get_Bold)(BOOL FAR* pbold);
     STDMETHOD(put_Bold)(BOOL bold);
     STDMETHOD(get_Italic)(BOOL FAR* pitalic);
     STDMETHOD(put_Italic)(BOOL italic);
     STDMETHOD(get_Underline)(BOOL FAR* punderline);
     STDMETHOD(put_Underline)(BOOL underline);
     STDMETHOD(get_Strikethrough)(BOOL FAR* pstrikethrough);
     STDMETHOD(put_Strikethrough)(BOOL strikethrough);
     STDMETHOD(get_Weight)(short FAR* pweight);
     STDMETHOD(put_Weight)(short weight);
     STDMETHOD(get_Charset)(short FAR* pcharset);
     STDMETHOD(put_Charset)(short charset);
     STDMETHOD(get_hFont)(HFONT FAR* phfont);
     STDMETHOD(Clone)(IFontA FAR* FAR* lplpfont);
     STDMETHOD(IsEqual)(IFontA FAR * lpFontOther);
     STDMETHOD(SetRatio)(long cyLogical, long cyHimetric);
     STDMETHOD(QueryTextMetrics)(LPTEXTMETRIC lptm);
     STDMETHOD(AddRefHfont)(HFONT hfont);
     STDMETHOD(ReleaseHfont)(HFONT hfont);

     inline CFontA(LPUNKNOWN pUnk, IFont* pWide) :
                 CAnsiInterface(ID_IFont, pUnk, (LPUNKNOWN)pWide) {};
     inline IFont* GetWide() const
                 { return (IFont*)m_pObj; };
};


////////////////////////////////////////////////////////////////////////////
/
// CPictureA class

class CPictureA : CAnsiInterface
{
public:
     // IPictureA methods
     STDMETHOD(get_Handle)(OLE_HANDLE FAR* phandle);
     STDMETHOD(get_hPal)(OLE_HANDLE FAR* phpal);
     STDMETHOD(get_Type)(short FAR* ptype);
     STDMETHOD(get_Width)(OLE_XSIZE_HIMETRIC FAR* pwidth);
     STDMETHOD(get_Height)(OLE_YSIZE_HIMETRIC FAR* pheight);
     STDMETHOD(Render)(HDC hdc, long x, long y, long cx, long cy,
           OLE_XPOS_HIMETRIC xSrc, OLE_YPOS_HIMETRIC ySrc,
           OLE_XSIZE_HIMETRIC cxSrc, OLE_YSIZE_HIMETRIC cySrc,
           LPRECT lprcWBounds);
     STDMETHOD(set_hPal)(OLE_HANDLE hpal);
     STDMETHOD(get_CurDC)(HDC FAR * phdcOut);
     STDMETHOD(SelectPicture)(
           HDC hdcIn, HDC FAR * phdcOut, OLE_HANDLE FAR * phbmpOut);
     STDMETHOD(get_KeepOriginalFormat)(BOOL * pfkeep);
     STDMETHOD(put_KeepOriginalFormat)(BOOL fkeep);
     STDMETHOD(PictureChanged)();
     STDMETHOD(SaveAsFile)(LPSTREAMA lpstream, BOOL fSaveMemCopy,
           LONG FAR * lpcbSize);
     STDMETHOD(get_Attributes)(DWORD FAR * lpdwAttr);

     inline CPictureA(LPUNKNOWN pUnk, IPicture* pWide) :
                 CAnsiInterface(ID_IPicture, pUnk, (LPUNKNOWN)pWide) {};

     inline IPicture* GetWide() const
                 { return (IPicture*)m_pObj; };
};
```

ANSICTL.CPP (OLEANSI Sample)

```
//
+---------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       AnsiCtl.cpp
//
//  Contents:   ANSI Wrappers for OLE Control Interfaces and APIs.
//
//  History:    01-Jun-94   johnels     Created.
//
//---------------------------------------------------------------------------
-

#include "Ole2Ansi.h"


////////////////////////////////////////////////////////////////////////////
/
// Get an entry point for the OLE Controls DLL

HINSTANCE _hOleControlDll = (HINSTANCE)-1;

#ifdef _DEBUG
#define OC_DLL "oc30d.dll"
#else
#define OC_DLL "oc30.dll"
#endif

HRESULT GetEntryPoint(FARPROC* lplpfn, LPSTR pszFunction)
{
     if (*lplpfn == NULL)
     {
           if (_hOleControlDll == (HINSTANCE)-1)
                 _hOleControlDll = LoadLibrary(OC_DLL);

           if (_hOleControlDll != NULL)
                 *lplpfn = GetProcAddress(_hOleControlDll, pszFunction);

#ifdef _DEBUG
           if (*lplpfn == NULL)
           {
                 char buf[MAX_STRING];
                 wsprintf(buf, "%s: Entry point not found!", pszFunction);
           }
#endif
     }

     return (*lplpfn != NULL) ? NOERROR : ResultFromScode(E_UNEXPECTED);
}
```

```
///////////////////////////////////////////////////////////////////////////
/
// Property sheet APIs

STDAPI OleCreatePropertyFrameA(HWND hwndOwner, UINT x, UINT y,
     LPCSTR lpszCaptionA, ULONG cObjects, LPUNKNOWN FAR* ppUnkA,
     ULONG cPages, LPCLSID pPageClsID, LCID lcid,
     DWORD dwReserved, LPVOID pvReserved)
{
     TraceSTDAPIEnter("OleCreatePropertyFrameA");

     OCPFIPARAMSA ocpfiA;
     ocpfiA.cbStructSize = sizeof(OCPFIPARAMSA);
     ocpfiA.hWndOwner = hwndOwner;
     ocpfiA.x = x;
     ocpfiA.y = y;
     ocpfiA.lpszCaption = (LPSTR)lpszCaptionA;
     ocpfiA.cObjects = cObjects;
     ocpfiA.lplpUnk = ppUnkA;
     ocpfiA.cPages = cPages;
     ocpfiA.lpPages = pPageClsID;
     ocpfiA.lcid = lcid;
     ocpfiA.dispidInitialProperty = DISPID_UNKNOWN;

     return OleCreatePropertyFrameIndirectA(&ocpfiA);
}

STDAPI OleCreatePropertyFrameIndirectA(LPOCPFIPARAMSA lpParams)
{
     static HRESULT (FAR STDAPICALLTYPE *
_lpfnOleCreatePropertyFrameIndirect)(
           LPOCPFIPARAMS) = NULL;

     TraceSTDAPIEnter("OleCreatePropertyFrameIndirectA");

     HRESULT hResult;

     WCHAR szCaption[MAX_STRING];
     if (lpParams->lpszCaption != NULL)
           ConvertStringToW(lpParams->lpszCaption, szCaption);
     else
           szCaption[0] = L'\0';

     ULONG cObjects = lpParams->cObjects;
     LPUNKNOWN FAR* ppUnk = new LPUNKNOWN[cObjects];

     ULONG i;
     for (i = 0; i < cObjects; i++)
           ppUnk[i] = NULL;

     for (i = 0; i < cObjects; i++)
     {
           hResult = WrapIUnknownWFromA(lpParams->lplpUnk[i], &ppUnk[i]);
           if (FAILED(hResult))
                 goto Error;
     }

     OCPFIPARAMS ocpfi;
     memcpy(&ocpfi, lpParams, lpParams->cbStructSize);
     ocpfi.lpszCaption = szCaption;
     ocpfi.lplpUnk = ppUnk;

     hResult = GetEntryPoint((FARPROC*)&_lpfnOleCreatePropertyFrameIndirect,
           "OleCreatePropertyFrameIndirect");

     if (FAILED(hResult))
           goto Error;

     hResult = (*_lpfnOleCreatePropertyFrameIndirect)(&ocpfi);

Error:
     for (i = 0; i < cObjects; i++)
           if (ppUnk[i] != NULL)
                 ppUnk[i]->Release();
     delete [] ppUnk;

     return hResult;
}


////////////////////////////////////////////////////////////////////////////
/
// Font APIs

STDAPI OleCreateFontIndirectA(LPFONTDESCA lpfd,
           REFIID riid, LPVOID FAR* lplpvObj)
{
     static HRESULT (FAR STDAPICALLTYPE * _lpfnOleCreateFontIndirect)(
                 LPFONTDESC, REFIID, LPVOID FAR*) = NULL;

     TraceSTDAPIEnter("OleCreateFontIndirectA");

     HRESULT hResult;
     LPUNKNOWN pUnk;
     IDINTERFACE idRef;

     WCHAR szName[MAX_STRING];
     if (lpfd->lpstrName != NULL)
           ConvertStringToW(lpfd->lpstrName, szName);
     else
           szName[0] = L'\0';

     FONTDESC fontDesc;
     memcpy(&fontDesc, lpfd, lpfd->cbSizeofstruct);
     fontDesc.lpstrName = szName;

     hResult = GetEntryPoint((FARPROC*)&_lpfnOleCreateFontIndirect,
           "OleCreateFontIndirect");
     if (FAILED(hResult))
           return hResult;

     hResult = (*_lpfnOleCreateFontIndirect)(&fontDesc, riid,
           (LPVOID FAR*)&pUnk);

     if (FAILED(hResult))
           return hResult;

     //
     //  Convert the 16 bytes GUID into an internal integer for speed.
     //
     idRef = WrapTranslateIID(riid);
     hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)lplpvObj);
     pUnk->Release();

     return hResult;
}


////////////////////////////////////////////////////////////////////////////
/
// CPropertyNotifySinkA class

STDMETHODIMP CPropertyNotifySinkA::OnChanged(DISPID dispid)
{
     TraceMethodEnter("CPropertyNotifySinkA::OnChanged", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPropertyNotifySink, OnChanged));
     return GetWide()->OnChanged(dispid);
}

STDMETHODIMP CPropertyNotifySinkA::OnRequestEdit(DISPID dispid)
{
     TraceMethodEnter("CPropertyNotifySinkA::OnRequestEdit", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPropertyNotifySink, OnRequestEdit));
     return GetWide()->OnRequestEdit(dispid);
}


////////////////////////////////////////////////////////////////////////////
/
// CProvideClassInfoA class

STDMETHODIMP CProvideClassInfoA::GetClassInfo(LPTYPEINFOA FAR* ppTI)
{
     TraceMethodEnter("CProvideClassInfoA::GetClassInfo", this);
     _DebugHook(GetWide(), MEMBER_PTR(IProvideClassInfo, GetClassInfo));

     LPTYPEINFO pTypeInfo;
     HRESULT hResult;

     hResult = GetWide()->GetClassInfo(&pTypeInfo);
     if (FAILED(hResult))
           return hResult;
     hResult = WrapITypeInfoAFromW(pTypeInfo, ppTI);
     pTypeInfo->Release();
     return hResult;
}

////////////////////////////////////////////////////////////////////////////
/
// CConnectionPointContainerA class

STDMETHODIMP CConnectionPointContainerA::EnumConnectionPoints(
                 LPENUMCONNECTIONPOINTSA FAR* ppEnum)
{
     TraceMethodEnter("CConnectionPointContainerA::EnumConnectionPoints",
this);
     _DebugHook(GetWide(), MEMBER_PTR(IConnectionPointContainer,
EnumConnectionPoints));

     LPENUMCONNECTIONPOINTS pEnumConnPts;
     HRESULT hResult;

     hResult = GetWide()->EnumConnectionPoints(&pEnumConnPts);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIEnumConnectionPointsAFromW(pEnumConnPts, ppEnum);
     pEnumConnPts->Release();
     return hResult;
}

STDMETHODIMP CConnectionPointContainerA::FindConnectionPoint(
                 REFIID iid, LPCONNECTIONPOINTA FAR* ppCP)
{
     TraceMethodEnter("CConnectionPointContainerA::FindConnectionPoint",
this);
     _DebugHook(GetWide(), MEMBER_PTR(IConnectionPointContainer,
FindConnectionPoint));

     LPCONNECTIONPOINT pConnPt;
     HRESULT hResult;

     hResult = GetWide()->FindConnectionPoint(iid, &pConnPt);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIConnectionPointAFromW(pConnPt, ppCP);
     pConnPt->Release();
     return hResult;
}


////////////////////////////////////////////////////////////////////////////
/
// CEnumConnectionPointsA class
```

```
STDMETHODIMP CEnumConnectionPointsA::Next(
                 ULONG cConnections,
                 LPCONNECTIONPOINTA FAR* rgpcn, ULONG FAR* lpcFetched)
{
     TraceMethodEnter("CEnumConnectionPointsA::Next", this);
     _DebugHook(GetWide(), MEMBER_PTR(IEnumConnectionPoints, Next));

     LPCONNECTIONPOINT FAR* rgpConnPt = new LPCONNECTIONPOINT[cConnections];
     HRESULT hResult;

     hResult = GetWide()->Next(cConnections, rgpConnPt, lpcFetched);
     if (FAILED(hResult))
     {
           delete [] rgpConnPt;
           return hResult;
     }

     ULONG i;
     for (i = 0; i < *lpcFetched; i++)   // Wrap each pointer in array
     {
           hResult = WrapIConnectionPointAFromW(rgpConnPt[i], &rgpcn[i]);
           rgpConnPt[i]->Release();
     }

     delete [] rgpConnPt;
     return hResult;
}

STDMETHODIMP CEnumConnectionPointsA::Skip(ULONG cConnections)
{
     TraceMethodEnter("CEnumConnectionPointsA::Skip", this);
     _DebugHook(GetWide(), MEMBER_PTR(IEnumConnectionPoints, Skip));
     return GetWide()->Skip(cConnections);
}

STDMETHODIMP CEnumConnectionPointsA::Reset()
{
     TraceMethodEnter("CEnumConnectionPointsA::Reset", this);
     _DebugHook(GetWide(), MEMBER_PTR(IEnumConnectionPoints, Reset));
     return GetWide()->Reset();
}

STDMETHODIMP CEnumConnectionPointsA::Clone(LPENUMCONNECTIONPOINTSA FAR*
ppEnum)
{
     TraceMethodEnter("CEnumConnectionPointsA::Clone", this);
     _DebugHook(GetWide(), MEMBER_PTR(IEnumConnectionPoints, Clone));

     LPENUMCONNECTIONPOINTS pEnumConnPts;
     HRESULT hResult;

     hResult = GetWide()->Clone(&pEnumConnPts);
     if (FAILED(hResult))
           return hResult;
     hResult = WrapIEnumConnectionPointsAFromW(pEnumConnPts, ppEnum);
     pEnumConnPts->Release();
     return hResult;
}


///////////////////////////////////////////////////////////////////////////
/
// CConnectionPointA class

STDMETHODIMP CConnectionPointA::GetConnectionInterface(IID FAR* pIID)
{
     TraceMethodEnter("CConnectionPointA::GetConnectionInterface", this);
     _DebugHook(GetWide(), MEMBER_PTR(IConnectionPoint,
GetConnectionInterface));
     return GetWide()->GetConnectionInterface(pIID);
}

STDMETHODIMP CConnectionPointA::GetConnectionPointContainer(
                 LPCONNECTIONPOINTCONTAINERA FAR* ppCPC)
{
     TraceMethodEnter("CConnectionPointA::GetConnectionPointContainer",
this);
     _DebugHook(GetWide(), MEMBER_PTR(IConnectionPoint,
GetConnectionPointContainer));

     LPCONNECTIONPOINTCONTAINER pConnPtCont;
     HRESULT hResult;

     hResult = GetWide()->GetConnectionPointContainer(&pConnPtCont);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIConnectionPointContainerAFromW(pConnPtCont, ppCPC);
     pConnPtCont->Release();
     return hResult;
}

STDMETHODIMP CConnectionPointA::Advise(LPUNKNOWN pUnkSinkA, DWORD FAR*
pdwCookie)
{
     TraceMethodEnter("CConnectionPointA::Advise", this);
     _DebugHook(GetWide(), MEMBER_PTR(IConnectionPoint, Advise));

     LPUNKNOWN pUnkSink;
     HRESULT hResult;

     hResult = WrapIUnknownWFromA(pUnkSinkA, &pUnkSink);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->Advise(pUnkSink, pdwCookie);
     pUnkSink->Release();
     return hResult;
}
```

```
STDMETHODIMP CConnectionPointA::Unadvise(DWORD dwCookie)
{
     TraceMethodEnter("CConnectionPointA::Unadvise", this);
     _DebugHook(GetWide(), MEMBER_PTR(IConnectionPoint, Unadvise));
     return GetWide()->Unadvise(dwCookie);
}

STDMETHODIMP CConnectionPointA::EnumConnections(LPENUMCONNECTIONSA FAR*
ppEnum)
{

     TraceMethodEnter("CConnectionPointA::EnumConnections", this);
     _DebugHook(GetWide(), MEMBER_PTR(IConnectionPoint, EnumConnections));

     LPENUMCONNECTIONS pEnumConns;
     HRESULT hResult;

     hResult = GetWide()->EnumConnections(&pEnumConns);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIEnumConnectionsAFromW(pEnumConns, ppEnum);
     pEnumConns->Release();
     return hResult;
}


/////////////////////////////////////////////////////////////////////////////
/
// CEnumConnectionsA class

STDMETHODIMP CEnumConnectionsA::Next(
                 ULONG cConnections,
                 LPCONNECTDATAA rgcd, ULONG FAR* lpcFetched)
{
     TraceMethodEnter("CEnumConnectionsA::Next", this);
     _DebugHook(GetWide(), MEMBER_PTR(IEnumConnections, Next));

     LPCONNECTDATA rgConnData = new CONNECTDATA[cConnections];
     HRESULT hResult;

     hResult = GetWide()->Next(cConnections, rgConnData, lpcFetched);
     if (FAILED(hResult))
     {
           delete [] rgConnData;
           return hResult;
     }

     ULONG i;
     for (i = 0; i < *lpcFetched; i++)   // Wrap each pointer in array
     {
           rgcd[i].dwCookie = rgConnData[i].dwCookie;
           hResult = WrapIUnknownAFromW(rgConnData[i].pUnk, &rgcd[i].pUnk);
           rgConnData[i].pUnk->Release();
     }

     delete [] rgConnData;
     return hResult;
}

STDMETHODIMP CEnumConnectionsA::Skip(ULONG cConnections)
{
     TraceMethodEnter("CEnumConnectionPointsA::Skip", this);
     _DebugHook(GetWide(), MEMBER_PTR(IEnumConnectionPoints, Skip));
     return GetWide()->Skip(cConnections);
}

STDMETHODIMP CEnumConnectionsA::Reset()
{
     TraceMethodEnter("CEnumConnectionPointsA::Reset", this);
     _DebugHook(GetWide(), MEMBER_PTR(IEnumConnectionPoints, Reset));
     return GetWide()->Reset();
}

STDMETHODIMP CEnumConnectionsA::Clone(LPENUMCONNECTIONSA FAR* ppEnum)
{
     TraceMethodEnter("CEnumConnectionsA::Clone", this);
     _DebugHook(GetWide(), MEMBER_PTR(IEnumConnections, Clone));

     LPENUMCONNECTIONS pEnumConns;
     HRESULT hResult;

     hResult = GetWide()->Clone(&pEnumConns);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIEnumConnectionsAFromW(pEnumConns, ppEnum);
     pEnumConns->Release();
     return hResult;
}

///////////////////////////////////////////////////////////////////////////
/
// COleControlA class

STDMETHODIMP COleControlA::GetControlInfo(LPCONTROLINFO pCI)
{
     TraceMethodEnter("COleControlA::GetControlInfo", this);
     _DebugHook(GetWide(), MEMBER_PTR(IOleControl, GetControlInfo));
     return GetWide()->GetControlInfo(pCI);
}

STDMETHODIMP COleControlA::OnMnemonic(LPMSG pMsg)
{
     TraceMethodEnter("COleControlA::OnMnemonic", this);
     _DebugHook(GetWide(), MEMBER_PTR(IOleControl, OnMnemonic));
     return GetWide()->OnMnemonic(pMsg);
}
```

```
STDMETHODIMP COleControlA::OnAmbientPropertyChange(DISPID dispid)
{
     TraceMethodEnter("COleControlA::OnAmbientPropertyChange", this);
     _DebugHook(GetWide(), MEMBER_PTR(IOleControl,
OnAmbientPropertyChange));
     return GetWide()->OnAmbientPropertyChange(dispid);
}

STDMETHODIMP COleControlA::FreezeEvents(BOOL bFreeze)
{
     TraceMethodEnter("COleControlA::FreezeEvents", this);
     _DebugHook(GetWide(), MEMBER_PTR(IOleControl, FreezeEvents));
     return GetWide()->FreezeEvents(bFreeze);
}

/////////////////////////////////////////////////////////////////////////////
/
// COleControlSiteA class

STDMETHODIMP COleControlSiteA::OnControlInfoChanged()
{
     TraceMethodEnter("COleControlSiteA::OnControlInfoChanged", this);
     _DebugHook(GetWide(), MEMBER_PTR(IOleControlSite,
OnControlInfoChanged));
     return GetWide()->OnControlInfoChanged();
}

STDMETHODIMP COleControlSiteA::LockInPlaceActive(BOOL fLock)

{
     TraceMethodEnter("COleControlSiteA::LockInPlaceActive", this);
     _DebugHook(GetWide(), MEMBER_PTR(IOleControlSite, LockInPlaceActive));
     return GetWide()->LockInPlaceActive(fLock);
}

STDMETHODIMP COleControlSiteA::GetExtendedControl(LPDISPATCHA FAR* ppDisp)
{
     TraceMethodEnter("COleControlSiteA::GetExtendedControl", this);
     _DebugHook(GetWide(), MEMBER_PTR(IOleControlSite, GetExtendedControl));

     LPDISPATCH pDisp;
     HRESULT hResult;

     hResult = GetWide()->GetExtendedControl(&pDisp);
     if (FAILED(hResult))
          return hResult;

     hResult = WrapIDispatchAFromW(pDisp, ppDisp);
     pDisp->Release();
     return hResult;
}

STDMETHODIMP COleControlSiteA::TransformCoords(POINTL FAR* lpptlHimetric,
     POINTF FAR* lpptfContainer, DWORD flags)
{
     TraceMethodEnter("COleControlSiteA::TransformCoords", this);
     _DebugHook(GetWide(), MEMBER_PTR(IOleControlSite, TransformCoords));
     return GetWide()->TransformCoords(lpptlHimetric, lpptfContainer,
flags);
}

STDMETHODIMP COleControlSiteA::TranslateAccelerator(LPMSG lpMsg,
     DWORD grfModifiers)
{
     TraceMethodEnter("COleControlSiteA::TranslateAccelerator", this);
     _DebugHook(GetWide(), MEMBER_PTR(IOleControlSite,
TranslateAccelerator));
     return GetWide()->TranslateAccelerator(lpMsg, grfModifiers);
}

STDMETHODIMP COleControlSiteA::OnFocus(BOOL fGotFocus)
{
     TraceMethodEnter("COleControlSiteA::OnFocus", this);
     _DebugHook(GetWide(), MEMBER_PTR(IOleControlSite, OnFocus));
     return GetWide()->OnFocus(fGotFocus);
}

STDMETHODIMP COleControlSiteA::ShowPropertyFrame()
{
     TraceMethodEnter("COleControlSiteA::ShowPropertyFrame", this);
     _DebugHook(GetWide(), MEMBER_PTR(IOleControlSite, ShowPropertyFrame));
     return GetWide()->ShowPropertyFrame();
}


////////////////////////////////////////////////////////////////////////////
/
// CSimpleFrameSiteA class

STDMETHODIMP CSimpleFrameSiteA::PreMessageFilter(HWND hwnd, UINT msg,
     WPARAM wp, LPARAM lp, LRESULT FAR* lplResult, DWORD FAR * lpdwCookie)
{
     TraceMethodEnter("CSimpleFrameSiteA::PreMessageFilter", this);
     _DebugHook(GetWide(), MEMBER_PTR(ISimpleFrameSite, PreMessageFilter));
     return GetWide()->PreMessageFilter(hwnd, msg, wp, lp, lplResult,
lpdwCookie);
}

STDMETHODIMP CSimpleFrameSiteA::PostMessageFilter(HWND hwnd, UINT msg,
     WPARAM wp, LPARAM lp, LRESULT FAR* lplResult, DWORD dwCookie)
{
     TraceMethodEnter("CSimpleFrameSiteA::PostMessageFilter", this);
     _DebugHook(GetWide(), MEMBER_PTR(ISimpleFrameSite, PostMessageFilter));
     return GetWide()->PostMessageFilter(hwnd, msg, wp, lp, lplResult,
dwCookie);
}


////////////////////////////////////////////////////////////////////////////
/
```

```
// CPersistStreamInitA class

STDMETHODIMP CPersistStreamInitA::GetClassID(LPCLSID lpClassID)
{
     TraceMethodEnter("CPersistStreamInitA::GetClassID", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPersistStreamInit, GetClassID));
     return GetWide()->GetClassID(lpClassID);
}

STDMETHODIMP CPersistStreamInitA::IsDirty()
{
     TraceMethodEnter("CPersistStreamInitA::IsDirty", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPersistStreamInit, IsDirty));
     return GetWide()->IsDirty();
}

STDMETHODIMP CPersistStreamInitA::Load(LPSTREAMA pStmA)
{
     TraceMethodEnter("CPersistStreamInitA::Load", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPersistStreamInit, Load));

     LPSTREAM pStm;
     HRESULT hResult;

     hResult = WrapIStreamWFromA(pStmA, &pStm);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->Load(pStm);
     pStm->Release();
     return hResult;
}

STDMETHODIMP CPersistStreamInitA::Save(LPSTREAMA pStmA, BOOL fClearDirty)
{
     TraceMethodEnter("CPersistStreamInitA::Save", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPersistStreamInit, Save));


     LPSTREAM pStm;
     HRESULT hResult;

     hResult = WrapIStreamWFromA(pStmA, &pStm);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->Save(pStm, fClearDirty);
     pStm->Release();
     return hResult;
}

STDMETHODIMP CPersistStreamInitA::GetSizeMax(ULARGE_INTEGER FAR* pcbSize)
{
     TraceMethodEnter("CPersistStreamInitA::GetSizeMax", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPersistStreamInit, GetSizeMax));
     return GetWide()->GetSizeMax(pcbSize);
}

STDMETHODIMP CPersistStreamInitA::InitNew()
{
     TraceMethodEnter("CPersistStreamInitA::InitNew", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPersistStreamInit, InitNew));
     return GetWide()->InitNew();
}

///////////////////////////////////////////////////////////////////////////////
/
// CClassFactory2A class

STDMETHODIMP CClassFactory2A::CreateInstance(LPUNKNOWN pUnkOuterA,
             REFIID riid, LPVOID FAR* ppvObject)
{
     TraceMethodEnter("CClassFactory2A::CreateInstance", this);
     _DebugHook(GetWide(), MEMBER_PTR(IClassFactory2, CreateInstance));

     LPUNKNOWN pUnkOuter;
     LPUNKNOWN pUnk;
     IDINTERFACE idRef;
     HRESULT hResult;

     hResult = WrapIUnknownWFromA(pUnkOuterA, &pUnkOuter);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->CreateInstance(pUnkOuter, riid, (LPVOID *)&pUnk);
     if (FAILED(hResult))
           goto Error;

     //
     //  Convert the 16 bytes GUID into an internal integer for speed.
     //
     idRef = WrapTranslateIID(riid);
     hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)ppvObject);
     pUnk->Release();

Error:
     if (pUnkOuter)
           pUnkOuter->Release();

     return hResult;
}


STDMETHODIMP CClassFactory2A::LockServer(BOOL fLock)
{
     TraceNotify("CClassFactory2A::LockServer");
     _DebugHook(GetWide(), MEMBER_PTR(IClassFactory2, LockServer));
     return GetWide()->LockServer(fLock);
}
```

```
STDMETHODIMP CClassFactory2A::GetLicInfo(LPLICINFO pLicInfo)
{
     TraceMethodEnter("COleClassFactory2A::GetLicInfo", this);
     _DebugHook(GetWide(), MEMBER_PTR(IClassFactory2, GetLicInfo));
     return GetWide()->GetLicInfo(pLicInfo);
}

STDMETHODIMP CClassFactory2A::RequestLicKey(DWORD dwReserved, BSTRA FAR*
pbstrKeyA)
{
     TraceMethodEnter("COleClassFactory2A::RequestLicKey", this);
     _DebugHook(GetWide(), MEMBER_PTR(IClassFactory2, RequestLicKey));

     HRESULT hResult;
     BSTR bstrKey, *pbstrKey;
     pbstrKey = (pbstrKeyA ? &bstrKey : NULL);

     hResult = GetWide()->RequestLicKey(dwReserved, pbstrKey);
     if (FAILED(hResult))
           return hResult;

     if (pbstrKey)
     {
           hResult = ConvertDispStringToA(bstrKey, pbstrKeyA);
           SysFreeString(bstrKey);
     }

     return hResult;
}

STDMETHODIMP CClassFactory2A::CreateInstanceLic(LPUNKNOWN pUnkOuterA,
           LPUNKNOWN pUnkReservedA, REFIID riid, BSTRA bstrKeyA,
           LPVOID FAR* ppvObject)
{
     TraceMethodEnter("COleClassFactory2A::CreateInstanceLic", this);
     _DebugHook(GetWide(), MEMBER_PTR(IClassFactory2, CreateInstanceLic));

     LPUNKNOWN pUnkOuter;
     LPUNKNOWN pUnkReserved;
     LPUNKNOWN pUnk;
     IDINTERFACE idRef;
     BSTR bstrKey;
     HRESULT hResult;

     hResult = WrapIUnknownWFromA(pUnkOuterA, &pUnkOuter);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIUnknownWFromA(pUnkReservedA, &pUnkReserved);
     if (FAILED(hResult))
           return hResult;

     hResult = ConvertDispStringToW(bstrKeyA, &bstrKey);
     if (FAILED(hResult))
           goto Error;
     hResult = GetWide()->CreateInstanceLic(pUnkOuter, pUnkReserved, riid,
                 bstrKey, (LPVOID *)&pUnk);

     if (FAILED(hResult))
           goto Error;

     //
     //  Convert the 16 bytes GUID into an internal integer for speed.
     //
     idRef = WrapTranslateIID(riid);
     hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)ppvObject);
     pUnk->Release();

Error:
     if (bstrKey)
           ConvertDispStringFreeW(bstrKey);

     if (pUnkReserved)
           pUnkReserved->Release();

     if (pUnkOuter)
           pUnkOuter->Release();

     return hResult;
}

////////////////////////////////////////////////////////////////////////////
/
// CSpecifyPropertyPagesA class

STDMETHODIMP CSpecifyPropertyPagesA::GetPages(CAUUID FAR* pPages)
{
     TraceMethodEnter("CSpecifyPropertyPagesA::GetPages", this);
     _DebugHook(GetWide(), MEMBER_PTR(ISpecifyPropertyPages, GetPages));
     return GetWide()->GetPages(pPages);
}

////////////////////////////////////////////////////////////////////////////
/
// CPerPropertyBrowsingA class

STDMETHODIMP CPerPropertyBrowsingA::GetDisplayString(DISPID dispid,
     BSTRA FAR* pbstrA)
{
     TraceMethodEnter("CPerPropertyBrowsingA::GetDisplayString", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPerPropertyBrowsing,
GetDisplayString));

     HRESULT hResult;
     BSTR bstr, *pbstr;
     pbstr = (pbstrA ? &bstr : NULL);

     hResult = GetWide()->GetDisplayString(dispid, pbstr);
     if (hResult != NOERROR)
     {
           if (pbstrA)
                 *pbstrA = NULL;
           return hResult;
     }

     HRESULT hResultConvert = NOERROR;

     if (pbstr)
     {
           hResultConvert = ConvertDispStringToA(bstr, pbstrA);
           SysFreeString(bstr);
     }

     return SUCCEEDED(hResultConvert) ? hResult : hResultConvert;
}

STDMETHODIMP CPerPropertyBrowsingA::MapPropertyToPage(DISPID dispid, LPCLSID
lpclsid)
{
     TraceMethodEnter("CPerPropertyBrowsingA::MapPropertyToPage", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPerPropertyBrowsing,
MapPropertyToPage));
     return GetWide()->MapPropertyToPage(dispid, lpclsid);
}

STDMETHODIMP CPerPropertyBrowsingA::GetPredefinedStrings(DISPID dispid,
     CALPSTR FAR* lpcaStringsOut, CADWORD FAR* lpcaCookiesOut)
{
     TraceMethodEnter("CPerPropertyBrowsingA::GetPredefinedStrings", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPerPropertyBrowsing,
GetPredefinedStrings));

     HRESULT hResult;
     CALPOLESTR caStringsOut;

     hResult = GetWide()->GetPredefinedStrings(dispid, &caStringsOut,
lpcaCookiesOut);
     if (hResult != NOERROR)
           return hResult;

     // Convert string array
     lpcaStringsOut->cElems = caStringsOut.cElems;
     hResult = ConvertStringArrayToA(caStringsOut.pElems, &lpcaStringsOut-
>pElems,
           caStringsOut.cElems);

     if (FAILED(hResult))
           delete lpcaCookiesOut->pElems;

     ConvertStringArrayFree(caStringsOut.pElems, caStringsOut.cElems);

     return hResult;
}
```

```
STDMETHODIMP CPerPropertyBrowsingA::GetPredefinedValue(DISPID dispid,
     DWORD dwCookie, VARIANTA FAR* lpvarOut)
{
     TraceMethodEnter("CPerPropertyBrowsingA::GetPredefinedValue", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPerPropertyBrowsing,
GetPredefinedValue));

     HRESULT hResult;

     hResult = GetWide()->GetPredefinedValue(dispid, dwCookie,
(LPVARIANT)&lpvarOut);
     if (FAILED(hResult))
           return hResult;

     return ConvertVariantToA(lpvarOut);
}

//////////////////////////////////////////////////////////////////////////////
/
// CPropertyPageSiteA class


STDMETHODIMP CPropertyPageSiteA::OnStatusChange(DWORD flags)
{
     TraceMethodEnter("CPropertyPageSiteA::OnStatusChange", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPropertyPageSite, OnStatusChange));
     return GetWide()->OnStatusChange(flags);
}

STDMETHODIMP CPropertyPageSiteA::GetLocaleID(LCID FAR* pLocaleID)
{
     TraceMethodEnter("CPropertyPageSiteA::GetLocaleID", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPropertyPageSite, GetLocaleID));
     return GetWide()->GetLocaleID(pLocaleID);
}

STDMETHODIMP CPropertyPageSiteA::GetPageContainer(LPUNKNOWN FAR* ppUnk)
{
     TraceMethodEnter("CPropertyPageSiteA::GetPageContainer", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPropertyPageSite, GetPageContainer));

     LPUNKNOWN pUnk;
     HRESULT hResult;

     hResult = GetWide()->GetPageContainer(&pUnk);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIUnknownAFromW(pUnk, ppUnk);
     pUnk->Release();
     return hResult;
}

STDMETHODIMP CPropertyPageSiteA::TranslateAccelerator(LPMSG lpMsg)
```

```
{
     TraceMethodEnter("CPropertyPageSiteA::TranslateAccelerator", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPropertyPageSite,
TranslateAccelerator));
     return GetWide()->TranslateAccelerator(lpMsg);
}


///////////////////////////////////////////////////////////////////////////
/
// CPropertyPage2A class

STDMETHODIMP CPropertyPage2A::SetPageSite(LPPROPERTYPAGESITEA pPageSiteA)
{
     TraceMethodEnter("CPropertyPage2A::SetPageSite", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPropertyPage2, SetPageSite));

     LPPROPERTYPAGESITE pPageSite;
     HRESULT hResult;

     WrapIPropertyPageSiteWFromA(pPageSiteA, &pPageSite);
     hResult = GetWide()->SetPageSite(pPageSite);

     pPageSite->Release();
     return hResult;
}

STDMETHODIMP CPropertyPage2A::Activate(HWND hwndParent, LPCRECT rect,
                 BOOL bModal)
{
     TraceMethodEnter("CPropertyPage2A::Activate", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPropertyPage2, Activate));
     return GetWide()->Activate(hwndParent, rect, bModal);
}

STDMETHODIMP CPropertyPage2A::Deactivate()
{
     TraceMethodEnter("CPropertyPage2A::Deactivate", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPropertyPage2, Deactivate));
     return GetWide()->Deactivate();
}

STDMETHODIMP CPropertyPage2A::GetPageInfo(LPPROPPAGEINFOA pPageInfoA)
{
     TraceMethodEnter("CPropertyPage2A::GetPageInfo", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPropertyPage2, GetPageInfo));

     HRESULT hResult;
     PROPPAGEINFO pageInfo;

     hResult = GetWide()->GetPageInfo(&pageInfo);
     if (FAILED(hResult))
          return hResult;

     hResult = ConvertPROPPAGEINFOToA(&pageInfo, pPageInfoA);
```

```
    ConvertPROPPAGEINFOFree(&pageInfo);
    return hResult;
}

STDMETHODIMP CPropertyPage2A::SetObjects(ULONG cObjects, LPUNKNOWN FAR*
ppUnkA)
{
    TraceMethodEnter("CPropertyPage2A::SetObjects", this);
    _DebugHook(GetWide(), MEMBER_PTR(IPropertyPage2, SetObjects));

    HRESULT hResult;
    LPUNKNOWN FAR* ppUnk = new LPUNKNOWN[cObjects];

    ULONG i;
    for (i = 0; i < cObjects; i++)
        ppUnk[i] = NULL;

    for (i = 0; i < cObjects; i++)
    {
        hResult = WrapIUnknownWFromA(ppUnkA[i], &ppUnk[i]);
        if (FAILED(hResult))
            goto Error;
    }

    hResult = GetWide()->SetObjects(cObjects, ppUnk);

Error:
    for (i = 0; i < cObjects; i++)
        if (ppUnk[i] != NULL)
            ppUnk[i]->Release();
    delete [] ppUnk;

    return hResult;
}

STDMETHODIMP CPropertyPage2A::Show(UINT nCmdShow)
{
    TraceMethodEnter("CPropertyPage2A::Show", this);
    _DebugHook(GetWide(), MEMBER_PTR(IPropertyPage2, Show));
    return GetWide()->Show(nCmdShow);
}

STDMETHODIMP CPropertyPage2A::Move(LPCRECT prect)
{
    TraceMethodEnter("CPropertyPage2A::Move", this);
    _DebugHook(GetWide(), MEMBER_PTR(IPropertyPage2, Move));
    return GetWide()->Move(prect);
}

STDMETHODIMP CPropertyPage2A::IsPageDirty()
{
    TraceMethodEnter("CPropertyPage2A::IsPageDirty", this);
    _DebugHook(GetWide(), MEMBER_PTR(IPropertyPage2, IsPageDirty));
    return GetWide()->IsPageDirty();
```

```
}

STDMETHODIMP CPropertyPage2A::Apply()
{
     TraceMethodEnter("CPropertyPage2A::Apply", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPropertyPage2, Apply));
     return GetWide()->Apply();
}

STDMETHODIMP CPropertyPage2A::Help(LPCSTR lpszHelpDirA)
{
     TraceMethodEnter("CPropertyPage2A::Help", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPropertyPage2, Help));

     WCHAR  szHelpDir[MAX_STRING];
     if (lpszHelpDirA != NULL)
           ConvertStringToW(lpszHelpDirA, szHelpDir);
     else
           szHelpDir[0] = L'\0';

     return GetWide()->Help(szHelpDir);
}

STDMETHODIMP CPropertyPage2A::TranslateAccelerator(LPMSG lpMsg)
{
     TraceMethodEnter("CPropertyPage2A::TranslateAccelerator", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPropertyPage2,
TranslateAccelerator));
     return GetWide()->TranslateAccelerator(lpMsg);
}

STDMETHODIMP CPropertyPage2A::EditProperty(DISPID dispid)
{
     TraceMethodEnter("CPropertyPage2A::EditProperty", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPropertyPage2, EditProperty));
     return GetWide()->EditProperty(dispid);
}


///////////////////////////////////////////////////////////////////////////
/
// CFontA class

STDMETHODIMP CFontA::get_Name(BSTRA FAR* pname)
{
     TraceMethodEnter("CFontA::get_Name", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, get_Name));

     HRESULT hResult;
     BSTR bstr, *pbstr;
     pbstr = (pname ? &bstr : NULL);

     hResult = GetWide()->get_Name(pbstr);
     if (FAILED(hResult))
           return hResult;
```

```
	if (pbstr)
	{
		hResult = ConvertDispStringToA(bstr, pname);
		SysFreeString(bstr);
	}

	return hResult;
}

STDMETHODIMP CFontA::put_Name(BSTRA name)
{
	TraceMethodEnter("CFontA::put_Name", this);
	_DebugHook(GetWide(), MEMBER_PTR(IFont, put_Name));

	HRESULT hResult;
	BSTR bstr;

	hResult = ConvertDispStringToW(name, &bstr);
	if (FAILED(hResult))
		return hResult;

	hResult = GetWide()->put_Name(bstr);
	ConvertDispStringFreeW(bstr);
	return hResult;
}

STDMETHODIMP CFontA::get_Size(CY FAR* psize)
{
	TraceMethodEnter("CFontA::get_Size", this);
	_DebugHook(GetWide(), MEMBER_PTR(IFont, get_Size));
	return GetWide()->get_Size(psize);
}

STDMETHODIMP CFontA::put_Size(CY size)
{
	TraceMethodEnter("CFontA::put_Size", this);
	_DebugHook(GetWide(), MEMBER_PTR(IFont, put_Size));
	return GetWide()->put_Size(size);
}

STDMETHODIMP CFontA::get_Bold(BOOL FAR* pbold)
{
	TraceMethodEnter("CFontA::get_Bold", this);
	_DebugHook(GetWide(), MEMBER_PTR(IFont, get_Bold));
	return GetWide()->get_Bold(pbold);
}

STDMETHODIMP CFontA::put_Bold(BOOL bold)
{
	TraceMethodEnter("CFontA::put_Bold", this);
	_DebugHook(GetWide(), MEMBER_PTR(IFont, put_Bold));
	return GetWide()->put_Bold(bold);
}
```

```
STDMETHODIMP CFontA::get_Italic(BOOL FAR* pitalic)
{
     TraceMethodEnter("CFontA::get_Italic", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, get_Italic));
     return GetWide()->get_Italic(pitalic);
}

STDMETHODIMP CFontA::put_Italic(BOOL italic)
{
     TraceMethodEnter("CFontA::put_Italic", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, put_Italic));
     return GetWide()->put_Italic(italic);
}


STDMETHODIMP CFontA::get_Underline(BOOL FAR* punderline)
{
     TraceMethodEnter("CFontA::get_Underline", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, get_Underline));
     return GetWide()->get_Underline(punderline);
}

STDMETHODIMP CFontA::put_Underline(BOOL underline)
{
     TraceMethodEnter("CFontA::put_Underline", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, put_Underline));
     return GetWide()->put_Underline(underline);
}

STDMETHODIMP CFontA::get_Strikethrough(BOOL FAR* pstrikethrough)
{
     TraceMethodEnter("CFontA::get_Strikethrough", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, get_Strikethrough));
     return GetWide()->get_Strikethrough(pstrikethrough);
}

STDMETHODIMP CFontA::put_Strikethrough(BOOL strikethrough)
{
     TraceMethodEnter("CFontA::put_Strikethrough", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, put_Strikethrough));
     return GetWide()->put_Strikethrough(strikethrough);
}

STDMETHODIMP CFontA::get_Weight(short FAR* pweight)
{
     TraceMethodEnter("CFontA::get_Weight", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, get_Weight));
     return GetWide()->get_Weight(pweight);
}

STDMETHODIMP CFontA::put_Weight(short weight)
{
     TraceMethodEnter("CFontA::put_Weight", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, put_Weight));
     return GetWide()->put_Weight(weight);
```

```
}

STDMETHODIMP CFontA::get_Charset(short FAR* pcharset)
{
     TraceMethodEnter("CFontA::get_Charset", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, get_Charset));
     return GetWide()->get_Charset(pcharset);
}

STDMETHODIMP CFontA::put_Charset(short charset)
{
     TraceMethodEnter("CFontA::put_Charset", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, put_Charset));
     return GetWide()->put_Charset(charset);
}

STDMETHODIMP CFontA::get_hFont(HFONT FAR* phfont)
{
     TraceMethodEnter("CFontA::get_hFont", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, get_hFont));
     return GetWide()->get_hFont(phfont);
}

STDMETHODIMP CFontA::Clone(IFontA FAR* FAR* lplpFont)
{
     TraceMethodEnter("CFontA::Clone", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, Clone));

     LPFONT lpFont;
     HRESULT hResult;

     hResult = GetWide()->Clone(&lpFont);
     if (FAILED(hResult))
          return hResult;

     hResult = WrapIFontAFromW(lpFont, lplpFont);
     lpFont->Release();
     return hResult;
}

STDMETHODIMP CFontA::IsEqual(IFontA FAR* lpFontOtherA)
{
     TraceMethodEnter("CFontA::IsEqual", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, IsEqual));

     HRESULT hResult;
     LPFONT lpFontOther;
     hResult = WrapIFontWFromA(lpFontOtherA, &lpFontOther);
     if (FAILED(hResult))
          return hResult;

     hResult = GetWide()->IsEqual(lpFontOther);
     lpFontOther->Release();

     return hResult;
```

```
}

STDMETHODIMP CFontA::SetRatio(long cyLogical, long cyHimetric)
{
     TraceMethodEnter("CFontA::SetRatio", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, SetRatio));
     return GetWide()->SetRatio(cyLogical, cyHimetric);
}

STDMETHODIMP CFontA::QueryTextMetrics(LPTEXTMETRIC lptm)
{
     TraceMethodEnter("CFontA::QueryTextMetrics", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, QueryTextMetrics));
     return GetWide()->QueryTextMetrics(lptm);
}

STDMETHODIMP CFontA::AddRefHfont(HFONT hfont)
{
     TraceMethodEnter("CFontA::AddRefHfont", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, AddRefHfont));
     return GetWide()->AddRefHfont(hfont);
}

STDMETHODIMP CFontA::ReleaseHfont(HFONT hfont)
{
     TraceMethodEnter("CFontA::ReleaseHfont", this);
     _DebugHook(GetWide(), MEMBER_PTR(IFont, ReleaseHfont));
     return GetWide()->ReleaseHfont(hfont);
}


//////////////////////////////////////////////////////////////////////////////
/
// CPictureA class

STDMETHODIMP CPictureA::get_Handle(OLE_HANDLE FAR* phandle)
{
     TraceMethodEnter("CPictureA::get_Handle", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPicture, get_Handle));
     return GetWide()->get_Handle(phandle);
}

STDMETHODIMP CPictureA::get_hPal(OLE_HANDLE FAR* phpal)

{
     TraceMethodEnter("CPictureA::get_hPal", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPicture, get_hPal));
     return GetWide()->get_hPal(phpal);
}

STDMETHODIMP CPictureA::get_Type(short FAR* ptype)
{
     TraceMethodEnter("CPictureA::get_Type", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPicture, get_Type));
     return GetWide()->get_Type(ptype);
```

```
}

STDMETHODIMP CPictureA::get_Width(OLE_XSIZE_HIMETRIC FAR* pwidth)
{
     TraceMethodEnter("CPictureA::get_Width", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPicture, get_Width));
     return GetWide()->get_Width(pwidth);
}

STDMETHODIMP CPictureA::get_Height(OLE_YSIZE_HIMETRIC FAR* pheight)
{
     TraceMethodEnter("CPictureA::get_Height", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPicture, get_Height));
     return GetWide()->get_Height(pheight);
}

STDMETHODIMP CPictureA::Render(HDC hdc, long x, long y, long cx, long cy,
     OLE_XPOS_HIMETRIC xSrc, OLE_YPOS_HIMETRIC ySrc,
     OLE_XSIZE_HIMETRIC cxSrc, OLE_YSIZE_HIMETRIC cySrc,
     LPRECT lprcWBounds)
{
     TraceMethodEnter("CPictureA::Render", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPicture, Render));
     return GetWide()->Render(hdc, x, y, cx, cy, xSrc, ySrc, cxSrc, cySrc,
           lprcWBounds);
}

STDMETHODIMP CPictureA::set_hPal(OLE_HANDLE hpal)
{
     TraceMethodEnter("CPictureA::set_hPal", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPicture, set_hPal));
     return GetWide()->set_hPal(hpal);
}

STDMETHODIMP CPictureA::get_CurDC(HDC FAR * phdcOut)
{
     TraceMethodEnter("CPictureA::get_CurDC", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPicture, get_CurDC));
     return GetWide()->get_CurDC(phdcOut);
}

STDMETHODIMP CPictureA::SelectPicture(
     HDC hdcIn, HDC FAR * phdcOut, OLE_HANDLE FAR * phbmpOut)
{
     TraceMethodEnter("CPictureA::SelectPicture", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPicture, SelectPicture));
     return GetWide()->SelectPicture(hdcIn, phdcOut, phbmpOut);
}

STDMETHODIMP CPictureA::get_KeepOriginalFormat(BOOL * pfkeep)
{
     TraceMethodEnter("CPictureA::get_KeepOriginalFormat", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPicture, get_KeepOriginalFormat));
     return GetWide()->get_KeepOriginalFormat(pfkeep);
}
```

```
STDMETHODIMP CPictureA::put_KeepOriginalFormat(BOOL fkeep)
{
     TraceMethodEnter("CPictureA::put_KeepOriginalFormat", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPicture, put_KeepOriginalFormat));
     return GetWide()->put_KeepOriginalFormat(fkeep);
}

STDMETHODIMP CPictureA::PictureChanged()
{
     TraceMethodEnter("CPictureA::PictureChanged", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPicture, PictureChanged));
     return GetWide()->PictureChanged();
}

STDMETHODIMP CPictureA::SaveAsFile(LPSTREAMA pStmA, BOOL fSaveMemCopy,
     LONG FAR * lpcbSize)
{
     TraceMethodEnter("CPictureA::SaveAsFile", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPicture, SaveAsFile));

     LPSTREAM pStm;
     HRESULT hResult;

     hResult = WrapIStreamWFromA(pStmA, &pStm);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->SaveAsFile(pStm, fSaveMemCopy, lpcbSize);
     pStm->Release();
     return hResult;
}

STDMETHODIMP CPictureA::get_Attributes(DWORD FAR * lpdwAttr)
{
     TraceMethodEnter("CPictureA::get_Attributes", this);
     _DebugHook(GetWide(), MEMBER_PTR(IPicture, get_Attributes));
     return GetWide()->get_Attributes(lpdwAttr);
}
```

ANSIDISP.H   (OLEANSI Sample)

```
//
+---------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       ansidisp.h
//
//  Contents:   ANSI Wrappers for Unicode Dispatch Interfaces and APIs.
//
//  Classes:    CTypeLibA        - ANSI wrapper object for ITypeLib.
//              CTypeInfoA       - ANSI wrapper object for TypeInfo.
//              CTypeCompA       - ANSI wrapper object for TypeComp.
//              CCreateTypeLibA  - ANSI wrapper object for CreateTypeLib.
//              CCreateTypeInfoA - ANSI wrapper object for CreateTypeInfo.
//              CEnumVARIANTA    - ANSI wrapper object for EnumVARIANT.
//              CDispatchA       - ANSI wrapper object for Dispatch.
//
//  History:    01-Nov-93   v-kentc     Created.
//
//---------------------------------------------------------------------------
-

#if defined(__cplusplus)
interface ITypeLibA;
interface ITypeInfoA;
interface ITypeCompA;
interface ICreateTypeLibA;
interface ICreateTypeInfoA;
interface IEnumVARIANTA;
interface IDispatchA;
#else
typedef interface ITypeLibA ITypeLibA;
typedef interface ITypeInfoA ITypeInfoA;
typedef interface ITypeCompA ITypeCompA;
typedef interface ICreateTypeLibA ICreateTypeLibA;
typedef interface ICreateTypeInfoA ICreateTypeInfoA;
typedef interface IEnumVARIANTA IEnumVARIANTA;
typedef interface IDispatchA IDispatchA;
#endif

typedef ITypeLibA * LPTYPELIBA;
typedef ITypeInfoA * LPTYPEINFOA;
typedef ITypeCompA * LPTYPECOMPA;
typedef ICreateTypeLibA * LPCREATETYPELIBA;
typedef ICreateTypeInfoA * LPCREATETYPEINFOA;
typedef IEnumVARIANTA * LPENUMVARIANTA;
typedef IDispatchA * LPDISPATCHA;


/*---------------------------------------------------------------------*/
/*                          ANSI BSTR API                               */
/*---------------------------------------------------------------------*/
```

```
typedef char FAR* BSTRA;
typedef BSTRA * LPBSTRA;

STDAPI_(BSTRA) SysAllocStringA(const char FAR*);
STDAPI_(BSTRA) SysAllocStringLenA(const char FAR*, unsigned int);
STDAPI_(int)   SysReAllocStringA(BSTRA FAR*, const char FAR*);
STDAPI_(int)   SysReAllocStringLenA(BSTRA FAR*, const char FAR*, unsigned
int);
STDAPI_(unsigned int) SysStringLenA(BSTRA);
STDAPI_(void)  SysFreeStringA(BSTRA);


/*---------------------------------------------------------------------*/
/*                         ANSI SafeArray API                          */
/*---------------------------------------------------------------------*/

typedef SAFEARRAY SAFEARRAYA;

STDAPI
SafeArrayAllocDescriptorA(unsigned int cDims, SAFEARRAYA FAR* FAR* ppsaOut);

STDAPI SafeArrayAllocDataA(SAFEARRAYA FAR* psa);

STDAPI_(SAFEARRAYA FAR*)
SafeArrayCreateA(
     VARTYPE vt,
     unsigned int cDims,
     SAFEARRAYBOUND FAR* rgsabound);

STDAPI SafeArrayDestroyDescriptorA(SAFEARRAYA FAR* psa);

STDAPI SafeArrayDestroyDataA(SAFEARRAYA FAR* psa);

STDAPI SafeArrayDestroyA(SAFEARRAYA FAR* psa);

STDAPI SafeArrayRedimA(SAFEARRAYA FAR* psa, SAFEARRAYBOUND FAR*
psaboundNew);

STDAPI_(unsigned int) SafeArrayGetDimA(SAFEARRAYA FAR* psa);

STDAPI_(unsigned int) SafeArrayGetElemsizeA(SAFEARRAYA FAR* psa);

STDAPI
SafeArrayGetUBoundA(
     SAFEARRAYA FAR* psa,
     unsigned int nDim,
     long FAR* plUbound);

STDAPI
SafeArrayGetLBoundA(
     SAFEARRAYA FAR* psa,
     unsigned int nDim,
     long FAR* plLbound);
```

```
STDAPI SafeArrayLockA(SAFEARRAYA FAR* psa);

STDAPI SafeArrayUnlockA(SAFEARRAYA FAR* psa);

STDAPI SafeArrayAccessDataA(SAFEARRAYA FAR* psa, void HUGEP* FAR* ppvData);

STDAPI SafeArrayUnaccessDataA(SAFEARRAYA FAR* psa);

STDAPI
SafeArrayGetElementA(
     SAFEARRAYA FAR* psa,
     long FAR* rgIndices,
     void FAR* pv);

STDAPI
SafeArrayPutElementA(
     SAFEARRAYA FAR* psa,
     long FAR* rgIndices,
     void FAR* pv);

STDAPI
SafeArrayCopyA(
     SAFEARRAYA FAR* psa,
     SAFEARRAYA FAR* FAR* ppsaOut);



/*---------------------------------------------------------------------*/
/*                         ANSI VARIANT API                            */
/*---------------------------------------------------------------------*/

typedef struct FARSTRUCT tagVARIANTA VARIANTA;
typedef struct FARSTRUCT tagVARIANTA FAR* LPVARIANTA;
typedef struct FARSTRUCT tagVARIANTA VARIANTARGA;
typedef struct FARSTRUCT tagVARIANTA FAR* LPVARIANTARGA;

struct FARSTRUCT tagVARIANTA{
     VARTYPE vt;
     unsigned short wReserved1;
     unsigned short wReserved2;
     unsigned short wReserved3;
     union {
       unsigned char bVal;              /* VT_UI1                */
       short    iVal;              /* VT_I2                */
       long     lVal;              /* VT_I4                */
       float    fltVal;            /* VT_R4                */
       double       dblVal;            /* VT_R8                */
       VARIANT_BOOL bool;              /* VT_BOOL              */
       SCODE    scode;             /* VT_ERROR             */
       CY       cyVal;             /* VT_CY                */
       DATE     date;              /* VT_DATE              */
       BSTRA    bstrVal;           /* VT_BSTR              */
       IUnknown     FAR* punkVal;      /* VT_UNKNOWN           */
       IDispatchA   FAR* pdispVal;     /* VT_DISPATCH          */
       SAFEARRAYA   FAR* parray;       /* VT_ARRAY|*           */

       unsigned char    FAR* pbVal;        /* VT_BYREF|VT_UI1       */
       short    FAR* piVal;        /* VT_BYREF|VT_I2       */
       long     FAR* plVal;        /* VT_BYREF|VT_I4       */
       float    FAR* pfltVal;      /* VT_BYREF|VT_R4       */
       double       FAR* pdblVal;      /* VT_BYREF|VT_R8       */
       VARIANT_BOOL FAR* pbool;        /* VT_BYREF|VT_BOOL     */
       SCODE    FAR* pscode;       /* VT_BYREF|VT_ERROR    */
       CY       FAR* pcyVal;       /* VT_BYREF|VT_CY       */
       DATE     FAR* pdate;        /* VT_BYREF|VT_DATE     */
       BSTRA    FAR* pbstrVal;     /* VT_BYREF|VT_BSTR     */
       IUnknown   FAR* FAR* ppunkVal;  /* VT_BYREF|VT_UNKNOWN  */
       IDispatchA FAR* FAR* ppdispVal; /* VT_BYREF|VT_DISPATCH */
       SAFEARRAYA FAR* FAR* pparray;   /* VT_BYREF|VT_ARRAY|*  */
       VARIANTA     FAR* pvarVal;      /* VT_BYREF|VT_VARIANT  */

       void     FAR* byref;        /* Generic ByRef        */
     }
#if defined(NONAMELESSUNION) || (defined(_MAC) && !defined(__cplusplus) && !
defined(_MSC_VER))
     u
#endif
     ;
};

STDAPI_(void)
VariantInitA(VARIANTARGA FAR* pvarg);

STDAPI
VariantClearA(VARIANTARGA FAR* pvarg);

STDAPI
VariantCopyA(
     VARIANTARGA FAR* pvargDest,
     VARIANTARGA FAR* pvargSrc);

STDAPI
VariantCopyIndA(
     VARIANTA FAR* pvarDest,
     VARIANTARGA FAR* pvargSrc);

STDAPI
VariantChangeTypeA(
     VARIANTARGA FAR* pvargDest,
     VARIANTARGA FAR* pvarSrc,
     unsigned short wFlags,
     VARTYPE vt);

STDAPI
VariantChangeTypeExA(
     VARIANTARGA FAR* pvargDest,
     VARIANTARGA FAR* pvarSrc,
     LCID lcid,
     unsigned short wFlags,
     VARTYPE vt);


/*---------------------------------------------------------------------*/
/*                   ANSI VARIANT Coercion API                         */
/*---------------------------------------------------------------------*/

#ifndef NOI1
STDAPI VarUI1FromStrA(char FAR* strIn, LCID lcid,
                              unsigned long dwFlags, unsigned char FAR*
pbOut);
STDAPI VarUI1FromDispA(IDispatchA FAR* pdispIn, LCID lcid, unsigned char
FAR* pbOut);
#endif //!NOI1

STDAPI VarI2FromStrA(char FAR* strIn, LCID lcid,
                              unsigned long dwFlags, short FAR* psOut);
STDAPI VarI2FromDispA(IDispatchA FAR* pdispIn, LCID lcid, short FAR* psOut);

STDAPI VarI4FromStrA(char FAR* strIn, LCID lcid,
                              unsigned long dwFlags, long FAR* plOut);
STDAPI VarI4FromDispA(IDispatchA FAR* pdispIn, LCID lcid, long FAR* plOut);

STDAPI VarR4FromStrA(char FAR* strIn, LCID lcid,
                              unsigned long dwFlags, float FAR* pfltOut);
STDAPI VarR4FromDispA(IDispatchA FAR* pdispIn, LCID lcid, float FAR*
pfltOut);


STDAPI VarR8FromStrA(char FAR* strIn, LCID lcid,
                              unsigned long dwFlags, double FAR* pdblOut);
STDAPI VarR8FromDispA(IDispatchA FAR* pdispIn, LCID lcid, double FAR*
pdblOut);

STDAPI VarDateFromStrA(char FAR* strIn, LCID lcid,
                                unsigned long dwFlags, DATE FAR* pdateOut);
STDAPI VarDateFromDispA(IDispatchA FAR* pdispIn, LCID lcid, DATE FAR*
pdateOut);
STDAPI VarCyFromStrA(char FAR* strIn, LCID lcid,
                              unsigned long dwFlags, CY FAR* pcyOut);
STDAPI VarCyFromDispA(IDispatchA FAR* pdispIn, LCID lcid, CY FAR* pcyOut);

#ifndef NOI1
STDAPI VarBstrFromUI1A(unsigned char bVal, LCID lcid,
                               unsigned long dwFlags, BSTRA FAR* pbstrOut);
#endif //!NOI1
STDAPI VarBstrFromI2A(short iVal, LCID lcid,
                               unsigned long dwFlags, BSTRA FAR* pbstrOut);
STDAPI VarBstrFromI4A(long lIn, LCID lcid,
                               unsigned long dwFlags, BSTRA FAR* pbstrOut);
STDAPI VarBstrFromR4A(float fltIn, LCID lcid,
                               unsigned long dwFlags, BSTRA FAR* pbstrOut);
STDAPI VarBstrFromR8A(double dblIn, LCID lcid,
                               unsigned long dwFlags, BSTRA FAR* pbstrOut);
STDAPI VarBstrFromCyA(CY cyIn, LCID lcid,
                               unsigned long dwFlags, BSTRA FAR* pbstrOut);
STDAPI VarBstrFromDateA(DATE dateIn, LCID lcid,
                                   unsigned long dwFlags, BSTRA FAR*
pbstrOut);
STDAPI VarBstrFromDispA(IDispatchA FAR* pdispIn, LCID lcid,
                                   unsigned long dwFlags, BSTRA FAR*
pbstrOut);
STDAPI VarBstrFromBoolA(VARIANT_BOOL boolIn, LCID lcid,
                                   unsigned long dwFlags, BSTRA FAR*
pbstrOut);

STDAPI VarBoolFromStrA(char FAR* strIn, LCID lcid,
                                unsigned long dwFlags, VARIANT_BOOL FAR*
pboolOut);
STDAPI VarBoolFromDispA(IDispatchA FAR* pdispIn, LCID lcid,
                                   VARIANT_BOOL FAR* pboolOut);


/***********************************************************************/
/*                                                                     */
/*                  ANSI Support for Dispatch.h Elements               */
/*                                                                     */
/***********************************************************************/

/*---------------------------------------------------------------------*/
/*                       ANSI Dispatch Typedefs                        */
/*---------------------------------------------------------------------*/

/*---------------------------------------------------------------------*/
/*                       ANSI Dispatch Typedefs                        */
/*---------------------------------------------------------------------*/

typedef struct tagEXCEPINFOA {
     unsigned short wCode;             /* An error code describing the
error. */
                               /* Either (but not both) the wCode or */
                               /* scode fields must be set */
     unsigned short wReserved;

     BSTRA bstrSource;       /* A textual, human readable name of the
                          source of the exception. It is up to the
                          IDispatch implementor to fill this in.
                          Typically this will be an application name. */

     BSTRA bstrDescription;   /* A textual, human readable description of
the
                          error. If no description is available, NULL
                          should be used. */

     BSTRA bstrHelpFile;      /* Fully qualified drive, path, and file name
                          of a help file with more information about
                          the error.  If no help is available, NULL
                          should be used. */

     unsigned long dwHelpContext;
```

```
                    /* help context of topic within the help file. */

     void * pvReserved;

     /* Use of this field allows an application to defer filling in
        the bstrDescription, bstrHelpFile, and dwHelpContext fields
        until they are needed.  This field might be used, for example,
        if loading the string for the error is a time-consuming
        operation. If deferred fill-in is not desired, this field should
        be set to NULL. */
#ifdef _MAC
# ifdef _MSC_VER
     HRESULT (STDAPICALLTYPE * pfnDeferredFillIn)(struct tagEXCEPINFOA *);
# else
     STDAPICALLTYPE HRESULT (* pfnDeferredFillIn)(struct tagEXCEPINFOA *);
# endif
#else
     HRESULT (STDAPICALLTYPE * pfnDeferredFillIn)(struct tagEXCEPINFOA *);
#endif

     SCODE scode;        /* An SCODE describing the error. */

} EXCEPINFOA, * LPEXCEPINFOA;

typedef struct tagVARDESCA {
     MEMBERID memid;
     CHAR * lpstrSchema;     /* reserved for future use */
     union {
       /* VAR_PERINSTANCE - the offset of this variable within the instance
*/
       unsigned long oInst;

       /* VAR_CONST - the value of the constant */
       VARIANTA * lpvarValue;


     };
     ELEMDESC elemdescVar;
     unsigned short wVarFlags;
     VARKIND varkind;
} VARDESCA, * LPVARDESCA;

typedef struct tagPARAMDATAA {
     CHAR * szName;      /* parameter name */
     VARTYPE vt;         /* parameter type */
} PARAMDATAA, * LPPARAMDATAA;

typedef struct tagMETHODDATAA {
     CHAR * szName;      /* method name */
     PARAMDATAA * ppdata;    /* pointer to an array of PARAMDATAs */
     DISPID dispid;      /* method ID */
     unsigned int iMeth;     /* method index */
     CALLCONV cc;        /* calling convention */
     unsigned int cArgs;     /* count of arguments */
     unsigned short wFlags;  /* same wFlags as on IDispatch::Invoke() */
     VARTYPE vtReturn;
} METHODDATAA, * LPMETHODDATAA;

typedef struct tagINTERFACEDATAA {
     METHODDATAA * pmethdata;    /* pointer to an array of METHODDATAs */
     unsigned int cMembers;  /* count of members */
} INTERFACEDATAA, * LPINTERFACEDATAA;

typedef union tagBINDPTRA {
     FUNCDESC * lpfuncdesc;
     VARDESCA * lpvardesc;
     ITypeCompA * lptcomp;
} BINDPTRA, * LPBINDPTRA;

typedef struct tagDISPPARAMSA{
     VARIANTARGA * rgvarg;
     DISPID * rgdispidNamedArgs;
     unsigned int cArgs;
     unsigned int cNamedArgs;
} DISPPARAMSA, * LPDISPPARAMSA;

typedef DISPPARAMS * LPDISPPARAMS;


/*----------------------------------------------------------------------*/
/*                             ITypeLibA                                */
/*----------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE ITypeLibA

DECLARE_INTERFACE_(ITypeLibA, IUnknown)
{
     //BEGIN_INTERFACE

     /* IUnknown methods */
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, void FAR* FAR* ppvObj)
PURE;
     STDMETHOD_(unsigned long, AddRef)(THIS) PURE;
     STDMETHOD_(unsigned long, Release)(THIS) PURE;

     /* ITypeLib methods */
     STDMETHOD_(unsigned int,GetTypeInfoCount)(THIS) PURE;

     STDMETHOD(GetTypeInfo)(THIS_
       unsigned int index, ITypeInfoA FAR* FAR* pptinfo) PURE;

     STDMETHOD(GetTypeInfoType)(THIS_
       unsigned int index, TYPEKIND FAR* ptypekind) PURE;

     STDMETHOD(GetTypeInfoOfGuid)(THIS_
       REFGUID guid, ITypeInfoA FAR* FAR* pptinfo) PURE;

     STDMETHOD(GetLibAttr)(THIS_
       TLIBATTR FAR* FAR* pptlibattr) PURE;
     STDMETHOD(GetTypeComp)(THIS_
       ITypeCompA FAR* FAR* pptcomp) PURE;

     STDMETHOD(GetDocumentation)(THIS_
       int index,
       BSTRA FAR* pbstrName,
       BSTRA FAR* pbstrDocString,
       unsigned long FAR* pdwHelpContext,
       BSTRA FAR* pbstrHelpFile) PURE;

     STDMETHOD(IsName)(THIS_
       CHAR FAR* szNameBuf,
       unsigned long lHashVal,
       int FAR* lpfName) PURE;

     STDMETHOD(FindName)(THIS_
       CHAR FAR* szNameBuf,
       unsigned long lHashVal,
       ITypeInfoA FAR* FAR* rgptinfo,
       MEMBERID FAR* rgmemid,
       unsigned short FAR* pcFound) PURE;

     STDMETHOD_(void, ReleaseTLibAttr)(THIS_ TLIBATTR FAR* ptlibattr) PURE;
};

typedef ITypeLibA FAR* LPTYPELIBA;


/*---------------------------------------------------------------------*/
/*                            ITypeInfoA                               */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE ITypeInfoA

DECLARE_INTERFACE_(ITypeInfoA, IUnknown)
{
     //BEGIN_INTERFACE

     /* IUnknown methods */
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, void FAR* FAR* ppvObj)
PURE;
     STDMETHOD_(unsigned long, AddRef)(THIS) PURE;
     STDMETHOD_(unsigned long, Release)(THIS) PURE;

     /* ITypeInfo methods */
     STDMETHOD(GetTypeAttr)(THIS_ TYPEATTR FAR* FAR* pptypeattr) PURE;

     STDMETHOD(GetTypeComp)(THIS_ ITypeCompA FAR* FAR* pptcomp) PURE;

     STDMETHOD(GetFuncDesc)(THIS_
       unsigned int index, FUNCDESC FAR* FAR* ppfuncdesc) PURE;

     STDMETHOD(GetVarDesc)(THIS_
       unsigned int index, VARDESCA FAR* FAR* ppvardesc) PURE;

     STDMETHOD(GetNames)(THIS_
       MEMBERID memid,
       BSTRA FAR* rgbstrNames,
       unsigned int cMaxNames,
       unsigned int FAR* pcNames) PURE;

     STDMETHOD(GetRefTypeOfImplType)(THIS_
       unsigned int index, HREFTYPE FAR* phreftype) PURE;

     STDMETHOD(GetImplTypeFlags)(THIS_
       unsigned int index, int FAR* pimpltypeflags) PURE;

     STDMETHOD(GetIDsOfNames)(THIS_
       char FAR* FAR* rgszNames,
       unsigned int cNames,
       MEMBERID FAR* rgmemid) PURE;

     STDMETHOD(Invoke)(THIS_
       void FAR* pvInstance,
       MEMBERID memid,
       unsigned short wFlags,
       DISPPARAMSA FAR *pdispparamsA,
       VARIANTA FAR *pvarResult,
       EXCEPINFOA FAR *pexcepinfo,
       unsigned int FAR *puArgErr) PURE;

     STDMETHOD(GetDocumentation)(THIS_
       MEMBERID memid,
       BSTRA FAR* pbstrName,
       BSTRA FAR* pbstrDocString,
       unsigned long FAR* pdwHelpContext,
       BSTRA FAR* pbstrHelpFile) PURE;

     STDMETHOD(GetDllEntry)(THIS_
       MEMBERID memid,
       INVOKEKIND invkind,
       BSTRA FAR* pbstrDllName,
       BSTRA FAR* pbstrName,
       unsigned short FAR* pwOrdinal) PURE;

     STDMETHOD(GetRefTypeInfo)(THIS_
       HREFTYPE hreftype, ITypeInfoA FAR* FAR* pptinfo) PURE;

     STDMETHOD(AddressOfMember)(THIS_
       MEMBERID memid, INVOKEKIND invkind, void FAR* FAR* ppv) PURE;

     STDMETHOD(CreateInstance)(THIS_
       IUnknown FAR* punkOuter,
       REFIID riid,
       void FAR* FAR* ppvObj) PURE;

     STDMETHOD(GetMops)(THIS_ MEMBERID memid, BSTRA FAR* pbstrMops) PURE;
```

```
     STDMETHOD(GetContainingTypeLib)(THIS_
       ITypeLibA FAR* FAR* pptlib, unsigned int FAR* pindex) PURE;

     STDMETHOD_(void, ReleaseTypeAttr)(THIS_ TYPEATTR FAR* ptypeattr) PURE;
     STDMETHOD_(void, ReleaseFuncDesc)(THIS_ FUNCDESC FAR* pfuncdesc) PURE;
     STDMETHOD_(void, ReleaseVarDesc)(THIS_ VARDESCA FAR* pvardesc) PURE;
};

typedef ITypeInfoA FAR* LPTYPEINFOA;


/*---------------------------------------------------------------------*/
/*                             ITypeCompA                              */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE ITypeCompA

DECLARE_INTERFACE_(ITypeCompA, IUnknown)
{
     //BEGIN_INTERFACE

     /* IUnknown methods */
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, void FAR* FAR* ppvObj)
PURE;
     STDMETHOD_(unsigned long, AddRef)(THIS) PURE;
     STDMETHOD_(unsigned long, Release)(THIS) PURE;

     /* ITypeComp methods */
     STDMETHOD(Bind)(THIS_
       CHAR FAR* szName,
       unsigned long lHashVal,
       unsigned short wflags,
       ITypeInfoA FAR* FAR* pptinfo,
       DESCKIND FAR* pdesckind,
       BINDPTRA FAR* pbindptr) PURE;

     STDMETHOD(BindType)(THIS_
       CHAR FAR* szName,
       unsigned long lHashVal,
       ITypeInfoA FAR* FAR* pptinfoA,
       ITypeCompA FAR* FAR* pptcompA) PURE;
};

typedef ITypeCompA FAR* LPTYPECOMPA;


/*---------------------------------------------------------------------*/
/*                          ICreateTypeLibA                            */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE ICreateTypeLibA
```

```
DECLARE_INTERFACE_(ICreateTypeLibA, IUnknown)
{
     //BEGIN_INTERFACE

     /* IUnknown methods */
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, void FAR* FAR* ppvObj)
PURE;
     STDMETHOD_(unsigned long, AddRef)(THIS) PURE;
     STDMETHOD_(unsigned long, Release)(THIS) PURE;

     /* ICreateTypeLib methods */
     STDMETHOD(CreateTypeInfo)(THIS_
       CHAR FAR* szName,
       TYPEKIND tkind,
       ICreateTypeInfoA FAR* FAR* lplpictinfo) PURE;

     STDMETHOD(SetName)(THIS_ CHAR FAR* szName) PURE;

     STDMETHOD(SetVersion)(THIS_
       unsigned short wMajorVerNum, unsigned short wMinorVerNum) PURE;

     STDMETHOD(SetGuid) (THIS_ REFGUID guid) PURE;

     STDMETHOD(SetDocString)(THIS_ CHAR FAR* szDoc) PURE;

     STDMETHOD(SetHelpFileName)(THIS_ CHAR FAR* szHelpFileName) PURE;

     STDMETHOD(SetHelpContext)(THIS_ unsigned long dwHelpContext) PURE;

     STDMETHOD(SetLcid)(THIS_ LCID lcid) PURE;


     STDMETHOD(SetLibFlags)(THIS_ unsigned int uLibFlags) PURE;

     STDMETHOD(SaveAllChanges)(THIS) PURE;
};

typedef ICreateTypeLibA FAR* LPCREATETYPELIBA;


/*---------------------------------------------------------------------*/
/*                         ICreateTypeInfoA                            */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE ICreateTypeInfoA

DECLARE_INTERFACE_(ICreateTypeInfoA, IUnknown)
{
     //BEGIN_INTERFACE

     /* IUnknown methods */
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, void FAR* FAR* ppvObj)
PURE;
     STDMETHOD_(unsigned long, AddRef)(THIS) PURE;
     STDMETHOD_(unsigned long, Release)(THIS) PURE;

     /* ICreateTypeInfo methods */
     STDMETHOD(SetGuid)(THIS_ REFGUID guid) PURE;

     STDMETHOD(SetTypeFlags)(THIS_ unsigned int uTypeFlags) PURE;

     STDMETHOD(SetDocString)(THIS_ CHAR FAR* pstrDoc) PURE;

     STDMETHOD(SetHelpContext)(THIS_ unsigned long dwHelpContext) PURE;

     STDMETHOD(SetVersion)(THIS_
       unsigned short wMajorVerNum, unsigned short wMinorVerNum) PURE;

     STDMETHOD(AddRefTypeInfo)(THIS_
       ITypeInfoA FAR* ptinfo, HREFTYPE FAR* phreftype) PURE;

     STDMETHOD(AddFuncDesc)(THIS_
       unsigned int index, FUNCDESC FAR* pfuncdesc) PURE;

     STDMETHOD(AddImplType)(THIS_
       unsigned int index, HREFTYPE hreftype) PURE;

     STDMETHOD(SetImplTypeFlags)(THIS_
       unsigned int index, int impltypeflags) PURE;

     STDMETHOD(SetAlignment)(THIS_ unsigned short cbAlignment) PURE;

     STDMETHOD(SetSchema)(THIS_ CHAR FAR* lpstrSchema) PURE;

     STDMETHOD(AddVarDesc)(THIS_
       unsigned int index, VARDESCA FAR* pvardesc) PURE;

     STDMETHOD(SetFuncAndParamNames)(THIS_
       unsigned int index, CHAR FAR* FAR* rgszNames, unsigned int cNames)
PURE;

     STDMETHOD(SetVarName)(THIS_
       unsigned int index, CHAR FAR* szName) PURE;

     STDMETHOD(SetTypeDescAlias)(THIS_
       TYPEDESC FAR* ptdescAlias) PURE;

     STDMETHOD(DefineFuncAsDllEntry)(THIS_
       unsigned int index, CHAR FAR* szDllName, CHAR FAR* szProcName) PURE;

     STDMETHOD(SetFuncDocString)(THIS_
       unsigned int index, CHAR FAR* szDocString) PURE;

     STDMETHOD(SetVarDocString)(THIS_
       unsigned int index, CHAR FAR* szDocString) PURE;

     STDMETHOD(SetFuncHelpContext)(THIS_
       unsigned int index, unsigned long dwHelpContext) PURE;
     STDMETHOD(SetVarHelpContext)(THIS_
       unsigned int index, unsigned long dwHelpContext) PURE;

     STDMETHOD(SetMops)(THIS_
       unsigned int index, BSTRA bstrMops) PURE;

     STDMETHOD(SetTypeIdldesc)(THIS_
       IDLDESC FAR* pidldesc) PURE;

     STDMETHOD(LayOut)(THIS) PURE;
};

typedef ICreateTypeInfoA FAR* LPCREATETYPEINFOA;


/*---------------------------------------------------------------------*/
/*                          IEnumVARIANTA                              */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE IEnumVARIANTA

DECLARE_INTERFACE_(IEnumVARIANTA, IUnknown)
{
     //BEGIN_INTERFACE

     /* IUnknown methods */
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, void FAR* FAR* ppvObj)
PURE;
     STDMETHOD_(unsigned long, AddRef)(THIS) PURE;
     STDMETHOD_(unsigned long, Release)(THIS) PURE;

     /* IEnumVARIANT methods */
     STDMETHOD(Next)(
       THIS_ unsigned long celt, VARIANTA FAR* rgvar, unsigned long FAR*
pceltFetched) PURE;
     STDMETHOD(Skip)(THIS_ unsigned long celt) PURE;
     STDMETHOD(Reset)(THIS) PURE;
     STDMETHOD(Clone)(THIS_ IEnumVARIANTA FAR* FAR* ppenum) PURE;
};

typedef IEnumVARIANTA FAR* LPENUMVARIANTA;


/*---------------------------------------------------------------------*/
/*                             IDispatchA                              */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE IDispatchA


DECLARE_INTERFACE_(IDispatchA, IUnknown)
{
     //BEGIN_INTERFACE
     /* IUnknown methods */
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, void * * ppvObj) PURE;
     STDMETHOD_(unsigned long, AddRef)(THIS) PURE;
     STDMETHOD_(unsigned long, Release)(THIS) PURE;

     /* IDispatch methods */
     STDMETHOD(GetTypeInfoCount)(THIS_ unsigned int * pctinfo) PURE;

     STDMETHOD(GetTypeInfo)(
       THIS_
       unsigned int itinfo,
       LCID lcid,
       ITypeInfoA * * pptinfo) PURE;

     STDMETHOD(GetIDsOfNames)(
       THIS_
       REFIID riid,
       char * * rgszNames,
       unsigned int cNames,
       LCID lcid,
       DISPID * rgdispid) PURE;

     STDMETHOD(Invoke)(
       THIS_
       DISPID dispidMember,
       REFIID riid,
       LCID lcid,
       unsigned short wFlags,
       DISPPARAMSA * pdispparamsA,
       VARIANTA * pvarResult,
       EXCEPINFOA * pexcepinfo,
       unsigned int * puArgErr) PURE;
};

typedef IDispatchA * LPDISPATCHA;


/*---------------------------------------------------------------------*/
/*                          ANSI Dispatch API                          */
/*---------------------------------------------------------------------*/

#define LHashValOfNameA(lcid, szName) \
     LHashValOfNameSysA(SYS_WIN32, lcid, szName)

STDAPI
LoadTypeLibA(const char * szFile, ITypeLibA * * pptlib);

STDAPI
LoadRegTypeLibA(
     REFGUID rguid,
     unsigned short wVerMajor,
     unsigned short wVerMinor,
     LCID lcid,
     ITypeLibA * * pptlib);
```

```
STDAPI
QueryPathOfRegTypeLibA(
     REFGUID guid,
     unsigned short wMaj,
     unsigned short wMin,
     LCID lcid,
     LPBSTRA lpbstrPathName);

STDAPI
RegisterTypeLibA(
     ITypeLibA * ptlib,
     char * szFullPath,
     char * szHelpDir);

STDAPI
CreateTypeLibA(SYSKIND syskind, const char * szFile, ICreateTypeLibA * *
ppctlib);


/*---------------------------------------------------------------------*/
/*                          ANSI Dispatch API                          */
/*---------------------------------------------------------------------*/

STDAPI
DispGetParamA(
     DISPPARAMSA * pdispparamsA,
     unsigned int position,
     VARTYPE vtTarg,
     VARIANTA * pvarResult,
     unsigned int * puArgErr);

STDAPI
DispGetIDsOfNamesA(
     ITypeInfoA * ptinfo,
     char * * rgszNames,
     unsigned int cNames,
     DISPID * rgdispid);

STDAPI
DispInvokeA(
     void * _this,
     ITypeInfoA * ptinfo,
     DISPID dispidMember,
     unsigned short wFlags,
     DISPPARAMSA * pparamsA,
     VARIANTA * pvarResult,
     EXCEPINFOA * pexcepinfo,
     unsigned int * puArgErr);

STDAPI
CreateDispTypeInfoA(
     INTERFACEDATAA * pidata,
     LCID lcid,
     ITypeInfoA * * pptinfo);
```

```
STDAPI
CreateStdDispatchA(
     IUnknown * punkOuter,
     void * pvThis,
     ITypeInfoA * ptinfo,
     IUnknown * * ppunkStdDisp);



//
//  Forward declarations
//
class CTypeLibA;
class CTypeInfoA;
class CTypeCompA;
class CCreateTypeLibA;
class CCreateTypeInfoA;
class CEnumVARIANTA;
class CDispatchA;


//
+-------------------------------------------------------------------------
//
//  Class:      CTypeLibA
//
//  Synopsis:   Class definition of ITypeLibA
//
//--------------------------------------------------------------------------
-
class CTypeLibA : CAnsiInterface
{
public:
     // *** ITypeLibA methods ***
     STDMETHOD_(unsigned int,GetTypeInfoCount)(VOID);

     STDMETHOD(GetTypeInfo)(
       unsigned int index, ITypeInfoA * * pptinfo);

     STDMETHOD(GetTypeInfoType)(
       unsigned int index, TYPEKIND * ptypekind);

     STDMETHOD(GetTypeInfoOfGuid)(
       REFGUID guid, ITypeInfoA * * pptinfo);

     STDMETHOD(GetLibAttr)(
       TLIBATTR * * pptlibattr);

     STDMETHOD(GetTypeComp)(
       ITypeCompA * * pptcomp);

     STDMETHOD(GetDocumentation)(
       int index,
       BSTRA * pbstrName,
       BSTRA * pbstrDocString,
       unsigned long * pdwHelpContext,
       BSTRA * pbstrHelpFile);

     STDMETHOD(IsName)(
       CHAR * szNameBuf,
       unsigned long lHashVal,
       int * lpfName);

     STDMETHOD(FindName)(
       CHAR * szNameBuf,
       unsigned long lHashVal,
       ITypeInfoA * * rgptinfo,
       MEMBERID * rgmemid,
       unsigned short * pcFound);

     STDMETHOD_(void, ReleaseTLibAttr)( TLIBATTR * ptlibattr);

     inline CTypeLibA(LPUNKNOWN pUnk, ITypeLib * pWide) :
                 CAnsiInterface(ID_ITypeLib, pUnk, (LPUNKNOWN)pWide) {};

     inline ITypeLib * GetWide() const
                 { return (ITypeLib *)m_pObj; };
};


//
+----------------------------------------------------------------------------
//
//  Class:      CTypeInfoA
//
//  Synopsis:   Class definition of ITypeInfoA
//
//-----------------------------------------------------------------------------
-
class CTypeInfoA : CAnsiInterface
{
public:
     // *** ITypeInfoA methods ***
     STDMETHOD(GetTypeAttr)(TYPEATTR FAR* FAR* pptypeattr);

     STDMETHOD(GetTypeComp)(ITypeCompA FAR* FAR* pptcomp);

     STDMETHOD(GetFuncDesc)(
       unsigned int index, FUNCDESC FAR* FAR* ppfuncdesc);

     STDMETHOD(GetVarDesc)(
       unsigned int index, VARDESCA FAR* FAR* ppvardesc);

     STDMETHOD(GetNames)(
       MEMBERID memid,
       BSTRA FAR* rgbstrNames,
       unsigned int cMaxNames,
       unsigned int FAR* pcNames);
```

```
STDMETHOD(GetRefTypeOfImplType)(
  unsigned int index, HREFTYPE FAR* phreftype);

STDMETHOD(GetImplTypeFlags)(
  unsigned int index, int FAR* pimpltypeflags);

STDMETHOD(GetIDsOfNames)(
  char FAR* FAR* rgszNames,
  unsigned int cNames,
  MEMBERID FAR* rgmemid);

STDMETHOD(Invoke)(
  void FAR* pvInstance,
  MEMBERID memid,
  unsigned short wFlags,
  DISPPARAMSA FAR *pdispparamsA,
  VARIANTA FAR *pvarResult,
  EXCEPINFOA FAR *pexcepinfo,
  unsigned int FAR *puArgErr);

STDMETHOD(GetDocumentation)(
  MEMBERID memid,
  BSTRA FAR* pbstrName,
  BSTRA FAR* pbstrDocString,
  unsigned long FAR* pdwHelpContext,
  BSTRA FAR* pbstrHelpFile);

STDMETHOD(GetDllEntry)(
  MEMBERID memid,
  INVOKEKIND invkind,
  BSTRA FAR* pbstrDllName,
  BSTRA FAR* pbstrName,
  unsigned short FAR* pwOrdinal);

STDMETHOD(GetRefTypeInfo)(
  HREFTYPE hreftype, ITypeInfoA FAR* FAR* pptinfo);

STDMETHOD(AddressOfMember)(
  MEMBERID memid, INVOKEKIND invkind, void FAR* FAR* ppv);

STDMETHOD(CreateInstance)(
  IUnknown FAR* punkOuter,
  REFIID riid,
  void FAR* FAR* ppvObj);

STDMETHOD(GetMops)(MEMBERID memid, BSTRA FAR* pbstrMops);

STDMETHOD(GetContainingTypeLib)(
  ITypeLibA FAR* FAR* pptlib, unsigned int FAR* pindex);

STDMETHOD_(void, ReleaseTypeAttr)(TYPEATTR FAR* ptypeattr);
STDMETHOD_(void, ReleaseFuncDesc)(FUNCDESC FAR* pfuncdesc);
STDMETHOD_(void, ReleaseVarDesc)(VARDESCA FAR* pvardesc);
```

```
     inline CTypeInfoA(LPUNKNOWN pUnk, ITypeInfo * pWide) :
                 CAnsiInterface(ID_ITypeInfo, pUnk, (LPUNKNOWN)pWide) {};

     inline ITypeInfo * GetWide() const
                 { return (ITypeInfo *)m_pObj; };
};


//
+---------------------------------------------------------------------------
//
//  Class:      CTypeCompA
//
//  Synopsis:   Class definition of ITypeCompA
//
//--------------------------------------------------------------------------
-
class CTypeCompA : CAnsiInterface
{
public:
     // *** ITypeCompA methods ***
     STDMETHOD(Bind)(
       CHAR * szName,
       unsigned long lHashVal,
       unsigned short wflags,
       ITypeInfoA * * pptinfo,
       DESCKIND * pdesckind,
       BINDPTRA * pbindptr);

     STDMETHOD(BindType)(
       CHAR * szName,
       unsigned long lHashVal,
       ITypeInfoA * * pptinfoA,
       ITypeCompA * * pptcompA);

     inline CTypeCompA(LPUNKNOWN pUnk, ITypeComp * pWide) :
                 CAnsiInterface(ID_ITypeComp, pUnk, (LPUNKNOWN)pWide) {};

     inline ITypeComp * GetWide() const
                 { return (ITypeComp *)m_pObj; };
};


//
+---------------------------------------------------------------------------
//
//  Class:      CCreateTypeLibA
//
//  Synopsis:   Class definition of ICreateTypeLibA
//
//--------------------------------------------------------------------------
-
class CCreateTypeLibA : CAnsiInterface
{
```

```
public:
     // *** ICreateTypeLibA methods ***
     STDMETHOD(CreateTypeInfo)(
       CHAR * szName,
       TYPEKIND tkind,
       ICreateTypeInfoA * * lplpictinfo);

     STDMETHOD(SetName)( CHAR * szName);

     STDMETHOD(SetVersion)(
       unsigned short wMajorVerNum, unsigned short wMinorVerNum);

     STDMETHOD(SetGuid) ( REFGUID guid);

     STDMETHOD(SetDocString)( CHAR * szDoc);

     STDMETHOD(SetHelpFileName)( CHAR * szHelpFileName);

     STDMETHOD(SetHelpContext)( unsigned long dwHelpContext);

     STDMETHOD(SetLcid)( LCID lcid);

     STDMETHOD(SetLibFlags)( unsigned int uLibFlags);

     STDMETHOD(SaveAllChanges)(VOID);

     inline CCreateTypeLibA(LPUNKNOWN pUnk, ICreateTypeLib * pWide) :
                 CAnsiInterface(ID_ICreateTypeLib, pUnk, (LPUNKNOWN)pWide)
{};

     inline ICreateTypeLib * GetWide() const
                 { return (ICreateTypeLib *)m_pObj; };
};


//
+--------------------------------------------------------------------------
//
//  Class:      CCreateTypeInfoA
//
//  Synopsis:   Class definition of ICreateTypeInfoA
//
//---------------------------------------------------------------------------
-
class CCreateTypeInfoA : CAnsiInterface
{
public:
     // *** ICreateTypeInfoA methods ***
     STDMETHOD(SetGuid)( REFGUID guid);

     STDMETHOD(SetTypeFlags)( unsigned int uTypeFlags);

     STDMETHOD(SetDocString)( CHAR * pstrDoc);

     STDMETHOD(SetHelpContext)( unsigned long dwHelpContext);
     STDMETHOD(SetVersion)(
       unsigned short wMajorVerNum, unsigned short wMinorVerNum);

     STDMETHOD(AddRefTypeInfo)(
       ITypeInfoA * ptinfo, HREFTYPE * phreftype);

     STDMETHOD(AddFuncDesc)(
       unsigned int index, FUNCDESC * pfuncdesc);

     STDMETHOD(AddImplType)(
       unsigned int index, HREFTYPE hreftype);

     STDMETHOD(SetImplTypeFlags)(
       unsigned int index, int impltypeflags);

     STDMETHOD(SetAlignment)( unsigned short cbAlignment);

     STDMETHOD(SetSchema)( CHAR * lpstrSchema);

     STDMETHOD(AddVarDesc)(
       unsigned int index, VARDESCA * pvardesc);

     STDMETHOD(SetFuncAndParamNames)(
       unsigned int index, CHAR * * rgszNames, unsigned int cNames);

     STDMETHOD(SetVarName)(
       unsigned int index, CHAR * szName);

     STDMETHOD(SetTypeDescAlias)(
       TYPEDESC * ptdescAlias);

     STDMETHOD(DefineFuncAsDllEntry)(
       unsigned int index, CHAR * szDllName, CHAR * szProcName);

     STDMETHOD(SetFuncDocString)(
       unsigned int index, CHAR * szDocString);

     STDMETHOD(SetVarDocString)(

       unsigned int index, CHAR * szDocString);

     STDMETHOD(SetFuncHelpContext)(
       unsigned int index, unsigned long dwHelpContext);

     STDMETHOD(SetVarHelpContext)(
       unsigned int index, unsigned long dwHelpContext);

     STDMETHOD(SetMops)(
       unsigned int index, BSTRA bstrMops);

     STDMETHOD(SetTypeIdldesc)(
       IDLDESC * pidldesc);

     STDMETHOD(LayOut)(VOID);
```

```
     inline CCreateTypeInfoA(LPUNKNOWN pUnk, ICreateTypeInfo * pWide) :
                 CAnsiInterface(ID_ICreateTypeInfo, pUnk, (LPUNKNOWN)pWide)
{};

     inline ICreateTypeInfo * GetWide() const
                 { return (ICreateTypeInfo *)m_pObj; };
};


//
+---------------------------------------------------------------------------
//
//  Class:      CEnumVARIANTA
//
//  Synopsis:   Class definition of IEnumVARIANTA
//
//----------------------------------------------------------------------------
-
class CEnumVARIANTA : CAnsiInterface
{
public:
     // *** IEnumVARIANTA methods ***
     STDMETHOD(Next)(
        unsigned long celt, VARIANTA * rgvar, unsigned long * pceltFetched);
     STDMETHOD(Skip)( unsigned long celt);
     STDMETHOD(Reset)(VOID);
     STDMETHOD(Clone)( IEnumVARIANTA * * ppenum);

     inline CEnumVARIANTA(LPUNKNOWN pUnk, IEnumVARIANT * pWide) :
                 CAnsiInterface(ID_IEnumVARIANT, pUnk, (LPUNKNOWN)pWide) {};

     inline IEnumVARIANT * GetWide() const
                 { return (IEnumVARIANT *)m_pObj; };
};


//
+---------------------------------------------------------------------------
//
//  Class:      CDispatchA
//
//  Synopsis:   Class definition of IDispatchA
//
//----------------------------------------------------------------------------
-
class CDispatchA : CAnsiInterface
{
public:
     // *** IDispatchA methods ***
     STDMETHOD(GetTypeInfoCount)(unsigned int * pctinfo);

     STDMETHOD(GetTypeInfo)(
       unsigned int itinfo,
       LCID lcid,
       ITypeInfoA * * pptinfo);

     STDMETHOD(GetIDsOfNames)(
       REFIID riid,
       char * * rgszNames,
       unsigned int cNames,
       LCID lcid,
       DISPID * rgdispid);

     STDMETHOD(Invoke)(
       DISPID dispidMember,
       REFIID riid,
       LCID lcid,
       unsigned short wFlags,
       DISPPARAMSA * pdispparamsA,
       VARIANTA * pvarResult,
       EXCEPINFOA * pexcepinfo,
       unsigned int * puArgErr);

     inline CDispatchA(LPUNKNOWN pUnk, IDispatch * pWide) :
                 CAnsiInterface(ID_IDispatch, pUnk, (LPUNKNOWN)pWide) {};

     inline IDispatch * GetWide() const
                 { return (IDispatch *)m_pObj; };
};


#ifndef NOERRORINFO
//
+---------------------------------------------------------------------------
//
//  Class:      CErrorInfoA
//
//  Synopsis:   Class definition of IErrorInfoA
//
//----------------------------------------------------------------------------
-
class CErrorInfoA : CAnsiInterface
{
public:
     // *** IErrorInfoA methods ***
     STDMETHOD(GetGUID)(GUID *pguid);
     STDMETHOD(GetSource)(BSTRA *pbstrSource);
     STDMETHOD(GetDescription)(BSTRA *pbstrDescription);
     STDMETHOD(GetHelpFile)(BSTRA *pbstrHelpFile);
     STDMETHOD(GetHelpContext)(DWORD *pdwHelpContext);

     inline CErrorInfoA(LPUNKNOWN pUnk, IErrorInfo * pObj) :
                 CAnsiInterface(ID_IErrorInfo, pUnk, (LPUNKNOWN)pObj) {};

     inline IErrorInfo * GetWide() const
                 { return (IErrorInfo *)m_pObj; };
};
```

```
//
+---------------------------------------------------------------------------
//
//  Class:      CCreateErrorInfoA
//
//  Synopsis:   Class definition of ICreateErrorInfoA
//
//----------------------------------------------------------------------------
-
class CCreateErrorInfoA : CAnsiInterface
{
public:
     // *** ICreateErrorInfoA methods ***
     STDMETHOD(SetGUID)(REFGUID rguid);
     STDMETHOD(SetSource)(LPSTR szSource);
     STDMETHOD(SetDescription)(LPSTR szDescription);
     STDMETHOD(SetHelpFile)(LPSTR szHelpFile);
     STDMETHOD(SetHelpContext)(DWORD dwHelpContext);


     inline CCreateErrorInfoA(LPUNKNOWN pUnk, ICreateErrorInfo * pObj) :
                 CAnsiInterface(ID_ICreateErrorInfo, pUnk, (LPUNKNOWN)pObj)
{};

     inline ICreateErrorInfo * GetWide() const
                 { return (ICreateErrorInfo *)m_pObj; };
};

#endif //!NOERRORINFO
```

ANSIDISP.CPP (OLEANSI Sample)

```
//
+---------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       ansidisp.cpp
//
//  Contents:   ANSI Wrappers for Dispatch Interfaces and APIs.
//
//  Classes:    CTypeLibA        - ANSI wrapper object for ITypeLib.
//              CTypeInfoA       - ANSI wrapper object for ITypeInfo.
//              CTypeCompA       - ANSI wrapper object for ITypeComp.
//              CCreateTypeLibA  - ANSI wrapper object for ICreateTypeLib.
//              CCreateTypeInfoA - ANSI wrapper object for ICreateTypeInfo.
//              CEnumVARIANTA    - ANSI wrapper object for IEnumVARIANT.
//              CDispatchA       - ANSI wrapper object for IDispatch.
//
//  History:    01-Nov-93   v-kentc     Created.
//              31-Mar-94   v-kentc     Result return fix ::AddRefTypeInfo.
//
//---------------------------------------------------------------------------
-

#include "Ole2Ansi.h"
```

ITypeLibA Implementation   (OLEANSI Sample)

```
//
***************************************************************************
//
//                   ITypeLibA Implementation
//
//
***************************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     CTypeLibA::GetTypeInfoCount, public
//
//  Synopsis:   Thunks GetTypeInfoCount to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP_(unsigned int) CTypeLibA::GetTypeInfoCount(VOID)
{
     TraceMethodEnter("CTypeLibA::GetTypeInfoCount", this);

     _DebugHook(GetWide(), MEMBER_PTR(ITypeLib, GetTypeInfoCount));

     return GetWide()->GetTypeInfoCount();
}


//
+--------------------------------------------------------------------------
//
//  Member:     CTypeLibA::GetTypeInfo, public
//
//  Synopsis:   Thunks GetTypeInfo to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CTypeLibA::GetTypeInfo(unsigned int index,
           ITypeInfoA * * pptinfo)
{
     TraceMethodEnter("CTypeLibA::GetTypeInfo", this);

     LPTYPEINFO pTypeInfo;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ITypeLib, GetTypeInfo));

     hResult = GetWide()->GetTypeInfo(index, &pTypeInfo);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapITypeInfoAFromW(pTypeInfo, pptinfo);

     pTypeInfo->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CTypeLibA::GetTypeInfoType, public
//
//  Synopsis:   Thunks GetTypeInfoType to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CTypeLibA::GetTypeInfoType(unsigned int index,
           TYPEKIND * ptypekind)
{
     TraceMethodEnter("CTypeLibA::GetTypeInfoType", this);

     _DebugHook(GetWide(), MEMBER_PTR(ITypeLib, GetTypeInfoType));

     return GetWide()->GetTypeInfoType(index, ptypekind);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CTypeLibA::GetTypeInfoOfGuid, public
//
//  Synopsis:   Thunks GetTypeInfoOfGuid to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CTypeLibA::GetTypeInfoOfGuid(
       REFGUID guid, ITypeInfoA * * pptinfo)
{
     TraceMethodEnter("CTypeLibA::GetTypeInfoOfGuid", this);

     LPTYPEINFO pTypeInfo;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ITypeLib, GetTypeInfoOfGuid));
     hResult = GetWide()->GetTypeInfoOfGuid(guid, &pTypeInfo);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapITypeInfoAFromW(pTypeInfo, pptinfo);

     pTypeInfo->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CTypeLibA::GetLibAttr, public
//
//  Synopsis:   Thunks GetLibAttr to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CTypeLibA::GetLibAttr(TLIBATTR * * pptlibattr)
{
     TraceMethodEnter("CTypeLibA::GetLibAttr", this);

     _DebugHook(GetWide(), MEMBER_PTR(ITypeLib, GetLibAttr));

     return GetWide()->GetLibAttr(pptlibattr);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CTypeLibA::GetTypeComp, public
//
//  Synopsis:   Thunks GetTypeComp to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CTypeLibA::GetTypeComp(ITypeCompA * * pptcomp)
{
     TraceMethodEnter("CTypeLibA::GetTypeComp", this);

     LPTYPECOMP pTypeComp;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ITypeLib, GetTypeComp));
     hResult = GetWide()->GetTypeComp(&pTypeComp);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapITypeCompAFromW(pTypeComp, pptcomp);

     pTypeComp->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CTypeLibA::GetDocumentation, public
//
//  Synopsis:   Thunks GetDocumentation to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CTypeLibA::GetDocumentation(int index, BSTRA * pbstrNameA,
           BSTRA * pbstrDocStringA, unsigned long * pdwHelpContext,
           BSTRA * pbstrHelpFileA)
{
     TraceMethodEnter("CTypeLibA::GetDocumentation", this);

     BSTR pstrName, *ppstrName;
     BSTR pstrDocString, *ppstrDocString;
     BSTR pstrHelpFile, *ppstrHelpFile;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ITypeLib, GetDocumentation));

     if (pbstrNameA)
           ppstrName = &pstrName;
     else
           ppstrName = NULL;

     if (pbstrDocStringA)
           ppstrDocString = &pstrDocString;
     else
           ppstrDocString = NULL;

     if (pbstrHelpFileA)
           ppstrHelpFile = &pstrHelpFile;
     else
           ppstrHelpFile = NULL;

     hResult = GetWide()->GetDocumentation(index, ppstrName, ppstrDocString,
                 pdwHelpContext, ppstrHelpFile);
     if (FAILED(hResult))
```

```
            return (hResult);

      if (ppstrName)
      {
            hResult = ConvertDispStringToA(pstrName, pbstrNameA);
            if (FAILED(hResult))
                  goto Error;

            SysFreeString(pstrName);
      }

      if (ppstrDocString)
      {
            hResult = ConvertDispStringToA(pstrDocString, pbstrDocStringA);
            if (FAILED(hResult))
                  goto Error1;

            SysFreeString(pstrDocString);
      }

      if (ppstrHelpFile)
      {
            hResult = ConvertDispStringToA(pstrHelpFile, pbstrHelpFileA);
            if (FAILED(hResult))
                  goto Error2;

            SysFreeString(pstrHelpFile);
      }

      return NOERROR;

Error2:
      if (pbstrDocStringA)
            ConvertDispStringFreeA(*pbstrDocStringA);

Error1:
      if (pbstrNameA)
            ConvertDispStringFreeA(*pbstrNameA);

Error:
      if (ppstrName)
            SysFreeString(pstrName);
      if (ppstrDocString)
            SysFreeString(pstrDocString);
      if (ppstrHelpFile)
            SysFreeString(pstrHelpFile);

      return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CTypeLibA::IsName, public
```

```
//
//  Synopsis:   Thunks IsName to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CTypeLibA::IsName(CHAR * szNameBufA, unsigned long lHashVal,
           int * lpfName)
{
     TraceMethodEnter("CTypeLibA::IsName", this);

     OLECHAR szNameBuf[MAX_STRING];
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ITypeLib, IsName));

     ConvertStringToW(szNameBufA, szNameBuf);

     hResult = GetWide()->IsName(szNameBuf, lHashVal, lpfName);

     // copy back result into original string (guaranteed to be same length)
     if (hResult == NOERROR && *lpfName != 0)
     {
           int Count = lstrlen(szNameBufA);

           WideCharToMultiByte(CP_ACP, 0, szNameBuf, Count, szNameBufA,
Count, NULL, NULL);
     }

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CTypeLibA::FindName, public
//
//  Synopsis:   Thunks FindName to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CTypeLibA::FindName(CHAR * szNameBufA, unsigned long lHashVal,
           ITypeInfoA * * rgptinfo, MEMBERID * rgmemid, unsigned short *
pcFound)
{
     TraceMethodEnter("CTypeLibA::FindName", this);

     OLECHAR       szNameBuf[MAX_STRING];
     LPTYPEINFO    pTypeInfo;
```

```
    LPTYPEINFOA   pTypeInfoA;
    HRESULT       hResult;


    _DebugHook(GetWide(), MEMBER_PTR(ITypeLib, FindName));

    ConvertStringToW(szNameBufA, szNameBuf);

    hResult = GetWide()->FindName(szNameBuf, lHashVal,
                    (LPTYPEINFO *)rgptinfo,
                    rgmemid,
                         pcFound);

    if (SUCCEEDED(hResult))
    {
         // now convert the array of ITypeInfo's to ITypeInfoA's in-place

         for (USHORT i = 0; i < *pcFound; i++)
         {
              pTypeInfo = (LPTYPEINFO)rgptinfo[i];
              if (SUCCEEDED(hResult))   // if we haven't failed already
              {
                   hResult =
WrapITypeInfoAFromW((LPTYPEINFO)rgptinfo[i], &pTypeInfoA);
                   if (FAILED(hResult))  // release any Ansi wrappers
we've made so far
                   {
                        for (USHORT j = 0; j < i; j++)
                        {
                             rgptinfo[j]->Release();
                        }
                   }
                   else
                   {
                        rgptinfo[i] = pTypeInfoA;
                   }
              }
              pTypeInfo->Release();
         }
    }

    return hResult;
}


//
+----------------------------------------------------------------------------
//
//  Member:     CTypeLibA::ReleaseTLibAttr, public
//
//  Synopsis:   Thunks ReleaseTLibAttr to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
```

```
//---------------------------------------------------------------------------
-
STDMETHODIMP_(void) CTypeLibA::ReleaseTLibAttr(TLIBATTR * ptlibattr)
{
     TraceMethodEnter("CTypeLibA::ReleaseTLibAttr", this);

     _DebugHook(GetWide(), MEMBER_PTR(ITypeLib, ReleaseTLibAttr));

     GetWide()->ReleaseTLibAttr(ptlibattr);
}
```

ITypeInfoA Implementation (OLEANSI Sample)

```
//
**********************************************************************
//
//                    ITypeInfoA Implementation
//
//
**********************************************************************

//
+---------------------------------------------------------------------
//
//  Member:     CTypeInfoA::GetTypeAttr, public
//
//  Synopsis:   Thunks GetTypeAttr to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------
-
STDMETHODIMP CTypeInfoA::GetTypeAttr(TYPEATTR * * pptypeattr)
{
     TraceMethodEnter("CTypeInfoA::GetTypeAttr", this);

     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, GetTypeAttr));

     return GetWide()->GetTypeAttr(pptypeattr);
}


//
+---------------------------------------------------------------------
//
//  Member:     CTypeInfoA::GetTypeComp, public
//
//  Synopsis:   Thunks GetTypeComp to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------
-
STDMETHODIMP CTypeInfoA::GetTypeComp(ITypeCompA * * pptcomp)
{
     TraceMethodEnter("CTypeInfoA::GetTypeComp", this);

     LPTYPECOMP pTypeComp;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, GetTypeComp));

     hResult = GetWide()->GetTypeComp(&pTypeComp);
     if (FAILED(hResult))
```

```
           return (hResult);

     hResult = WrapITypeCompAFromW(pTypeComp, pptcomp);

     pTypeComp->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::GetFuncDesc, public
//
//  Synopsis:   Thunks GetFuncDesc to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CTypeInfoA::GetFuncDesc(unsigned int index,
           FUNCDESC * * ppfuncdesc)
{
     TraceMethodEnter("CTypeInfoA::GetFuncDesc", this);

     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, GetFuncDesc));

     return GetWide()->GetFuncDesc(index, ppfuncdesc);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::GetVarDesc, public
//
//  Synopsis:   Thunks GetVarDesc to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CTypeInfoA::GetVarDesc(unsigned int index,
           VARDESCA * * ppvardescA)
{
     TraceMethodEnter("CTypeInfoA::GetVarDesc", this);

     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, GetVarDesc));

     hResult = GetWide()->GetVarDesc(index, (VARDESC * *)ppvardescA);
     if (FAILED(hResult))
```

```
            return (hResult);

      hResult = ConvertVARDESCToA(*ppvardescA);

      return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::GetNames, public
//
//  Synopsis:   Thunks GetNames to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CTypeInfoA::GetNames(MEMBERID memid, BSTRA * rgbstrNames,
            unsigned int cMaxNames, unsigned int * pcNames)
{
      TraceMethodEnter("CTypeInfoA::GetNames", this);

      HRESULT hResult;


      _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, GetNames));

      hResult = GetWide()->GetNames(memid, (BSTR *)rgbstrNames, cMaxNames,
                  pcNames);
      if (FAILED(hResult))
            return (hResult);

      return ConvertDispStringArrayToA(rgbstrNames, *pcNames);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::GetRefTypeOfImplType, public
//
//  Synopsis:   Thunks GetRefTypeOfImplType to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CTypeInfoA::GetRefTypeOfImplType(unsigned int index,
            HREFTYPE * phreftype)
{
      TraceMethodEnter("CTypeInfoA::GetRefTypeOfImplType", this);
```

```
     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, GetRefTypeOfImplType));

     return GetWide()->GetRefTypeOfImplType(index, phreftype);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::GetImplTypeFlags, public
//
//  Synopsis:   Thunks GetImplTypeFlags to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CTypeInfoA::GetImplTypeFlags(unsigned int index,
           int * pimpltypeflags)
{
     TraceMethodEnter("CTypeInfoA::GetImplTypeFlags", this);

     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, GetImplTypeFlags));

     return GetWide()->GetImplTypeFlags(index, pimpltypeflags);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::GetIDsOfNames, public
//
//  Synopsis:   Thunks GetIDsOfNames to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CTypeInfoA::GetIDsOfNames(CHAR * * rgszNamesA,
           unsigned int cNames, MEMBERID * rgmemid)
{
     TraceMethodEnter("CTypeInfoA::GetIDsOfNames", this);

     LPWSTR * rgszNames;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, GetIDsOfNames));

     hResult = ConvertStringArrayToW(rgszNamesA, &rgszNames, cNames);
     if (FAILED(hResult))
           return (hResult);

     hResult = GetWide()->GetIDsOfNames(rgszNames, cNames, rgmemid);
     ConvertStringArrayFree(rgszNames, cNames);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::Invoke, public
//
//  Synopsis:   Thunks Invoke to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CTypeInfoA::Invoke(void * pvInstance, MEMBERID memid,
           unsigned short wFlags, DISPPARAMSA *pdispparamsA,
           VARIANTA *pvarResultA, EXCEPINFOA *pexcepinfoA,
           unsigned int *puArgErr)
{
     TraceMethodEnter("CTypeInfoA::Invoke", this);

     LPDISPPARAMS pDispParams;
     LPVARIANT pByrefTemp;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, Invoke));

     hResult = ConvertDispParamsToW(pdispparamsA, &pDispParams,
&pByrefTemp);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->Invoke(pvInstance, memid, wFlags, pDispParams,
                 (LPVARIANT)pvarResultA, (EXCEPINFO *)pexcepinfoA,
puArgErr);

     if (hResult == DISP_E_EXCEPTION)
           ConvertExcepInfoToA(pexcepinfoA);

     if (FAILED(hResult))
           goto Error;

     if (pByrefTemp)
     {
           hResult = ConvertDispParamsCopyBack(pdispparamsA, pByrefTemp);
           if (FAILED(hResult))
                 goto Error;
     }

     hResult = ConvertVariantToA(pvarResultA);
     if (FAILED(hResult))
           goto Error;

Error:
     ConvertDispParamsFree(pDispParams, pByrefTemp);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::GetDocumentation, public
//
//  Synopsis:   Thunks GetDocumentation to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CTypeInfoA::GetDocumentation(MEMBERID memid, BSTRA *
pbstrNameA,
           BSTRA * pbstrDocStringA, unsigned long * pdwHelpContext,
           BSTRA * pbstrHelpFileA)
{
     TraceMethodEnter("CTypeInfoA::GetDocumentation", this);

     BSTR pstrName, *ppstrName;
     BSTR pstrDocString, *ppstrDocString;
     BSTR pstrHelpFile, *ppstrHelpFile;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, GetDocumentation));

     if (pbstrNameA)
           ppstrName = &pstrName;
     else
           ppstrName = NULL;

     if (pbstrDocStringA)
           ppstrDocString = &pstrDocString;
     else
           ppstrDocString = NULL;

     if (pbstrHelpFileA)
           ppstrHelpFile = &pstrHelpFile;
     else
           ppstrHelpFile = NULL;

     hResult = GetWide()->GetDocumentation(memid, ppstrName, ppstrDocString,
                 pdwHelpContext, ppstrHelpFile);
     if (FAILED(hResult))
```

```
            return (hResult);

      if (ppstrName)
      {
            hResult = ConvertDispStringToA(pstrName, pbstrNameA);
            if (FAILED(hResult))
                  goto Error;

            SysFreeString(pstrName);
      }

      if (ppstrDocString)
      {
            hResult = ConvertDispStringToA(pstrDocString, pbstrDocStringA);
            if (FAILED(hResult))
                  goto Error1;

            SysFreeString(pstrDocString);
      }

      if (ppstrHelpFile)
      {
            hResult = ConvertDispStringToA(pstrHelpFile, pbstrHelpFileA);
            if (FAILED(hResult))
                  goto Error2;

            SysFreeString(pstrHelpFile);
      }

      return NOERROR;

Error2:
      if (pbstrDocStringA)
            ConvertDispStringFreeA(*pbstrDocStringA);

Error1:
      if (pbstrNameA)
            ConvertDispStringFreeA(*pbstrNameA);

Error:
      if (ppstrName)
            SysFreeString(pstrName);
      if (ppstrDocString)
            SysFreeString(pstrDocString);
      if (ppstrHelpFile)
            SysFreeString(pstrHelpFile);

      return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::GetDllEntry, public
```

```
//
//  Synopsis:   Thunks GetDllEntry to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CTypeInfoA::GetDllEntry(MEMBERID memid, INVOKEKIND invkind,
           BSTRA * pbstrDllNameA, BSTRA * pbstrNameA,
           unsigned short * pwOrdinal)
{
     TraceMethodEnter("CTypeInfoA::GetDllEntry", this);

     BSTR pstrDllName, *ppstrDllName;
     BSTR pstrName, *ppstrName;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, GetDllEntry));

     if (pbstrDllNameA)
           ppstrDllName = &pstrDllName;
     else
           ppstrDllName = NULL;

     if (pbstrNameA)
           ppstrName = &pstrName;
     else
           ppstrName = NULL;

     hResult = GetWide()->GetDllEntry(memid, invkind, ppstrDllName,
ppstrName,
                 pwOrdinal);
     if (FAILED(hResult))
           return (hResult);

     if (pbstrDllNameA)
     {
           hResult = ConvertDispStringToA(pstrDllName, pbstrDllNameA);
           if (FAILED(hResult))
                 goto Error;

           SysFreeString(pstrDllName);
     }

     if (pbstrNameA)
     {
           hResult = ConvertDispStringToA(pstrName, pbstrNameA);
           if (FAILED(hResult))
                 goto Error1;

           SysFreeString(pstrName);
     }
     return NOERROR;

Error1:
     if (pbstrNameA)
           ConvertDispStringFreeA(*pbstrNameA);

Error:
     SysFreeString(pstrDllName);
     SysFreeString(pstrName);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::GetRefTypeInfo, public
//
//  Synopsis:   Thunks GetRefTypeInfo to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CTypeInfoA::GetRefTypeInfo(HREFTYPE hreftype,
           ITypeInfoA * * pptinfo)
{
     TraceMethodEnter("CTypeInfoA::GetRefTypeInfo", this);

     LPTYPEINFO pTypeInfo;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, GetRefTypeInfo));

     hResult = GetWide()->GetRefTypeInfo(hreftype, &pTypeInfo);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapITypeInfoAFromW(pTypeInfo, pptinfo);

     pTypeInfo->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::AddressOfMember, public
//
//  Synopsis:   Thunks AddressOfMember to Unicode method.
//
```

```
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CTypeInfoA::AddressOfMember(MEMBERID memid, INVOKEKIND invkind,
           void * * ppv)
{
     TraceMethodEnter("CTypeInfoA::AddressOfMember", this);

     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, AddressOfMember));

     return GetWide()->AddressOfMember(memid, invkind, ppv);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::CreateInstance, public
//
//  Synopsis:   Thunks CreateInstance to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CTypeInfoA::CreateInstance(IUnknown * punkOuter, REFIID riid,
           void * * ppvObj)
{
     HRESULT hResult;
     IUnknown * pvObjW;

     TraceMethodEnter("CTypeInfoA::CreateInstance", this);

     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, CreateInstance));

     hResult = GetWide()->CreateInstance(punkOuter, riid, (void **)&pvObjW);
     if (SUCCEEDED(hResult))
     {
           hResult = WrapInterfaceAFromW(WrapTranslateIID(riid), pvObjW,
(IUnknown**)ppvObj);
           pvObjW->Release();
     }
     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::GetMops, public
//
//  Synopsis:   Thunks GetMops to Unicode method.
//
//  Returns:    OLE2 Result Code.
```

```
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CTypeInfoA::GetMops(MEMBERID memid, BSTRA * pbstrMopsA)
{
     TraceMethodEnter("CTypeInfoA::GetMops", this);

     BSTR bstrMops;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, GetMops));

     hResult = GetWide()->GetMops(memid, &bstrMops);
     if (FAILED(hResult))
           return hResult;

     hResult = ConvertDispStringToA(bstrMops, pbstrMopsA);

     ConvertDispStringFreeW(bstrMops);

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::GetContainingTypeLib, public
//
//  Synopsis:   Thunks GetContainingTypeLib to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CTypeInfoA::GetContainingTypeLib(ITypeLibA * * pptlib,
           unsigned int * pindex)
{
     TraceMethodEnter("CTypeInfoA::GetContainingTypeLib", this);

     LPTYPELIB pTypeLib;
     LPTYPELIB *ppTypeLib;
     HRESULT hResult;

     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, GetContainingTypeLib));

     if (pptlib)
           ppTypeLib = &pTypeLib;
     else
           ppTypeLib = NULL;

     hResult = GetWide()->GetContainingTypeLib(ppTypeLib, pindex);
     if (FAILED(hResult))
```

```
            return (hResult);

     if (pptlib)
     {
            hResult = WrapITypeLibAFromW(pTypeLib, pptlib);
            pTypeLib->Release();
     }

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::ReleaseTypeAttr, public
//
//  Synopsis:   Thunks ReleaseTypeAttr to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP_(void) CTypeInfoA::ReleaseTypeAttr(TYPEATTR * ptypeattr)
{
     TraceMethodEnter("CTypeInfoA::ReleaseTypeAttr", this);

     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, ReleaseTypeAttr));

     GetWide()->ReleaseTypeAttr(ptypeattr);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::ReleaseFuncDesc, public
//
//  Synopsis:   Thunks ReleaseFuncDesc to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP_(void) CTypeInfoA::ReleaseFuncDesc(FUNCDESC * pfuncdesc)
{
     TraceMethodEnter("CTypeInfoA::ReleaseFuncDesc", this);

     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, ReleaseFuncDesc));

     GetWide()->ReleaseFuncDesc(pfuncdesc);
}
```

```
//
+---------------------------------------------------------------------------
//
//  Member:     CTypeInfoA::ReleaseVarDesc, public
//
//  Synopsis:   Thunks ReleaseVarDesc to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP_(void) CTypeInfoA::ReleaseVarDesc(VARDESCA * pvardesc)
{
     TraceMethodEnter("CTypeInfoA::ReleaseVarDesc", this);

     _DebugHook(GetWide(), MEMBER_PTR(ITypeInfo, ReleaseVarDesc));

     GetWide()->ReleaseVarDesc((VARDESC *)pvardesc);
}
```

ITypeCompA Implementation   (OLEANSI Sample)

```
//
***************************************************************************
//
//                   ITypeCompA Implementation
//
//
***************************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     CTypeCompA::Bind, public
//
//  Synopsis:   Thunks Bind to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CTypeCompA::Bind(CHAR * szNameA, unsigned long lHashVal,
           unsigned short wflags, ITypeInfoA * * pptinfo, DESCKIND *
pdesckind,
           BINDPTRA * pbindptr)
{
     TraceMethodEnter("CTypeCompA::Bind", this);

     OLECHAR    szName[MAX_STRING];
     LPTYPEINFO pTypeInfo;
     HRESULT    hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ITypeComp, Bind));

     ConvertStringToW(szNameA, szName);

     hResult = GetWide()->Bind(szName, lHashVal, wflags, &pTypeInfo,
                 pdesckind, (BINDPTR *)pbindptr);

     // NOTE: if *pdesckind == DESKIND_NONE, then *pTypeInfo and *pbindptr
     // are indeterminate.
     if (FAILED(hResult) || *pdesckind == DESCKIND_NONE)
           return hResult;

     if (pTypeInfo == NULL)
     {
           *pptinfo = NULL;
     }
     else
     {
           hResult = WrapITypeInfoAFromW(pTypeInfo, pptinfo);
           if (FAILED(hResult))
                 goto Error;
```

```
	}

	// convert *pbindptr from BINDPTR to BINDPTRA
	switch (*pdesckind)
	{
	case DESCKIND_VARDESC:
	case DESCKIND_IMPLICITAPPOBJ:
		hResult = ConvertVARDESCToA(pbindptr->lpvardesc);
		break;

	case DESCKIND_TYPECOMP:
		{
		LPTYPECOMP pTypeComp;
		pTypeComp = (LPTYPECOMP)pbindptr->lptcomp;
		hResult = WrapITypeCompAFromW(pTypeComp, &pbindptr->lptcomp);
		pTypeComp->Release();
		break;
		}

	//case DESCKIND_NONE:
	//case DESCKIND_FUNCDESC:
	// nothing to do
	}

Error:
	if (pTypeInfo)
		pTypeInfo->Release();

	return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CTypeCompA::BindType, public
//
//  Synopsis:   Thunks BindType to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CTypeCompA::BindType(CHAR * szNameA, unsigned long lHashVal,
		ITypeInfoA * * pptinfoA, ITypeCompA * * pptcompA)
{
	TraceMethodEnter("CTypeCompA::BindType", this);

	OLECHAR    szName[MAX_STRING];
	LPTYPEINFO pTypeInfo;
	LPTYPECOMP pTypeComp;
	HRESULT    hResult;


	_DebugHook(GetWide(), MEMBER_PTR(ITypeComp, BindType));
```

```
    ConvertStringToW(szNameA, szName);

    hResult = GetWide()->BindType(szName, lHashVal,  &pTypeInfo,
&pTypeComp);
    if (FAILED(hResult))
        return hResult;

    if (pTypeInfo == NULL)
    {
        *pptinfoA = NULL;
    }
    else
    {
        hResult = WrapITypeInfoAFromW(pTypeInfo, pptinfoA);
        if (FAILED(hResult))
            goto Error;
    }

    if (pTypeComp == NULL)
    {
        *pptcompA = NULL;
    }
    else
    {
        hResult = WrapITypeCompAFromW(pTypeComp, pptcompA);
        pTypeComp->Release();
    }

Error:
    if (pTypeInfo)
        pTypeInfo->Release();

    return hResult;
}
```

ICreateTypeLibA Implementation (OLEANSI Sample)

```
//
**********************************************************************
//
//                    ICreateTypeLibA Implementation
//
//
**********************************************************************
//
+---------------------------------------------------------------------
//
//  Member:     CCreateTypeLibA::CreateTypeInfo, public
//
//  Synopsis:   Thunks CreateTypeInfo to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeLibA::CreateTypeInfo(CHAR * szNameA, TYPEKIND tkind,
           ICreateTypeInfoA * * lplpictinfo)
{
     TraceMethodEnter("CCreateTypeLibA::CreateTypeInfo", this);

     OLECHAR    szName[MAX_STRING];
     LPCREATETYPEINFO pTypeInfo;
     HRESULT    hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeLib, CreateTypeInfo));

     ConvertStringToW(szNameA, szName);

     hResult = GetWide()->CreateTypeInfo(szName, tkind,  &pTypeInfo);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapICreateTypeInfoAFromW(pTypeInfo, lplpictinfo);

     if (pTypeInfo)
           pTypeInfo->Release();

     return hResult;
}


//
+---------------------------------------------------------------------
//
//  Member:     CCreateTypeLibA::SetName, public
//
//  Synopsis:   Thunks SetName to Unicode method.
//
```

```
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeLibA::SetName(CHAR * szNameA)
{
     TraceMethodEnter("CCreateTypeLibA::SetName", this);

     OLECHAR szName[MAX_STRING];


     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeLib, SetName));

     ConvertStringToW(szNameA, szName);

     return GetWide()->SetName(szName);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CCreateTypeLibA::SetVersion, public
//
//  Synopsis:   Thunks SetVersion to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeLibA::SetVersion(unsigned short wMajorVerNum,
           unsigned short wMinorVerNum)
{
     TraceMethodEnter("CCreateTypeLibA::SetVersion", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeLib, SetVersion));

     return GetWide()->SetVersion(wMajorVerNum, wMinorVerNum);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CCreateTypeLibA::SetGuid, public
//
//  Synopsis:   Thunks SetGuid to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeLibA::SetGuid(REFGUID guid)
{
     TraceMethodEnter("CCreateTypeLibA::SetGuid", this);
     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeLib, SetGuid));

     return GetWide()->SetGuid(guid);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeLibA::SetDocString, public
//
//  Synopsis:   Thunks SetDocString to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeLibA::SetDocString(CHAR * szDocA)
{
     TraceMethodEnter("CCreateTypeLibA::SetDocString", this);

     LPWSTR     lpszDoc;
     HRESULT    hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeLib, SetDocString));

     hResult = ConvertStringToW(szDocA, &lpszDoc);
     if (FAILED(hResult))
           return (hResult);

     hResult = GetWide()->SetDocString(lpszDoc);

     ConvertStringFree(lpszDoc);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeLibA::SetHelpFileName, public
//
//  Synopsis:   Thunks SetHelpFileName to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeLibA::SetHelpFileName(CHAR * szHelpFileNameA)
{
     TraceMethodEnter("CCreateTypeLibA::SetHelpFileName", this);
     OLECHAR szHelpFileName[MAX_STRING];


     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeLib, SetHelpFileName));

     ConvertStringToW(szHelpFileNameA, szHelpFileName);

     return GetWide()->SetHelpFileName(szHelpFileName);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeLibA::SetHelpContext, public
//
//  Synopsis:   Thunks SetHelpContext to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeLibA::SetHelpContext(unsigned long dwHelpContext)
{
     TraceMethodEnter("CCreateTypeLibA::SetHelpContext", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeLib, SetHelpContext));

     return GetWide()->SetHelpContext(dwHelpContext);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeLibA::SetLcid, public
//
//  Synopsis:   Thunks SetLcid to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeLibA::SetLcid(LCID lcid)
{
     TraceMethodEnter("CCreateTypeLibA::SetLcid", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeLib, SetLcid));

     return GetWide()->SetLcid(lcid);
}
```

```
//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeLibA::SetLibFlags, public
//
//  Synopsis:   Thunks SetLibFlags to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeLibA::SetLibFlags(unsigned int uLibFlags)
{
     TraceMethodEnter("CCreateTypeLibA::SetLibFlags", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeLib, SetLibFlags));

     return GetWide()->SetLibFlags(uLibFlags);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeLibA::SaveAllChanges, public
//
//  Synopsis:   Thunks SaveAllChanges to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeLibA::SaveAllChanges(VOID)
{
     TraceMethodEnter("CCreateTypeLibA::SaveAllChanges", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeLib, SaveAllChanges));

     return GetWide()->SaveAllChanges();
}
```

ICreateTypeInfoA Implementation (OLEANSI Sample)

```
//
**********************************************************************
//
//                    ICreateTypeInfoA Implementation
//
//
**********************************************************************

//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::SetGuid, public
//
//  Synopsis:   Thunks SetGuid to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::SetGuid(REFGUID guid)
{
     TraceMethodEnter("CCreateTypeInfoA::SetGuid", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, SetGuid));

     return GetWide()->SetGuid(guid);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::SetTypeFlags, public
//
//  Synopsis:   Thunks SetTypeFlags to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::SetTypeFlags(unsigned int uTypeFlags)
{
     TraceMethodEnter("CCreateTypeInfoA::SetTypeFlags", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, SetTypeFlags));

     return GetWide()->SetTypeFlags(uTypeFlags);
}


//
+---------------------------------------------------------------------------
```

```
//
//  Member:     CCreateTypeInfoA::SetDocString, public
//
//  Synopsis:   Thunks SetDocString to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::SetDocString(CHAR * pstrDocA)
{
     TraceMethodEnter("CCreateTypeInfoA::SetDocString", this);

     LPWSTR   lpszDoc;
     HRESULT    hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, SetDocString));

     hResult = ConvertStringToW(pstrDocA, &lpszDoc);
     if (FAILED(hResult))
           return (hResult);

     hResult = GetWide()->SetDocString(lpszDoc);

     ConvertStringFree(lpszDoc);

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::SetHelpContext, public
//
//  Synopsis:   Thunks SetHelpContext to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::SetHelpContext(unsigned long dwHelpContext)
{
     TraceMethodEnter("CCreateTypeInfoA::SetHelpContext", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, SetHelpContext));

     return GetWide()->SetHelpContext(dwHelpContext);
}


//
+--------------------------------------------------------------------------
//
```

```
//  Member:     CCreateTypeInfoA::SetVersion, public
//
//  Synopsis:   Thunks SetVersion to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//-------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::SetVersion(unsigned short wMajorVerNum,
           unsigned short wMinorVerNum)
{
     TraceMethodEnter("CCreateTypeInfoA::SetVersion", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, SetVersion));

     return GetWide()->SetVersion(wMajorVerNum, wMinorVerNum);
}


//
+------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::AddRefTypeInfo, public
//
//  Synopsis:   Thunks AddRefTypeInfo to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//-------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::AddRefTypeInfo(ITypeInfoA * ptinfo,
           HREFTYPE * phreftype)
{
     TraceMethodEnter("CCreateTypeInfoA::AddRefTypeInfo", this);

     ITypeInfo * pTypeInfo;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, AddRefTypeInfo));

     // ptinfo->QueryInterface(IID_ITypeInfo, (PVOID *)&pTypeInfo);
     if( FAILED( hResult = WrapITypeInfoWFromA( ptinfo, &pTypeInfo )))
       return hResult;

     hResult = GetWide()->AddRefTypeInfo(pTypeInfo, phreftype);

     pTypeInfo->Release();

     return hResult;
}
```

```
//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::AddFuncDesc, public
//
//  Synopsis:   Thunks AddFuncDesc to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::AddFuncDesc(unsigned int index,
           FUNCDESC * pfuncdesc)
{
     TraceMethodEnter("CCreateTypeInfoA::AddFuncDesc", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, AddFuncDesc));

     return GetWide()->AddFuncDesc(index, pfuncdesc);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::AddImplType, public
//
//  Synopsis:   Thunks AddImplType to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::AddImplType(unsigned int index,
           HREFTYPE hreftype)
{
     TraceMethodEnter("CCreateTypeInfoA::AddImplType", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, AddImplType));

     return GetWide()->AddImplType(index, hreftype);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::SetImplTypeFlags, public
//
//  Synopsis:   Thunks SetImplTypeFlags to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
```

```
STDMETHODIMP CCreateTypeInfoA::SetImplTypeFlags(unsigned int index,
           int impltypeflags)
{
     TraceMethodEnter("CCreateTypeInfoA::SetImplTypeFlags", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, SetImplTypeFlags));

     return GetWide()->SetImplTypeFlags(index, impltypeflags);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::SetAlignment, public
//
//  Synopsis:   Thunks SetAlignment to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::SetAlignment(unsigned short cbAlignment)
{
     TraceMethodEnter("CCreateTypeInfoA::SetAlignment", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, SetAlignment));

     return GetWide()->SetAlignment(cbAlignment);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::SetSchema, public
//
//  Synopsis:   Thunks SetSchema to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::SetSchema(CHAR * lpstrSchemaA)
{
     TraceMethodEnter("CCreateTypeInfoA::SetSchema", this);

     LPWSTR   lpszSchema;

     HRESULT    hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, SetSchema));

     hResult = ConvertStringToW(lpstrSchemaA, &lpszSchema);
     if (FAILED(hResult))
           return (hResult);

     hResult = GetWide()->SetSchema(lpszSchema);

     ConvertStringFree(lpszSchema);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::AddVarDesc, public
//
//  Synopsis:   Thunks AddVarDesc to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::AddVarDesc(unsigned int index,
           VARDESCA * pvardescA)
{
     TraceMethodEnter("CCreateTypeInfoA::AddVarDesc", this);

     LPVARDESC pVarDesc;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, AddVarDesc));

     hResult = ConvertVARDESCToW(pvardescA, &pVarDesc);
     if (FAILED(hResult))
           return (hResult);

     hResult = GetWide()->AddVarDesc(index, pVarDesc);

     ConvertVARDESCFree(pVarDesc);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::SetFuncAndParamNames, public
//
//  Synopsis:   Thunks SetFuncAndParamNames to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
```

```
//--------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::SetFuncAndParamNames(unsigned int index,
           CHAR * * rgszNamesA, unsigned int cNames)
{
     TraceMethodEnter("CCreateTypeInfoA::SetFuncAndParamNames", this);

     LPWSTR * rgszNames;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo,
SetFuncAndParamNames));

     hResult = ConvertStringArrayToW(rgszNamesA, &rgszNames, cNames);
     if (FAILED(hResult))
           return (hResult);

     hResult = GetWide()->SetFuncAndParamNames(index, rgszNames, cNames);

     ConvertStringArrayFree(rgszNames, cNames);

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::SetVarName, public
//
//  Synopsis:   Thunks SetVarName to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::SetVarName(unsigned int index, CHAR *
szNameA)
{
     TraceMethodEnter("CCreateTypeInfoA::SetVarName", this);

     OLECHAR szName[MAX_STRING];


     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, SetVarName));

     ConvertStringToW(szNameA, szName);

     return GetWide()->SetVarName(index, szName);
}


//
+--------------------------------------------------------------------------
```

```
//
//  Member:     CCreateTypeInfoA::SetTypeDescAlias, public
//
//  Synopsis:   Thunks SetTypeDescAlias to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::SetTypeDescAlias(TYPEDESC * ptdescAlias)
{
     TraceMethodEnter("CCreateTypeInfoA::SetTypeDescAlias", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, SetTypeDescAlias));

     return GetWide()->SetTypeDescAlias(ptdescAlias);
}



//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::DefineFuncAsDllEntry, public
//
//  Synopsis:   Thunks DefineFuncAsDllEntry to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::DefineFuncAsDllEntry(unsigned int index,
           CHAR * szDllNameA, CHAR * szProcNameA)
{
     TraceMethodEnter("CCreateTypeInfoA::DefineFuncAsDllEntry", this);

     OLECHAR szDllName[MAX_STRING];
     LPWSTR  lpszProcName;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo,
DefineFuncAsDllEntry));

     ConvertStringToW(szDllNameA, szDllName);

     if (HIWORD(szProcNameA))
     {
           // not an ordinal
           hResult = ConvertStringToW(szProcNameA, &lpszProcName);
           if (FAILED(hResult))
                 goto Error;
     }
     else
     {
           // really an ordinal
           lpszProcName = (LPWSTR)szProcNameA;
     }

     hResult = GetWide()->DefineFuncAsDllEntry(index, szDllName,
lpszProcName);

     if (HIWORD(szProcNameA))
     {
           // not an ordinal
           ConvertStringFree(lpszProcName);
     }

Error:
     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::SetFuncDocString, public
//
//  Synopsis:   Thunks SetFuncDocString to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::SetFuncDocString(unsigned int index,
           CHAR * szDocStringA)
{
     TraceMethodEnter("CCreateTypeInfoA::SetFuncDocString", this);

     LPWSTR   lpszDocString;
     HRESULT    hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, SetFuncDocString));

     hResult = ConvertStringToW(szDocStringA, &lpszDocString);
     if (FAILED(hResult))
           return (hResult);

     hResult = GetWide()->SetFuncDocString(index, lpszDocString);

     ConvertStringFree(lpszDocString);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
```

```
//  Member:     CCreateTypeInfoA::SetVarDocString, public
//
//  Synopsis:   Thunks SetVarDocString to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::SetVarDocString(unsigned int index,
           CHAR * szDocStringA)
{
     TraceMethodEnter("CCreateTypeInfoA::SetVarDocString", this);

     LPWSTR   lpszDocString;
     HRESULT    hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, SetVarDocString));

     hResult = ConvertStringToW(szDocStringA, &lpszDocString);
     if (FAILED(hResult))
           return (hResult);

     hResult = GetWide()->SetVarDocString(index, lpszDocString);

     ConvertStringFree(lpszDocString);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::SetFuncHelpContext, public
//
//  Synopsis:   Thunks SetFuncHelpContext to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::SetFuncHelpContext(unsigned int index,
           unsigned long dwHelpContext)
{
     TraceMethodEnter("CCreateTypeInfoA::SetFuncHelpContext", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, SetFuncHelpContext));

     return GetWide()->SetFuncHelpContext(index, dwHelpContext);
}
```

```
//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::SetVarHelpContext, public
//
//  Synopsis:   Thunks SetVarHelpContext to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::SetVarHelpContext(unsigned int index,
           unsigned long dwHelpContext)
{
     TraceMethodEnter("CCreateTypeInfoA::SetVarHelpContext", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, SetVarHelpContext));

     return GetWide()->SetVarHelpContext(index, dwHelpContext);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::SetMops, public
//
//  Synopsis:   Thunks SetMops to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::SetMops(unsigned int index, BSTRA bstrMopsA)
{
     TraceMethodEnter("CCreateTypeInfoA::SetMops", this);

     BSTR bstrMops;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, SetMops));

     hResult = ConvertStringToW(bstrMopsA, &bstrMops);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->SetMops(index, bstrMops);

     ConvertStringFree(bstrMops);

     return hResult;

}
```

```
//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::SetTypeIdldesc, public
//
//  Synopsis:   Thunks SetTypeIdldesc to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::SetTypeIdldesc(IDLDESC * pidldesc)
{
     TraceMethodEnter("CCreateTypeInfoA::SetTypeIdldesc", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, SetTypeIdldesc));

     return GetWide()->SetTypeIdldesc(pidldesc);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CCreateTypeInfoA::LayOut, public
//
//  Synopsis:   Thunks LayOut to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CCreateTypeInfoA::LayOut(VOID)
{
     TraceMethodEnter("CCreateTypeInfoA::LayOut", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateTypeInfo, LayOut));

     return GetWide()->LayOut();
}
```

IEnumVARIANTA Implementation (OLEANSI Sample)

```
//
***************************************************************************
//
//                   IEnumVARIANTA Implementation
//
//
***************************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     CEnumVARIANTA::Next, public
//
//  Synopsis:   Thunks Next to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CEnumVARIANTA::Next(unsigned long celt, VARIANTA * rgvar,
           unsigned long * pceltFetched)
{
     TraceMethodEnter("CEnumVARIANTA::Next", this);

     ULONG   celtFetched;
     HRESULT hReturn;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IEnumVARIANT, Next));

     if (pceltFetched == NULL)
           pceltFetched = &celtFetched;

     hReturn = GetWide()->Next(celt, (VARIANT *)rgvar, pceltFetched);
     if (FAILED(hReturn))
           return (hReturn);

     hResult = ConvertVariantArrayToA(rgvar, *pceltFetched);
     if (FAILED(hResult))
           return (hResult);

     return hReturn;
}



//
+--------------------------------------------------------------------------
//
//  Member:     CEnumVARIANTA::Skip, public
//
```

```
//  Synopsis:   Thunks Skip to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CEnumVARIANTA::Skip(unsigned long celt)
{
     TraceMethodEnter("CEnumVARIANTA::Skip", this);

     _DebugHook(GetWide(), MEMBER_PTR(IEnumVARIANT, Skip));

     return GetWide()->Skip(celt);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CEnumVARIANTA::Reset, public
//
//  Synopsis:   Thunks Reset to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CEnumVARIANTA::Reset(VOID)
{
     TraceMethodEnter("CEnumVARIANTA::Reset", this);

     _DebugHook(GetWide(), MEMBER_PTR(IEnumVARIANT, Reset));

     return GetWide()->Reset();
}


//
+---------------------------------------------------------------------------
//
//  Member:     CEnumVARIANTA::Clone, public
//
//  Synopsis:   Thunks Clone to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CEnumVARIANTA::Clone(IEnumVARIANTA * * ppenmA)
{
     TraceMethodEnter("CEnumVARIANTA::Clone", this);

     IEnumVARIANT * penm;
```

```
    _DebugHook(GetWide(), MEMBER_PTR(IEnumVARIANT, Clone));

    HRESULT hResult = GetWide()->Clone(&penm);
    if (FAILED(hResult))
          return (hResult);

    hResult = WrapIEnumVARIANTAFromW(penm, ppenmA);

    penm->Release();

    return hResult;
}
```

IDispatchA Implementation (OLEANSI Sample)

```
//
***************************************************************************
//
//                   IDispatchA Implementation
//
//
***************************************************************************

//
+---------------------------------------------------------------------------
//
//  Member:     CDispatchA::GetTypeInfoCount, public
//
//  Synopsis:   Thunks GetTypeInfoCount to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CDispatchA::GetTypeInfoCount(unsigned int * pctinfo)
{
     TraceMethodEnter("CDispatchA::GetTypeInfoCount", this);

     _DebugHook(GetWide(), MEMBER_PTR(IDispatch, GetTypeInfoCount));

     return GetWide()->GetTypeInfoCount(pctinfo);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CDispatchA::GetTypeInfo, public
//
//  Synopsis:   Thunks GetTypeInfo to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CDispatchA::GetTypeInfo(unsigned int itinfo, LCID lcid,
           ITypeInfoA * * pptinfo)
{
     TraceMethodEnter("CDispatchA::GetTypeInfo", this);

     LPTYPEINFO pTypeInfo;
     HRESULT    hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IDispatch, GetTypeInfo));

     hResult = GetWide()->GetTypeInfo(itinfo, lcid, &pTypeInfo);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapITypeInfoAFromW(pTypeInfo, pptinfo);

     pTypeInfo->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CDispatchA::GetIDsOfNames, public
//
//  Synopsis:   Thunks GetIDsOfNames to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CDispatchA::GetIDsOfNames(REFIID riid, char * * rgszNamesA,
           unsigned int cNames, LCID lcid, DISPID * rgdispid)
{
     TraceMethodEnter("CDispatchA::GetIDsOfNames", this);

     LPWSTR * rgszNames;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IDispatch, GetIDsOfNames));

     hResult = ConvertStringArrayToW(rgszNamesA, lcid, &rgszNames, cNames);
     if (FAILED(hResult))
           return (hResult);

     hResult = GetWide()->GetIDsOfNames(riid, rgszNames, cNames, lcid,
rgdispid);

     ConvertStringArrayFree(rgszNames, cNames);

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CDispatchA::Invoke, public
//
//  Synopsis:   Thunks Invoke to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
```

```
//  History:    24-Jun-94   patl    Fixed case where GetWide->Invoke
//                  returns an error, and an exception.
//                  Previously, the exception would be
//                  left wide, not converted to ANSI.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CDispatchA::Invoke(DISPID dispidMember, REFIID riid, LCID lcid,
           unsigned short wFlags, DISPPARAMSA * pdispparamsA,
           VARIANTA * pvarResultA, EXCEPINFOA * pexcepinfoA,
           unsigned int * puArgErr)
{
     TraceMethodEnter("CDispatchA::Invoke", this);

     LPDISPPARAMS pDispParams;
     LPVARIANT    pByrefTemp;
     HRESULT      hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IDispatch, Invoke));

     hResult = ConvertDispParamsToW(pdispparamsA, &pDispParams,
&pByrefTemp);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->Invoke(dispidMember, riid, lcid, wFlags,
pDispParams,
                 (LPVARIANT)pvarResultA, (EXCEPINFO *)pexcepinfoA,
puArgErr);

     if (hResult == DISP_E_EXCEPTION)
           ConvertExcepInfoToA(pexcepinfoA);

     if (FAILED(hResult))
           goto Error;

     if (pByrefTemp)
     {
           hResult = ConvertDispParamsCopyBack(pdispparamsA, pByrefTemp);
           if (FAILED(hResult))
                 goto Error;
     }

     hResult = ConvertVariantToA(pvarResultA);
     if (FAILED(hResult))
           goto Error;

Error:

     ConvertDispParamsFree(pDispParams, pByrefTemp);

     return hResult;
}
```

IErrorInfoA Implementation   (OLEANSI Sample)#ifndef NOERRORINFO

```
//
***************************************************************************
//
//                       IErrorInfoA Implementation
//
//
***************************************************************************

STDMETHODIMP
CErrorInfoA::GetGUID(GUID *pguid)
{
     TraceMethodEnter("CErrorInfoA::GetGUID", this);

     _DebugHook(GetWide(), MEMBER_PTR(IErrorInfo, GetGUID));

     return GetWide()->GetGUID(pguid);
}

STDMETHODIMP
CErrorInfoA::GetSource(BSTRA *pbstraSource)
{
     TraceMethodEnter("CErrorInfoA::GetSource", this);

     HRESULT hresult;
     BSTR bstrSource;

     bstrSource = NULL;

     _DebugHook(GetWide(), MEMBER_PTR(IErrorInfo, GetSource));

     hresult = GetWide()->GetSource(&bstrSource);
     if(FAILED(hresult))
     return hresult;

     hresult = ConvertDispStringToA(bstrSource, pbstraSource);

     SysFreeString(bstrSource);

     return hresult;
}

STDMETHODIMP
CErrorInfoA::GetDescription(BSTRA *pbstraDescription)
{
     TraceMethodEnter("CErrorInfoA::GetDescription", this);

     HRESULT hresult;
     BSTR bstrDescription;

     bstrDescription = NULL;

     _DebugHook(GetWide(), MEMBER_PTR(IErrorInfo, GetDescription));
     hresult = GetWide()->GetDescription(&bstrDescription);
     if(FAILED(hresult))
     return hresult;

     hresult = ConvertDispStringToA(bstrDescription, pbstraDescription);

     SysFreeString(bstrDescription);

     return hresult;
}

STDMETHODIMP
CErrorInfoA::GetHelpFile(BSTRA *pbstraHelpFile)
{
     TraceMethodEnter("CErrorInfoA::GetHelpFile", this);

     HRESULT hresult;
     BSTR bstrHelpFile;

     bstrHelpFile = NULL;

     _DebugHook(GetWide(), MEMBER_PTR(IErrorInfo, GetHelpFile));

     hresult = GetWide()->GetHelpFile(&bstrHelpFile);
     if(FAILED(hresult))
     return hresult;

     hresult = ConvertDispStringToA(bstrHelpFile, pbstraHelpFile);

     SysFreeString(bstrHelpFile);

     return hresult;
}

STDMETHODIMP
CErrorInfoA::GetHelpContext(DWORD *pdwHelpContext)
{
     TraceMethodEnter("CErrorInfoA::GetHelpContext", this);

     _DebugHook(GetWide(), MEMBER_PTR(IErrorInfo, GetHelpContext));

     return GetWide()->GetHelpContext(pdwHelpContext);
}
```

ICreateErrorInfo Implementation   (OLEANSI Sample)

```
//
***************************************************************************
//
//                      ICreateErrorInfo Implementation
//
//
***************************************************************************

STDMETHODIMP
CCreateErrorInfoA::SetGUID(REFGUID rguid)
{
     TraceMethodEnter("CCreateErrorInfoA::SetGUID", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateErrorInfo, SetGUID));

     return GetWide()->SetGUID(rguid);
}

STDMETHODIMP
CCreateErrorInfoA::SetSource(LPSTR szSource)
{
     TraceMethodEnter("CCreateErrorInfoA::SetSource", this);

     HRESULT hresult;
     LPWSTR szwSource;

     szwSource = NULL;

     _DebugHook(GetWide(), MEMBER_PTR(ICreateErrorInfo, SetSource));

     if(szSource != NULL){
     hresult = ConvertStringToW(szSource, &szwSource);
     if(FAILED(hresult))
           return hresult;
     }

     hresult = GetWide()->SetSource(szwSource);

     if(szwSource != NULL)
       ConvertStringFree(szwSource);

     return hresult;
}

STDMETHODIMP
CCreateErrorInfoA::SetDescription(LPSTR szDescription)
{
     TraceMethodEnter("CCreateErrorInfoA::SetDescription", this);

     HRESULT hresult;
     LPWSTR szwDescription;
     szwDescription = NULL;

     _DebugHook(GetWide(), MEMBER_PTR(ICreateErrorInfo, SetDescription));

     if(szDescription != NULL){
     hresult = ConvertStringToW(szDescription, &szwDescription);
     if(FAILED(hresult))
           return hresult;
     }

     hresult = GetWide()->SetDescription(szwDescription);

     if(szwDescription != NULL)
     ConvertStringFree(szwDescription);

     return hresult;
}

STDMETHODIMP
CCreateErrorInfoA::SetHelpFile(LPSTR szHelpFile)
{
     TraceMethodEnter("CCreateErrorInfoA::SetHelpFile", this);

     HRESULT hresult;
     LPWSTR szwHelpFile;

     szwHelpFile = NULL;

     _DebugHook(GetWide(), MEMBER_PTR(ICreateErrorInfo, SetHelpFile));

     if(szHelpFile != NULL){
     hresult = ConvertStringToW(szHelpFile, &szwHelpFile);
     if(FAILED(hresult))
           return hresult;
     }

     hresult = GetWide()->SetHelpFile(szwHelpFile);

     if(szwHelpFile != NULL)
     ConvertStringFree(szwHelpFile);

     return hresult;
}

STDMETHODIMP
CCreateErrorInfoA::SetHelpContext(DWORD dwHelpContext)
{
     TraceMethodEnter("CCreateErrorInfoA::SetHelpContext", this);

     _DebugHook(GetWide(), MEMBER_PTR(ICreateErrorInfo, SetHelpContext));

     return GetWide()->SetHelpContext(dwHelpContext);
}
```

```
STDAPI
SetErrorInfoA(DWORD dwReserved, IErrorInfoA *perrinfoA)
{
     TraceSTDAPIEnter("SetErrorInfo");

     HRESULT hresult;
     IErrorInfo *perrinfoW;

     hresult = WrapIErrorInfoWFromA(perrinfoA, &perrinfoW);
     if(FAILED(hresult))
     return hresult;
     hresult = SetErrorInfo(dwReserved, perrinfoW);
     perrinfoW->Release(); // ownership has transferred to the system.
     return hresult;
}

STDAPI
GetErrorInfoA(DWORD dwReserved, IErrorInfoA **pperrinfoA)
{
     TraceSTDAPIEnter("GetErrorInfo");

     HRESULT hresult;
     IErrorInfo *perrinfoW;

     hresult = GetErrorInfo(dwReserved, &perrinfoW);
     if(hresult != NOERROR) // S_FALSE means no Error Info obj available
     return hresult;

     hresult = WrapIErrorInfoAFromW(perrinfoW, pperrinfoA);
     if(FAILED(hresult))
     return hresult;

     perrinfoW->Release(); // now held by perrinfoA
     return NOERROR;
}

STDAPI
CreateErrorInfoA(ICreateErrorInfoA **pperrinfoA)
{
     TraceSTDAPIEnter("CreateErrorInfo");

     HRESULT hresult;
     ICreateErrorInfo *perrinfoW;

     hresult = CreateErrorInfo(&perrinfoW);
     if(FAILED(hresult))
     return hresult;

     hresult = WrapICreateErrorInfoAFromW(perrinfoW, pperrinfoA);
     if(FAILED(hresult))
     return hresult;

     perrinfoW->Release(); // now held by pperrinfoA
     return NOERROR;
}
#endif  //!NOERRORINFO
```

ANSIDVOB.H   (OLEANSI Sample)

```
//
+--------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       ansidvob.h
//
//  Contents:   ANSI Wrappers for Unicode DvObj Interfaces and APIs.
//
//  Classes:    CEnumSTATDATAA
//              CDataObjectA
//              CViewObject2A
//              CAdviseSink2A
//              CDataAdviseHolder
//              COleCache2
//              COleCacheControl
//
//  Functions:  None.
//
//  History:    01-Nov-93   v-kentc     Created.
//
//--------------------------------------------------------------------------
-


#if defined(__cplusplus)
interface IAdviseSink2A;
#else
typedef interface IAdviseSink2A IAdviseSink2A;
#endif

typedef IAdviseSink2A * LPADVISESINKA;

// Data/View target device; determines the device for drawing or gettting
data
typedef struct FARSTRUCT tagDVTARGETDEVICEA
{
     DWORD tdSize;
     WORD tdDeviceNameOffset;
     WORD tdDriverNameOffset;
     WORD tdPortNameOffset;
     WORD tdExtDevmodeOffset;
     BYTE tdData[1];
} DVTARGETDEVICEA, * LPDVTARGETDEVICEA;

typedef DVTARGETDEVICE * LPDVTARGETDEVICE;

// Format, etc.; completely specifices the kind of data desired, including
tymed
typedef struct FARSTRUCT tagFORMATETCA
{
     CLIPFORMAT          cfFormat;
     DVTARGETDEVICEA FAR* ptd;
     DWORD               dwAspect;
     LONG                lindex;
     DWORD               tymed;
} FORMATETCA, FAR* LPFORMATETCA;

typedef struct tagSTGMEDIUMA
{
     DWORD   tymed;
     union
     {
           HANDLE  hGlobal;
     LPSTR   lpszFileName;
     IStreamA * pstm;
     IStorageA * pstg;
     }
#ifndef __cplusplus
#ifdef NONAMELESSUNION
     u       // add a tag when name less unions not supported
#endif
#endif
     ;
     IUnknown * pUnkForRelease;
} STGMEDIUMA, * LPSTGMEDIUMA;

typedef struct tagSTATDATAA
{                                    // field used by:
     FORMATETCA formatetc;            // EnumAdvise, EnumData (cache),
EnumFormats
     DWORD advf;                      // EnumAdvise, EnumData (cache)
     LPADVISESINKA pAdvSink;          // EnumAdvise
     DWORD dwConnection;              // EnumAdvise
} STATDATAA;

typedef  STATDATAA * LPSTATDATAA;


/*---------------------------------------------------------------------*/
/*                IEnumSTATDATAA                                       */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IEnumSTATDATAA

DECLARE_INTERFACE_(IEnumSTATDATAA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppv) PURE;
     STDMETHOD_(ULONG, AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG, Release) (THIS) PURE;

     // *** IEnumSTATDATA methods ***
     STDMETHOD(Next) (THIS_ ULONG celt, STATDATAA  * rgelt, ULONG *
pceltFetched) PURE;
     STDMETHOD(Skip) (THIS_ ULONG celt) PURE;
     STDMETHOD(Reset) (THIS) PURE;
     STDMETHOD(Clone) (THIS_ IEnumSTATDATAA * * ppenum) PURE;
};
typedef        IEnumSTATDATAA * LPENUMSTATDATAA;


/*---------------------------------------------------------------------*/
/*                             IEnumFORMATETCA                          */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IEnumFORMATETCA

DECLARE_INTERFACE_(IEnumFORMATETCA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IEnumFORMATETC methods ***
     STDMETHOD(Next) (THIS_ ULONG celt,
                               FORMATETCA * rgelt,
                               ULONG * pceltFetched) PURE;
     STDMETHOD(Skip) (THIS_ ULONG celt) PURE;
     STDMETHOD(Reset) (THIS) PURE;
     STDMETHOD(Clone) (THIS_ IEnumFORMATETCA * * ppenm) PURE;
};
typedef IEnumFORMATETCA * LPENUMFORMATETCA;



/*---------------------------------------------------------------------*/
/*                 IDataObjectA                                         */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IDataObjectA

DECLARE_INTERFACE_(IDataObjectA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG, Release) (THIS) PURE;

     // *** IDataObject methods ***
     STDMETHOD(GetData) (THIS_ LPFORMATETCA pformatetcIn,
                       LPSTGMEDIUMA pmediumA ) PURE;
     STDMETHOD(GetDataHere) (THIS_ LPFORMATETCA pformatetc,
                       LPSTGMEDIUMA pmediumA ) PURE;
     STDMETHOD(QueryGetData) (THIS_ LPFORMATETCA pformatetc ) PURE;
     STDMETHOD(GetCanonicalFormatEtc) (THIS_ LPFORMATETCA pformatetc,
                                    LPFORMATETCA pformatetcOut) PURE;
     STDMETHOD(SetData) (THIS_ LPFORMATETCA pformatetc, STGMEDIUMA  *
pmediumA,
                                         BOOL fRelease) PURE;
     STDMETHOD(EnumFormatEtc) (THIS_ DWORD dwDirection,
                                         LPENUMFORMATETCA * ppenumFormatEtc)
PURE;

     STDMETHOD(DAdvise) (THIS_ FORMATETCA * pFormatetc, DWORD advf,
                 LPADVISESINKA pAdvSinkA, DWORD * pdwConnection) PURE;
     STDMETHOD(DUnadvise) (THIS_ DWORD dwConnection) PURE;
     STDMETHOD(EnumDAdvise) (THIS_ LPENUMSTATDATAA * ppenumAdviseA) PURE;
};
typedef      IDataObjectA * LPDATAOBJECTA;


/*-----------------------------------------------------------------------*/
/*                             IViewObject2A                             */
/*-----------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IViewObject2A

DECLARE_INTERFACE_(IViewObject2A, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IViewObject methods ***
     STDMETHOD(Draw) (THIS_ DWORD dwDrawAspect, LONG lindex,
                         void * pvAspect, DVTARGETDEVICEA  * ptd,
                         HDC hicTargetDev,
                         HDC hdcDraw,
                         LPCRECTL lprcBounds,
                         LPCRECTL lprcWBounds,
                         BOOL (CALLBACK * pfnContinue) (DWORD),
                         DWORD dwContinue) PURE;

     STDMETHOD(GetColorSet) (THIS_ DWORD dwDrawAspect, LONG lindex,
                         void * pvAspect, DVTARGETDEVICEA  * ptd,
                         HDC hicTargetDev,
                         LPLOGPALETTE * ppColorSet) PURE;

     STDMETHOD(Freeze)(THIS_ DWORD dwDrawAspect, LONG lindex,
                         void * pvAspect,
                         DWORD * pdwFreeze) PURE;
     STDMETHOD(Unfreeze) (THIS_ DWORD dwFreeze) PURE;
     STDMETHOD(SetAdvise) (THIS_ DWORD aspects, DWORD advf,
                 LPADVISESINKA pAdvSinkA) PURE;
     STDMETHOD(GetAdvise) (THIS_ DWORD * pAspects, DWORD * pAdvf,
                 LPADVISESINKA * ppAdvSinkA) PURE;

     // *** IViewObject2 methods ***
     STDMETHOD(GetExtent) (THIS_ DWORD dwDrawAspect, LONG lindex,
                             DVTARGETDEVICEA  * ptd, LPSIZEL lpsizel) PURE;

};
typedef      IViewObject2A * LPVIEWOBJECTA;
typedef      IViewObject2A * LPVIEWOBJECT2A;


/*---------------------------------------------------------------------*/
/*                           IAdviseSink2A                             */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IAdviseSink2A

DECLARE_INTERFACE_(IAdviseSink2A, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppv) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IAdviseSink methods ***
     STDMETHOD_(void,OnDataChange)(THIS_ FORMATETCA * pFormatetc,
                      STGMEDIUMA * pStgmedA) PURE;
     STDMETHOD_(void,OnViewChange)(THIS_ DWORD dwAspect, LONG lindex) PURE;
     STDMETHOD_(void,OnRename)(THIS_ LPMONIKERA pmkA) PURE;
     STDMETHOD_(void,OnSave)(THIS) PURE;
     STDMETHOD_(void,OnClose)(THIS) PURE;

     // *** IAdviseSink2 methods ***
     STDMETHOD_(void,OnLinkSrcChange)(THIS_ LPMONIKERA pmkA) PURE;
};
typedef      IAdviseSink2A * LPADVISESINK2A;


/*---------------------------------------------------------------------*/
/*                IDataAdviseHolderA               */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IDataAdviseHolderA

DECLARE_INTERFACE_(IDataAdviseHolderA, IUnknown)
{

     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppv) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IDataAdviseHolder methods ***
     STDMETHOD(Advise)(THIS_ LPDATAOBJECTA pDataObjectA, FORMATETCA * pFetc,
           DWORD advf, LPADVISESINKA pAdviseA, DWORD * pdwConnection) PURE;
     STDMETHOD(Unadvise)(THIS_ DWORD dwConnection) PURE;
     STDMETHOD(EnumAdvise)(THIS_ LPENUMSTATDATAA * ppenumAdviseA) PURE;
     STDMETHOD(SendOnDataChange)(THIS_ LPDATAOBJECTA pDataObjectA, DWORD
dwReserved, DWORD advf) PURE;
};
typedef      IDataAdviseHolderA * LPDATAADVISEHOLDERA;


/*---------------------------------------------------------------------*/
/*                IOleCache2A                  */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IOleCache2A

DECLARE_INTERFACE_(IOleCache2A, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG, Release) (THIS) PURE;

     // *** IOleCache methods ***
     STDMETHOD(Cache) (THIS_ LPFORMATETCA lpFormatetc, DWORD advf, LPDWORD
lpdwConnection) PURE;
     STDMETHOD(Uncache) (THIS_ DWORD dwConnection) PURE;
     STDMETHOD(EnumCache) (THIS_ LPENUMSTATDATAA * ppenumStatDataA) PURE;
     STDMETHOD(InitCache) (THIS_ LPDATAOBJECTA pDataObjectA) PURE;
     STDMETHOD(SetData) (THIS_ LPFORMATETCA pformatetc, STGMEDIUMA  *
pmediumA,
                                         BOOL fRelease) PURE;

     // *** IOleCache2 methods ***
     STDMETHOD(UpdateCache) (THIS_ LPDATAOBJECTA pDataObjectA, DWORD
grfUpdf,
                                         LPVOID pReserved) PURE;
     STDMETHOD(DiscardCache) (THIS_ DWORD dwDiscardOptions) PURE;

};
typedef      IOleCache2A * LPOLECACHEA;
typedef      IOleCache2A * LPOLECACHE2A;


/*---------------------------------------------------------------------*/
/*                IOleCacheControlA                */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IOleCacheControlA

DECLARE_INTERFACE_(IOleCacheControlA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG, Release) (THIS) PURE;
     // *** IDataObject methods ***
     STDMETHOD(OnRun) (THIS_ LPDATAOBJECTA pDataObjectA) PURE;
     STDMETHOD(OnStop) (THIS) PURE;
};
typedef      IOleCacheControlA * LPOLECACHECONTROLA;



//
//  Forward declarations
//
class CEnumSTATDATAA;
class CDataObjectA;
class CViewObject2A;
class CAdviseSink2A;
class COleCache2A;
class COleCacheControlA;
class CDataAdviseHolderA;



//
+---------------------------------------------------------------------------
//
//  Class:      CEnumSTATDATAA
//
//  Synopsis:   Class definition of IEnumSTATDATAA
//
//----------------------------------------------------------------------------
-
class CEnumSTATDATAA : CAnsiInterface
{
public:
     // *** IEnumSTATDATAA methods ***
     STDMETHOD(Next) (ULONG celt,
                                STATDATAA * rgelt,
                                ULONG * pceltFetched);
     STDMETHOD(Skip) (ULONG celt);
     STDMETHOD(Reset) (VOID);
     STDMETHOD(Clone) (IEnumSTATDATAA * * ppenm);

     inline CEnumSTATDATAA(LPUNKNOWN pUnk, IEnumSTATDATA * pWide) :
                 CAnsiInterface(ID_IEnumSTATDATA, pUnk, (LPUNKNOWN)pWide)
{};

     inline IEnumSTATDATA * GetWide() const
                 { return (IEnumSTATDATA *)m_pObj; };
};



//
+---------------------------------------------------------------------------
//
```

```
//  Class:      CEnumFORMATETCA
//
//  Synopsis:   Class definition of IEnumFORMATETCA
//
//---------------------------------------------------------------------------
-
class CEnumFORMATETCA : CAnsiInterface
{
public:
     // *** IEnumFORMATETCA methods ***
     STDMETHOD(Next) (ULONG celt,
                                FORMATETCA* rgelt,
                                ULONG FAR* pceltFetched);
     STDMETHOD(Skip) (ULONG celt);
     STDMETHOD(Reset) (VOID);
     STDMETHOD(Clone) (IEnumFORMATETCA * * ppenm);

     inline CEnumFORMATETCA(LPUNKNOWN pUnk, IEnumFORMATETC * pObj) :
                 CAnsiInterface(ID_IEnumFORMATETC, pUnk, (LPUNKNOWN)pObj)
{};

     inline IEnumFORMATETC * GetWide() const
                 { return (IEnumFORMATETC *)m_pObj; };
};



//
+--------------------------------------------------------------------------
//
//  Class:      CDataObjectA
//
//  Synopsis:   Class definition of IDataObjectA
//
//---------------------------------------------------------------------------
-
class CDataObjectA : CAnsiInterface
{
public:
     // *** IDataObject methods ***
     STDMETHOD(GetData) (LPFORMATETCA pformatetcIn,
                       LPSTGMEDIUMA pmediumA );
     STDMETHOD(GetDataHere) (LPFORMATETCA pformatetc,
                       LPSTGMEDIUMA pmediumA );
     STDMETHOD(QueryGetData) (LPFORMATETCA pformatetc );
     STDMETHOD(GetCanonicalFormatEtc) (LPFORMATETCA pformatetc,
                                         LPFORMATETCA pformatetcOut);
     STDMETHOD(SetData) (LPFORMATETCA pformatetc, STGMEDIUMA * pmediumA,
                                         BOOL fRelease);
     STDMETHOD(EnumFormatEtc) (DWORD dwDirection,
                                         LPENUMFORMATETCA *
ppenumFormatEtc);

     STDMETHOD(DAdvise) (FORMATETCA * pFormatetc, DWORD advf,
                 LPADVISESINKA pAdvSinkA, DWORD * pdwConnection);
     STDMETHOD(DUnadvise) (DWORD dwConnection);
     STDMETHOD(EnumDAdvise) (LPENUMSTATDATAA * ppenumAdviseA);

     inline CDataObjectA(LPUNKNOWN pUnk, IDataObject * pWide) :
                 CAnsiInterface(ID_IDataObject, pUnk, (LPUNKNOWN)pWide) {};

     inline IDataObject * GetWide() const
                 { return (IDataObject *)m_pObj; };
};



//
+---------------------------------------------------------------------------
//
//  Class:      CViewObject2A
//
//  Synopsis:   Class definition of IViewObject2A
//
//----------------------------------------------------------------------------
-
class CViewObject2A : CAnsiInterface
{
public:
     // *** IViewObject methods ***
     STDMETHOD(Draw) (DWORD dwDrawAspect, LONG lindex,
                             void * pvAspect, DVTARGETDEVICEA * ptd,
                             HDC hicTargetDev,
                             HDC hdcDraw,
                             LPCRECTL lprcBounds,
                             LPCRECTL lprcWBounds,
                             BOOL (CALLBACK * pfnContinue) (DWORD),
                             DWORD dwContinue);

     STDMETHOD(GetColorSet) (DWORD dwDrawAspect, LONG lindex,
                             void * pvAspect, DVTARGETDEVICEA * ptd,
                             HDC hicTargetDev,
                             LPLOGPALETTE * ppColorSet);

     STDMETHOD(Freeze)(DWORD dwDrawAspect, LONG lindex,
                             void * pvAspect,
                             DWORD * pdwFreeze);
     STDMETHOD(Unfreeze) (DWORD dwFreeze);
     STDMETHOD(SetAdvise) (DWORD aspects, DWORD advf,
                 LPADVISESINKA pAdvSinkA);
     STDMETHOD(GetAdvise) (DWORD * pAspects, DWORD * pAdvf,
                 LPADVISESINKA * ppAdvSinkA);

     // *** IViewObject2 methods ***
     STDMETHOD(GetExtent) (DWORD dwDrawAspect, LONG lindex,
                             DVTARGETDEVICEA * ptd, LPSIZEL lpsizel);


     inline CViewObject2A(LPUNKNOWN pUnk, IViewObject2 * pWide) :
```

```
                  CAnsiInterface(ID_IViewObject2, pUnk, (LPUNKNOWN)pWide) {};

     inline IViewObject2 * GetWide() const
                  { return (IViewObject2 *)m_pObj; };
};



//
+-------------------------------------------------------------------------
//
//  Class:      CAdviseSink2A
//
//  Synopsis:   Class definition of IAdviseSink2A
//
//--------------------------------------------------------------------------
-
class CAdviseSink2A : CAnsiInterface
{
public:
     // *** IAdviseSink methods ***
     STDMETHOD_(void,OnDataChange)(FORMATETCA * pFormatetc,
                        STGMEDIUMA * pStgmedA);
     STDMETHOD_(void,OnViewChange)(DWORD dwAspect, LONG lindex);
     STDMETHOD_(void,OnRename)(LPMONIKERA pmkA);
     STDMETHOD_(void,OnSave)(VOID);
     STDMETHOD_(void,OnClose)(VOID);

     // *** IAdviseSink2 methods ***
     STDMETHOD_(void,OnLinkSrcChange)(LPMONIKERA pmkA);

     inline CAdviseSink2A(LPUNKNOWN pUnk, IAdviseSink2 * pWide) :
                  CAnsiInterface(ID_IAdviseSink2, pUnk, (LPUNKNOWN)pWide) {};

     inline IAdviseSink2 * GetWide() const
                  { return (IAdviseSink2 *)m_pObj; };
};



//
+-------------------------------------------------------------------------
//
//  Class:      CDataAdviseHolderA
//
//  Synopsis:   Class definition of IDataAdviseHolderA
//
//--------------------------------------------------------------------------
-
class CDataAdviseHolderA : CAnsiInterface
{
public:
     // *** IDataAdviseHolder methods ***
     STDMETHOD(Advise)(LPDATAOBJECTA pDataObjectA, FORMATETCA * pFetc,
           DWORD advf, LPADVISESINKA pAdviseA, DWORD * pdwConnection);
     STDMETHOD(Unadvise)(DWORD dwConnection);
     STDMETHOD(EnumAdvise)(LPENUMSTATDATAA * ppenumAdviseA);

     STDMETHOD(SendOnDataChange)(LPDATAOBJECTA pDataObjectA, DWORD
dwReserved, DWORD advf);

     inline CDataAdviseHolderA(LPUNKNOWN pUnk, IDataAdviseHolder * pWide) :
                 CAnsiInterface(ID_IDataAdviseHolder, pUnk,
(LPUNKNOWN)pWide) {};

     inline IDataAdviseHolder * GetWide() const
                 { return (IDataAdviseHolder *)m_pObj; };
};



//
+--------------------------------------------------------------------------
//
//  Class:      COleCache2A
//
//  Synopsis:   Class definition of IOleCache2A
//
//---------------------------------------------------------------------------
-
class COleCache2A : CAnsiInterface
{
public:
     // *** IOleCache methods ***
     STDMETHOD(Cache) (LPFORMATETCA lpFormatetc, DWORD advf, LPDWORD
lpdwConnection);
     STDMETHOD(Uncache) (DWORD dwConnection);
     STDMETHOD(EnumCache) (LPENUMSTATDATAA * ppenumStatDataA);
     STDMETHOD(InitCache) (LPDATAOBJECTA pDataObjectA);
     STDMETHOD(SetData) (LPFORMATETCA pformatetc, STGMEDIUMA * pmediumA,
                                         BOOL fRelease);

     // *** IOleCache2 methods ***
     STDMETHOD(UpdateCache) (LPDATAOBJECTA pDataObjectA, DWORD grfUpdf,
                                         LPVOID pReserved);
     STDMETHOD(DiscardCache) (DWORD dwDiscardOptions);


     inline COleCache2A(LPUNKNOWN pUnk, IOleCache2 * pWide) :
                 CAnsiInterface(ID_IOleCache2, pUnk, (LPUNKNOWN)pWide) {};

     inline IOleCache2 * GetWide() const
                 { return (IOleCache2 *)m_pObj; };
};



//
+--------------------------------------------------------------------------
```

```
//
//  Class:      COleCacheControlA
//
//  Synopsis:   Class definition of IOleCacheControlA
//
//---------------------------------------------------------------------------
-
class COleCacheControlA : CAnsiInterface
{
public:
     // *** IDataObject methods ***
     STDMETHOD(OnRun) (LPDATAOBJECTA pDataObjectA);
     STDMETHOD(OnStop) (VOID);

     inline COleCacheControlA(LPUNKNOWN pUnk, IOleCacheControl * pWide) :
                 CAnsiInterface(ID_IOleCacheControl, pUnk, (LPUNKNOWN)pWide)
{};

     inline IOleCacheControl * GetWide() const
                 { return (IOleCacheControl *)m_pObj; };
};
```

ANSIDVOB.CPP (OLEANSI Sample)

```
//
+---------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       ansidvob.h
//
//  Contents:   ANSI Wrappers for Unicode DvObj Interfaces and APIs.
//
//  Classes:    CEnumSTATDATAA
//              CDataObjectA
//              CViewObject2A
//              CAdviseSink2A
//              CDataAdviseHolder
//              COleCache2
//              COleCacheControl
//
//  Functions:  CreateDataAdviseHolderA
//              CreateDataCacheA
//              IEnumSTATDATAAFromW
//              IDataObjectAFromW
//              IViewObjectAFromW
//              IViewObject2AFromW
//              IAdviseSinkAFromW
//              IAdviseSink2AFromW
//              IDataAdviseHolderAFromW
//              IOleCacheAFromW
//              IOleCache2AFromW
//              IOleCacheControlAFromW
//
//  History:    01-Nov-93   v-kentc     Created.
//
//----------------------------------------------------------------------------
-

#include "Ole2Ansi.h"
```

IEnumSTATDATAA Implementation (OLEANSI Sample)

```
//
***************************************************************************
//
//                   IEnumSTATDATAA Implementation
//
//
***************************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     CEnumSTATDATAA::Next, public
//
//  Synopsis:   Thunks Next to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CEnumSTATDATAA::Next(
           ULONG celt,
           STATDATAA * rgelt,
           ULONG * pceltFetched)
{
     TraceMethodEnter("CEnumSTATDATAA::Next", this);

     ULONG   celtFetched;
     HRESULT hReturn;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IEnumSTATDATA, Next));

     if (pceltFetched == NULL)
           pceltFetched = &celtFetched;

     hReturn = GetWide()->Next(celt, (STATDATA *)rgelt, pceltFetched);
     if (FAILED(hReturn))
           return (hReturn);

     hResult = ConvertSTATDATAArrayToA(rgelt, *pceltFetched);
     if (FAILED(hResult))
           return (hResult);

     return hReturn;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CEnumSTATDATAA::Skip, public
```

```
//
//  Synopsis:   Thunks Skip to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CEnumSTATDATAA::Skip(ULONG celt)
{
     _DebugHook(GetWide(), MEMBER_PTR(IEnumSTATDATA, Skip));

     return (GetWide()->Skip(celt));
}


//
+--------------------------------------------------------------------------
//
//  Member:     CEnumSTATDATAA::Reset, public
//
//  Synopsis:   Thunks Reset to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CEnumSTATDATAA::Reset(VOID)
{
     _DebugHook(GetWide(), MEMBER_PTR(IEnumSTATDATA, Reset));

     return (GetWide()->Reset());
}


//
+--------------------------------------------------------------------------
//
//  Member:     CEnumSTATDATAA::Clone, public
//
//  Synopsis:   Thunks Clone to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CEnumSTATDATAA::Clone(IEnumSTATDATAA * * ppstatA)
{
     TraceMethodEnter("CEnumSTATDATAA::Clone", this);

     IEnumSTATDATA * pwide;


     _DebugHook(GetWide(), MEMBER_PTR(IEnumSTATDATA, Clone));

     *ppstatA = NULL;
```

```
    HRESULT hResult = GetWide()->Clone(&pwide);
    if (FAILED(hResult))
        return (hResult);

    hResult = WrapIEnumSTATDATAAFromW(pwide, ppstatA);

    if (pwide)
        pwide->Release();

    return hResult;
}
```

IEnumFORMATETCA Implementation   (OLEANSI Sample)

```
//
***************************************************************************
//
//                   IEnumFORMATETCA Implementation
//
//
***************************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     CEnumFORMATETCA::Next, public
//
//  Synopsis:   Thunks Next to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CEnumFORMATETCA::Next(
           ULONG celt,
           FORMATETCA * rgelt,
           ULONG * pceltFetched)
{
     TraceNotify("CEnumFORMATETCA::Next");

     ULONG   celtFetched;
     HRESULT hReturn;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IEnumFORMATETC, Next));

     if (pceltFetched == NULL)
           pceltFetched = &celtFetched;

     hReturn = GetWide()->Next(celt, (FORMATETC *)rgelt, pceltFetched);
     if (FAILED(hReturn))
           return (hReturn);

     hResult = ConvertFORMATETCArrayToA(rgelt, *pceltFetched);
     if (FAILED(hResult))
           return (hResult);

     return hReturn;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CEnumFORMATETCA::Skip, public
```

```
//
//  Synopsis:   Thunks Skip to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//-----------------------------------------------------------------------------
-
STDMETHODIMP CEnumFORMATETCA::Skip(ULONG celt)
{
     TraceNotify("CEnumFORMATETCA::Skip");

     _DebugHook(GetWide(), MEMBER_PTR(IEnumFORMATETC, Skip));

     return GetWide()->Skip(celt);
}


//
+----------------------------------------------------------------------------
//
//  Member:     CEnumFORMATETCA::Reset, public
//
//  Synopsis:   Thunks Reset to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//-----------------------------------------------------------------------------
-
STDMETHODIMP CEnumFORMATETCA::Reset(VOID)
{
     TraceNotify("CEnumFORMATETCA::Reset");

     _DebugHook(GetWide(), MEMBER_PTR(IEnumFORMATETC, Reset));

     return GetWide()->Reset();
}


//
+----------------------------------------------------------------------------
//
//  Member:     CEnumFORMATETCA::Clone, public
//
//  Synopsis:   Thunks Clone to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//-----------------------------------------------------------------------------
-
STDMETHODIMP CEnumFORMATETCA::Clone(IEnumFORMATETCA * * ppenmA)
{
     TraceNotify("CEnumFORMATETCA::Clone");

     IEnumFORMATETC * penm;
```

```
    _DebugHook(GetWide(), MEMBER_PTR(IEnumFORMATETC, Clone));

    *ppenmA = NULL;

    HRESULT hResult = GetWide()->Clone(&penm);
    if (FAILED(hResult))
        return (hResult);

    hResult = WrapIEnumFORMATETCAFromW(penm, ppenmA);

    if (penm)
        penm->Release();

    return hResult;
}
```

IDataObjectA Implementation (OLEANSI Sample)

```
//
**********************************************************************
//
//                    IDataObjectA Implementation
//
//
**********************************************************************

//
+----------------------------------------------------------------------
//
//  Member:     CDataObjectA::GetData, public
//
//  Synopsis:   Thunks GetData to Unicode method.
//
//  Returns:    OLE result code.
//
//-----------------------------------------------------------------------
-
STDMETHODIMP CDataObjectA::GetData(LPFORMATETCA pformatetcIn,
           LPSTGMEDIUMA pmediumA)
{
     TraceMethodEnter("CDataObjectA::GetData", this);

     FORMATETC FormatEtc;
     HRESULT   hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IDataObject, GetData));

     hResult = ConvertFORMATETCToW(pformatetcIn, &FormatEtc);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->GetData(&FormatEtc, (LPSTGMEDIUM)pmediumA);

     ConvertFORMATETCFree(&FormatEtc);

     if (FAILED(hResult))
           return hResult;

     return ConvertSTGMEDIUMToA(pformatetcIn->cfFormat, pmediumA);
}


//
+----------------------------------------------------------------------
//
//  Member:     CDataObjectA::GetDataHere, public
//
//  Synopsis:   Thunks GetDataHere to Unicode method.
//
```

```
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CDataObjectA::GetDataHere(LPFORMATETCA pformatetc,
           LPSTGMEDIUMA pmediumA)
{
     TraceMethodEnter("CDataObjectA::GetDataHere", this);

     FORMATETC FormatEtc;
     STGMEDIUM Medium;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IDataObject, GetDataHere));

     hResult = ConvertFORMATETCToW(pformatetc, &FormatEtc);
     if (FAILED(hResult))
           return hResult;

     hResult = ConvertSTGMEDIUMToW(pformatetc->cfFormat, pmediumA, &Medium);
     if (FAILED(hResult))
           goto Error;

     hResult = GetWide()->GetDataHere(&FormatEtc, &Medium);

     ConvertSTGMEDIUMFree(pformatetc->cfFormat, &Medium);

Error:
     ConvertFORMATETCFree(&FormatEtc);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CDataObjectA::QueryGetData, public
//
//  Synopsis:   Thunks QueryGetData to Unicode method.
//
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CDataObjectA::QueryGetData(LPFORMATETCA pformatetc)
{
     TraceMethodEnter("CDataObjectA::QueryGetData", this);

     FORMATETC FormatEtc;
     HRESULT   hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IDataObject, QueryGetData));
     hResult = ConvertFORMATETCToW(pformatetc, &FormatEtc);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->QueryGetData(&FormatEtc);

     ConvertFORMATETCFree(&FormatEtc);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CDataObjectA::GetCanonicalFormatEtc, public
//
//  Synopsis:   Thunks GetCanonicalFormatEtc to Unicode method.
//
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CDataObjectA::GetCanonicalFormatEtc(LPFORMATETCA pformatetc,
           LPFORMATETCA pformatetcOut)
{
     TraceMethodEnter("CDataObjectA::GetCanonicalFormatEtc", this);


     FORMATETC FormatEtc;
     FORMATETC FormatEtcOut;
     HRESULT   hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IDataObject, GetCanonicalFormatEtc));

     hResult = ConvertFORMATETCToW(pformatetc, &FormatEtc);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->GetCanonicalFormatEtc(&FormatEtc, &FormatEtcOut);

     ConvertFORMATETCFree(&FormatEtc);

        if (hResult != S_OK)
           return hResult;

     hResult = ConvertFORMATETCToA(&FormatEtcOut, pformatetcOut);

     ConvertFORMATETCFree(&FormatEtcOut);

     return hResult;
}
```

```
//
+--------------------------------------------------------------------------
//
//  Member:     CDataObjectA::SetData, public
//
//  Synopsis:   Thunks SetData to Unicode method.
//
//  Returns:    OLE result code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CDataObjectA::SetData(LPFORMATETCA pformatetc,
           STGMEDIUMA * pmediumA, BOOL fRelease)
{
     TraceMethodEnter("CDataObjectA::SetData", this);

     FORMATETC FormatEtc;
     STGMEDIUM Medium;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IDataObject, SetData));

     hResult = ConvertFORMATETCToW(pformatetc, &FormatEtc);
     if (FAILED(hResult))
           return hResult;

     hResult = ConvertSTGMEDIUMToW(pformatetc->cfFormat, pmediumA, &Medium);
     if (FAILED(hResult))
           goto Error;

     hResult = GetWide()->SetData(&FormatEtc, &Medium, fRelease);

     ConvertSTGMEDIUMFree(pformatetc->cfFormat, &Medium);

Error:
     ConvertFORMATETCFree(&FormatEtc);

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CDataObjectA::EnumFormatEtc, public
//
//  Synopsis:   Thunks EnumFormatEtc to Unicode method.
//
//  Returns:    OLE result code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CDataObjectA::EnumFormatEtc(DWORD dwDirection,
            LPENUMFORMATETCA * ppenumFormatEtcA)
{
     TraceMethodEnter("CDataObjectA::EnumFormatEtc", this);

     LPENUMFORMATETC pEnumFormatEtc;
     HRESULT hResult;

     _DebugHook(GetWide(), MEMBER_PTR(IDataObject, EnumFormatEtc));

     hResult = GetWide()->EnumFormatEtc(dwDirection, &pEnumFormatEtc);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIEnumFORMATETCAFromW(pEnumFormatEtc, ppenumFormatEtcA);

     if (pEnumFormatEtc)
           pEnumFormatEtc->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CDataObjectA::DAdvise, public
//
//  Synopsis:   Thunks DAdvise to Unicode method.
//
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CDataObjectA::DAdvise(FORMATETCA * pFormatetc, DWORD advf,
            LPADVISESINKA pAdvSinkA, DWORD * pdwConnection)
{
     TraceMethodEnter("CDataObjectA::DAdvise", this);

     FORMATETC FormatEtc;
     LPADVISESINK pAdvSink;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IDataObject, DAdvise));

     hResult = ConvertFORMATETCToW(pFormatetc, &FormatEtc);
     if (FAILED(hResult))
           return hResult;

           hResult = WrapIAdviseSinkWFromA(pAdvSinkA, &pAdvSink);
     if (FAILED(hResult))
           goto Error;

     hResult = GetWide()->DAdvise(&FormatEtc, advf, pAdvSink,
pdwConnection);
```

```
     if (pAdvSink)
           pAdvSink->Release();

Error:
     ConvertFORMATETCFree(&FormatEtc);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CDataObjectA::DUnadvise, public
//
//  Synopsis:   Thunks DUnadvise to Unicode method.
//
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CDataObjectA::DUnadvise(DWORD dwConnection)
{
     _DebugHook(GetWide(), MEMBER_PTR(IDataObject, DUnadvise));

     return GetWide()->DUnadvise(dwConnection);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CDataObjectA::EnumDAdvise, public
//
//  Synopsis:   Thunks EnumDAdvise to Unicode method.
//
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CDataObjectA::EnumDAdvise(LPENUMSTATDATAA * ppenumAdviseA)
{
     TraceMethodEnter("CDataObjectA::EnumDAdvise", this);

     LPENUMSTATDATA penumAdvise = NULL;
     HRESULT hResult;



     _DebugHook(GetWide(), MEMBER_PTR(IDataObject, EnumDAdvise));

     hResult = GetWide()->EnumDAdvise(&penumAdvise);
     if (FAILED(hResult))
```

```
            return hResult;

        if (penumAdvise)
        {
            hResult = WrapIEnumSTATDATAAFromW(penumAdvise, ppenumAdviseA);
            penumAdvise->Release();
        }
        else
            *ppenumAdviseA = NULL;

        return hResult;
}
```

IViewObject2A Implementation (OLEANSI Sample)

```
//
**********************************************************************
//
//                    IViewObject2A Implementation
//
//
**********************************************************************

//
+---------------------------------------------------------------------
//
//  Member:     CViewObject2A::Draw, public
//
//  Synopsis:   Thunks Draw to Unicode method.
//
//  Returns:    OLE result code.
//
//--------------------------------------------------------------------
-
STDMETHODIMP CViewObject2A::Draw(DWORD dwDrawAspect, LONG lindex,
           void * pvAspect, DVTARGETDEVICEA * ptd, HDC hicTargetDev, HDC
hdcDraw,
           LPCRECTL lprcBounds, LPCRECTL lprcWBounds,
           BOOL (CALLBACK * pfnContinue)(DWORD), DWORD dwContinue)
{
     TraceMethodEnter("CViewObject2A::Draw", this);

     LPDVTARGETDEVICE pDevice;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IViewObject, Draw));

     if (ptd)
     {
           hResult = ConvertDVTARGETDEVICEToW(ptd, &pDevice);
           if (FAILED(hResult))
                 return hResult;
     }
     else
           pDevice = NULL;

     hResult = GetWide()->Draw(dwDrawAspect, lindex, pvAspect, pDevice,
                 hicTargetDev, hdcDraw, lprcBounds, lprcWBounds,
pfnContinue,
                 dwContinue);

     ConvertDVTARGETDEVICEFree(pDevice);

     return hResult;
}
```

```
//
+---------------------------------------------------------------------------
//
//  Member:     CViewObject2A::GetColorSet, public
//
//  Synopsis:   Thunks GetColorSet to Unicode method.
//
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CViewObject2A::GetColorSet(DWORD dwDrawAspect, LONG lindex,
            void * pvAspect, DVTARGETDEVICEA * ptd, HDC hicTargetDev,
            LPLOGPALETTE * ppColorSet)
{
     TraceMethodEnter("CViewObject2A::GetColorSet", this);

     LPDVTARGETDEVICE pDevice;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IViewObject, GetColorSet));

     if (ptd)
     {
           hResult = ConvertDVTARGETDEVICEToW(ptd, &pDevice);
           if (FAILED(hResult))
                 return hResult;
     }
     else
           pDevice = NULL;

     hResult = GetWide()->GetColorSet(dwDrawAspect, lindex, pvAspect,
pDevice,
                 hicTargetDev, ppColorSet);

     ConvertDVTARGETDEVICEFree(pDevice);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CViewObject2A::Freeze, public
//
//  Synopsis:   Thunks Freeze to Unicode method.
//
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CViewObject2A::Freeze(DWORD dwDrawAspect, LONG lindex,
            void * pvAspect, DWORD * pdwFreeze)
{
     _DebugHook(GetWide(), MEMBER_PTR(IViewObject, Freeze));

     return GetWide()->Freeze(dwDrawAspect, lindex, pvAspect, pdwFreeze);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CViewObject2A::Unfreeze, public
//
//  Synopsis:   Thunks Unfreeze to Unicode method.
//
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CViewObject2A::Unfreeze(DWORD dwFreeze)
{
     _DebugHook(GetWide(), MEMBER_PTR(IViewObject, Unfreeze));

     return GetWide()->Unfreeze(dwFreeze);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CViewObject2A::SetAdvise, public
//
//  Synopsis:   Thunks SetAdvise to Unicode method.
//
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CViewObject2A::SetAdvise(DWORD aspects, DWORD advf,
            LPADVISESINKA pAdvSinkA)
{
     TraceMethodEnter("CViewObject2A::SetAdvise", this);

     LPADVISESINK pAdvSink;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IViewObject, SetAdvise));

           hResult = WrapIAdviseSinkWFromA(pAdvSinkA, &pAdvSink);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->SetAdvise(aspects, advf, pAdvSink);
     if (pAdvSink)
           pAdvSink->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CViewObject2A::GetAdvise, public
//
//  Synopsis:   Thunks GetAdvise to Unicode method.
//
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CViewObject2A::GetAdvise(DWORD * pAspects, DWORD * pAdvf,
           LPADVISESINKA * ppAdvSinkA)
{
     TraceMethodEnter("CViewObject2A::GetAdvise", this);

     LPADVISESINK pAdvSink, * ppAdvSink;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IViewObject, GetAdvise));

     if (ppAdvSinkA)
     {
           ppAdvSink = &pAdvSink;
           pAdvSink = NULL;
     }
     else
           ppAdvSink = NULL;

     hResult = GetWide()->GetAdvise(pAspects, pAdvf, ppAdvSink);
     if (FAILED(hResult))
           return hResult;

     if (ppAdvSinkA)
     {
           hResult = WrapIAdviseSinkAFromW(pAdvSink, ppAdvSinkA);

           if (pAdvSink)
                 pAdvSink->Release();
     }

     return hResult;
}
```

```
//
+---------------------------------------------------------------------------
//
//  Member:     CViewObject2A::GetExtent, public
//
//  Synopsis:   Thunks GetExtent to Unicode method.
//
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CViewObject2A::GetExtent(DWORD dwDrawAspect, LONG lindex,
           DVTARGETDEVICEA * ptd, LPSIZEL lpsizel)
{
     TraceMethodEnter("CViewObject2A::GetExtent", this);

     LPDVTARGETDEVICE pDevice;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IViewObject2, GetExtent));

     if (ptd)
     {
           hResult = ConvertDVTARGETDEVICEToW(ptd, &pDevice);
           if (FAILED(hResult))
                 return hResult;
     }
     else
           pDevice = NULL;

     hResult = GetWide()->GetExtent(dwDrawAspect, lindex, pDevice, lpsizel);

     ConvertDVTARGETDEVICEFree(pDevice);

     return hResult;
}
```

IAdviseSink2A Implementation (OLEANSI Sample)

```
//
****************************************************************************
//
//                   IAdviseSink2A Implementation
//
//
****************************************************************************

//
+---------------------------------------------------------------------------
//
//  Member:     CAdviseSink2A::OnDataChange, public
//
//  Synopsis:   Thunks OnDataChange to Unicode method.
//
//  Returns:    None.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP_(void) CAdviseSink2A::OnDataChange(FORMATETCA * pFormatetc,
           STGMEDIUMA * pStgmedA)
{
     TraceMethodEnter("CAdviseSink2A::OnDataChange", this);

     FORMATETC FormatEtc;
     STGMEDIUM Medium;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IAdviseSink, OnDataChange));

     hResult = ConvertFORMATETCToW(pFormatetc, &FormatEtc);
     if (FAILED(hResult))
           return;

     hResult = ConvertSTGMEDIUMToW(pFormatetc->cfFormat, pStgmedA, &Medium);
     if (FAILED(hResult))
           goto Error;

     GetWide()->OnDataChange(&FormatEtc, &Medium);

     ConvertSTGMEDIUMFree(pFormatetc->cfFormat, &Medium);

Error:
     ConvertFORMATETCFree(&FormatEtc);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CAdviseSink2A::OnViewChange, public
```

```
//
//  Synopsis:   Thunks OnViewChange to Unicode method.
//
//  Returns:    None.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP_(void) CAdviseSink2A::OnViewChange(DWORD dwAspect, LONG lindex)
{
     _DebugHook(GetWide(), MEMBER_PTR(IAdviseSink, OnViewChange));

     GetWide()->OnViewChange(dwAspect, lindex);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CAdviseSink2A::OnRename, public
//
//  Synopsis:   Thunks OnRename to Unicode method.
//
//  Returns:    None.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP_(void) CAdviseSink2A::OnRename(LPMONIKERA pmkA)
{
     TraceMethodEnter("CAdviseSink2A::OnRename", this);

     LPMONIKER pmk;


     _DebugHook(GetWide(), MEMBER_PTR(IAdviseSink, OnRename));

     WrapIMonikerWFromA(pmkA, &pmk);

     GetWide()->OnRename(pmk);

     if (pmk)
           pmk->Release();
}


//
+---------------------------------------------------------------------------
//
//  Member:     CAdviseSink2A::OnSave, public
//
//  Synopsis:   Thunks OnSave to Unicode method.
//
//  Returns:    None.
//
//----------------------------------------------------------------------------
-
```

```
STDMETHODIMP_(void) CAdviseSink2A::OnSave(VOID)
{
     _DebugHook(GetWide(), MEMBER_PTR(IAdviseSink2, OnSave));

     GetWide()->OnSave();
}


//
+---------------------------------------------------------------------------
//
//  Member:     CAdviseSink2A::OnClose, public
//
//  Synopsis:   Thunks OnClose to Unicode method.
//
//  Returns:    None.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP_(void) CAdviseSink2A::OnClose(VOID)
{
     _DebugHook(GetWide(), MEMBER_PTR(IAdviseSink2, OnClose));

     GetWide()->OnClose();
}



//
+---------------------------------------------------------------------------
//
//  Member:     CAdviseSink2A::OnLinkSrcChange, public
//
//  Synopsis:   Thunks OnLinkSrcChange to Unicode method.
//
//  Returns:    None.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP_(void) CAdviseSink2A::OnLinkSrcChange(LPMONIKERA pmkA)
{
     TraceMethodEnter("CAdviseSink2A::OnLinkSrcChange", this);

     LPMONIKER pmk;


     _DebugHook(GetWide(), MEMBER_PTR(IAdviseSink2, OnLinkSrcChange));

     WrapIMonikerWFromA(pmkA, &pmk);

     GetWide()->OnLinkSrcChange(pmk);

     if (pmk)
          pmk->Release();
}
```

IDataAdviseHolderA Implementation (OLEANSI Sample)

```
//
**********************************************************************
//
//                    IDataAdviseHolderA Implementation
//
//
**********************************************************************

//
+---------------------------------------------------------------------
//
//  Member:     CDataAdviseHolderA::Advise, public
//
//  Synopsis:   Thunks Advise to Unicode method.
//
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------
-
STDMETHODIMP CDataAdviseHolderA::Advise(LPDATAOBJECTA pDataObjectA,
           FORMATETCA * pFetc, DWORD advf, LPADVISESINKA pAdviseA,
           DWORD * pdwConnection)
{
     TraceMethodEnter("CDataAdviseHolderA::Advise", this);

     LPDATAOBJECT pDataObject;
     FORMATETC    FormatEtc;
     LPADVISESINK pAdvise;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IDataAdviseHolder, Advise));

     hResult = WrapIDataObjectWFromA(pDataObjectA, &pDataObject);
     if (FAILED(hResult))
           return hResult;

     hResult = ConvertFORMATETCToW(pFetc, &FormatEtc);
     if (FAILED(hResult))
           goto Error;

           hResult = WrapIAdviseSinkWFromA(pAdviseA, &pAdvise);
     if (FAILED(hResult))
           goto Error1;

     hResult = GetWide()->Advise(pDataObject, &FormatEtc, advf, pAdvise,
pdwConnection);

     if (pAdvise)
           pAdvise->Release();

Error1:
     if (pDataObject)
           pDataObject->Release();

Error:
     ConvertFORMATETCFree(&FormatEtc);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CDataAdviseHolderA::Unadvise, public
//
//  Synopsis:   Thunks Unadvise to Unicode method.
//
//  Returns:    OLE result code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CDataAdviseHolderA::Unadvise(DWORD dwConnection)
{
     _DebugHook(GetWide(), MEMBER_PTR(IDataAdviseHolder, Unadvise));

     return GetWide()->Unadvise(dwConnection);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CDataAdviseHolderA::EnumAdvise, public
//
//  Synopsis:   Thunks EnumAdvise to Unicode method.
//
//  Returns:    OLE result code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CDataAdviseHolderA::EnumAdvise(
           LPENUMSTATDATAA * ppenumAdviseA)
{
     TraceMethodEnter("CDataAdviseHolderA::EnumAdvise", this);

     LPENUMSTATDATA penumAdvise = NULL;
     HRESULT hResult;



     _DebugHook(GetWide(), MEMBER_PTR(IDataAdviseHolder, EnumAdvise));

     hResult = GetWide()->EnumAdvise(&penumAdvise);
     if (FAILED(hResult))
           return hResult;
     if (penumAdvise)
     {
           hResult = WrapIEnumSTATDATAAFromW(penumAdvise, ppenumAdviseA);
           penumAdvise->Release();
     }
     else
     *ppenumAdviseA = NULL;

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CDataAdviseHolderA::SendOnDataChange, public
//
//  Synopsis:   Thunks SendOnDataChange to Unicode method.
//
//  Returns:    OLE result code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CDataAdviseHolderA::SendOnDataChange(LPDATAOBJECTA
pDataObjectA,
           DWORD dwReserved, DWORD advf)
{
     TraceMethodEnter("CDataAdviseHolderA::SendOnDataChange", this);


     LPDATAOBJECT pDataObject;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IDataAdviseHolder, SendOnDataChange));

     hResult = WrapIDataObjectWFromA(pDataObjectA, &pDataObject);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->SendOnDataChange(pDataObject, dwReserved, advf);

     if (pDataObject)
           pDataObject->Release();

     return hResult;
}
```

IOleCache2A Implementation   (OLEANSI Sample)

```
//
***************************************************************************
//
//                    IOleCache2A Implementation
//
//
***************************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     COleCache2A::Cache, public
//
//  Synopsis:   Thunks Cache to Unicode method.
//
//  Returns:    OLE result code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP COleCache2A::Cache(LPFORMATETCA lpFormatetc, DWORD advf,
           LPDWORD lpdwConnection)
{
     TraceMethodEnter("COleCache2A::Cache", this);

     FORMATETC FormatEtc;
     HRESULT   hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleCache, Cache));

     hResult = ConvertFORMATETCToW(lpFormatetc, &FormatEtc);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->Cache(&FormatEtc, advf, lpdwConnection);

     ConvertFORMATETCFree(&FormatEtc);

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     COleCache2A::Uncache, public
//
//  Synopsis:   Thunks Uncache to Unicode method.
//
//  Returns:    OLE result code.
//
```

```
//----------------------------------------------------------------------------
-
STDMETHODIMP COleCache2A::Uncache(DWORD dwConnection)
{
     _DebugHook(GetWide(), MEMBER_PTR(IOleCache, Uncache));

     return GetWide()->Uncache(dwConnection);
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleCache2A::EnumCache, public
//
//  Synopsis:   Thunks EnumCache to Unicode method.
//
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleCache2A::EnumCache(LPENUMSTATDATAA * ppenumStatDataA)
{
     TraceMethodEnter("COleCache2A::EnumCache", this);

     LPENUMSTATDATA penumStatData = NULL;
     HRESULT hResult;



     _DebugHook(GetWide(), MEMBER_PTR(IOleCache, EnumCache));

     hResult = GetWide()->EnumCache(&penumStatData);
     if (FAILED(hResult))
            return hResult;

     if (penumStatData)
     {
            hResult = WrapIEnumSTATDATAAFromW(penumStatData,
ppenumStatDataA);
            penumStatData->Release();
     }
     else
            *ppenumStatDataA = NULL;

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleCache2A::InitCache, public
//
//  Synopsis:   Thunks InitCache to Unicode method.
```

```
//
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleCache2A::InitCache(LPDATAOBJECTA pDataObjectA)
{
     TraceMethodEnter("COleCache2A::InitCache", this);

     LPDATAOBJECT pDataObject;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleCache, InitCache));

     hResult = WrapIDataObjectWFromA(pDataObjectA, &pDataObject);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->InitCache(pDataObject);

     if (pDataObject)
           pDataObject->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleCache2A::SetData, public
//
//  Synopsis:   Thunks SetData to Unicode method.
//
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleCache2A::SetData(LPFORMATETCA pformatetc,
           STGMEDIUMA * pmediumA, BOOL fRelease)
{
     TraceMethodEnter("COleCache2A::SetData", this);

     FORMATETC FormatEtc;
     STGMEDIUM Medium;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleCache, SetData));

     hResult = ConvertFORMATETCToW(pformatetc, &FormatEtc);
     if (FAILED(hResult))
           return hResult;
     hResult = ConvertSTGMEDIUMToW(pformatetc->cfFormat, pmediumA, &Medium);
     if (FAILED(hResult))
           goto Error;

     hResult = GetWide()->SetData(&FormatEtc, &Medium, fRelease);

     ConvertSTGMEDIUMFree(pformatetc->cfFormat, &Medium);

Error:
     ConvertFORMATETCFree(&FormatEtc);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleCache2A::UpdateCache, public
//
//  Synopsis:   Thunks UpdateCache to Unicode method.
//
//  Returns:    OLE result code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleCache2A::UpdateCache(LPDATAOBJECTA pDataObjectA,
           DWORD grfUpdf, LPVOID pReserved)
{
     TraceMethodEnter("COleCache2A::UpdateCache", this);

     LPDATAOBJECT pDataObject;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleCache2, UpdateCache));

     hResult = WrapIDataObjectWFromA(pDataObjectA, &pDataObject);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->UpdateCache(pDataObject, grfUpdf, pReserved);

     if (pDataObject)
           pDataObject->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleCache2A::DiscardCache, public
```

```
//
//  Synopsis:   Thunks DiscardCache to Unicode method.
//
//  Returns:    OLE result code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP COleCache2A::DiscardCache(DWORD dwDiscardOptions)
{
     _DebugHook(GetWide(), MEMBER_PTR(IOleCache2, DiscardCache));

     return GetWide()->DiscardCache(dwDiscardOptions);
}
```

IOleCacheControlA Implementation (OLEANSI Sample)

```
//
*********************************************************************
//
//                    IOleCacheControlA Implementation
//
//
*********************************************************************

//
+--------------------------------------------------------------------
//
//  Member:     COleCacheControlA::OnRun, public
//
//  Synopsis:   Thunks OnRun to Unicode method.
//
//  Returns:    OLE result code.
//
//---------------------------------------------------------------------
-
STDMETHODIMP COleCacheControlA::OnRun(LPDATAOBJECTA pDataObjectA)
{
     TraceMethodEnter("COleCacheControlA::OnRun", this);

     LPDATAOBJECT pDataObject;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleCacheControl, OnRun));

     hResult = WrapIDataObjectWFromA(pDataObjectA, &pDataObject);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->OnRun(pDataObject);

     if (pDataObject)
           pDataObject->Release();

     return hResult;
}


//
+--------------------------------------------------------------------
//
//  Member:     COleCacheControlA::OnStop, public
//
//  Synopsis:   Thunks OnStop to Unicode method.
//
//  Returns:    OLE result code.
//
```

```
//--------------------------------------------------------------------------
-
STDMETHODIMP COleCacheControlA::OnStop(VOID)
{
     _DebugHook(GetWide(), MEMBER_PTR(IOleCacheControl, OnStop));

     return GetWide()->OnStop();
}
```

DvObj API Thunks.   (OLEANSI Sample)

```
//
****************************************************************************
//
//                           DvObj API Thunks.
//
//
****************************************************************************
STDAPI CreateDataAdviseHolderA(LPDATAADVISEHOLDERA * ppDAHolder)
{
     TraceSTDAPIEnter("CreateDataAdviseHolderA");
     LPDATAADVISEHOLDER pDAHolder;
     HRESULT hResult;


     *ppDAHolder = NULL;

     hResult = CreateDataAdviseHolder(&pDAHolder);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIDataAdviseHolderAFromW(pDAHolder, ppDAHolder);

     if (pDAHolder)
           pDAHolder->Release();

     return hResult;
}
```

ANSIMONI.H (OLEANSI Sample)

```
//
+---------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       ansimoni.h
//
//  Contents:   ANSI Wrappers for Unicode Moniker Interfaces and APIs.
//
//  Classes:    CBindCtxA
//              CMonikerA
//              CRunningObjectTableA
//              CEnumMonikerA
//
//  Functions:  BindMonikerA
//              MkParseDisplayNameA
//              MonikerRelativePathToA
//              MonikerCommonPrefixWithA
//              CreateBindCtxA
//              CreateGenericCompositeA
//              GetClassFileA
//              CreateFileMonikerA
//              CreateItemMonikerA
//              CreateAntiMonikerA
//              CreatePointerMonikerA
//              GetRunningObjectTableA
//              IBindCtxAFromW
//              IMonikerAFromW
//              IRunningObjectTableAFromW
//              IEnumMonikerAFromW
//
//  History:    01-Nov-93   v-kentc     Created.
//
//---------------------------------------------------------------------------
-


#if defined(__cplusplus)
interface IDataObjectA;
#else
typedef interface IDataObjectA IDataObjectA;
#endif

typedef IDataObjectA * LPDATAOBJECTA;

typedef struct tagOLEVERBA
{
     LONG    lVerb;
     LPSTR   lpszVerbName;
     DWORD   fuFlags;
     DWORD   grfAttribs;
} OLEVERBA, * LPOLEVERBA;
```

```
/*----------------------------------------------------------------------*/
/*                          IDropTargetA                                */
/*----------------------------------------------------------------------*/

#undef INTERFACE
#define INTERFACE   IDropTargetA

DECLARE_INTERFACE_(IDropTargetA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IDropTarget methods ***
     STDMETHOD(DragEnter) (THIS_ LPDATAOBJECTA pDataObjA, DWORD grfKeyState,
POINTL pt, LPDWORD pdwEffect) PURE;
     STDMETHOD(DragOver) (THIS_ DWORD grfKeyState, POINTL pt, LPDWORD
pdwEffect) PURE;
     STDMETHOD(DragLeave) (THIS) PURE;
     STDMETHOD(Drop) (THIS_ LPDATAOBJECTA pDataObj, DWORD grfKeyState,
POINTL pt, LPDWORD pdwEffect) PURE;
};
typedef IDropTargetA * LPDROPTARGETA;


/*----------------------------------------------------------------------*/
/*                        IPersistStorageA                              */
/*----------------------------------------------------------------------*/

#undef INTERFACE
#define INTERFACE   IPersistStorageA

DECLARE_INTERFACE_(IPersistStorageA, IPersist)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IPersist methods ***
     STDMETHOD(GetClassID) (THIS_ LPCLSID lpClassID) PURE;

     // *** IPersistStorage methods ***
     STDMETHOD(IsDirty) (THIS) PURE;
     STDMETHOD(InitNew) (THIS_ LPSTORAGEA pStgA) PURE;
     STDMETHOD(Load) (THIS_ LPSTORAGEA pStgA) PURE;
     STDMETHOD(Save) (THIS_ LPSTORAGEA pStgSaveA, BOOL fSameAsLoad) PURE;
     STDMETHOD(SaveCompleted) (THIS_ LPSTORAGEA pStgNewA) PURE;
     STDMETHOD(HandsOffStorage) (THIS) PURE;
};
typedef IPersistStorageA * LPPERSISTSTORAGEA;
```

```
/*------------------------------------------------------------------*/
/*                         IPersistStreamA                          */
/*------------------------------------------------------------------*/

#undef INTERFACE
#define INTERFACE   IPersistStreamA

DECLARE_INTERFACE_(IPersistStreamA, IPersist)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IPersist methods ***
     STDMETHOD(GetClassID) (THIS_ LPCLSID lpClassID) PURE;

     // *** IPersistStream methods ***
     STDMETHOD(IsDirty) (THIS) PURE;
     STDMETHOD(Load) (THIS_ LPSTREAMA pStmA) PURE;
     STDMETHOD(Save) (THIS_ LPSTREAMA pStmA,
                         BOOL fClearDirty) PURE;
     STDMETHOD(GetSizeMax) (THIS_ ULARGE_INTEGER * pcbSize) PURE;
};
typedef IPersistStreamA * LPPERSISTSTREAMA;


/*------------------------------------------------------------------*/
/*                         IPersistFileA                            */
/*------------------------------------------------------------------*/

#undef INTERFACE
#define INTERFACE   IPersistFileA

DECLARE_INTERFACE_(IPersistFileA, IPersist)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IPersist methods ***
     STDMETHOD(GetClassID) (THIS_ LPCLSID lpClassID) PURE;

     // *** IPersistFile methods ***
     STDMETHOD(IsDirty) (THIS) PURE;
     STDMETHOD(Load) (THIS_ LPCSTR lpszFileName, DWORD grfMode) PURE;
     STDMETHOD(Save) (THIS_ LPCSTR lpszFileName, BOOL fRemember) PURE;
     STDMETHOD(SaveCompleted) (THIS_ LPCSTR lpszFileName) PURE;
     STDMETHOD(GetCurFile) (THIS_ LPSTR * lplpszFileName) PURE;
};
typedef IPersistFileA * LPPERSISTFILEA;
```

```
#if defined(__cplusplus)
interface IMonikerA;
interface IEnumMonikerA;
interface IRunningObjectTableA;
#else
typedef interface IMonikerA IMonikerA;
typedef interface IEnumMonikerA IEnumMonikerA;
typedef interface IRunningObjectTableA IRunningObjectTableA;
#endif

typedef IMonikerA * LPMONIKERA;
typedef IEnumMonikerA * LPENUMMONIKERA;
typedef IRunningObjectTableA * LPRUNNINGOBJECTTABLEA;


/*---------------------------------------------------------------------*/
/*                          IBindCtxA                                  */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IBindCtxA

DECLARE_INTERFACE_(IBindCtxA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IBindCtx methods ***
     STDMETHOD(RegisterObjectBound) (THIS_ LPUNKNOWN punk) PURE;
     STDMETHOD(RevokeObjectBound) (THIS_ LPUNKNOWN punk) PURE;
     STDMETHOD(ReleaseBoundObjects) (THIS) PURE;

     STDMETHOD(SetBindOptions) (THIS_ LPBIND_OPTS pbindopts) PURE;
     STDMETHOD(GetBindOptions) (THIS_ LPBIND_OPTS pbindopts) PURE;
     STDMETHOD(GetRunningObjectTable) (THIS_ LPRUNNINGOBJECTTABLEA  *
           pprotA) PURE;
     STDMETHOD(RegisterObjectParam) (THIS_ LPSTR lpszKey, LPUNKNOWN punk)
PURE;
     STDMETHOD(GetObjectParam) (THIS_ LPSTR lpszKey, LPUNKNOWN * ppunk)
PURE;
     STDMETHOD(EnumObjectParam) (THIS_ LPENUMSTRINGA * ppenumA) PURE;
     STDMETHOD(RevokeObjectParam) (THIS_ LPSTR lpszKey) PURE;
};
typedef IBindCtxA * LPBCA;
typedef IBindCtxA * LPBINDCTXA;


/*---------------------------------------------------------------------*/
/*                          IMonikerA                                  */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
```

```
#define INTERFACE   IMonikerA

DECLARE_INTERFACE_(IMonikerA, IPersistStreamA)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IPersist methods ***
     STDMETHOD(GetClassID) (THIS_ LPCLSID lpClassID) PURE;

     // *** IPersistStream methods ***
     STDMETHOD(IsDirty) (THIS) PURE;
     STDMETHOD(Load) (THIS_ LPSTREAMA pStmA) PURE;
     STDMETHOD(Save) (THIS_ LPSTREAMA pStmA,
                            BOOL fClearDirty) PURE;
     STDMETHOD(GetSizeMax) (THIS_ ULARGE_INTEGER * pcbSize) PURE;

     // *** IMoniker methods ***
     STDMETHOD(BindToObject) (THIS_ LPBCA pbca, LPMONIKERA pmkToLeftA,
           REFIID riidResult, LPVOID * ppvResult) PURE;
     STDMETHOD(BindToStorage) (THIS_ LPBCA pbca, LPMONIKERA pmkToLeftA,
           REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD(Reduce) (THIS_ LPBCA pbca, DWORD dwReduceHow, LPMONIKERA *
           ppmkToLeft, LPMONIKERA * ppmkReducedA) PURE;
     STDMETHOD(ComposeWith) (THIS_ LPMONIKERA pmkRightA, BOOL
fOnlyIfNotGeneric,
           LPMONIKERA * ppmkCompositeA) PURE;
     STDMETHOD(Enum) (THIS_ BOOL fForward, LPENUMMONIKERA * ppenumMonikerA)
           PURE;
     STDMETHOD(IsEqual) (THIS_ LPMONIKERA pmkOtherMonikerA) PURE;
     STDMETHOD(Hash) (THIS_ LPDWORD pdwHash) PURE;
     STDMETHOD(IsRunning) (THIS_ LPBCA pbca, LPMONIKERA pmkToLeft,
LPMONIKERA
           pmkNewlyRunning) PURE;

     STDMETHOD(GetTimeOfLastChange) (THIS_ LPBCA pbca, LPMONIKERA
pmkToLeftA,
           FILETIME * pfiletime) PURE;
     STDMETHOD(Inverse) (THIS_ LPMONIKERA * ppmkA) PURE;
     STDMETHOD(CommonPrefixWith) (THIS_ LPMONIKERA pmkOther, LPMONIKERA *
           ppmkPrefix) PURE;
     STDMETHOD(RelativePathTo) (THIS_ LPMONIKERA pmkOther, LPMONIKERA *
           ppmkRelPath) PURE;
     STDMETHOD(GetDisplayName) (THIS_ LPBCA pbca, LPMONIKERA pmkToLeftA,
           LPSTR * lplpszDisplayName) PURE;
     STDMETHOD(ParseDisplayName) (THIS_ LPBCA pbca, LPMONIKERA pmkToLeftA,
           LPSTR lpszDisplayName, ULONG * pchEaten,
           LPMONIKERA * ppmkOutA) PURE;
     STDMETHOD(IsSystemMoniker) (THIS_ LPDWORD pdwMksys) PURE;
};


/*---------------------------------------------------------------------*/
```

```
/*                        IRunningObjectTableA                                */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IRunningObjectTableA

DECLARE_INTERFACE_(IRunningObjectTableA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IRunningObjectTable methods ***
     STDMETHOD(Register) (THIS_ DWORD grfFlags, LPUNKNOWN punkObject,
           LPMONIKERA pmkObjectNameA, DWORD * pdwRegister) PURE;
     STDMETHOD(Revoke) (THIS_ DWORD dwRegister) PURE;
     STDMETHOD(IsRunning) (THIS_ LPMONIKERA pmkObjectNameA) PURE;
     STDMETHOD(GetObject) (THIS_ LPMONIKERA pmkObjectNameA,
           LPUNKNOWN * ppunkObject) PURE;
     STDMETHOD(NoteChangeTime) (THIS_ DWORD dwRegister, FILETIME *
pfiletime) PURE;
     STDMETHOD(GetTimeOfLastChange) (THIS_ LPMONIKERA pmkObjectNameA,
FILETIME * pfiletime) PURE;
     STDMETHOD(EnumRunning) (THIS_ LPENUMMONIKERA * ppenumMonikerA ) PURE;
};
typedef IRunningObjectTableA * LPRUNNINGOBJECTTABLEA;


/*---------------------------------------------------------------------*/
/*                          IEnumMonikerA                               */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IEnumMonikerA

DECLARE_INTERFACE_(IEnumMonikerA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IEnumOleDataObject methods ***
     STDMETHOD(Next) (THIS_ ULONG celt, LPMONIKERA * rgeltA, ULONG *
pceltFetched) PURE;
     STDMETHOD(Skip) (THIS_ ULONG celt) PURE;
     STDMETHOD(Reset) (THIS) PURE;
     STDMETHOD(Clone) (THIS_ IEnumMonikerA * * ppenmA) PURE;
};
typedef IEnumMonikerA * LPENUMMONIKERA;



//
```

```
//  Forward declarations
//
class CBindCtxA;
class CMonikerA;
class CRunningObjectTableA;
class CEnumMonikerA;



//
+-------------------------------------------------------------------------
//
//  Class:      CBindCtxA
//
//  Synopsis:   Class definition of IBindCtxA
//
//--------------------------------------------------------------------------
-
class CBindCtxA : CAnsiInterface
{
public:
     // *** IBindCtx methods ***
     STDMETHOD(RegisterObjectBound) (LPUNKNOWN punk);
     STDMETHOD(RevokeObjectBound) (LPUNKNOWN punk);
     STDMETHOD(ReleaseBoundObjects) (VOID);

     STDMETHOD(SetBindOptions) (LPBIND_OPTS pbindopts);
     STDMETHOD(GetBindOptions) (LPBIND_OPTS pbindopts);
     STDMETHOD(GetRunningObjectTable) (LPRUNNINGOBJECTTABLEA  *
           pprotA);
     STDMETHOD(RegisterObjectParam) (LPSTR lpszKey, LPUNKNOWN punk);
     STDMETHOD(GetObjectParam) (LPSTR lpszKey, LPUNKNOWN * ppunk);
     STDMETHOD(EnumObjectParam) (LPENUMSTRINGA * ppenumA);
     STDMETHOD(RevokeObjectParam) (LPSTR lpszKey);

     inline CBindCtxA(LPUNKNOWN pUnk, IBindCtx * pWide) :
                 CAnsiInterface(ID_IBindCtx, pUnk, (LPUNKNOWN)pWide) {};

     inline IBindCtx * GetWide() const
                 { return (IBindCtx *)m_pObj; };
};



//
+-------------------------------------------------------------------------
//
//  Class:      CMonikerA
//
//  Synopsis:   Class definition of IMonikerA
//
//--------------------------------------------------------------------------
-
class CMonikerA : CAnsiInterface
{
```

```
public:
     // *** IPersist methods ***
     STDMETHOD(GetClassID) (LPCLSID lpClassID);

     // *** IPersistStream methods ***
     STDMETHOD(IsDirty) (VOID);
     STDMETHOD(Load) (LPSTREAMA pStmA);
     STDMETHOD(Save) (LPSTREAMA pStmA,
                            BOOL fClearDirty);
     STDMETHOD(GetSizeMax) (ULARGE_INTEGER * pcbSize);

     // *** IMoniker methods ***
     STDMETHOD(BindToObject) (LPBCA pbca, LPMONIKERA pmkToLeftA,
           REFIID riidResult, LPVOID * ppvResult);
     STDMETHOD(BindToStorage) (LPBCA pbca, LPMONIKERA pmkToLeftA,
           REFIID riid, LPVOID * ppvObj);
     STDMETHOD(Reduce) (LPBCA pbca, DWORD dwReduceHowFar, LPMONIKERA *
           ppmkToLeft, LPMONIKERA * ppmkReducedA);
     STDMETHOD(ComposeWith) (LPMONIKERA pmkRightA, BOOL fOnlyIfNotGeneric,
           LPMONIKERA * ppmkCompositeA);
     STDMETHOD(Enum) (BOOL fForward, LPENUMMONIKERA * ppenumMonikerA);
     STDMETHOD(IsEqual) (LPMONIKERA pmkOtherMonikerA);
     STDMETHOD(Hash) (LPDWORD pdwHash);
     STDMETHOD(IsRunning) (LPBCA pbca, LPMONIKERA pmkToLeft, LPMONIKERA
           pmkNewlyRunning);
     STDMETHOD(GetTimeOfLastChange) (LPBCA pbca, LPMONIKERA pmkToLeftA,
           FILETIME * pfiletime);
     STDMETHOD(Inverse) (LPMONIKERA * ppmkA);
     STDMETHOD(CommonPrefixWith) (LPMONIKERA pmkOther, LPMONIKERA *
           ppmkPrefix);
     STDMETHOD(RelativePathTo) (LPMONIKERA pmkOther, LPMONIKERA *
           ppmkRelPath);
     STDMETHOD(GetDisplayName) (LPBCA pbca, LPMONIKERA pmkToLeftA,
           LPSTR * lplpszDisplayName);
     STDMETHOD(ParseDisplayName) (LPBCA pbca, LPMONIKERA pmkToLeftA,
           LPSTR lpszDisplayName, ULONG * pchEaten,
           LPMONIKERA * ppmkOutA);
     STDMETHOD(IsSystemMoniker) (LPDWORD pdwMksys);

     inline CMonikerA(LPUNKNOWN pUnk, IMoniker * pWide) :
                 CAnsiInterface(ID_IMoniker, pUnk, (LPUNKNOWN)pWide) {};

     inline IMoniker * GetWide() const
                 { return (IMoniker *)m_pObj; };
};



//
+----------------------------------------------------------------------------
//
//  Class:      CRunningObjectTableA
//
//  Synopsis:   Class definition of IRunningObjectTableA
```

```
//
//--------------------------------------------------------------------------
-
class CRunningObjectTableA : CAnsiInterface
{
public:
     // *** IRunningObjectTable methods ***
     STDMETHOD(Register) (DWORD grfFlags, LPUNKNOWN punkObject,
           LPMONIKERA pmkObjectNameA, DWORD * pdwRegister);
     STDMETHOD(Revoke) (DWORD dwRegister);
     STDMETHOD(IsRunning) (LPMONIKERA pmkObjectNameA);
     STDMETHOD(GetObject) (LPMONIKERA pmkObjectNameA,
           LPUNKNOWN * ppunkObject);
     STDMETHOD(NoteChangeTime) (DWORD dwRegister, FILETIME * pfiletime);
     STDMETHOD(GetTimeOfLastChange) (LPMONIKERA pmkObjectNameA, FILETIME *
pfiletime);
     STDMETHOD(EnumRunning) (LPENUMMONIKERA * ppenumMonikerA );

     inline CRunningObjectTableA(LPUNKNOWN pUnk, IRunningObjectTable *
pWide) :
                 CAnsiInterface(ID_IRunningObjectTable, pUnk,
(LPUNKNOWN)pWide) {};

     inline IRunningObjectTable * GetWide() const
                 { return (IRunningObjectTable *)m_pObj; };
};



//
+-------------------------------------------------------------------------
//
//  Class:      CEnumMonikerA
//
//  Synopsis:   Class definition of IEnumMonikerA
//
//--------------------------------------------------------------------------
-
class CEnumMonikerA : CAnsiInterface
{
public:
     // *** IEnumOleDataObject methods ***
     STDMETHOD(Next) (ULONG celt, LPMONIKERA * rgeltA, ULONG *
pceltFetched);
     STDMETHOD(Skip) (ULONG celt);
     STDMETHOD(Reset) (VOID);
     STDMETHOD(Clone) (IEnumMonikerA * * ppenmA);

     inline CEnumMonikerA(LPUNKNOWN pUnk, IEnumMoniker * pWide) :
                 CAnsiInterface(ID_IEnumMoniker, pUnk, (LPUNKNOWN)pWide) {};

     inline IEnumMoniker * GetWide() const
                 { return (IEnumMoniker *)m_pObj; };
};
```

ANSIMONI.CPP (OLEANSI Sample)

```
//
+---------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       ansimoni.h
//
//  Contents:   ANSI Wrappers for Unicode Moniker Interfaces and APIs.
//
//  Classes:    CBindCtxA
//              CMonikerA
//              CRunningObjectTableA
//              CEnumMonikerA
//
//  Functions:  BindMonikerA
//              MkParseDisplayNameA
//              MonikerRelativePathToA
//              MonikerCommonPrefixWithA
//              CreateBindCtxA
//              CreateGenericCompositeA
//              GetClassFileA
//              CreateFileMonikerA
//              CreateItemMonikerA
//              CreateAntiMonikerA
//              CreatePointerMonikerA
//              GetRunningObjectTableA
//              IBindCtxAFromW
//              IMonikerAFromW
//              IRunningObjectTableAFromW
//              IEnumMonikerAFromW
//
//  History:    01-Nov-93   v-kentc     Created.
//                      28-Mar-94   v-kentc     Wrap all IUnknowns.
//
//---------------------------------------------------------------------------
-

#include "Ole2Ansi.h"
```

IBindCtxA Implementation (OLEANSI Sample)

```
//
**********************************************************************
//
//                    IBindCtxA Implementation
//
//
**********************************************************************

//
+---------------------------------------------------------------------
//
//  Member:     CBindCtxA::RegisterObjectBound, public
//
//  Synopsis:   Thunks RegisterObjectBound to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------
-
STDMETHODIMP CBindCtxA::RegisterObjectBound(LPUNKNOWN punkA)
{
     TraceMethodEnter("CBindCtxA::RegisterObjectBound", this);

     LPUNKNOWN punk;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IBindCtx, RegisterObjectBound));

     hResult = WrapIUnknownWFromA(punkA, &punk);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->RegisterObjectBound(punk);

     punk->Release();

     return hResult;
}


//
+---------------------------------------------------------------------
//
//  Member:     CBindCtxA::RevokeObjectBound, public
//
//  Synopsis:   Thunks RevokeObjectBound to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------
-
```

```
STDMETHODIMP CBindCtxA::RevokeObjectBound(LPUNKNOWN punkA)
{
     TraceMethodEnter("CBindCtxA::RevokeObjectBound", this);

     LPUNKNOWN punk;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IBindCtx, RevokeObjectBound));

     hResult = WrapIUnknownWFromA(punkA, &punk);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->RevokeObjectBound(punk);

     punk->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CBindCtxA::ReleaseBoundObjects, public
//
//  Synopsis:   Thunks ReleaseBoundObjects to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CBindCtxA::ReleaseBoundObjects()
{
     _DebugHook(GetWide(), MEMBER_PTR(IBindCtx, ReleaseBoundObjects));

     return GetWide()->ReleaseBoundObjects();
}


//
+---------------------------------------------------------------------------
//
//  Member:     CBindCtxA::SetBindOptions, public
//
//  Synopsis:   Thunks SetBindOptions to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CBindCtxA::SetBindOptions(LPBIND_OPTS pbindopts)
{
     _DebugHook(GetWide(), MEMBER_PTR(IBindCtx, SetBindOptions));
     return GetWide()->SetBindOptions(pbindopts);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CBindCtxA::GetBindOptions, public
//
//  Synopsis:   Thunks GetBindOptions to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CBindCtxA::GetBindOptions(LPBIND_OPTS pbindopts)
{
     _DebugHook(GetWide(), MEMBER_PTR(IBindCtx, GetBindOptions));

     return GetWide()->GetBindOptions(pbindopts);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CBindCtxA::GetRunningObjectTable, public
//
//  Synopsis:   Thunks GetRunningObjectTable to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CBindCtxA::GetRunningObjectTable(LPRUNNINGOBJECTTABLEA *
pprotA)
{
     TraceMethodEnter("CBindCtxA::GetRunningObjectTable", this);

     LPRUNNINGOBJECTTABLE prot;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IBindCtx, GetRunningObjectTable));

     *pprotA = NULL;

     hResult = GetWide()->GetRunningObjectTable(&prot);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIRunningObjectTableAFromW(prot, pprotA);
     if (prot)
          prot->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CBindCtxA::RegisterObjectParam, public
//
//  Synopsis:   Thunks RegisterObjectParam to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CBindCtxA::RegisterObjectParam(LPSTR lpszKeyA, LPUNKNOWN punkA)
{
     TraceMethodEnter("CBindCtxA::RegisterObjectParam", this);

     LPOLESTR lpszKey;
     LPUNKNOWN punk;
     HRESULT  hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IBindCtx, RegisterObjectParam));

     hResult = ConvertStringToW(lpszKeyA, &lpszKey);
     if (FAILED(hResult))
          return (hResult);

     hResult = WrapIUnknownWFromA(punkA, &punk);
     if (FAILED(hResult))
          goto Error;

     hResult = GetWide()->RegisterObjectParam(lpszKey, punk);

     punk->Release();

Error:
     ConvertStringFree(lpszKey);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CBindCtxA::GetObjectParam, public
//
//  Synopsis:   Thunks GetObjectParam to Unicode method.
//
```

```
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CBindCtxA::GetObjectParam(LPSTR lpszKeyA, LPUNKNOWN * ppunkA)
{
     TraceMethodEnter("CBindCtxA::GetObjectParam", this);

     LPOLESTR  lpszKey;
     LPUNKNOWN punk;
     HRESULT   hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IBindCtx, GetObjectParam));

     *ppunkA = NULL;

     hResult = ConvertStringToW(lpszKeyA, &lpszKey);
     if (FAILED(hResult))
           return (hResult);

     hResult = GetWide()->GetObjectParam(lpszKey, &punk);
     if (FAILED(hResult))
           goto Error;

     hResult = WrapIUnknownAFromW(punk, ppunkA);
     punk->Release();

Error:
     ConvertStringFree(lpszKey);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CBindCtxA::EnumObjectParam, public
//
//  Synopsis:   Thunks EnumObjectParam to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CBindCtxA::EnumObjectParam(LPENUMSTRINGA * ppenumA)
{
     TraceMethodEnter("CBindCtxA::EnumObjectParam", this);

     LPENUMSTRING penum;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IBindCtx, EnumObjectParam));
```

```
    *ppenumA = NULL;

    hResult = GetWide()->EnumObjectParam(&penum);
    if (FAILED(hResult))
        return hResult;

    if (penum)
    {
        hResult = WrapIEnumStringAFromW(penum, ppenumA);
        penum->Release();
    }

    return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CBindCtxA::RevokeObjectParam, public
//
//  Synopsis:   Thunks RevokeObjectParam to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CBindCtxA::RevokeObjectParam(LPSTR lpszKeyA)
{
    TraceMethodEnter("CBindCtxA::RevokeObjectParam", this);

    LPOLESTR lpszKey;
    HRESULT hResult;


    _DebugHook(GetWide(), MEMBER_PTR(IBindCtx, RevokeObjectParam));

    hResult = ConvertStringToW(lpszKeyA, &lpszKey);
    if (FAILED(hResult))
        return (hResult);

    hResult = GetWide()->RevokeObjectParam(lpszKey);

    ConvertStringFree(lpszKey);

    return hResult;
}
```

IMonikerA Implementation   (OLEANSI Sample)

```
//
***************************************************************************
//
//                    IMonikerA Implementation
//
//
***************************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     CMonikerA::GetClassID, public
//
//  Synopsis:   Thunks GetClassID to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::GetClassID(LPCLSID lpClassID)
{
     _DebugHook(GetWide(), MEMBER_PTR(IPersist, GetClassID));

     return GetWide()->GetClassID(lpClassID);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CMonikerA::IsDirty, public
//
//  Synopsis:   Thunks IsDirty to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::IsDirty(VOID)
{
     _DebugHook(GetWide(), MEMBER_PTR(IPersistStream, IsDirty));

     return GetWide()->IsDirty();
}


//
+--------------------------------------------------------------------------
//
//  Member:     CMonikerA::Load, public
//
//  Synopsis:   Thunks Load to Unicode method.
```

```
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::Load(LPSTREAMA pStrmA)
{
     TraceMethodEnter("CMonikerA::Load", this);

     LPSTREAM pStrm;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IPersistStream, Load));

     hResult = WrapIStreamWFromA(pStrmA, &pStrm);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->Load(pStrm);

     if (pStrm)
           pStrm->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CMonikerA::Save, public
//
//  Synopsis:   Thunks Save to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::Save(LPSTREAMA pStrmA, BOOL fClearDirty)
{
     TraceMethodEnter("CMonikerA::Save", this);

     LPSTREAM pStrm;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IPersistStream, Save));

     hResult = WrapIStreamWFromA(pStrmA, &pStrm);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->Save(pStrm, fClearDirty);
     if (pStrm)
           pStrm->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CMonikerA::GetSizeMax, public
//
//  Synopsis:   Thunks GetSizeMax to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::GetSizeMax(ULARGE_INTEGER * pcbSize)
{
     _DebugHook(GetWide(), MEMBER_PTR(IPersistStream, GetSizeMax));

     return GetWide()->GetSizeMax(pcbSize);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CMonikerA::BindToObject, public
//
//  Synopsis:   Thunks BindToObject to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::BindToObject(LPBCA pbcA, LPMONIKERA pmkToLeftA,
           REFIID riidResult, LPVOID * ppvResult)
{
     TraceMethodEnter("CMonikerA::BindToObject", this);

     LPBC        pbc;
     LPMONIKER   pmk;
     LPUNKNOWN   punk;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IMoniker, BindToObject));

     *ppvResult = NULL;

     hResult = WrapIBindCtxWFromA(pbcA, &pbc);
     if (FAILED(hResult))
```

```
            return hResult;

      hResult = WrapIMonikerWFromA(pmkToLeftA, &pmk);
      if (FAILED(hResult))
            goto Error;

      hResult = GetWide()->BindToObject(pbc, pmk, riidResult, (LPVOID
*)&punk);
      if (FAILED(hResult))
            goto Error1;

      if (punk)
      {
            IDINTERFACE idRef = WrapTranslateIID(riidResult);
            hResult = WrapInterfaceAFromW(idRef, punk, (LPUNKNOWN
*)ppvResult);
            punk->Release();
      }

Error1:
      if (pmk)
            pmk->Release();

Error:
      if (pbc)
            pbc->Release();

      return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CMonikerA::BindToStorage, public
//
//  Synopsis:   Thunks BindToStorage to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::BindToStorage(LPBCA pbcA, LPMONIKERA pmkToLeftA,
            REFIID riid, LPVOID * ppvObj)
{
      TraceMethodEnter("CMonikerA::BindToStorage", this);

      LPBC pbc;
      LPMONIKER pmk;
      LPUNKNOWN punk;
      HRESULT hResult;


      _DebugHook(GetWide(), MEMBER_PTR(IMoniker, BindToStorage));
```

```
     *ppvObj = NULL;

     hResult = WrapIBindCtxWFromA(pbcA, &pbc);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIMonikerWFromA(pmkToLeftA, &pmk);
     if (FAILED(hResult))
           goto Error;

     hResult = GetWide()->BindToStorage(pbc, pmk, riid, (LPVOID *)&punk);
     if (FAILED(hResult))
           goto Error1;

     if (punk)
     {
           IDINTERFACE idRef = WrapTranslateIID(riid);
           hResult = WrapInterfaceAFromW(idRef, punk, (LPUNKNOWN *)ppvObj);
           punk->Release();
     }

Error1:
     if (pmk)
           pmk->Release();

Error:
     if (pbc)
           pbc->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CMonikerA::Reduce, public
//
//  Synopsis:   Thunks Reduce to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::Reduce(LPBCA pbcA, DWORD dwReduceHowFar,
           LPMONIKERA * ppmkToLeftA, LPMONIKERA * ppmkReducedA)
{
     TraceMethodEnter("CMonikerA::Reduce", this);

     LPBC pbc;
     LPMONIKER pmkToLeft, * ppmkToLeft;
     LPMONIKER pmkReduced, * ppmkReduced;
     HRESULT hReturn;
     HRESULT hResult;
```

```
    _DebugHook(GetWide(), MEMBER_PTR(IMoniker, Reduce));

    if (ppmkToLeftA)
        *ppmkToLeftA = NULL;

    if (ppmkReducedA)
        *ppmkReducedA = NULL;

    hResult = WrapIBindCtxWFromA(pbcA, &pbc);
    if (FAILED(hResult))
        return hResult;

    if (ppmkToLeftA)
        ppmkToLeft = &pmkToLeft;
    else
        ppmkToLeft = NULL;

    if (ppmkReducedA)
        ppmkReduced = &pmkReduced;
    else
        ppmkReduced = NULL;

    hReturn = GetWide()->Reduce(pbc, dwReduceHowFar, ppmkToLeft,
            ppmkReduced);
    if (FAILED(hReturn))
        goto Error;

    if (ppmkToLeft)
    {
        hResult = WrapIMonikerAFromW(pmkToLeft, ppmkToLeftA);
        if (FAILED(hResult))
            goto Error1;
    }

    if (ppmkReduced)
    {
        hResult = WrapIMonikerAFromW(pmkReduced, ppmkReducedA);
        if (FAILED(hResult))
            goto Error1;
    }

Error1:
    if (ppmkToLeft && *ppmkToLeft)
        pmkToLeft->Release();

    if (ppmkReduced && *ppmkReduced)
        pmkReduced->Release();

Error:
    if (pbc)
        pbc->Release();

    if (FAILED(hResult))
```

```
           hReturn = hResult;

     return hReturn;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CMonikerA::ComposeWith, public
//
//  Synopsis:   Thunks ComposeWith to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::ComposeWith(LPMONIKERA pmkRightA,
           BOOL fOnlyIfNotGeneric, LPMONIKERA * ppmkCompositeA)
{
     TraceMethodEnter("CMonikerA::ComposeWith", this);

     LPMONIKER pmk;
     LPMONIKER pmkComposite;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IMoniker, ComposeWith));

     *ppmkCompositeA = NULL;

     hResult = WrapIMonikerWFromA(pmkRightA, &pmk);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->ComposeWith(pmk, fOnlyIfNotGeneric,
&pmkComposite);
     if (FAILED(hResult))
           goto Error;

     hResult = WrapIMonikerAFromW(pmkComposite, ppmkCompositeA);

     if (pmkComposite)
           pmkComposite->Release();

Error:
     if (pmk)
           pmk->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
```

```
//
//  Member:     CMonikerA::Enum, public
//
//  Synopsis:   Thunks Enum to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::Enum(BOOL fForward, LPENUMMONIKERA * ppenumMonikerA)
{
     TraceMethodEnter("CMonikerA::Enum", this);

     LPENUMMONIKER penum;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IMoniker, Enum));

     *ppenumMonikerA = NULL;

     hResult = GetWide()->Enum(fForward, &penum);
     if (FAILED(hResult))
           return hResult;

     if (penum)
     {
           hResult = WrapIEnumMonikerAFromW(penum, ppenumMonikerA);
           penum->Release();
     }

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CMonikerA::IsEqual, public
//
//  Synopsis:   Thunks IsEqual to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::IsEqual(LPMONIKERA pmkOtherMonikerA)
{
     TraceMethodEnter("CMonikerA::IsEqual", this);

     LPMONIKER pmk;
     HRESULT hResult;
     _DebugHook(GetWide(), MEMBER_PTR(IMoniker, IsEqual));

     hResult = WrapIMonikerWFromA(pmkOtherMonikerA, &pmk);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->IsEqual(pmk);

     pmk->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CMonikerA::Hash, public
///
//  Synopsis:   Thunks Hash to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::Hash(LPDWORD pdwHash)
{
     _DebugHook(GetWide(), MEMBER_PTR(IMoniker, Hash));

     return GetWide()->Hash(pdwHash);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CMonikerA::IsRunning, public
//
//  Synopsis:   Thunks IsRunning to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::IsRunning(LPBCA pbcA, LPMONIKERA pmkToLeftA,
           LPMONIKERA pmkNewlyRunningA)
{
     TraceMethodEnter("CMonikerA::IsRunning", this);

     LPBC pbc;
     LPMONIKER pmk;
     LPMONIKER pmkRunning;
     HRESULT hResult;
     _DebugHook(GetWide(), MEMBER_PTR(IMoniker, IsRunning));

     hResult = WrapIBindCtxWFromA(pbcA, &pbc);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIMonikerWFromA(pmkToLeftA, &pmk);
     if (FAILED(hResult))
           goto Error;

     hResult = WrapIMonikerWFromA(pmkNewlyRunningA, &pmkRunning);
     if (FAILED(hResult))
           goto Error1;

     hResult = GetWide()->IsRunning(pbc, pmk, pmkRunning);

     if (pmkRunning)
           pmkRunning->Release();

Error1:
     if (pmk)
           pmk->Release();

Error:
     if (pbc)
           pbc->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CMonikerA::GetTimeOfLastChange, public
//
//  Synopsis:   Thunks GetTimeOfLastChange to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::GetTimeOfLastChange(LPBCA pbcA,
           LPMONIKERA pmkToLeftA, FILETIME * pfiletime)
{
     TraceMethodEnter("CMonikerA::GetTimeOfLastChange", this);

     LPBC pbc;
     LPMONIKER pmk;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IMoniker, GetTimeOfLastChange));

     hResult = WrapIBindCtxWFromA(pbcA, &pbc);
```

```
	if (FAILED(hResult))
		return hResult;

	hResult = WrapIMonikerWFromA(pmkToLeftA, &pmk);
	if (FAILED(hResult))
		goto Error;

	hResult = GetWide()->GetTimeOfLastChange(pbc, pmk, pfiletime);

	if (pmk)
		pmk->Release();

Error:
	if (pbc)
		pbc->Release();

	return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CMonikerA::Inverse, public
//
//  Synopsis:   Thunks Inverse to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::Inverse(LPMONIKERA * ppmkA)
{
	TraceMethodEnter("CMonikerA::Inverse", this);

	LPMONIKER pmk;
	HRESULT hResult;


	_DebugHook(GetWide(), MEMBER_PTR(IMoniker, Inverse));

	*ppmkA = NULL;

	hResult = GetWide()->Inverse(&pmk);
	if (FAILED(hResult))
		return hResult;

	hResult = WrapIMonikerAFromW(pmk, ppmkA);

	if (pmk)
		pmk->Release();

	return hResult;
}
```

```
//
+----------------------------------------------------------------------------
//
//  Member:     CMonikerA::CommonPrefixWith, public
//
//  Synopsis:   Thunks CommonPrefixWith to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::CommonPrefixWith(LPMONIKERA pmkOtherA,
           LPMONIKERA * ppmkPrefixA)
{
     TraceMethodEnter("CMonikerA::CommonPrefixWith", this);

     LPMONIKER pmkOther;
     LPMONIKER pmkPrefix;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IMoniker, CommonPrefixWith));

     *ppmkPrefixA = NULL;

     hResult = WrapIMonikerWFromA(pmkOtherA, &pmkOther);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->CommonPrefixWith(pmkOther, &pmkPrefix);
     if (FAILED(hResult))
           goto Error;

     hResult = WrapIMonikerAFromW(pmkPrefix, ppmkPrefixA);

     if (pmkPrefix)
           pmkPrefix->Release();

Error:
     if (pmkOther)
           pmkOther->Release();

     return hResult;
}


//
+----------------------------------------------------------------------------
//
//  Member:     CMonikerA::RelativePathTo, public
//
//  Synopsis:   Thunks RelativePathTo to Unicode method.
//
```

```
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::RelativePathTo(LPMONIKERA pmkOtherA,
           LPMONIKERA * ppmkRelPathA)
{
     TraceMethodEnter("CMonikerA::RelativePathTo", this);

     LPMONIKER pmkOther;
     LPMONIKER pmkRelPath;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IMoniker, RelativePathTo));

     *ppmkRelPathA = NULL;

     hResult = WrapIMonikerWFromA(pmkOtherA, &pmkOther);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->RelativePathTo(pmkOther, &pmkRelPath);
     if (FAILED(hResult))
           goto Error;

     hResult = WrapIMonikerAFromW(pmkRelPath, ppmkRelPathA);

     if (pmkRelPath)
           pmkRelPath->Release();

Error:
     pmkOther->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CMonikerA::GetDisplayName, public
//
//  Synopsis:   Thunks GetDisplayName to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::GetDisplayName(LPBCA pbcA, LPMONIKERA pmkToLeftA,
           LPSTR * lplpszDisplayNameA)
{
     TraceMethodEnter("CMonikerA::GetDisplayName", this);

     LPBC pbc;
     LPMONIKER pmkToLeft;
     LPOLESTR lpszDisplayName;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IMoniker, GetDisplayName));

     *lplpszDisplayNameA = NULL;

     hResult = WrapIBindCtxWFromA(pbcA, &pbc);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIMonikerWFromA(pmkToLeftA, &pmkToLeft);
     if (FAILED(hResult))
           goto Error;

     hResult = GetWide()->GetDisplayName(pbc, pmkToLeft, &lpszDisplayName);
     if (FAILED(hResult))
           goto Error1;

     hResult = ConvertStringToA(lpszDisplayName, lplpszDisplayNameA);

     ConvertStringFree(lpszDisplayName);

Error1:
     if (pmkToLeft)
           pmkToLeft->Release();

Error:
     if (pbc)
           pbc->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CMonikerA::ParseDisplayName, public
//
//  Synopsis:   Thunks ParseDisplayName to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::ParseDisplayName(LPBCA pbcA, LPMONIKERA pmkToLeftA,
           LPSTR lpszDisplayNameA, ULONG * pchEaten, LPMONIKERA * ppmkOutA)
{
     TraceMethodEnter("CMonikerA::ParseDisplayName", this);

     LPBC pbc;
     LPMONIKER pmkToLeft;
     LPOLESTR lpszDisplayName;
     LPMONIKER pmkOut;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IMoniker, ParseDisplayName));

     *ppmkOutA = NULL;

     hResult = WrapIBindCtxWFromA(pbcA, &pbc);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIMonikerWFromA(pmkToLeftA, &pmkToLeft);
     if (FAILED(hResult))
           goto Error;

     hResult = ConvertStringToW(lpszDisplayNameA, &lpszDisplayName);
     if (FAILED(hResult))
           goto Error1;

     hResult = GetWide()->ParseDisplayName(pbc, pmkToLeft, lpszDisplayName,
                 pchEaten, &pmkOut);
     if (FAILED(hResult))
           goto Error2;

     hResult = WrapIMonikerAFromW(pmkOut, ppmkOutA);

     if (pmkOut)
           pmkOut->Release();

Error2:
     ConvertStringFree(lpszDisplayName);

Error1:
     if (pmkToLeft)
           pmkToLeft->Release();

Error:
     if (pbc)
           pbc->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CMonikerA::IsSystemMoniker, public
//
//  Synopsis:   Thunks IsSystemMoniker to Unicode method.
//
//  Returns:    OLE2 Result Code.
```

```
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CMonikerA::IsSystemMoniker(LPDWORD pdwMksys)
{
     _DebugHook(GetWide(), MEMBER_PTR(IMoniker, IsSystemMoniker));

     return GetWide()->IsSystemMoniker(pdwMksys);
}
```

IRunningObjectTableA Implementation   (OLEANSI Sample)

```
//
*****************************************************************************
//
//                    IRunningObjectTableA Implementation
//
//
*****************************************************************************

//
+---------------------------------------------------------------------------
//
//  Member:     CRunningObjectTableA::Register, public
//
//  Synopsis:   Thunks Register to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CRunningObjectTableA::Register(DWORD grfFlags,
           LPUNKNOWN punkObjectA, LPMONIKERA pmkObjectNameA, DWORD *
pdwRegister)
{
     TraceMethodEnter("CRunningObjectTableA::Register", this);

     LPMONIKER pmkObjectName;
     LPUNKNOWN punk;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IRunningObjectTable, Register));

     hResult = WrapIMonikerWFromA(pmkObjectNameA, &pmkObjectName);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIUnknownWFromA(punkObjectA, &punk);
     if (FAILED(hResult))
           goto Error;

     hResult = GetWide()->Register(grfFlags, punk, pmkObjectName,
                 pdwRegister);

     punk->Release();

Error:
     if (pmkObjectName)
           pmkObjectName->Release();

     return hResult;
}
```

```
//
+-------------------------------------------------------------------------
//
//  Member:     CRunningObjectTableA::Revoke, public
//
//  Synopsis:   Thunks Revoke to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CRunningObjectTableA::Revoke(DWORD dwRegister)
{
     TraceMethodEnter("CRunningObjectTableA::Revoke", this);

     _DebugHook(GetWide(), MEMBER_PTR(IRunningObjectTable, Revoke));

     return GetWide()->Revoke(dwRegister);
}


//
+-------------------------------------------------------------------------
//
//  Member:     CRunningObjectTableA::IsRunning, public
//
//  Synopsis:   Thunks IsRunning to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CRunningObjectTableA::IsRunning(LPMONIKERA pmkObjectNameA)
{
     TraceMethodEnter("CRunningObjectTableA::IsRunning", this);

     LPMONIKER pmkObjectName;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IRunningObjectTable, IsRunning));

     hResult = WrapIMonikerWFromA(pmkObjectNameA, &pmkObjectName);
     if (FAILED(hResult))
            return hResult;

     hResult = GetWide()->IsRunning(pmkObjectName);

     if (pmkObjectName)
            pmkObjectName->Release();

     return hResult;
}
```

```
//
+---------------------------------------------------------------------------
//
//  Member:     CRunningObjectTableA::GetObject, public
//
//  Synopsis:   Thunks GetObject to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CRunningObjectTableA::GetObject(LPMONIKERA pmkObjectNameA,
           LPUNKNOWN * ppunkObjectA)
{
     TraceMethodEnter("CRunningObjectTableA::GetObject", this);

     LPMONIKER pmkObjectName;
     LPUNKNOWN punk;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IRunningObjectTable, GetObject));

     *ppunkObjectA = NULL;

     hResult = WrapIMonikerWFromA(pmkObjectNameA, &pmkObjectName);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->GetObject(pmkObjectName, &punk);
     if (FAILED(hResult))
           goto Error;

     hResult = WrapIUnknownAFromW(punk, ppunkObjectA);
     punk->Release();

Error:
     if (pmkObjectName)
           pmkObjectName->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CRunningObjectTableA::NoteChangeTime, public
//
//  Synopsis:   Thunks NoteChangeTime to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
```

```
//----------------------------------------------------------------------------
-
STDMETHODIMP CRunningObjectTableA::NoteChangeTime(DWORD dwRegister,
           FILETIME * pfiletime)
{
     _DebugHook(GetWide(), MEMBER_PTR(IRunningObjectTable, NoteChangeTime));

     return GetWide()->NoteChangeTime(dwRegister, pfiletime);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CRunningObjectTableA::GetTimeOfLastChange, public
//
//  Synopsis:   Thunks GetTimeOfLastChange to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CRunningObjectTableA::GetTimeOfLastChange(
           LPMONIKERA pmkObjectNameA, FILETIME * pfiletime)
{
     TraceMethodEnter("CRunningObjectTableA::GetTimeOfLastChange", this);

     LPMONIKER pmkObjectName;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IRunningObjectTable,
GetTimeOfLastChange));

     hResult = WrapIMonikerWFromA(pmkObjectNameA, &pmkObjectName);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->GetTimeOfLastChange(pmkObjectName, pfiletime);

     if (pmkObjectName)
           pmkObjectName->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CRunningObjectTableA::EnumRunning, public
//
//  Synopsis:   Thunks EnumRunning to Unicode method.
//
//  Returns:    OLE2 Result Code.
```

```
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CRunningObjectTableA::EnumRunning(
           LPENUMMONIKERA * ppenumMonikerA)
{
     TraceMethodEnter("CRunningObjectTableA::EnumRunning", this);

     LPENUMMONIKER penumMoniker;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IRunningObjectTable, EnumRunning));

     *ppenumMonikerA = NULL;

     hResult = GetWide()->EnumRunning(&penumMoniker);
     if (FAILED(hResult))
           return hResult;

     if (penumMoniker)
     {
           hResult = WrapIEnumMonikerAFromW(penumMoniker, ppenumMonikerA);
           penumMoniker->Release();
     }

     return hResult;
}
```

IEnumMonikerA Implementation (OLEANSI Sample)

```
//
**********************************************************************
//
//                   IEnumMonikerA Implementation
//
//
**********************************************************************

//
+---------------------------------------------------------------------
//
//  Member:     CEnumMonikerA::Next, public
//
//  Synopsis:   Thunks Next to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------
-
STDMETHODIMP CEnumMonikerA::Next(
           ULONG celt,
           LPMONIKERA * rgelt,
           ULONG * pceltFetched)
{
     TraceMethodEnter("CEnumMonikerA::Next", this);

     ULONG   celtFetched;
     HRESULT hReturn;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IEnumMoniker, Next));

     if (pceltFetched == NULL)
           pceltFetched = &celtFetched;

     hReturn = GetWide()->Next(celt, (LPMONIKER *)rgelt, pceltFetched);
     if (FAILED(hReturn))
           return hReturn;

     hResult = ConvertMonikerArrayToA(rgelt, *pceltFetched);
     if (FAILED(hResult))
           return hResult;

     return hReturn;
}


//
+---------------------------------------------------------------------
//
//  Member:     CEnumMonikerA::Skip, public
```

```
//
//  Synopsis:   Thunks Skip to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CEnumMonikerA::Skip(ULONG celt)
{
     _DebugHook(GetWide(), MEMBER_PTR(IEnumMoniker, Skip));

     return GetWide()->Skip(celt);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CEnumMonikerA::Reset, public
//
//  Synopsis:   Thunks Reset to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CEnumMonikerA::Reset(VOID)
{
     _DebugHook(GetWide(), MEMBER_PTR(IEnumMoniker, Reset));

     return GetWide()->Reset();
}


//
+--------------------------------------------------------------------------
//
//  Member:     CEnumMonikerA::Clone, public
//
//  Synopsis:   Thunks Clone to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CEnumMonikerA::Clone(IEnumMonikerA * * ppenmA)
{
     TraceMethodEnter("CEnumMonikerA::Clone", this);

     IEnumMoniker * penm;


     _DebugHook(GetWide(), MEMBER_PTR(IEnumMoniker, Clone));

     *ppenmA = NULL;
```

```
	HRESULT hResult = GetWide()->Clone(&penm);
	if (FAILED(hResult))
		return (hResult);

	hResult = WrapIEnumMonikerAFromW(penm, ppenmA);

	if (penm)
		penm->Release();

	return hResult;
}
```

Moniker API Thunks.   (OLEANSI Sample)

```
//
*********************************************************************
//
//                          Moniker API Thunks.
//
//
*********************************************************************


//
+--------------------------------------------------------------------
//
//  Routine:    BindMonikerA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------
-
STDAPI BindMonikerA(LPMONIKERA pmkA, DWORD grfOpt, REFIID iidResult,
           LPVOID * ppvResult)
{
     TraceSTDAPIEnter("BindMonikerA");
     LPMONIKER pmk;
     LPUNKNOWN punk;
     HRESULT hResult;


     *ppvResult = NULL;

     hResult = WrapIMonikerWFromA(pmkA, &pmk);
     if (FAILED(hResult))
           return hResult;

     hResult = BindMoniker(pmk, grfOpt, iidResult, (LPVOID *)&punk);
     if (FAILED(hResult))
           goto Error;

     if (punk)
     {
           IDINTERFACE idRef = WrapTranslateIID(iidResult);

           hResult = WrapInterfaceAFromW(idRef, punk, (LPUNKNOWN
*)ppvResult);

           punk->Release();
     }

Error:
     if (pmk)
           pmk->Release();
     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    MkParseDisplayNameA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI MkParseDisplayNameA(LPBCA pbcA, LPSTR szUserNameA, ULONG * pchEaten,
           LPMONIKERA * ppmkA)
{
     TraceSTDAPIEnter("MkParseDisplayNameA");
     LPBC  pbc;
     LPOLESTR lpszUserName;
     LPMONIKER pmk;
     HRESULT hResult;


     *ppmkA = NULL;

     hResult = WrapIBindCtxWFromA(pbcA, &pbc);
     if (FAILED(hResult))
           return hResult;

     hResult = ConvertStringToW(szUserNameA, &lpszUserName);
     if (FAILED(hResult))
           goto Error;

     hResult = MkParseDisplayName(pbc, lpszUserName, pchEaten, &pmk);
     if (FAILED(hResult))
           goto Error1;

     hResult = WrapIMonikerAFromW(pmk, ppmkA);

     if (pmk)
           pmk->Release();

Error1:
     ConvertStringFree(lpszUserName);

Error:
     if (pbc)
           pbc->Release();

     return hResult;
}
```

```
//
+---------------------------------------------------------------------------
//
//  Routine:    MonikerRelativePathToA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI MonikerRelativePathToA(LPMONIKERA pmkSrcA, LPMONIKERA pmkDestA,
           LPMONIKERA * ppmkRelPathA, BOOL fCalledFromMethod)
{
     TraceSTDAPIEnter("MonikerRelativePathToA");
     LPMONIKER pmkSrc;
     LPMONIKER pmkDest;
     LPMONIKER pmkRelPath;
     HRESULT hReturn;
     HRESULT hResult;


     *ppmkRelPathA = NULL;

     hResult = WrapIMonikerWFromA(pmkSrcA, &pmkSrc);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIMonikerWFromA(pmkDestA, &pmkDest);
     if (FAILED(hResult))
           goto Error;


     hReturn = MonikerRelativePathTo(pmkSrc, pmkDest, &pmkRelPath,
fCalledFromMethod);
     if (FAILED(hReturn))
           goto Error1;

     hResult = WrapIMonikerAFromW(pmkRelPath, ppmkRelPathA);

     if (pmkRelPath)
           pmkRelPath->Release();

Error1:
     if (pmkDest)
           pmkDest->Release();

Error:
     if (pmkSrc)
           pmkSrc->Release();

     if (FAILED(hResult))
           hReturn = hResult;
     return hReturn;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    MonikerCommonPrefixWithA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI MonikerCommonPrefixWithA(LPMONIKERA pmkThisA, LPMONIKERA pmkOtherA,
           LPMONIKERA * ppmkCommonA)
{
     TraceSTDAPIEnter("MonikerCommonPrefixWithA");
     LPMONIKER pmkThis;
     LPMONIKER pmkOther;
     LPMONIKER pmkCommon;
     HRESULT hReturn;
     HRESULT hResult;


     *ppmkCommonA = NULL;

     hResult = WrapIMonikerWFromA(pmkThisA, &pmkThis);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIMonikerWFromA(pmkOtherA, &pmkOther);
     if (FAILED(hResult))
           goto Error;

     hReturn = MonikerCommonPrefixWith(pmkThis, pmkOther, &pmkCommon);
     if (FAILED(hReturn))
           goto Error1;

     hResult = WrapIMonikerAFromW(pmkCommon, ppmkCommonA);

     if (pmkCommon)
           pmkCommon->Release();

Error1:
     if (pmkOther)
           pmkOther->Release();

Error:
     if (pmkThis)
           pmkThis->Release();

     if (FAILED(hResult))
           hReturn = hResult;
     return hReturn;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    CreateBindCtxA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI CreateBindCtxA(DWORD reserved, LPBCA * ppbcA)
{
     TraceSTDAPIEnter("CreateBindCtxA");
     LPBC pbc;
     HRESULT hResult;


     *ppbcA = NULL;

     hResult = CreateBindCtx(reserved, &pbc);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIBindCtxAFromW(pbc, ppbcA);

     if (pbc)
           pbc->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    CreateGenericCompositeA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI CreateGenericCompositeA(LPMONIKERA pmkFirstA, LPMONIKERA pmkRestA,
           LPMONIKERA * ppmkCompositeA)
{
     TraceSTDAPIEnter("CreateGenericCompositeA");
     LPMONIKER pmkFirst;
     LPMONIKER pmkRest;
     LPMONIKER pmkComposite;
     HRESULT hResult;


     *ppmkCompositeA = NULL;

     hResult = WrapIMonikerWFromA(pmkFirstA, &pmkFirst);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIMonikerWFromA(pmkRestA, &pmkRest);
     if (FAILED(hResult))
           goto Error;

     hResult = CreateGenericComposite(pmkFirst, pmkRest, &pmkComposite);
     if (FAILED(hResult))
           goto Error1;

     hResult = WrapIMonikerAFromW(pmkComposite, ppmkCompositeA);

     if (pmkComposite)
           pmkComposite->Release();

Error1:
     if (pmkRest)
           pmkRest->Release();

Error:
     if (pmkFirst)
           pmkFirst->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Routine:    GetClassFileA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
STDAPI GetClassFileA(LPCSTR szFilenameA, CLSID * pclsid)
{
     TraceSTDAPIEnter("GetClassFileA");
     LPOLESTR lpszFilename;
     HRESULT hResult;


     hResult = ConvertStringToW(szFilenameA, &lpszFilename);
     if (FAILED(hResult))
           return hResult;

     hResult = GetClassFile(lpszFilename, pclsid);

     ConvertStringFree(lpszFilename);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    CreateFileMonikerA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
STDAPI CreateFileMonikerA(LPSTR lpszPathNameA, LPMONIKERA * ppmkA)
{
     TraceSTDAPIEnter("CreateFileMonikerA");
     LPOLESTR lpszPathName;
     LPMONIKER pmk;
     HRESULT hReturn;
     HRESULT hResult;


     *ppmkA = NULL;

     hResult = ConvertStringToW(lpszPathNameA, &lpszPathName);
     if (FAILED(hResult))
           return hResult;

     hReturn = CreateFileMoniker(lpszPathName, &pmk);
     if (FAILED(hReturn))
           goto Error;

     hResult = WrapIMonikerAFromW(pmk, ppmkA);

     if (pmk)
           pmk->Release();

Error:
     ConvertStringFree(lpszPathName);

     if (FAILED(hResult))
           hReturn = hResult;

     return hReturn;
}
```

```
//
+---------------------------------------------------------------------------
//
//  Routine:    CreateItemMonikerA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI CreateItemMonikerA(LPSTR lpszDelimA, LPSTR lpszItemA,
           LPMONIKERA * ppmkA)
{
     TraceSTDAPIEnter("CreateItemMonikerA");
     LPOLESTR lpszDelim;
     LPOLESTR lpszItem;
     LPMONIKER pmk;
     HRESULT hResult;


     *ppmkA = NULL;

     hResult = ConvertStringToW(lpszDelimA, &lpszDelim);
     if (FAILED(hResult))
           return hResult;

     hResult = ConvertStringToW(lpszItemA, &lpszItem);
     if (FAILED(hResult))
           goto Error;

     hResult = CreateItemMoniker(lpszDelim, lpszItem, &pmk);
     if (FAILED(hResult))
           goto Error1;

     hResult = WrapIMonikerAFromW(pmk, ppmkA);

     if (pmk)
           pmk->Release();

Error1:
     ConvertStringFree(lpszItem);


Error:
     ConvertStringFree(lpszDelim);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
```

```
//  Routine:    CreateAntiMonikerA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI CreateAntiMonikerA(LPMONIKERA * ppmkA)
{
     TraceSTDAPIEnter("CreateAntiMonikerA");
     LPMONIKER pmk;
     HRESULT hResult;


     *ppmkA = NULL;

     hResult = CreateAntiMoniker(&pmk);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIMonikerAFromW(pmk, ppmkA);

     if (pmk)
           pmk->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    CreatePointerMonikerA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI CreatePointerMonikerA(LPUNKNOWN punkA, LPMONIKERA * ppmkA)
{
     TraceSTDAPIEnter("CreatePointerMonikerA");


     LPUNKNOWN punk;
     LPMONIKER pmk;
     HRESULT hResult;


     *ppmkA = NULL;

     hResult = WrapIUnknownWFromA(punkA, &punk);
     if (FAILED(hResult))
```

```
            return hResult;

      hResult = CreatePointerMoniker(punk, &pmk);
      if (FAILED(hResult))
            goto Error;

      hResult = WrapIMonikerAFromW(pmk, ppmkA);

      if (pmk)
            pmk->Release();

Error:
      punk->Release();

      return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    GetRunningObjectTableA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Returns:    See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI GetRunningObjectTableA(DWORD reserved, LPRUNNINGOBJECTTABLEA *
pprotA)
{
      TraceSTDAPIEnter("GetRunningObjectTableA");
      LPRUNNINGOBJECTTABLE prot;
      HRESULT hResult;


      *pprotA = NULL;

      hResult = GetRunningObjectTable(reserved, &prot);
      if (FAILED(hResult))
            return hResult;

      hResult = WrapIRunningObjectTableAFromW(prot, pprotA);

      if (prot)
            prot->Release();

      return hResult;
}
```

ANSIOA.CPP (OLEANSI Sample)

```
//
+---------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//  Information Contained Herein Is Proprietary and Confidential.
//
//  File:       ansioa.cpp
//
//  Contents:   ANSI version of OLE Automation APIs.
//              See OLE2 docs for details of Unicode equalivent APIs.
//
//  Functions:  SysAllocStringA
//              SysAllocStringLenA
//              SysStringLenA
//              SysReAllocStringA
//              SysReAllocStringLenA
//              SysFreeStringA
//
//              SafeArrayAllocDescriptorA
//              SafeArrayAllocDataA
//              SafeArrayCreateA
//              SafeArrayDestroyDescriptorA
//              SafeArrayDestroyDataA
//              SafeArrayDestroyA
//              SafeArrayRedimA
//              SafeArrayGetDimA
//              SafeArrayGetElemsizeA
//              SafeArrayGetUBoundA
//              SafeArrayGetLBoundA
//              SafeArrayLockA
//              SafeArrayUnlockA
//              SafeArrayAccessDataA
//              SafeArrayUnaccessDataA
//              SafeArrayGetElement
//              SafeArrayPutElementA
//              SafeArrayCopyA
//
//              VariantInitA
//              VariantClearA
//              VariantCopyA
//              VariantCopyIndA
//              VariantChangeTypeA
//              VariantChangeTypeExA
//
//              VarUI1FromStrA
//              VarUI1FromDispA
//              VarI2FromStrA
//              VarI2FromDispA
//              VarI4FromStrA
//              VarI4FromDispA
//              VarR4FromStrA
//              VarR4FromDispA
//              VarR8FromStrA
//              VarR8FromDispA
//              VarDateFromStrA
//              VarDateFromDispA
//              VarCyFromStrA
//              VarCyFromDispA
//              VarBstrFromUI1A
//              VarBstrFromI2A
//              VarBstrFromI4A
//              VarBstrFromR4A
//              VarBstrFromR8A
//              VarBstrFromCyA
//              VarBstrFromDateA
//              VarBstrFromDispA
//              VarBstrFromBoolA
//              VarBoolFromStrA
//              VarBoolFromDispA
//
//              LHashValOfNameSysA
//              LoadTypeLibA
//              LoadRegTypeLibA
//              QueryPathOfRegTypeLibA
//              RegisterTypeLibA
//              CreateTypeLibA
//
//              DispGetParamA
//              DispGetIDsOfNamesA
//              DispInvokeA
//              CreateDispTypeInfoA
//              CreateStdDispatchA
//
//  History:    12-Jan-94   tomteng     Created.
//
//--------------------------------------------------------------------------
-

#include "Ole2Ansi.h"

extern "C" int __cdecl _fltused = 1; // avoid dragging in C-runtime

/*----------------------------------------------------------------------*/
/*                       ANSI BSTR API                                  */
/*----------------------------------------------------------------------*/


STDAPI_(BSTRA) SysAllocStringA(const char * psz)
{
     TraceSTDAPIEnter("SysAllocStringA");

     if (psz == NULL)
           return NULL;

     return (BSTRA)SysAllocStringByteLen(psz, lstrlen(psz));
}
```

```
STDAPI_(BSTRA) SysAllocStringLenA(const char * psz, unsigned int len)
{
     TraceSTDAPIEnter("SysAllocStringLenA");

     BSTRA bstr = (BSTRA)SysAllocStringByteLen(psz, len);
     return bstr;
}


STDAPI_(int) SysReAllocStringA(BSTRA * pbstr, const char * psz)
{
     TraceSTDAPIEnter("SysReAllocStringA");

     return SysReAllocStringLenA(pbstr, psz, (psz ? lstrlen(psz) : 0));
}


STDAPI_(int) SysReAllocStringLenA(BSTRA * pbstr, const char * psz, unsigned
int len)
{
     TraceSTDAPIEnter("SysReAllocStringLenA");

     BSTRA bstrNew = (BSTRA) SysAllocStringLenA(psz, len);
     if (bstrNew == NULL)
     return FALSE;

     SysFreeString((BSTR) *pbstr);
     *pbstr = bstrNew;

     return TRUE;
}



STDAPI_(unsigned int) SysStringLenA(BSTRA bstr)
{
     TraceSTDAPIEnter("int");

     return SysStringByteLen((BSTR)bstr);
}


STDAPI_(void) SysFreeStringA(BSTRA bstr)
{
     TraceSTDAPIEnter("SysFreeStringA");

     SysFreeString((BSTR)bstr);
}


/*---------------------------------------------------------------------*/
/*                         ANSI SafeArray API                          */
/*---------------------------------------------------------------------*/
```

```
STDAPI SafeArrayAllocDescriptorA(unsigned int cDims, SAFEARRAYA * * ppsaOut)
{
     TraceSTDAPIEnter("SafeArrayAllocDescriptorA");

     return (SafeArrayAllocDescriptor(cDims, (SAFEARRAY * *) ppsaOut));
}


STDAPI SafeArrayAllocDataA(SAFEARRAYA * psa)
{
     TraceSTDAPIEnter("SafeArrayAllocDataA");

     return (SafeArrayAllocData((SAFEARRAY *) psa));
}


STDAPI_(SAFEARRAYA *) SafeArrayCreateA(
     VARTYPE vt,
     unsigned int cDims,
     SAFEARRAYBOUND * rgsabound)
{
     TraceSTDAPIEnter("SafeArrayCreateA");

     return ((SAFEARRAYA *) SafeArrayCreate(vt, cDims, rgsabound));
}


STDAPI SafeArrayDestroyDescriptorA(SAFEARRAYA * psa)
{
     TraceSTDAPIEnter("SafeArrayDestroyDescriptorA");

     return (SafeArrayDestroyDescriptor((SAFEARRAY *) psa));
}


STDAPI SafeArrayDestroyDataA(SAFEARRAYA * psa)
{
     TraceSTDAPIEnter("SafeArrayDestroyDataA");

     return (SafeArrayDestroyData((SAFEARRAY *) psa));
}


STDAPI SafeArrayDestroyA(SAFEARRAYA * psa)
{
     TraceSTDAPIEnter("SafeArrayDestroyA");

  return (SafeArrayDestroy((SAFEARRAY *) psa));
}


STDAPI SafeArrayRedimA(SAFEARRAYA * psa, SAFEARRAYBOUND * psaboundNew)
{
     TraceSTDAPIEnter("SafeArrayRedimA");
     return (SafeArrayRedim((SAFEARRAY *) psa, psaboundNew));
}


STDAPI_(UINT) SafeArrayGetDimA(SAFEARRAYA * psa)
{
     TraceSTDAPIEnter("SafeArrayGetDimA");

     return (SafeArrayGetDim((SAFEARRAY *) psa));
}


STDAPI_(UINT) SafeArrayGetElemsizeA(SAFEARRAYA * psa)
{
     TraceSTDAPIEnter("SafeArrayGetElemsizeA");

     return (SafeArrayGetElemsize((SAFEARRAY *) psa));
}


STDAPI SafeArrayGetUBoundA(
     SAFEARRAYA * psa,
     UINT nDim,
     long * plUbound)
{
     TraceSTDAPIEnter("SafeArrayGetUBoundA");

     return (SafeArrayGetUBound((SAFEARRAY *) psa, nDim, plUbound));
}


STDAPI SafeArrayGetLBoundA(
     SAFEARRAYA * psa,
     UINT nDim,
     long * plLbound)
{
     TraceSTDAPIEnter("SafeArrayGetLBoundA");

     return (SafeArrayGetLBound((SAFEARRAY *) psa, nDim, plLbound));
}


STDAPI SafeArrayLockA(SAFEARRAYA * psa)
{
     TraceSTDAPIEnter("SafeArrayLockA");

     return (SafeArrayLock((SAFEARRAY *) psa));
}


STDAPI SafeArrayUnlockA(SAFEARRAYA * psa)
{
     TraceSTDAPIEnter("SafeArrayUnlockA");
     return (SafeArrayUnlock((SAFEARRAY *) psa));
}


STDAPI SafeArrayAccessDataA(SAFEARRAYA * psa, void HUGEP* * ppvData)
{
     TraceSTDAPIEnter("SafeArrayAccessDataA");

     return (SafeArrayAccessData((SAFEARRAY *) psa, ppvData));
}


STDAPI SafeArrayUnaccessDataA(SAFEARRAYA * psa)
{
     TraceSTDAPIEnter("SafeArrayUnaccessDataA");

     return (SafeArrayUnaccessData((SAFEARRAY *) psa));
}


STDAPI SafeArrayGetElementA(
     SAFEARRAYA * psa,
     long * rgIndices,
     void * pv)
{
     TraceSTDAPIEnter("SafeArrayGetElementA");

     return (SafeArrayGetElement((SAFEARRAY *) psa, rgIndices, pv));
}


STDAPI SafeArrayPutElementA(
     SAFEARRAYA * psa,
     long * rgIndices,
     void * pv)
{
     TraceSTDAPIEnter("SafeArrayPutElementA");

     return (SafeArrayPutElement((SAFEARRAY *) psa, rgIndices, pv));
}


STDAPI SafeArrayCopyA(
     SAFEARRAYA * psa,
     SAFEARRAYA * * ppsaOut)
{
     TraceSTDAPIEnter("SafeArrayCopyA");

     return (SafeArrayCopy((SAFEARRAY *) psa, (SAFEARRAY * *) ppsaOut));
}



/*---------------------------------------------------------------------*/
```

```
/*                          ANSI VARIANT API                              */
/*---------------------------------------------------------------------*/


STDAPI_(void) VariantInitA(VARIANTARGA * pvarg)
{
     TraceSTDAPIEnter("VariantInitA");

     VariantInit((VARIANTARG *)  pvarg);
}


STDAPI VariantClearA(VARIANTARGA * pvarg)
{
     TraceSTDAPIEnter("VariantClearA");

     return VariantClear((VARIANTARG *) pvarg);
}


STDAPI VariantCopyA(
     VARIANTARGA * pvargDest,
     VARIANTARGA * pvargSrc)
{
     TraceSTDAPIEnter("VariantCopyA");

     return VariantCopy((VARIANTARG *) pvargDest, (VARIANTARG *) pvargSrc);
}


STDAPI VariantCopyIndA(
     VARIANTA * pvargDest,
     VARIANTARGA * pvargSrc)

{
     TraceSTDAPIEnter("VariantCopyIndA");

     return VariantCopyInd((VARIANT *) pvargDest, (VARIANTARG *) pvargSrc);
}

STDAPI VariantChangeTypeA(
     VARIANTARGA * pvargDestA,
     VARIANTARGA * pvargSrcA,
     unsigned short wFlags,
     VARTYPE vt)
{
     TraceSTDAPIEnter("VariantChangeTypeA");

     HRESULT hResult;
     VARIANT varTempW;

     // CONSIDER: optimize this for the case of simple Variants (src, dest,
     // CONSIDER: and vt all non-string, non-array, non-object variants)
     VariantInit(&varTempW);
     hResult = VariantCopy((VARIANT *) &varTempW, (VARIANTARG *) pvargSrcA);
```

```
    if (hResult == NOERROR)
    {
        hResult = ConvertVariantToW(&varTempW);
        if (hResult == NOERROR)
        {
          hResult = VariantChangeType(&varTempW, &varTempW, wFlags, vt);
            if (hResult == NOERROR)
            {
                hResult = ConvertVariantToA((VARIANTA *)&varTempW);
                if (hResult == NOERROR)
                    hResult = VariantCopy((VARIANT *)pvargDestA,
&varTempW);
            }
        }
    }

    VariantClear(&varTempW);
    return hResult;
}


STDAPI VariantChangeTypeExA(
    VARIANTARGA * pvargDestA,
    VARIANTARGA * pvargSrcA,
    LCID lcid,
    unsigned short wFlags,
    VARTYPE vt)
{
    TraceSTDAPIEnter("VariantChangeTypeExA");

    HRESULT hResult;
    VARIANT varTempW;

    // CONSIDER: optimize this for the case of simple Variants (src, dest,
    // CONSIDER: and vt all non-string, non-array, non-object variants)
    VariantInit(&varTempW);
    hResult = VariantCopy((VARIANT *) &varTempW, (VARIANTARG *) pvargSrcA);
    if (hResult == NOERROR)
    {
        hResult = ConvertVariantToW(&varTempW);
        if (hResult == NOERROR)
        {
            hResult = VariantChangeTypeEx(&varTempW, &varTempW, lcid,
wFlags, vt);
            if (hResult == NOERROR)
            {
                hResult = ConvertVariantToA((VARIANTA *)&varTempW);
                if (hResult == NOERROR)
                    hResult = VariantCopy((VARIANT *)pvargDestA,
&varTempW);
            }
        }
    }
```

```
     VariantClear(&varTempW);
     return hResult;
}



/*---------------------------------------------------------------------*/
/*                   ANSI VARIANT Coercion API                         */
/*---------------------------------------------------------------------*/

#ifndef NOI1
STDAPI VarUI1FromStrA(char * strIn, LCID lcid,
                   unsigned long dwFlags, unsigned char * pbOut)
{
     TraceSTDAPIEnter("VarUI1FromStrA");

     LPOLESTR   lpwStrIn;
     HRESULT    hResult;


     hResult = ConvertStringToW(strIn, &lpwStrIn);
     if (FAILED(hResult))
           return (hResult);

     hResult = VarUI1FromStr(lpwStrIn, lcid, dwFlags, pbOut);

     ConvertStringFree(lpwStrIn);

     return hResult;

}

STDAPI VarUI1FromDispA(IDispatchA * pdispIn, LCID lcid, unsigned char *
pbOut)
{
     TraceSTDAPIEnter("VarUI1FromDispA");

     HRESULT    hResult;
     IDispatch* pDisp;

     hResult = WrapIDispatchWFromA(pdispIn, &pDisp);
     if (FAILED(hResult))
           return (hResult);

     hResult = VarUI1FromDisp(pDisp, lcid, pbOut);
     pDisp->Release();

     return hResult;
}
#endif //!NOI1

STDAPI VarI2FromStrA(char * strIn, LCID lcid,
                   unsigned long dwFlags, short * psOut)
{
     TraceSTDAPIEnter("VarI2FromStrA");
     LPOLESTR   lpwStrIn;
     HRESULT    hResult;


     hResult = ConvertStringToW(strIn, &lpwStrIn);
     if (FAILED(hResult))
           return (hResult);

     hResult = VarI2FromStr(lpwStrIn, lcid, dwFlags, psOut);

     ConvertStringFree(lpwStrIn);

     return hResult;

}


STDAPI VarI2FromDispA(IDispatchA * pdispIn, LCID lcid, short * psOut)
{
     TraceSTDAPIEnter("VarI2FromDispA");

     HRESULT    hResult;
     IDispatch* pDisp;

     hResult = WrapIDispatchWFromA(pdispIn, &pDisp);
     if (FAILED(hResult))
           return (hResult);

     hResult = VarI2FromDisp(pDisp, lcid, psOut);
     pDisp->Release();

     return hResult;
}


STDAPI VarI4FromStrA(char * strIn, LCID lcid,
                   unsigned long dwFlags, long * plOut)
{
     TraceSTDAPIEnter("VarI4FromStrA");

     LPOLESTR   lpwStrIn;
     HRESULT    hResult;


     hResult = ConvertStringToW(strIn, &lpwStrIn);
     if (FAILED(hResult))
           return (hResult);

     hResult = VarI4FromStr(lpwStrIn, lcid, dwFlags, plOut);

     ConvertStringFree(lpwStrIn);

     return hResult;
```

```
}


STDAPI VarI4FromDispA(IDispatchA * pdispIn, LCID lcid, long * plOut)
{
     TraceSTDAPIEnter("VarI4FromDispA");


     HRESULT    hResult;
     IDispatch* pDisp;

     hResult = WrapIDispatchWFromA(pdispIn, &pDisp);
     if (FAILED(hResult))
           return (hResult);

     hResult = VarI4FromDisp(pDisp, lcid, plOut);
     pDisp->Release();

     return hResult;
}


STDAPI VarR4FromStrA(char * strIn, LCID lcid,
                              unsigned long dwFlags, float * pfltOut)
{
     TraceSTDAPIEnter("VarR4FromStrA");

     LPOLESTR   lpwStrIn;
     HRESULT    hResult;


     hResult = ConvertStringToW(strIn, &lpwStrIn);
     if (FAILED(hResult))
           return (hResult);

     hResult = VarR4FromStr(lpwStrIn, lcid, dwFlags, pfltOut);

     ConvertStringFree(lpwStrIn);

     return hResult;

}


STDAPI VarR4FromDispA(IDispatchA * pdispIn, LCID lcid, float * pfltOut)
{
     TraceSTDAPIEnter("VarR4FromDispA");

     HRESULT    hResult;
     IDispatch* pDisp;

     hResult = WrapIDispatchWFromA(pdispIn, &pDisp);
     if (FAILED(hResult))
           return (hResult);
     hResult = VarR4FromDisp(pDisp, lcid, pfltOut);
     pDisp->Release();

     return hResult;
}


STDAPI VarR8FromStrA(char * strIn, LCID lcid,
                   unsigned long dwFlags, double * pdblOut)
{
     TraceSTDAPIEnter("VarR8FromStrA");

     LPOLESTR   lpwStrIn;
     HRESULT    hResult;


     hResult = ConvertStringToW(strIn, &lpwStrIn);
     if (FAILED(hResult))
           return (hResult);

     hResult = VarR8FromStr(lpwStrIn, lcid, dwFlags, pdblOut);

     ConvertStringFree(lpwStrIn);

     return hResult;

}


STDAPI VarR8FromDispA(IDispatchA * pdispIn, LCID lcid, double * pdblOut)
{
     TraceSTDAPIEnter("VarR8FromDispA");

     HRESULT    hResult;
     IDispatch* pDisp;

     hResult = WrapIDispatchWFromA(pdispIn, &pDisp);
     if (FAILED(hResult))
           return (hResult);

     hResult = VarR8FromDisp(pDisp, lcid, pdblOut);
     pDisp->Release();

     return hResult;
}


STDAPI VarDateFromStrA(char * strIn, LCID lcid,
                       unsigned long dwFlags, DATE * pdateOut)
{
     TraceSTDAPIEnter("VarDateFromStrA");

     LPOLESTR   lpwStrIn;
     HRESULT    hResult;
     hResult = ConvertStringToW(strIn, &lpwStrIn);
     if (FAILED(hResult))
           return (hResult);

     hResult = VarDateFromStr(lpwStrIn, lcid, dwFlags, pdateOut);

     ConvertStringFree(lpwStrIn);

     return hResult;

}


STDAPI VarDateFromDispA(IDispatchA * pdispIn, LCID lcid, DATE * pdateOut)
{
     TraceSTDAPIEnter("VarDateFromDispA");

     HRESULT    hResult;
     IDispatch* pDisp;


     hResult = WrapIDispatchWFromA(pdispIn, &pDisp);
     if (FAILED(hResult))
           return (hResult);

     hResult = VarDateFromDisp(pDisp, lcid, pdateOut);
     pDisp->Release();

     return hResult;
}


STDAPI VarCyFromStrA(char * strIn, LCID lcid,
                   unsigned long dwFlags, CY * pcyOut)
{
     TraceSTDAPIEnter("VarCyFromStrA");

     LPOLESTR   lpwStrIn;
     HRESULT    hResult;



     hResult = ConvertStringToW(strIn, &lpwStrIn);
     if (FAILED(hResult))
           return (hResult);

     hResult = VarCyFromStr(lpwStrIn, lcid, dwFlags, pcyOut);

     ConvertStringFree(lpwStrIn);

     return hResult;

}
```

```
STDAPI VarCyFromDispA(IDispatchA * pdispIn, LCID lcid, CY * pcyOut)
{
     TraceSTDAPIEnter("VarCyFromDispA");

     HRESULT    hResult;
     IDispatch* pDisp;


     hResult = WrapIDispatchWFromA(pdispIn, &pDisp);
     if (FAILED(hResult))
           return (hResult);

     hResult = VarCyFromDisp(pDisp, lcid, pcyOut);
     pDisp->Release();

     return hResult;

}


#ifndef NOI1
STDAPI VarBstrFromUI1A(unsigned char bVal, LCID lcid,
                    unsigned long dwFlags, BSTRA * pbstrOut)
{
     TraceSTDAPIEnter("VarBstrFromUI1A");

     HRESULT    hResult;
     BSTR       bstr;

     hResult = VarBstrFromUI1(bVal, lcid, dwFlags, &bstr);
     if (FAILED(hResult))
           return (hResult);

     hResult = ConvertDispStringToA(bstr, pbstrOut);
     SysFreeString(bstr);

     return hResult;
}
#endif //!NOI1


STDAPI VarBstrFromI2A(short iVal, LCID lcid,
                    unsigned long dwFlags, BSTRA * pbstrOut)
{
     TraceSTDAPIEnter("VarBstrFromI2A");

     HRESULT    hResult;
     BSTR       bstr;

     hResult = VarBstrFromI2(iVal, lcid, dwFlags, &bstr);
     if (FAILED(hResult))
           return (hResult);

     hResult = ConvertDispStringToA(bstr, pbstrOut);
```

```
    SysFreeString(bstr);

    return hResult;
}


STDAPI VarBstrFromI4A(long lIn, LCID lcid,
                    unsigned long dwFlags, BSTRA * pbstrOut)
{
    TraceSTDAPIEnter("VarBstrFromI4A");

    HRESULT    hResult;
    BSTR       bstr;

    hResult = VarBstrFromI4(lIn, lcid, dwFlags, &bstr);
    if (FAILED(hResult))
        return (hResult);

    hResult = ConvertDispStringToA(bstr, pbstrOut);
    SysFreeString(bstr);

    return hResult;
}


STDAPI VarBstrFromR4A(float fltIn, LCID lcid,
                    unsigned long dwFlags, BSTRA * pbstrOut)
{
    TraceSTDAPIEnter("VarBstrFromR4A");

    HRESULT    hResult;
    BSTR       bstr;

    hResult = VarBstrFromR4(fltIn, lcid, dwFlags, &bstr);
    if (FAILED(hResult))
        return (hResult);

    hResult = ConvertDispStringToA(bstr, pbstrOut);
    SysFreeString(bstr);

    return hResult;
}


STDAPI VarBstrFromR8A(double dblIn, LCID lcid,
                    unsigned long dwFlags, BSTRA * pbstrOut)
{
    TraceSTDAPIEnter("VarBstrFromR8A");

    HRESULT    hResult;
    BSTR       bstr;

    hResult = VarBstrFromR8(dblIn, lcid, dwFlags, &bstr);
    if (FAILED(hResult))
        return (hResult);
    hResult = ConvertDispStringToA(bstr, pbstrOut);
    SysFreeString(bstr);

    return hResult;
}


STDAPI VarBstrFromCyA(CY cyIn, LCID lcid,
                           unsigned long dwFlags, BSTRA * pbstrOut)
{
    TraceSTDAPIEnter("VarBstrFromCyA");

    HRESULT    hResult;
    BSTR       bstr;

    hResult = VarBstrFromCy(cyIn, lcid, dwFlags, &bstr);
    if (FAILED(hResult))
         return (hResult);

    hResult = ConvertDispStringToA(bstr, pbstrOut);
    SysFreeString(bstr);

    return hResult;
}



STDAPI VarBstrFromDateA(DATE dateIn, LCID lcid,
                      unsigned long dwFlags, BSTRA * pbstrOut)
{
    TraceSTDAPIEnter("VarBstrFromDateA");

    HRESULT    hResult;
    BSTR       bstr;

    hResult = VarBstrFromDate(dateIn, lcid, dwFlags, &bstr);
    if (FAILED(hResult))
         return (hResult);

    hResult = ConvertDispStringToA(bstr, pbstrOut);
    SysFreeString(bstr);

    return hResult;
}


STDAPI VarBstrFromDispA(IDispatchA * pdispIn, LCID lcid,
                      unsigned long dwFlags, BSTRA * pbstrOut)
{
    TraceSTDAPIEnter("VarBstrFromDispA");

    HRESULT    hResult;
    BSTR       bstr;
    IDispatch* pDisp;
```

```
	hResult = WrapIDispatchWFromA(pdispIn, &pDisp);
	if (FAILED(hResult))
		return (hResult);

	hResult = VarBstrFromDisp(pDisp, lcid, dwFlags, &bstr);
	if (FAILED(hResult))
		return (hResult);

	pDisp->Release();

	hResult = ConvertDispStringToA(bstr, pbstrOut);
	SysFreeString(bstr);

	return hResult;
}


STDAPI VarBstrFromBoolA(VARIANT_BOOL boolIn, LCID lcid,
                        unsigned long dwFlags, BSTRA * pbstrOut)
{
	TraceSTDAPIEnter("VarBstrFromBoolA");

	HRESULT    hResult;
	BSTR       bstr;

	hResult = VarBstrFromBool(boolIn, lcid, dwFlags, &bstr);
	if (FAILED(hResult))
		return (hResult);

	hResult = ConvertDispStringToA(bstr, pbstrOut);
	SysFreeString(bstr);

	return hResult;
}


STDAPI VarBoolFromStrA(char * strIn, LCID lcid,
                       unsigned long dwFlags, VARIANT_BOOL * pboolOut)
{
	TraceSTDAPIEnter("VarBoolFromStrA");

	LPOLESTR   lpwStrIn;
	HRESULT    hResult;


	hResult = ConvertStringToW(strIn, &lpwStrIn);
	if (FAILED(hResult))
		return (hResult);

	hResult = VarBoolFromStr(lpwStrIn, lcid, dwFlags, pboolOut);

	ConvertStringFree(lpwStrIn);
```

```
     return hResult;

}


STDAPI VarBoolFromDispA(IDispatchA * pdispIn, LCID lcid,
                        VARIANT_BOOL * pboolOut)
{
     TraceSTDAPIEnter("VarBoolFromDispA");

     HRESULT    hResult;
     IDispatch* pDisp;

     hResult = WrapIDispatchWFromA(pdispIn, &pDisp);
     if (FAILED(hResult))
           return (hResult);

     hResult = VarBoolFromDisp(pDisp, lcid, pboolOut);
     pDisp->Release();

     return hResult;
}


/*---------------------------------------------------------------------*/
/*                         ANSI Typelib API                            */
/*---------------------------------------------------------------------*/



STDAPI LoadTypeLibA(const char * szFileA, ITypeLibA * * pptlib)
{
     TraceSTDAPIEnter("LoadTypeLibA");

     LPOLESTR   lpszFile;
     LPTYPELIB  pTypeLib;
     HRESULT    hResult;


     hResult = ConvertStringToW(szFileA, &lpszFile);
     if (FAILED(hResult))
           return (hResult);

     hResult = LoadTypeLib(lpszFile, &pTypeLib);
     if (FAILED(hResult))
           goto Error;

     hResult = WrapITypeLibAFromW(pTypeLib, pptlib);

     pTypeLib->Release();

Error:
     ConvertStringFree(lpszFile);

     return hResult;
```

```
}


STDAPI LoadRegTypeLibA(REFGUID rguid, unsigned short wVerMajor,
           unsigned short wVerMinor, LCID lcid, ITypeLibA * * pptlibA)
{
     TraceSTDAPIEnter("LoadRegTypeLibA");

     LPTYPELIB  pTypeLib;
     HRESULT    hResult;


     hResult = LoadRegTypeLib(rguid, wVerMajor, wVerMinor, lcid, &pTypeLib);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapITypeLibAFromW(pTypeLib, pptlibA);

     pTypeLib->Release();

     return hResult;
}


STDAPI QueryPathOfRegTypeLibA(REFGUID guid, unsigned short wMaj,
           unsigned short wMin, LCID lcid, LPBSTRA lpbstrPathNameA)
{
     TraceSTDAPIEnter("QueryPathOfRegTypeLibA");

     BSTR       bstrPathName;
     HRESULT    hResult;


     hResult = QueryPathOfRegTypeLib(guid, wMaj, wMin, lcid,
                 &bstrPathName);
     if (FAILED(hResult))
           return (hResult);

     hResult = ConvertDispStringToA(bstrPathName, lpbstrPathNameA);

     SysFreeString(bstrPathName);

     return hResult;
}


STDAPI RegisterTypeLibA(ITypeLibA * ptlibA, char * szFullPathA,
           char * szHelpDirA)
{
     TraceSTDAPIEnter("RegisterTypeLibA");

     LPTYPELIB pTypeLib;
     LPOLESTR  lpszFullPath;
     LPOLESTR  lpszHelpDir;
     HRESULT   hResult;


     hResult = WrapITypeLibWFromA(ptlibA, &pTypeLib);
     if (FAILED(hResult))
           return (hResult);

     hResult = ConvertStringToW(szFullPathA, &lpszFullPath);
     if (FAILED(hResult))
           goto Error;

     hResult = ConvertStringToW(szHelpDirA, &lpszHelpDir);
     if (FAILED(hResult))
           goto Error1;

     hResult = RegisterTypeLib(pTypeLib, lpszFullPath, lpszHelpDir);

     ConvertStringFree(lpszHelpDir);

Error1:
     ConvertStringFree(lpszFullPath);

Error:
     pTypeLib->Release();

     return hResult;
}


STDAPI CreateTypeLibA(SYSKIND syskind, const char * szFileA,
           ICreateTypeLibA * * ppctlibA)
{
     TraceSTDAPIEnter("CreateTypeLibA");

     LPOLESTR         lpszFile;
     LPCREATETYPELIB  pCreateTypeLib;
     HRESULT          hResult;


     hResult = ConvertStringToW(szFileA, &lpszFile);
     if (FAILED(hResult))
           return (hResult);

     hResult = CreateTypeLib(syskind, lpszFile, &pCreateTypeLib);
     if (FAILED(hResult))
           goto Error;

     hResult = WrapICreateTypeLibAFromW(pCreateTypeLib, ppctlibA);

     pCreateTypeLib->Release();

Error:
     ConvertStringFree(lpszFile);

     return hResult;
```

```
}


STDAPI_(unsigned long) LHashValOfNameSysHACK(SYSKIND syskind, LCID lcid,
const char FAR* szName)
{
     TraceSTDAPIEnter("LHashValOfNameSysHACK");

     return LHashValOfNameSysA(syskind, lcid, szName);
}


/*---------------------------------------------------------------------*/
/*                        ANSI Dispatch API                            */
/*---------------------------------------------------------------------*/

STDAPI DispGetParamA(DISPPARAMSA * pdispparamsA, UINT position,
           VARTYPE vtTarg, VARIANTA * pvarResult, UINT * puArgErr)
{
     TraceSTDAPIEnter("DispGetParamA");

     LPDISPPARAMS pDispParams;
     LPVARIANT    pByrefTemp;
     HRESULT hResult;



     hResult = ConvertDispParamsToW(pdispparamsA, &pDispParams,
&pByrefTemp);
     if (FAILED(hResult))
           return hResult;

     hResult = DispGetParam(pDispParams, position, vtTarg,
                                 (LPVARIANT)pvarResult, puArgErr);
     if (FAILED(hResult))
           goto Error;

     hResult = ConvertVariantToA(pvarResult);

Error:
     ConvertDispParamsFree(pDispParams, pByrefTemp);

     return hResult;
}


STDAPI DispGetIDsOfNamesA(ITypeInfoA * ptinfoA, char * * rgszNamesA,
           UINT cNames, DISPID * rgdispid)
{
     TraceSTDAPIEnter("DispGetIDsOfNamesA");

     LPTYPEINFO pTypeInfo;
     LPOLESTR * rgszNames;
     HRESULT    hResult;
     hResult = WrapITypeInfoWFromA(ptinfoA, &pTypeInfo);
     if (FAILED(hResult))
           return (hResult);

     hResult = ConvertStringArrayToW(rgszNamesA, &rgszNames, cNames);
     if (FAILED(hResult))
           goto Error;

     hResult = DispGetIDsOfNames(pTypeInfo, rgszNames, cNames, rgdispid);

     ConvertStringArrayFree(rgszNames, cNames);

Error:
     pTypeInfo->Release();

     return hResult;
}


STDAPI DispInvokeA(void * _this, ITypeInfoA * ptinfoA, DISPID dispidMember,
           unsigned short wFlags, DISPPARAMSA * pparamsA, VARIANTA *
pvarResultA,
           EXCEPINFOA * pexcepinfoA, UINT * puArgErr)
{
     TraceSTDAPIEnter("DispInvokeA");

     LPTYPEINFO   pTypeInfo;
     LPDISPPARAMS pDispParams;
     LPVARIANT    pByrefTemp;
     HRESULT      hResult;


     hResult = WrapITypeInfoWFromA(ptinfoA, &pTypeInfo);
     if (FAILED(hResult))
           return (hResult);

     hResult = ConvertDispParamsToW(pparamsA, &pDispParams, &pByrefTemp);
     if (FAILED(hResult))
           goto Error;

     hResult = DispInvoke(_this, pTypeInfo, dispidMember, wFlags,
pDispParams,
                        (LPVARIANT)pvarResultA, (EXCEPINFO *)pexcepinfoA,
puArgErr);

        if (hResult == DISP_E_EXCEPTION)
            ConvertExcepInfoToA(pexcepinfoA);

     if (FAILED(hResult))
           goto Error1;

     if (pByrefTemp)
     {
           hResult = ConvertDispParamsCopyBack(pparamsA, pByrefTemp);
           if (FAILED(hResult))
                 goto Error1;
     }

     hResult = ConvertVariantToA(pvarResultA);

Error1:
     ConvertDispParamsFree(pDispParams, pByrefTemp);

Error:
     pTypeInfo->Release();

     return hResult;
}


STDAPI CreateDispTypeInfoA(INTERFACEDATAA * pidataA, LCID lcid,
           ITypeInfoA * * pptinfoA)
{
     TraceSTDAPIEnter("CreateDispTypeInfoA");

     INTERFACEDATA * pData;
     LPTYPEINFO  pTypeInfo;
     HRESULT    hResult;


     hResult = ConvertInterfaceDataToW(pidataA, lcid, &pData);
     if (FAILED(hResult))
           return (hResult);

     hResult = CreateDispTypeInfo(pData, lcid, &pTypeInfo);
     if (FAILED(hResult))
           goto Error;

     hResult = WrapITypeInfoAFromW(pTypeInfo, pptinfoA);

     pTypeInfo->Release();

     if (FAILED(hResult))
           goto Error;

     //
     //  Save the Unicode InterfaceData structure in the TypeInfo wrapper,
     //  to be delete on pTypeInfo is released.
     //
     ((CInterface *)*pptinfoA)->SetInterfaceData(pData);

     return hResult;

Error:
     ConvertInterfaceDataFree(pData);

     return hResult;
}
```

```
STDAPI CreateStdDispatchA(IUnknown * punkOuterA, void * pvThis,
           ITypeInfoA * ptinfoA, IUnknown * * ppunkStdDisp)
{
     TraceSTDAPIEnter("CreateStdDispatchA");

     LPTYPEINFO pTypeInfo;
     LPUNKNOWN  punkDisp;
     IDispatch* pDispatch;
     LPINTERFACEDATA pidata;
     HRESULT    hResult;


     hResult = WrapITypeInfoWFromA(ptinfoA, &pTypeInfo);
     if (FAILED(hResult))
           return hResult;

     hResult = CreateStdDispatch(punkOuterA, pvThis, pTypeInfo,
                       &punkDisp);
     if (FAILED(hResult))
           goto Error;

     punkDisp->QueryInterface( IID_IDispatch, (LPVOID *)&pDispatch );
     punkDisp->Release();
     if (FAILED(hResult))
           goto Error;

     hResult = WrapIDispatchAFromW(pDispatch, (IDispatchA **)ppunkStdDisp);

     pDispatch->Release();

     if (FAILED(hResult))
           goto Error;

     //
     //  Move any interface data from the TypeInto to the Dispatch wrapper.
     //  This solves the problem of the TypeInfo wrapper being deleted
     //  before the Dispatch wrapper.
     //
     pidata = ((CInterface *)ptinfoA)->GetInterfaceData();
     ((CInterface *)ptinfoA)->SetInterfaceData(0);
     ((CInterface *)*ppunkStdDisp)->SetInterfaceData(pidata);

Error:
     pTypeInfo->Release();

     return hResult;
}

/*---------------------------------------------------------------------*/
/*                    Active Object Registration API                   */
/*---------------------------------------------------------------------*/

STDAPI RegisterActiveObjectA(IUnknown * punkA, REFCLSID rclsid,
           unsigned long dwFlags, unsigned long * pdwRegister)
{
     TraceSTDAPIEnter("RegisterActiveObjectA");

     LPUNKNOWN pUnk;
     HRESULT hResult;


     hResult = WrapIUnknownWFromA(punkA, &pUnk);
     if (FAILED(hResult))
           return hResult;

     hResult = RegisterActiveObject(pUnk, rclsid, dwFlags, pdwRegister);

     pUnk->Release();

     return hResult;
}


STDAPI GetActiveObjectA(REFCLSID rclsid, void * pvReserved,
           IUnknown * FAR* ppunkA)
{
     TraceSTDAPIEnter("GetActiveObjectA");

     LPUNKNOWN pUnk;
     HRESULT hResult;


     hResult = GetActiveObject(rclsid, pvReserved, &pUnk);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIUnknownAFromW(pUnk, ppunkA);

     pUnk->Release();

     return hResult;
}

// only required for Chicago support, since Chicago doesn't support
forwarders
// but available on all platforms

STDAPI DispGetIDsOfNamesX(ITypeInfo * ptinfo, OLECHAR * * rgszNames,
           UINT cNames, DISPID * rgdispid)
{
     return DispGetIDsOfNames(ptinfo, rgszNames, cNames, rgdispid);
}

STDAPI LoadTypeLibX(OLECHAR * szFile, ITypeLib * * pptlib)
{
     return LoadTypeLib(szFile, pptlib);
}
```

```
STDAPI RegisterTypeLibX(ITypeLib * ptlib, OLECHAR * szFullPath,
          OLECHAR * szHelpDir)
{
     return RegisterTypeLib(ptlib, szFullPath, szHelpDir);
}

STDAPI VariantChangeTypeX(
     VARIANTARG * pvargDest,
     VARIANTARG * pvargSrc,
     unsigned short wFlags,
     VARTYPE vt)
{
     return VariantChangeType(pvargDest, pvargSrc, wFlags, vt);
}

STDAPI_(unsigned long) LHashValOfNameSysX(SYSKIND syskind, LCID lcid, const
OLECHAR * szName)
{
     return LHashValOfNameSys(syskind, lcid, szName);
}

STDAPI DispInvokeX(void * _this, ITypeInfo * ptinfo, DISPID dispidMember,
          unsigned short wFlags, DISPPARAMS * pparams, VARIANT *
pvarResult,
          EXCEPINFO * pexcepinfo, UINT * puArgErr)

{
     return DispInvoke(_this, ptinfo, dispidMember, wFlags, pparams,
pvarResult, pexcepinfo, puArgErr);
}

STDAPI_(BSTR) SysAllocStringLenW(const OLECHAR FAR* sz, unsigned int len)
{
     return SysAllocStringLen(sz, len);
}
```

ANSIOLE.H (OLEANSI Sample)

```
//
+-------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       ansiole1.h
//
//  Contents:   ANSI Wrappers for Unicode Ole2 Interfaces and APIs.
//
//  Classes:    CDropTargetA
//              CPersistStorageA
//              CPersistStreamA
//              CPersistFileA
//              CEnumOLEVERBA
//              COleObjectA
//              COleClientSiteA
//              CRunnableObjectA
//              COleItemCOntainerA
//              COleAdviseHolderA
//              COleLinkA
//              COleInPlaceObjectA
//              COleInPlaceActiveObjectA
//              COleInPlaceFrameA
//              COleInPlaceSiteA
//
//  History:    01-Nov-93   v-kentc     Created.
//
//--------------------------------------------------------------------------
-



#if defined(__cplusplus)
interface IOleClientSiteA;
interface IOleItemContainerA;
interface IOleInPlaceFrameA;
#else
typedef interface IOleClientSiteA IOleClientSiteA;
typedef interface IOleItemContainerA IOleItemContainerA;
typedef interface IOleInPlaceFrameA IOleInPlaceFrameA;
#endif

typedef IOleClientSiteA * LPOLECLIENTSITEA;
typedef IOleItemContainerA * LPOLEITEMCONTAINERA;
typedef IOleInPlaceFrameA * LPOLEINPLACEFRAMEA;

#define LPPARSEDISPLAYNAMEA     LPOLEITEMCONTAINERA
#define LPOLECONTAINERA         LPOLEITEMCONTAINERA

#define LPOLEWINDOWA            LPOLEINPLACEFRAMEA
#define LPOLEINPLACEUIWINDOWA   LPOLEINPLACEFRAMEA
```

```
#define LPPERSISTSTREAMA        LPMONIKERA




/*---------------------------------------------------------------------*/
/*                         IEnumOLEVERBA                               */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IEnumOLEVERBA

DECLARE_INTERFACE_(IEnumOLEVERBA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IEnumOLEVERB methods ***
     STDMETHOD(Next) (THIS_ ULONG celt, LPOLEVERBA rgelt, ULONG *
pceltFetched) PURE;
     STDMETHOD(Skip) (THIS_ ULONG celt) PURE;
     STDMETHOD(Reset) (THIS) PURE;
     STDMETHOD(Clone) (THIS_ IEnumOLEVERBA * * ppenm) PURE;
};
typedef IEnumOLEVERBA * LPENUMOLEVERBA;


/*---------------------------------------------------------------------*/
/*                         IOleObjectA                                 */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IOleObjectA

DECLARE_INTERFACE_(IOleObjectA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IOleObject methods ***
     STDMETHOD(SetClientSite) (THIS_ LPOLECLIENTSITEA pClientSiteA) PURE;
     STDMETHOD(GetClientSite) (THIS_ LPOLECLIENTSITEA * ppCLientSiteA) PURE;
     STDMETHOD(SetHostNames) (THIS_ LPCSTR szContainerApp, LPCSTR
szContainerObj) PURE;
     STDMETHOD(Close) (THIS_ DWORD dwSaveOption) PURE;
     STDMETHOD(SetMoniker) (THIS_ DWORD dwWhichMoniker, LPMONIKERA pmkA)
PURE;
     STDMETHOD(GetMoniker) (THIS_ DWORD dwAssign, DWORD dwWhichMoniker,
                       LPMONIKERA * ppmkA) PURE;
     STDMETHOD(InitFromData) (THIS_ LPDATAOBJECTA pDataObjectA,
                       BOOL fCreation,
                       DWORD dwReserved) PURE;
     STDMETHOD(GetClipboardData) (THIS_ DWORD dwReserved,
                       LPDATAOBJECTA * ppDataObjectA) PURE;
     STDMETHOD(DoVerb) (THIS_ LONG iVerb,
                       LPMSG lpmsg,
                       LPOLECLIENTSITEA pActiveSiteA,
                       LONG lindex,
                       HWND hwndParent,
                       LPCRECT lprcPosRect) PURE;
     STDMETHOD(EnumVerbs) (THIS_ LPENUMOLEVERBA * ppenumOleVerbA) PURE;
     STDMETHOD(Update) (THIS) PURE;
     STDMETHOD(IsUpToDate) (THIS) PURE;
     STDMETHOD(GetUserClassID) (THIS_ CLSID * pClsid) PURE;
     STDMETHOD(GetUserType) (THIS_ DWORD dwFormOfType, LPSTR * pszUserType)
PURE;
     STDMETHOD(SetExtent) (THIS_ DWORD dwDrawAspect, LPSIZEL lpsizel) PURE;
     STDMETHOD(GetExtent) (THIS_ DWORD dwDrawAspect, LPSIZEL lpsizel) PURE;

     STDMETHOD(Advise)(THIS_ LPADVISESINKA pAdvSinkA, DWORD * pdwConnection)
PURE;
     STDMETHOD(Unadvise)(THIS_ DWORD dwConnection) PURE;
     STDMETHOD(EnumAdvise) (THIS_ LPENUMSTATDATAA * ppenumAdviseA) PURE;
     STDMETHOD(GetMiscStatus) (THIS_ DWORD dwAspect, DWORD * pdwStatus)
PURE;
     STDMETHOD(SetColorScheme) (THIS_ LPLOGPALETTE lpLogpal) PURE;
};
typedef IOleObjectA * LPOLEOBJECTA;


/*---------------------------------------------------------------------*/
/*                         IOleClientSiteA                             */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IOleClientSiteA

DECLARE_INTERFACE_(IOleClientSiteA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IOleClientSite methods ***
     STDMETHOD(SaveObject) (THIS) PURE;
     STDMETHOD(GetMoniker) (THIS_ DWORD dwAssign, DWORD dwWhichMoniker,
                       LPMONIKERA * ppmkA) PURE;
     STDMETHOD(GetContainer) (THIS_ LPOLECONTAINERA * ppContainerA) PURE;
     STDMETHOD(ShowObject) (THIS) PURE;
     STDMETHOD(OnShowWindow) (THIS_ BOOL fShow) PURE;
     STDMETHOD(RequestNewObjectLayout) (THIS) PURE;
};
typedef IOleClientSiteA * LPOLECLIENTSITEA;
```

```
/*---------------------------------------------------------------------*/
/*                         IRunnableObjectA                            */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IRunnableObjectA

DECLARE_INTERFACE_(IRunnableObjectA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IRunnableObject methods ***
     STDMETHOD(GetRunningClass) (THIS_ LPCLSID lpClsid) PURE;
     STDMETHOD(Run) (THIS_ LPBINDCTXA pbcA) PURE;
     STDMETHOD_(BOOL, IsRunning) (THIS) PURE;
     STDMETHOD(LockRunning)(THIS_ BOOL fLock, BOOL fLastUnlockCloses) PURE;
     STDMETHOD(SetContainedObject)(THIS_ BOOL fContained) PURE;
};
typedef IRunnableObjectA * LPRUNNABLEOBJECTA;


/*---------------------------------------------------------------------*/
/*                         IOleItemContainerA                          */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IOleItemContainerA

DECLARE_INTERFACE_(IOleItemContainerA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IParseDisplayName method ***
     STDMETHOD(ParseDisplayName) (THIS_ LPBCA pbcA, LPSTR lpszDisplayName,
           ULONG * pchEaten, LPMONIKERA * ppmkOutA) PURE;

     // *** IOleContainer methods ***
     STDMETHOD(EnumObjects) (THIS_ DWORD grfFlags, LPENUMUNKNOWN *
ppenumUnknown) PURE;
     STDMETHOD(LockContainer) (THIS_ BOOL fLock) PURE;

     // *** IOleItemContainer methods ***
     STDMETHOD(GetObject) (THIS_ LPSTR lpszItem, DWORD dwSpeedNeeded,
           LPBINDCTXA pbcA, REFIID riid, LPVOID * ppvObject) PURE;
     STDMETHOD(GetObjectStorage) (THIS_ LPSTR lpszItem, LPBINDCTXA pbcA,
           REFIID riid, LPVOID * ppvStorage) PURE;
     STDMETHOD(IsRunning) (THIS_ LPSTR lpszItem) PURE;
};
```

```
typedef IOleItemContainerA * LPOLEITEMCONTAINERA;


/*---------------------------------------------------------------------*/
/*                        IOleAdviseHolderA                            */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IOleAdviseHolderA

DECLARE_INTERFACE_(IOleAdviseHolderA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppv) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IOleAdviseHolder methods ***
     STDMETHOD(Advise)(THIS_ LPADVISESINKA pAdviseA, DWORD * pdwConnection)
PURE;
     STDMETHOD(Unadvise)(THIS_ DWORD dwConnection) PURE;
     STDMETHOD(EnumAdvise)(THIS_ LPENUMSTATDATAA * ppenumAdviseA) PURE;

     STDMETHOD(SendOnRename)(THIS_ LPMONIKERA pmkA) PURE;
     STDMETHOD(SendOnSave)(THIS) PURE;
     STDMETHOD(SendOnClose)(THIS) PURE;
};
typedef IOleAdviseHolderA * LPOLEADVISEHOLDERA;


/*---------------------------------------------------------------------*/
/*                           IOleLinkA                                 */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IOleLinkA

DECLARE_INTERFACE_(IOleLinkA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IOleLink methods ***
     STDMETHOD(SetUpdateOptions) (THIS_ DWORD dwUpdateOpt) PURE;
     STDMETHOD(GetUpdateOptions) (THIS_ LPDWORD pdwUpdateOpt) PURE;
     STDMETHOD(SetSourceMoniker) (THIS_ LPMONIKERA pmkA, REFCLSID rclsid)
PURE;
     STDMETHOD(GetSourceMoniker) (THIS_ LPMONIKERA * ppmkA) PURE;
     STDMETHOD(SetSourceDisplayName) (THIS_ LPCSTR lpszDisplayName) PURE;
     STDMETHOD(GetSourceDisplayName) (THIS_ LPSTR * lplpszDisplayName) PURE;
     STDMETHOD(BindToSource) (THIS_ DWORD bindflags, LPBINDCTXA pbcA) PURE;
     STDMETHOD(BindIfRunning) (THIS) PURE;
     STDMETHOD(GetBoundSource) (THIS_ LPUNKNOWN * ppUnk) PURE;
     STDMETHOD(UnbindSource) (THIS) PURE;
     STDMETHOD(Update) (THIS_ LPBINDCTXA pbcA) PURE;
};
typedef IOleLinkA * LPOLELINKA;


/*---------------------------------------------------------------------*/
/*                        IOleInPlaceObjectA                           */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IOleInPlaceObjectA

DECLARE_INTERFACE_(IOleInPlaceObjectA, IOleWindow)
{
   // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IOleWindow methods ***
     STDMETHOD(GetWindow) (THIS_ HWND * lphwnd) PURE;
     STDMETHOD(ContextSensitiveHelp) (THIS_ BOOL fEnterMode) PURE;

     // *** IOleInPlaceObject methods ***
     STDMETHOD(InPlaceDeactivate) (THIS) PURE;
     STDMETHOD(UIDeactivate) (THIS) PURE;
     STDMETHOD(SetObjectRects) (THIS_ LPCRECT lprcPosRect, LPCRECT
lprcClipRect) PURE;
     STDMETHOD(ReactivateAndUndo) (THIS) PURE;

};
typedef IOleInPlaceObjectA * LPOLEINPLACEOBJECTA;


/*---------------------------------------------------------------------*/
/*                        IOleInPlaceActiveObjectA                     */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IOleInPlaceActiveObjectA

DECLARE_INTERFACE_(IOleInPlaceActiveObjectA, IOleWindow)
{
   // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IOleWindow methods ***
     STDMETHOD(GetWindow) (THIS_ HWND * lphwnd) PURE;
     STDMETHOD(ContextSensitiveHelp) (THIS_ BOOL fEnterMode) PURE;

     // *** IOleInPlaceActiveObject methods ***
     STDMETHOD(TranslateAccelerator) (THIS_ LPMSG lpmsg) PURE;
     STDMETHOD(OnFrameWindowActivate) (THIS_ BOOL fActivate) PURE;
     STDMETHOD(OnDocWindowActivate) (THIS_ BOOL fActivate) PURE;
     STDMETHOD(ResizeBorder) (THIS_ LPCRECT lprectBorder,
LPOLEINPLACEUIWINDOWA lpUIWindowA, BOOL fFrameWindow) PURE;
     STDMETHOD(EnableModeless) (THIS_ BOOL fEnable) PURE;

};
typedef IOleInPlaceActiveObjectA * LPOLEINPLACEACTIVEOBJECTA;


/*---------------------------------------------------------------------*/
/*                         IOleInPlaceFrameA                           */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IOleInPlaceFrameA

DECLARE_INTERFACE_(IOleInPlaceFrameA, IOleWindow)
{
   // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IOleWindow methods ***
     STDMETHOD(GetWindow) (THIS_ HWND * lphwnd) PURE;
     STDMETHOD(ContextSensitiveHelp) (THIS_ BOOL fEnterMode) PURE;

     // *** IOleInPlaceUIWindow methods ***
     STDMETHOD(GetBorder) (THIS_ LPRECT lprectBorder) PURE;
     STDMETHOD(RequestBorderSpace) (THIS_ LPCBORDERWIDTHS lpborderwidths)
PURE;
     STDMETHOD(SetBorderSpace) (THIS_ LPCBORDERWIDTHS lpborderwidths) PURE;
     STDMETHOD(SetActiveObject) (THIS_ LPOLEINPLACEACTIVEOBJECTA
lpActiveObjectA,
                             LPCSTR lpszObjName) PURE;


     // *** IOleInPlaceFrame methods ***
     STDMETHOD(InsertMenus) (THIS_ HMENU hmenuShared, LPOLEMENUGROUPWIDTHS
lpMenuWidths) PURE;
     STDMETHOD(SetMenu) (THIS_ HMENU hmenuShared, HOLEMENU holemenu, HWND
hwndActiveObject) PURE;
     STDMETHOD(RemoveMenus) (THIS_ HMENU hmenuShared) PURE;
     STDMETHOD(SetStatusText) (THIS_ LPCSTR lpszStatusText) PURE;

     STDMETHOD(EnableModeless) (THIS_ BOOL fEnable) PURE;
     STDMETHOD(TranslateAccelerator) (THIS_ LPMSG lpmsg, WORD wID) PURE;
};
typedef IOleInPlaceFrameA * LPOLEINPLACEFRAMEA;


/*---------------------------------------------------------------------*/
/*                         IOleInPlaceSiteA                            */
```

```
/*--------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IOleInPlaceSiteA

DECLARE_INTERFACE_(IOleInPlaceSiteA, IOleWindow)
{
   // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IOleWindow methods ***
     STDMETHOD(GetWindow) (THIS_ HWND * lphwnd) PURE;
     STDMETHOD(ContextSensitiveHelp) (THIS_ BOOL fEnterMode) PURE;

     // *** IOleInPlaceSite methods ***
     STDMETHOD(CanInPlaceActivate) (THIS) PURE;
     STDMETHOD(OnInPlaceActivate) (THIS) PURE;
     STDMETHOD(OnUIActivate) (THIS) PURE;
     STDMETHOD(GetWindowContext) (THIS_ LPOLEINPLACEFRAMEA * lplpFrameA,
                                 LPOLEINPLACEUIWINDOWA * lplpDocA,
                                 LPRECT lprcPosRect,
                                 LPRECT lprcClipRect,
                                 LPOLEINPLACEFRAMEINFO lpFrameInfo) PURE;
     STDMETHOD(Scroll) (THIS_ SIZE scrollExtent) PURE;
     STDMETHOD(OnUIDeactivate) (THIS_ BOOL fUndoable) PURE;
     STDMETHOD(OnInPlaceDeactivate) (THIS) PURE;
     STDMETHOD(DiscardUndoState) (THIS) PURE;
     STDMETHOD(DeactivateAndUndo) (THIS) PURE;
     STDMETHOD(OnPosRectChange) (THIS_ LPCRECT lprcPosRect) PURE;
};
typedef IOleInPlaceSiteA * LPOLEINPLACESITEA;



//
//  Forward declarations
//
class CDropTargetA;
class CPersistStorageA;
class CPersistFileA;
class CEnumOLEVERBA;
class COleObjectA;
class COleClientSiteA;
class CRunnableObjectA;
class COleItemContainerA;
class COleAdviseHolderA;
class COleLinkA;
class COleInPlaceObjectA;
class COleInPlaceActiveObjectA;
class COleInPlaceFrameA;
class COleInPlaceSiteA;
```

```
//
+---------------------------------------------------------------------------
//
//  Class:      CDropTargetA
//
//  Synopsis:   Class definition of IDropTargetA
//
//----------------------------------------------------------------------------
-
class CDropTargetA : CAnsiInterface
{
public:
     // *** IDropTarget methods ***
     STDMETHOD(DragEnter) (LPDATAOBJECTA pDataObjA, DWORD grfKeyState,
POINTL pt, LPDWORD pdwEffect);
     STDMETHOD(DragOver) (DWORD grfKeyState, POINTL pt, LPDWORD pdwEffect);
     STDMETHOD(DragLeave) (VOID);
     STDMETHOD(Drop) (LPDATAOBJECTA pDataObj, DWORD grfKeyState, POINTL pt,
LPDWORD pdwEffect);

     inline CDropTargetA(LPUNKNOWN pUnk, IDropTarget * pWide) :
                 CAnsiInterface(ID_IDropTarget, pUnk, (LPUNKNOWN)pWide) {};

     inline IDropTarget * GetWide() const
                 { return (IDropTarget *)m_pObj; };
};



//
+---------------------------------------------------------------------------
//
//  Class:      CPersistStorageA
//
//  Synopsis:   Class definition of IPersistStorageA
//
//----------------------------------------------------------------------------
-
class CPersistStorageA : CAnsiInterface
{
public:
     // *** IPersist methods ***
     STDMETHOD(GetClassID) (LPCLSID lpClassID);

     // *** IPersistStorage methods ***
     STDMETHOD(IsDirty) (VOID);
     STDMETHOD(InitNew) (LPSTORAGEA pStgA);
     STDMETHOD(Load) (LPSTORAGEA pStgA);
     STDMETHOD(Save) (LPSTORAGEA pStgSaveA, BOOL fSameAsLoad);
     STDMETHOD(SaveCompleted) (LPSTORAGEA pStgNewA);
     STDMETHOD(HandsOffStorage) (VOID);

     inline CPersistStorageA(LPUNKNOWN pUnk, IPersistStorage * pWide) :
```

```
                 CAnsiInterface(ID_IPersistStorage, pUnk, (LPUNKNOWN)pWide)
{};

     inline IPersistStorage * GetWide() const
                 { return (IPersistStorage *)m_pObj; };
};



//
+--------------------------------------------------------------------------
//
//  Class:      CPersistFileA
//
//  Synopsis:   Class definition of IPersistFileA
//
//---------------------------------------------------------------------------
-
class CPersistFileA : CAnsiInterface
{
public:
     // *** IPersist methods ***
     STDMETHOD(GetClassID) (LPCLSID lpClassID);

     // *** IPersistFile methods ***
     STDMETHOD(IsDirty) (VOID);
     STDMETHOD(Load) (LPCSTR lpszFileName, DWORD grfMode);
     STDMETHOD(Save) (LPCSTR lpszFileName, BOOL fRemember);
     STDMETHOD(SaveCompleted) (LPCSTR lpszFileName);
     STDMETHOD(GetCurFile) (LPSTR FAR* lplpszFileName);

     inline CPersistFileA(LPUNKNOWN pUnk, IPersistFile * pWide) :
                 CAnsiInterface(ID_IPersistFile, pUnk, (LPUNKNOWN)pWide) {};

     inline IPersistFile * GetWide() const
                 { return (IPersistFile *)m_pObj; };
};



//
+--------------------------------------------------------------------------
//
//  Class:      CEnumOLEVERBA
//
//  Synopsis:   Class definition of IEnumOLEVERBA
//
//---------------------------------------------------------------------------
-
class CEnumOLEVERBA : CAnsiInterface
{
public:
     // *** IEnumOLEVERB methods ***
     STDMETHOD(Next) (ULONG celt, LPOLEVERBA rgelt, ULONG * pceltFetched);
     STDMETHOD(Skip) (ULONG celt);
     STDMETHOD(Reset) (VOID);
     STDMETHOD(Clone) (IEnumOLEVERBA * * ppenm);

     inline CEnumOLEVERBA(LPUNKNOWN pUnk, IEnumOLEVERB * pWide) :
                CAnsiInterface(ID_IEnumOLEVERB, pUnk, (LPUNKNOWN)pWide) {};

     inline IEnumOLEVERB * GetWide() const
                { return (IEnumOLEVERB *)m_pObj; };
};



//
+---------------------------------------------------------------------------
//
//  Class:      COleObjectA
//
//  Synopsis:   Class definition of IOleObjectA
//
//----------------------------------------------------------------------------
-
class COleObjectA : CAnsiInterface
{
public:
     // *** IOleObject methods ***
     STDMETHOD(SetClientSite) (LPOLECLIENTSITEA pClientSiteA);
     STDMETHOD(GetClientSite) (LPOLECLIENTSITEA * ppCLientSiteA);
     STDMETHOD(SetHostNames) (LPCSTR szContainerApp, LPCSTR szContainerObj);
     STDMETHOD(Close) (DWORD dwSaveOption);
     STDMETHOD(SetMoniker) (DWORD dwWhichMoniker, LPMONIKERA pmkA);
     STDMETHOD(GetMoniker) (DWORD dwAssign, DWORD dwWhichMoniker,
                    LPMONIKERA * ppmkA);
     STDMETHOD(InitFromData) (LPDATAOBJECTA pDataObjectA,
                    BOOL fCreation,
                    DWORD dwReserved);
     STDMETHOD(GetClipboardData) (DWORD dwReserved,
                    LPDATAOBJECTA * ppDataObjectA);
     STDMETHOD(DoVerb) (LONG iVerb,
                    LPMSG lpmsg,
                    LPOLECLIENTSITEA pActiveSiteA,
                    LONG lindex,
                    HWND hwndParent,
                    LPCRECT lprcPosRect);
     STDMETHOD(EnumVerbs) (LPENUMOLEVERBA * ppenumOleVerbA);
     STDMETHOD(Update) (VOID);
     STDMETHOD(IsUpToDate) (VOID);
     STDMETHOD(GetUserClassID) (CLSID * pClsid);
     STDMETHOD(GetUserType) (DWORD dwFormOfType, LPSTR * pszUserType);
     STDMETHOD(SetExtent) (DWORD dwDrawAspect, LPSIZEL lpsizel);
     STDMETHOD(GetExtent) (DWORD dwDrawAspect, LPSIZEL lpsizel);

     STDMETHOD(Advise)(LPADVISESINKA pAdvSinkA, DWORD * pdwConnection);
     STDMETHOD(Unadvise)(DWORD dwConnection);
     STDMETHOD(EnumAdvise) (LPENUMSTATDATAA * ppenumAdviseA);
     STDMETHOD(GetMiscStatus) (DWORD dwAspect, DWORD * pdwStatus);
     STDMETHOD(SetColorScheme) (LPLOGPALETTE lpLogpal);

     inline COleObjectA(LPUNKNOWN pUnk, IOleObject * pWide) :
                CAnsiInterface(ID_IOleObject, pUnk, (LPUNKNOWN)pWide) {};

     inline IOleObject * GetWide() const
                { return (IOleObject *)m_pObj; };
};



//
+---------------------------------------------------------------------------
//
//  Class:      COleClientSiteA
//
//  Synopsis:   Class definition of IOleClientSiteA
//
//----------------------------------------------------------------------------
-
class COleClientSiteA : CAnsiInterface
{
public:
     // *** IOleClientSite methods ***
     STDMETHOD(SaveObject) (VOID);
     STDMETHOD(GetMoniker) (DWORD dwAssign, DWORD dwWhichMoniker,
                       LPMONIKERA * ppmkA);
     STDMETHOD(GetContainer) (LPOLECONTAINERA * ppContainerA);
     STDMETHOD(ShowObject) (VOID);
     STDMETHOD(OnShowWindow) (BOOL fShow);
     STDMETHOD(RequestNewObjectLayout) (VOID);

     inline COleClientSiteA(LPUNKNOWN pUnk, IOleClientSite * pWide) :
                CAnsiInterface(ID_IOleClientSite, pUnk, (LPUNKNOWN)pWide)
{};

     inline IOleClientSite * GetWide() const
                { return (IOleClientSite *)m_pObj; };

};



//
+---------------------------------------------------------------------------
//
//  Class:      CRunnableObjectA
//
//  Synopsis:   Class definition of IRunnableObjectA
//
//----------------------------------------------------------------------------
-
class CRunnableObjectA : CAnsiInterface
{
```

```
public:
     // *** IRunnableObject methods ***
     STDMETHOD(GetRunningClass) (LPCLSID lpClsid);
     STDMETHOD(Run) (LPBINDCTXA pbcA);
     STDMETHOD_(BOOL, IsRunning) (VOID);
     STDMETHOD(LockRunning)(BOOL fLock, BOOL fLastUnlockCloses);
     STDMETHOD(SetContainedObject)(BOOL fContained);

     inline CRunnableObjectA(LPUNKNOWN pUnk, IRunnableObject * pWide) :
                 CAnsiInterface(ID_IRunnableObject, pUnk, (LPUNKNOWN)pWide)
{};

     inline IRunnableObject * GetWide() const
                 { return (IRunnableObject *)m_pObj; };
};



//
+--------------------------------------------------------------------------
//
//  Class:      COleItemContainerA
//
//  Synopsis:   Class definition of IOleItemContainerA
//
//--------------------------------------------------------------------------
-
class COleItemContainerA : CAnsiInterface
{
public:
     // *** IParseDisplayName method ***
     STDMETHOD(ParseDisplayName) (LPBCA pbcA, LPSTR lpszDisplayName,
           ULONG * pchEaten, LPMONIKERA * ppmkAOut);

     // *** IOleContainer methods ***
     STDMETHOD(EnumObjects) (DWORD grfFlags, LPENUMUNKNOWN * ppenumUnknown);
     STDMETHOD(LockContainer) (BOOL fLock);

     // *** IOleItemContainer methods ***
     STDMETHOD(GetObject) (LPSTR lpszItem, DWORD dwSpeedNeeded,
           LPBINDCTXA pbcA, REFIID riid, LPVOID * ppvObject);
     STDMETHOD(GetObjectStorage) (LPSTR lpszItem, LPBINDCTXA pbcA,
           REFIID riid, LPVOID * ppvStorage);
     STDMETHOD(IsRunning) (LPSTR lpszItem);

     inline COleItemContainerA(LPUNKNOWN pUnk, IOleItemContainer * pWide) :
                 CAnsiInterface(ID_IOleItemContainer, pUnk,
(LPUNKNOWN)pWide) {};

     inline IOleItemContainer * GetWide() const
                 { return (IOleItemContainer *)m_pObj; };
};
```

```
//
+---------------------------------------------------------------------------
//
//  Class:      COleAdviseHolderA
//
//  Synopsis:   Class definition of IOleAdviseHolderA
//
//---------------------------------------------------------------------------
-
class COleAdviseHolderA : CAnsiInterface
{
public:
     // *** IOleAdviseHolder methods ***
     STDMETHOD(Advise)(LPADVISESINKA pAdviseA, DWORD * pdwConnection);
     STDMETHOD(Unadvise)(DWORD dwConnection);
     STDMETHOD(EnumAdvise)(LPENUMSTATDATAA * ppenumAdviseA);

     STDMETHOD(SendOnRename)(LPMONIKERA pmkA);
     STDMETHOD(SendOnSave)(VOID);
     STDMETHOD(SendOnClose)(VOID);

     inline COleAdviseHolderA(LPUNKNOWN pUnk, IOleAdviseHolder * pWide) :
                 CAnsiInterface(ID_IOleAdviseHolder, pUnk, (LPUNKNOWN)pWide)
{};

     inline IOleAdviseHolder * GetWide() const
                 { return (IOleAdviseHolder *)m_pObj; };
};



//
+---------------------------------------------------------------------------
//
//  Class:      COleLinkA
//
//  Synopsis:   Class definition of IOleLinkA
//
//---------------------------------------------------------------------------
-
class COleLinkA : CAnsiInterface
{
public:
     // *** IOleLink methods ***
     STDMETHOD(SetUpdateOptions) (DWORD dwUpdateOpt);
     STDMETHOD(GetUpdateOptions) (LPDWORD pdwUpdateOpt);
     STDMETHOD(SetSourceMoniker) (LPMONIKERA pmkA, REFCLSID rclsid);
     STDMETHOD(GetSourceMoniker) (LPMONIKERA * ppmkA);
     STDMETHOD(SetSourceDisplayName) (LPCSTR lpszDisplayName);
     STDMETHOD(GetSourceDisplayName) (LPSTR * lplpszDisplayName);
     STDMETHOD(BindToSource) (DWORD bindflags, LPBINDCTXA pbcA);
     STDMETHOD(BindIfRunning) (VOID);
     STDMETHOD(GetBoundSource) (LPUNKNOWN * ppUnk);
     STDMETHOD(UnbindSource) (VOID);
     STDMETHOD(Update) (LPBINDCTXA pbcA);
     inline COleLinkA(LPUNKNOWN pUnk, IOleLink * pWide) :
                 CAnsiInterface(ID_IOleLink, pUnk, (LPUNKNOWN)pWide) {};

     inline IOleLink * GetWide() const
                 { return (IOleLink *)m_pObj; };
};



//
+--------------------------------------------------------------------------
//
//  Class:      COleInPlaceObjectA
//
//  Synopsis:   Class definition of IOleInPlaceObjectA
//
//--------------------------------------------------------------------------
-
class COleInPlaceObjectA : CAnsiInterface
{
public:
     // *** IOleWindow methods ***
     STDMETHOD(GetWindow) (HWND * lphwnd);
     STDMETHOD(ContextSensitiveHelp) (BOOL fEnterMode);

     // *** IOleInPlaceObject methods ***
     STDMETHOD(InPlaceDeactivate) (VOID);
     STDMETHOD(UIDeactivate) (VOID);
     STDMETHOD(SetObjectRects) (LPCRECT lprcPosRect, LPCRECT lprcClipRect);
     STDMETHOD(ReactivateAndUndo) (VOID);

     inline COleInPlaceObjectA(LPUNKNOWN pUnk, IOleInPlaceObject * pWide) :
                 CAnsiInterface(ID_IOleInPlaceObject, pUnk,
(LPUNKNOWN)pWide) {};

     inline IOleInPlaceObject * GetWide() const
                 { return (IOleInPlaceObject *)m_pObj; };
};


//
+--------------------------------------------------------------------------
//
//  Class:      COleInPlaceActiveObjectA
//
//  Synopsis:   Class definition of IOleInPlaceActiveObjectA
//
//--------------------------------------------------------------------------
-
class COleInPlaceActiveObjectA : CAnsiInterface
{
public:
     // *** IOleWindow methods ***
     STDMETHOD(GetWindow) (HWND * lphwnd);
     STDMETHOD(ContextSensitiveHelp) (BOOL fEnterMode);

     // *** IOleInPlaceActiveObject methods ***
     STDMETHOD(TranslateAccelerator) (LPMSG lpmsg);
     STDMETHOD(OnFrameWindowActivate) (BOOL fActivate);
     STDMETHOD(OnDocWindowActivate) (BOOL fActivate);
     STDMETHOD(ResizeBorder) (LPCRECT lprectBorder, LPOLEINPLACEUIWINDOWA
lpUIWindowA, BOOL fFrameWindow);
     STDMETHOD(EnableModeless) (BOOL fEnable);

     inline COleInPlaceActiveObjectA(LPUNKNOWN pUnk, IOleInPlaceActiveObject
* pWide) :
                 CAnsiInterface(ID_IOleInPlaceActiveObject, pUnk,
(LPUNKNOWN)pWide) {};

     inline IOleInPlaceActiveObject * GetWide() const
                 { return (IOleInPlaceActiveObject *)m_pObj; };
};


//
+---------------------------------------------------------------------------
//
//  Class:      COleInPlaceFrameA
//
//  Synopsis:   Class definition of IOleInPlaceFrameA
//
//---------------------------------------------------------------------------
-
class COleInPlaceFrameA : CAnsiInterface
{
public:
     // *** IOleWindow methods ***
     STDMETHOD(GetWindow) (HWND * lphwnd);
     STDMETHOD(ContextSensitiveHelp) (BOOL fEnterMode);

     // *** IOleInPlaceUIWindow methods ***
     STDMETHOD(GetBorder) (LPRECT lprectBorder);
     STDMETHOD(RequestBorderSpace) (LPCBORDERWIDTHS lpborderwidths);
     STDMETHOD(SetBorderSpace) (LPCBORDERWIDTHS lpborderwidths);
     STDMETHOD(SetActiveObject) (LPOLEINPLACEACTIVEOBJECTA lpActiveObjectA,
                             LPCSTR lpszObjName);

     // *** IOleInPlaceFrame methods ***
     STDMETHOD(InsertMenus) (HMENU hmenuShared, LPOLEMENUGROUPWIDTHS
lpMenuWidths);
     STDMETHOD(SetMenu) (HMENU hmenuShared, HOLEMENU holemenu, HWND
hwndActiveObject);
     STDMETHOD(RemoveMenus) (HMENU hmenuShared);
     STDMETHOD(SetStatusText) (LPCSTR lpszStatusText);
     STDMETHOD(EnableModeless) (BOOL fEnable);
     STDMETHOD(TranslateAccelerator) (LPMSG lpmsg, WORD wID);

     inline COleInPlaceFrameA(LPUNKNOWN pUnk, IOleInPlaceFrame * pWide) :
```

```
                CAnsiInterface(ID_IOleInPlaceFrame, pUnk, (LPUNKNOWN)pWide)
{};

     inline IOleInPlaceFrame * GetWide() const
                { return (IOleInPlaceFrame *)m_pObj; };
};



//
+---------------------------------------------------------------------------
//
//  Class:      COleInPlaceSiteA
//
//  Synopsis:   Class definition of IOleInPlaceSiteA
//
//----------------------------------------------------------------------------
-
class COleInPlaceSiteA : CAnsiInterface
{
public:
     // *** IOleWindow methods ***
     STDMETHOD(GetWindow) (HWND * lphwnd);
     STDMETHOD(ContextSensitiveHelp) (BOOL fEnterMode);

     // *** IOleInPlaceSite methods ***
     STDMETHOD(CanInPlaceActivate) (VOID);
     STDMETHOD(OnInPlaceActivate) (VOID);
     STDMETHOD(OnUIActivate) (VOID);
     STDMETHOD(GetWindowContext) (LPOLEINPLACEFRAMEA * lplpFrameA,
                              LPOLEINPLACEUIWINDOWA * lplpDocA,
                              LPRECT lprcPosRect,
                              LPRECT lprcClipRect,
                              LPOLEINPLACEFRAMEINFO lpFrameInfo);
     STDMETHOD(Scroll) (SIZE scrollExtent);
     STDMETHOD(OnUIDeactivate) (BOOL fUndoable);
     STDMETHOD(OnInPlaceDeactivate) (VOID);
     STDMETHOD(DiscardUndoState) (VOID);
     STDMETHOD(DeactivateAndUndo) (VOID);
     STDMETHOD(OnPosRectChange) (LPCRECT lprcPosRect);


     inline COleInPlaceSiteA(LPUNKNOWN pUnk, IOleInPlaceSite * pWide) :
                CAnsiInterface(ID_IOleInPlaceSite, pUnk, (LPUNKNOWN)pWide)
{};

     inline IOleInPlaceSite * GetWide() const
                { return (IOleInPlaceSite *)m_pObj; };
};
```

ANSIOLE1.CPP (OLEANSI Sample)

```
//
+----------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       ansiole1.cpp
//
//  Contents:   ANSI Wrappers for Unicode Ole2 Interfaces and APIs.
//
//  Classes:    CDropTargetA
//              CPersistStorageA
//              CPersistFileA
//              CEnumOLEVERBA
//              COleObjectA
//              CRunnableObjectA
//
//  Functions:  IDropTargetAFromW
//              IPersistStorageAFromW
//              IPersistFileAFromW
//              IEnumOLEVERBAFromW
//              IOleObjectAFromW
//              IRunnableObjectAFromW
//
//  History:    01-Nov-93   v-kentc     Created.
//
//---------------------------------------------------------------------------
-

#include "Ole2Ansi.h"
```

IDropTargetA Implementation (OLEANSI Sample)

```
//
*********************************************************************
//
//                    IDropTargetA Implementation
//
//
*********************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     CDropTargetA::DragEnter, public
//
//  Synopsis:   Thunks DragEnter to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CDropTargetA::DragEnter(LPDATAOBJECTA pDataObjA,
           DWORD grfKeyState, POINTL pt, LPDWORD pdwEffect)
{
     TraceMethodEnter("CDropTargetA::DragEnter", this);

     LPDATAOBJECT pDataObj;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IDropTarget, DragEnter));

     hResult = WrapIDataObjectWFromA(pDataObjA, &pDataObj);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->DragEnter(pDataObj, grfKeyState, pt, pdwEffect);

     if (pDataObj)
           pDataObj->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CDropTargetA::DragOver, public
//
//  Synopsis:   Thunks DragOver to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
```

```
//--------------------------------------------------------------------------
-
STDMETHODIMP CDropTargetA::DragOver(DWORD grfKeyState, POINTL pt,
           LPDWORD pdwEffect)
{
     TraceMethodEnter("CDropTargetA::DragOver", this);

     _DebugHook(GetWide(), MEMBER_PTR(IDropTarget, DragOver));

     return GetWide()->DragOver(grfKeyState, pt, pdwEffect);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CDropTargetA::DragLeave, public
//
//  Synopsis:   Thunks DragLeave to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CDropTargetA::DragLeave(VOID)
{
     TraceMethodEnter("CDropTargetA::DragLeave", this);

     _DebugHook(GetWide(), MEMBER_PTR(IDropTarget, DragLeave));

     return GetWide()->DragLeave();
}


//
+--------------------------------------------------------------------------
//
//  Member:     CDropTargetA::Drop, public
//
//  Synopsis:   Thunks Drop to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CDropTargetA::Drop(LPDATAOBJECTA pDataObjA, DWORD grfKeyState,
           POINTL pt, LPDWORD pdwEffect)
{
     TraceMethodEnter("CDropTargetA::Drop", this);

     LPDATAOBJECT pDataObj;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IDropTarget, Drop));
     hResult = WrapIDataObjectWFromA(pDataObjA, &pDataObj);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->Drop(pDataObj, grfKeyState, pt, pdwEffect);

     if (pDataObj)
           pDataObj->Release();

     return hResult;
}
```

IPersistStorageA Implementation (OLEANSI Sample)

```
//
**********************************************************************
//
//                    IPersistStorageA Implementation
//
//
**********************************************************************

//
+---------------------------------------------------------------------------
//
//  Member:     CPersistStorageA::GetClassID, public
//
//  Synopsis:   Thunks GetClassID to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CPersistStorageA::GetClassID(LPCLSID lpClassID)
{
     TraceMethodEnter("CPersistStorageA::GetClassID", this);

     _DebugHook(GetWide(), MEMBER_PTR(IPersist, GetClassID));

     return GetWide()->GetClassID(lpClassID);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CPersistStorageA::IsDirty, public
//
//  Synopsis:   Thunks IsDirty to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CPersistStorageA::IsDirty(VOID)
{
     TraceMethodEnter("CPersistStorageA::IsDirty", this);

     _DebugHook(GetWide(), MEMBER_PTR(IPersistStorage, IsDirty));

     return GetWide()->IsDirty();
}


//
+---------------------------------------------------------------------------
```

```
//
//  Member:     CPersistStorageA::InitNew, public
//
//  Synopsis:   Thunks InitNew to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CPersistStorageA::InitNew(LPSTORAGEA pStgA)
{
     TraceMethodEnter("CPersistStorageA::InitNew", this);

     LPSTORAGE pStg;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IPersistStorage, InitNew));

     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->InitNew(pStg);

     if (pStg)
           pStg->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CPersistStorageA::Load, public
//
//  Synopsis:   Thunks Load to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CPersistStorageA::Load(LPSTORAGEA pStgA)
{
     TraceMethodEnter("CPersistStorageA::Load", this);

     LPSTORAGE pStg;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IPersistStorage, Load));

     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
```

```
            return hResult;

      hResult = GetWide()->Load(pStg);

      if (pStg)
            pStg->Release();

      return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CPersistStreamA::Save, public
//
//  Synopsis:   Thunks Save to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CPersistStorageA::Save(LPSTORAGEA pStgSaveA, BOOL fSameAsLoad)
{
      TraceMethodEnter("CPersistStorageA::Save", this);

      LPSTORAGE pStgSave;
      HRESULT hResult;


      _DebugHook(GetWide(), MEMBER_PTR(IPersistStorage, Save));

      hResult = WrapIStorageWFromA(pStgSaveA, &pStgSave);
      if (FAILED(hResult))
            return hResult;


      hResult = GetWide()->Save(pStgSave, fSameAsLoad);

      if (pStgSave)
            pStgSave->Release();

      return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CPersistStreamA::SaveCompleted, public
//
//  Synopsis:   Thunks SaveCompleted to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
```

```
//---------------------------------------------------------------------------
-
STDMETHODIMP CPersistStorageA::SaveCompleted(LPSTORAGEA pStgNewA)
{
     TraceMethodEnter("CPersistStorageA::SaveCompleted", this);

     LPSTORAGE pStgNew;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IPersistStorage, SaveCompleted));

     hResult = WrapIStorageWFromA(pStgNewA, &pStgNew);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->SaveCompleted(pStgNew);

     if (pStgNew)
           pStgNew->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CPersistStreamA::HandsOffStorage, public
//
//  Synopsis:   Thunks HandsOffStorage to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CPersistStorageA::HandsOffStorage(VOID)
{
     TraceMethodEnter("CPersistStorageA::HandsOffStorage", this);

     _DebugHook(GetWide(), MEMBER_PTR(IPersistStorage, HandsOffStorage));

     return GetWide()->HandsOffStorage();
}
```

IPersistFileA Implementation   (OLEANSI Sample)

```
//
**********************************************************************
//
//                    IPersistFileA Implementation
//
//
**********************************************************************

//
+---------------------------------------------------------------------
//
//  Member:     CPersistFileA::GetClassID, public
//
//  Synopsis:   Thunks GetClassID to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------
-
STDMETHODIMP CPersistFileA::GetClassID(LPCLSID lpClassID)
{
     TraceMethodEnter("CPersistFileA::GetClassID", this);

     _DebugHook(GetWide(), MEMBER_PTR(IPersist, GetClassID));

     return GetWide()->GetClassID(lpClassID);
}


//
+---------------------------------------------------------------------
//
//  Member:     CPersistFileA::IsDirty, public
//
//  Synopsis:   Thunks IsDirty to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------
-
STDMETHODIMP CPersistFileA::IsDirty(VOID)
{
     TraceMethodEnter("CPersistFileA::IsDirty", this);

     _DebugHook(GetWide(), MEMBER_PTR(IPersistFile, IsDirty));

     return GetWide()->IsDirty();
}


//
+---------------------------------------------------------------------
```

```
//
//  Member:     CPersistFileA::Load, public
//
//  Synopsis:   Thunks Load to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CPersistFileA::Load(LPCSTR lpszFileNameA, DWORD grfMode)
{
     TraceMethodEnter("CPersistFileA::Load", this);

     LPOLESTR lpszFileName;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IPersistFile, Load));

     hResult = ConvertStringToW(lpszFileNameA, &lpszFileName);
     if (FAILED(hResult))
           return (hResult);

     hResult = GetWide()->Load(lpszFileName, grfMode);

     ConvertStringFree(lpszFileName);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CPersistFileA::Save, public
//
//  Synopsis:   Thunks Save to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CPersistFileA::Save(LPCSTR lpszFileNameA, BOOL fRemember)
{
     TraceMethodEnter("CPersistFileA::Save", this);

     LPOLESTR lpszFileName;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IPersistFile, Save));

     hResult = ConvertStringToW(lpszFileNameA, &lpszFileName);
     if (FAILED(hResult))
           return (hResult);
     hResult = GetWide()->Save(lpszFileName, fRemember);

     ConvertStringFree(lpszFileName);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CPersistFileA::SaveCompleted, public
//
//  Synopsis:   Thunks SaveCompleted to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CPersistFileA::SaveCompleted(LPCSTR lpszFileNameA)
{
     TraceMethodEnter("CPersistFileA::SaveCompleted", this);

     LPOLESTR lpszFileName;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IPersistFile, SaveCompleted));

     hResult = ConvertStringToW(lpszFileNameA, &lpszFileName);
     if (FAILED(hResult))
            return (hResult);

     hResult = GetWide()->SaveCompleted(lpszFileName);

     ConvertStringFree(lpszFileName);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CPersistFileA::GetCurFile, public
//
//  Synopsis:   Thunks GetCurFile to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CPersistFileA::GetCurFile(LPSTR FAR* lplpszFileNameA)
```

```
{
    TraceMethodEnter("CPersistFileA::GetCurFile", this);

    LPOLESTR lpszFileName;
    HRESULT hResult;


    _DebugHook(GetWide(), MEMBER_PTR(IPersistFile, GetCurFile));

    hResult = GetWide()->GetCurFile(&lpszFileName);
    if (FAILED(hResult))
        return (hResult);

    hResult = ConvertStringToA(lpszFileName, lplpszFileNameA);

    ConvertStringFree(lpszFileName);

    return hResult;
}
```

IEnumOLEVERBA Implementation (OLEANSI Sample)

```
//
*********************************************************************
//
//                    IEnumOLEVERBA Implementation
//
//
*********************************************************************

//
+--------------------------------------------------------------------
//
//  Member:     CEnumOLEVERBA::Next, public
//
//  Synopsis:   Thunks Next to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------
-
STDMETHODIMP CEnumOLEVERBA::Next(
           ULONG celt,
           OLEVERBA * rgelt,
           ULONG * pceltFetched)
{
     TraceMethodEnter("CEnumOLEVERBA::Next", this);
     ULONG   celtFetched;
     HRESULT hReturn;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IEnumOLEVERB, Next));

     if (pceltFetched == NULL)
           pceltFetched = &celtFetched;

     hReturn = GetWide()->Next(celt, (OLEVERB *)rgelt, pceltFetched);
     if (FAILED(hReturn))
           return (hReturn);

     hResult = ConvertOLEVERBArrayToA(rgelt, *pceltFetched);
     if (FAILED(hResult))
           return (hResult);

     return hReturn;
}


//
+--------------------------------------------------------------------
//
//  Member:     CEnumOLEVERBA::Skip, public
//
```

```
//  Synopsis:   Thunks Skip to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CEnumOLEVERBA::Skip(ULONG celt)
{
     TraceMethodEnter("CEnumOLEVERBA::Skip", this);

     _DebugHook(GetWide(), MEMBER_PTR(IEnumOLEVERB, Skip));

     return GetWide()->Skip(celt);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CEnumOLEVERBA::Reset, public
//
//  Synopsis:   Thunks Reset to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CEnumOLEVERBA::Reset(VOID)
{
     TraceMethodEnter("CEnumOLEVERBA::Reset", this);

     _DebugHook(GetWide(), MEMBER_PTR(IEnumOLEVERB, Reset));

     return GetWide()->Reset();
}


//
+--------------------------------------------------------------------------
//
//  Member:     CEnumOLEVERBA::Clone, public
//
//  Synopsis:   Thunks Clone to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CEnumOLEVERBA::Clone(IEnumOLEVERBA * * ppobjA)
{
     TraceMethodEnter("CEnumOLEVERBA::Clone", this);

     IEnumOLEVERB * pobj;
```

```
	_DebugHook(GetWide(), MEMBER_PTR(IEnumOLEVERB, Clone));

	*ppobjA = NULL;

	HRESULT hResult = GetWide()->Clone(&pobj);
	if (FAILED(hResult))
		return (hResult);

	hResult = WrapIEnumOLEVERBAFromW(pobj, ppobjA);

	if (pobj)
		pobj->Release();

	return hResult;
}
```

IOleObjectA Implementation (OLEANSI Sample)

```
//
***************************************************************************
//
//                   IOleObjectA Implementation
//
//
***************************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     COleObjectA::SetClientSite, public
//
//  Synopsis:   Thunks SetClientSite to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::SetClientSite(LPOLECLIENTSITEA pClientSiteA)
{
     TraceMethodEnter("COleObjectA::SetClientSite", this);

     LPOLECLIENTSITE pClientSite;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, SetClientSite));

     hResult = WrapIOleClientSiteWFromA(pClientSiteA, &pClientSite);
     if (FAILED(hResult))
           return (hResult);

     hResult = GetWide()->SetClientSite(pClientSite);

     if (pClientSite)
        pClientSite->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     COleObjectA::GetClientSite, public
//
//  Synopsis:   Thunks GetClientSite to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
```

```
//--------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::GetClientSite(LPOLECLIENTSITEA * ppClientSiteA)
{
     TraceMethodEnter("COleObjectA::GetClientSite", this);

     LPOLECLIENTSITE pClientSite;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, GetClientSite));

     *ppClientSiteA = NULL;

     hResult = GetWide()->GetClientSite(&pClientSite);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapIOleClientSiteAFromW(pClientSite, ppClientSiteA);

     if (pClientSite)
           pClientSite->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     COleObjectA::SetHostNames, public
//
//  Synopsis:   Thunks SetHostNames to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::SetHostNames(LPCSTR szContainerAppA,
           LPCSTR szContainerObjA)
{
     TraceMethodEnter("COleObjectA::SetHostNames", this);

     LPOLESTR szContainerApp;
     LPOLESTR szContainerObj = 0;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, SetHostNames));

     hResult = ConvertStringToW(szContainerAppA, &szContainerApp);
     if (FAILED(hResult))
           return (hResult);

     hResult = ConvertStringToW(szContainerObjA, &szContainerObj);
     if (FAILED(hResult))
           goto Error;

     hResult = GetWide()->SetHostNames(szContainerApp, szContainerObj);

Error:
     ConvertStringFree(szContainerApp);
     ConvertStringFree(szContainerObj);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleObjectA::Close, public
//
//  Synopsis:   Thunks Close to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::Close(DWORD dwSaveOption)
{
     TraceMethodEnter("COleObjectA::Close", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, Close));

     return GetWide()->Close(dwSaveOption);
}



//
+---------------------------------------------------------------------------
//
//  Member:     COleObjectA::SetMoniker, public
//
//  Synopsis:   Thunks SetMoniker to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::SetMoniker(DWORD dwWhichMoniker, LPMONIKERA pmkA)
{
     TraceMethodEnter("COleObjectA::SetMoniker", this);

     LPMONIKER pmk;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, SetMoniker));
     hResult = WrapIMonikerWFromA(pmkA, &pmk);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->SetMoniker(dwWhichMoniker, pmk);

     if (pmk)
           pmk->Release();

     return hResult;
}


//
+----------------------------------------------------------------------------
//
//  Member:     COleObjectA::GetMoniker, public
//
//  Synopsis:   Thunks GetMoniker to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::GetMoniker(DWORD dwAssign, DWORD dwWhichMoniker,
           LPMONIKERA * ppmkA)
{
     TraceMethodEnter("COleObjectA::GetMoniker", this);

     LPMONIKER pmk;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, GetMoniker));

     *ppmkA = NULL;

     hResult = GetWide()->GetMoniker(dwAssign, dwWhichMoniker, &pmk);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapIMonikerAFromW(pmk, ppmkA);

     if (pmk)
           pmk->Release();

     return hResult;
}


//
+----------------------------------------------------------------------------
//
//  Member:     COleObjectA::InitFromData, public
```

```
//
//  Synopsis:   Thunks InitFromData to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::InitFromData(LPDATAOBJECTA pDataObjectA,
           BOOL fCreation, DWORD dwReserved)
{
     TraceMethodEnter("COleObjectA::InitFromData", this);

     LPDATAOBJECT pDataObject;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, InitFromData));

     hResult = WrapIDataObjectWFromA(pDataObjectA, &pDataObject);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->InitFromData(pDataObject, fCreation, dwReserved);

     if (pDataObject)
           pDataObject->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     COleObjectA::GetClipboardData, public
//
//  Synopsis:   Thunks GetClipboardData to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::GetClipboardData(DWORD dwReserved,
           LPDATAOBJECTA * ppDataObjectA)
{
     TraceMethodEnter("COleObjectA::GetClipboardData", this);

     LPDATAOBJECT pDataObject;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, GetClipboardData));

     *ppDataObjectA = NULL;
```

```
    hResult = GetWide()->GetClipboardData(dwReserved, &pDataObject);
    if (FAILED(hResult))
          return (hResult);


    hResult = WrapIDataObjectAFromW(pDataObject, ppDataObjectA);

    if (pDataObject)
          pDataObject->Release();

    return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleObjectA::DoVerb, public
//
//  Synopsis:   Thunks DoVerb to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::DoVerb(LONG iVerb, LPMSG lpmsg,
           LPOLECLIENTSITEA pActiveSiteA, LONG lindex, HWND hwndParent,
           LPCRECT lprcPosRect)
{
    TraceMethodEnter("COleObjectA::DoVerb", this);

    LPOLECLIENTSITE pActiveSite;
    HRESULT hResult;


    _DebugHook(GetWide(), MEMBER_PTR(IOleObject, DoVerb));

    hResult = WrapIOleClientSiteWFromA(pActiveSiteA, &pActiveSite);
    if (FAILED(hResult))
          return hResult;

    hResult = GetWide()->DoVerb(iVerb, lpmsg, pActiveSite, lindex,
hwndParent,
                 lprcPosRect);

    if (pActiveSite)
          pActiveSite->Release();

    return hResult;
}


//
+---------------------------------------------------------------------------
//
```

```
//  Member:     COleObjectA::EnumVerbs, public
//
//  Synopsis:   Thunks EnumVerbs to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::EnumVerbs(LPENUMOLEVERBA * ppenumOleVerbA)
{
     TraceMethodEnter("COleObjectA::EnumVerbs", this);

     LPENUMOLEVERB penumOleVerb = NULL;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, EnumVerbs));

     hResult = GetWide()->EnumVerbs(&penumOleVerb);
     if (FAILED(hResult) || hResult == OLE_S_USEREG)
            return (hResult);

     if (penumOleVerb)
     {
            hResult = WrapIEnumOLEVERBAFromW(penumOleVerb, ppenumOleVerbA);

            penumOleVerb->Release();
     }
     else
            *ppenumOleVerbA = NULL;

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleObjectA::Update, public
//
//  Synopsis:   Thunks Update to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::Update(VOID)
{
     TraceMethodEnter("COleObjectA::Update", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, Update));

     return GetWide()->Update();
}
```

```
//
+---------------------------------------------------------------------------
//
//  Member:     COleObjectA::IsUpToDate, public
//
//  Synopsis:   Thunks IsUpToDate to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::IsUpToDate(VOID)
{
     TraceMethodEnter("COleObjectA::IsUpToDate", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, IsUpToDate));

     return GetWide()->IsUpToDate();
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleObjectA::GetUserClassID, public
//
//  Synopsis:   Thunks GetUserClassID to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::GetUserClassID(CLSID * pClsid)
{
     TraceMethodEnter("COleObjectA::GetUserClassID", this);


     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, GetUserClassID));

     return GetWide()->GetUserClassID(pClsid);
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleObjectA::GetUserType, public
//
//  Synopsis:   Thunks GetUserType to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
```

```
STDMETHODIMP COleObjectA::GetUserType(DWORD dwFormOfType,
           LPSTR * pszUserTypeA)
{
     TraceMethodEnter("COleObjectA::GetUserType", this);

     LPOLESTR lpszUserType;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, GetUserType));

     hResult = GetWide()->GetUserType(dwFormOfType, &lpszUserType);
     if (FAILED(hResult) || hResult == OLE_S_USEREG)
           return (hResult);

     hResult = ConvertStringToA(lpszUserType, pszUserTypeA);

     ConvertStringFree(lpszUserType);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleObjectA::SetExtent, public
//
//  Synopsis:   Thunks SetExtent to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::SetExtent(DWORD dwDrawAspect, LPSIZEL lpsizel)
{
     TraceMethodEnter("COleObjectA::SetExtent", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, SetExtent));

     return GetWide()->SetExtent(dwDrawAspect, lpsizel);
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleObjectA::GetExtent, public
//
//  Synopsis:   Thunks GetExtent to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
```

```
STDMETHODIMP COleObjectA::GetExtent(DWORD dwDrawAspect, LPSIZEL lpsizel)
{
     TraceMethodEnter("COleObjectA::GetExtent", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, GetExtent));

     return GetWide()->GetExtent(dwDrawAspect, lpsizel);
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleObjectA::Advise, public
//
//  Synopsis:   Thunks Advise to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::Advise(LPADVISESINKA pAdvSinkA,
           DWORD * pdwConnection)
{
     TraceMethodEnter("COleObjectA::Advise", this);

     LPADVISESINK pAdvSink;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, Advise));

     hResult = WrapIAdviseSinkWFromA(pAdvSinkA, &pAdvSink);
     if (FAILED(hResult))
           return (hResult);

     hResult = GetWide()->Advise(pAdvSink, pdwConnection);

     if (pAdvSink)
           pAdvSink->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleObjectA::Unadvise, public
//
//  Synopsis:   Thunks Unadvise to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
```

```
//----------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::Unadvise(DWORD dwConnection)
{
     TraceMethodEnter("COleObjectA::Unadvise", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, Unadvise));

     return GetWide()->Unadvise(dwConnection);
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleObjectA::EnumAdvise, public
//
//  Synopsis:   Thunks EnumAdvise to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::EnumAdvise(LPENUMSTATDATAA * ppenumAdviseA)
{
     TraceMethodEnter("COleObjectA::EnumAdvise", this);

     LPENUMSTATDATA penumAdvise = NULL;
     HRESULT hResult;



     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, EnumAdvise));

     hResult = GetWide()->EnumAdvise(&penumAdvise);
     if (FAILED(hResult))
           return (hResult);

     if (penumAdvise)
     {
           hResult = WrapIEnumSTATDATAAFromW(penumAdvise, ppenumAdviseA);

           penumAdvise->Release();
     }
     else
           *ppenumAdviseA = NULL;

     return hResult;
}


//
+---------------------------------------------------------------------------
//
```

```
//  Member:     COleObjectA::GetMiscStatus, public
//
//  Synopsis:   Thunks GetMiscStatus to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::GetMiscStatus(DWORD dwAspect, DWORD * pdwStatus)
{
     TraceMethodEnter("COleObjectA::GetMiscStatus", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, GetMiscStatus));

     return GetWide()->GetMiscStatus(dwAspect, pdwStatus);
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleObjectA::SetColorScheme, public
//
//  Synopsis:   Thunks SetColorScheme to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleObjectA::SetColorScheme(LPLOGPALETTE lpLogpal)
{
     TraceMethodEnter("COleObjectA::SetColorScheme", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleObject, SetColorScheme));

     return GetWide()->SetColorScheme(lpLogpal);
}
```

IOleClientSiteA Implementation (OLEANSI Sample)

```
//
*********************************************************************
//
//                    IOleClientSiteA Implementation
//
//
*********************************************************************

//
+--------------------------------------------------------------------
//
//  Member:     COleClientSiteA::SaveObject, public
//
//  Synopsis:   Thunks SaveObject to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------
-
STDMETHODIMP COleClientSiteA::SaveObject(VOID)
{
     TraceMethodEnter("COleClientSiteA::SaveObject", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleClientSite, SaveObject));

     return GetWide()->SaveObject();
}


//
+--------------------------------------------------------------------
//
//  Member:     COleClientSiteA::GetMoniker, public
//
//  Synopsis:   Thunks GetMoniker to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------
-
STDMETHODIMP COleClientSiteA::GetMoniker(DWORD dwAssign,
           DWORD dwWhichMoniker, LPMONIKERA * ppmkA)
{
     TraceMethodEnter("COleClientSiteA::GetMoniker", this);

     LPMONIKER pmk;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleClientSite, GetMoniker));

     *ppmkA = NULL;
     hResult = GetWide()->GetMoniker(dwAssign, dwWhichMoniker, &pmk);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapIMonikerAFromW(pmk, ppmkA);

     if (pmk)
           pmk->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleClientSiteA::GetContainer, public
//
//  Synopsis:   Thunks GetContainer to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP COleClientSiteA::GetContainer(LPOLECONTAINERA * ppContainerA)
{
     TraceMethodEnter("COleClientSiteA::GetContainer", this);

     LPOLECONTAINER pContainer;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleClientSite, GetContainer));

     *ppContainerA = NULL;

     hResult = GetWide()->GetContainer(&pContainer);
     if (FAILED(hResult))
     {
           *ppContainerA = NULL;
           return (hResult);
     }

     hResult = WrapIOleContainerAFromW(pContainer, ppContainerA);

     if (pContainer)
           pContainer->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
```

```
//
//  Member:     COleClientSiteA::ShowObject, public
//
//  Synopsis:   Thunks ShowObject to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleClientSiteA::ShowObject(VOID)
{
     TraceMethodEnter("COleClientSiteA::ShowObject", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleClientSite, ShowObject));

     return GetWide()->ShowObject();
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleClientSiteA::OnShowWindow, public
//
//  Synopsis:   Thunks OnShowWindow to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleClientSiteA::OnShowWindow(BOOL fShow)
{
     TraceMethodEnter("COleClientSiteA::OnShowWindow", this);


     _DebugHook(GetWide(), MEMBER_PTR(IOleClientSite, OnShowWindow));

     return GetWide()->OnShowWindow(fShow);
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleClientSiteA::RequestNewObjectLayout, public
//
//  Synopsis:   Thunks RequestNewObjectLayout to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleClientSiteA::RequestNewObjectLayout(VOID)
{
     TraceMethodEnter("COleClientSiteA::RequestNewObjectLayout", this);
     _DebugHook(GetWide(), MEMBER_PTR(IOleClientSite, 
RequestNewObjectLayout));

     return GetWide()->RequestNewObjectLayout();
}
```

IRunnableObjectA Implementation (OLEANSI Sample)

```
//
***************************************************************************
//
//                     IRunnableObjectA Implementation
//
//
***************************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     CRunnableObjectA::GetRunningClass, public
//
//  Synopsis:   Thunks GetRunningClass to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CRunnableObjectA::GetRunningClass(LPCLSID lpClsid)
{
     _DebugHook(GetWide(), MEMBER_PTR(IRunnableObject, GetRunningClass));

     return GetWide()->GetRunningClass(lpClsid);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CRunnableObjectA::Run, public
//
//  Synopsis:   Thunks Run to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CRunnableObjectA::Run(LPBINDCTXA pbcA)
{
     TraceMethodEnter("CRunnableObjectA::Run", this);

     LPBINDCTX pbc;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IRunnableObject, Run));

     hResult = WrapIBindCtxWFromA(pbcA, &pbc);
     if (FAILED(hResult))
           return (hResult);
     hResult = GetWide()->Run(pbc);

     if (pbc)
          pbc->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CRunnableObjectA::IsRunning, public
//
//  Synopsis:   Thunks IsRunning to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP_(BOOL) CRunnableObjectA::IsRunning(VOID)
{
     _DebugHook(GetWide(), MEMBER_PTR(IRunnableObject, IsRunning));

     return GetWide()->IsRunning();
}


//
+---------------------------------------------------------------------------
//
//  Member:     CRunnableObjectA::LockRunning, public
//
//  Synopsis:   Thunks LockRunning to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CRunnableObjectA::LockRunning(BOOL fLock,
           BOOL fLastUnlockCloses)
{
     _DebugHook(GetWide(), MEMBER_PTR(IRunnableObject, LockRunning));

     return GetWide()->LockRunning(fLock, fLastUnlockCloses);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CRunnableObjectA::SetContainedObject, public
//
//  Synopsis:   Thunks SetContainedObject to Unicode method.
//
```

```
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CRunnableObjectA::SetContainedObject(BOOL fContained)
{
     _DebugHook(GetWide(), MEMBER_PTR(IRunnableObject, SetContainedObject));

     return GetWide()->SetContainedObject(fContained);
}
```

ANSIOLE2.CPP   (OLEANSI Sample)

```
//
+---------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       ansiole1.cpp
//
//  Contents:   ANSI Wrappers for Unicode Ole2 Interfaces and APIs.
//
//  Classes:    COleItemContainerA
//              COleAdviseHolderA
//              COleLinkA
//              COleInPlaceObjectA
//              COleInPlaceActiveObjectA
//              COleInPlaceFrameA
//              COleInPlaceSiteA
//
//  History:    01-Nov-93   v-kentc     Created.
//
//---------------------------------------------------------------------------
-

#include "Ole2Ansi.h"
```

IOleItemContainerA Implementation (OLEANSI Sample)

```
//
**********************************************************************
//
//                    IOleItemContainerA Implementation
//
//
**********************************************************************

//
+-----------------------------------------------------------------------
//
//  Member:     COleItemContainerA::ParseDisplayName, public
//
//  Synopsis:   Thunks ParseDisplayName to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------
-
STDMETHODIMP COleItemContainerA::ParseDisplayName(LPBCA pbcA,
           LPSTR lpszDisplayNameA, ULONG * pchEaten, LPMONIKERA * ppmkOutA)
{
     TraceMethodEnter("COleItemContainerA::ParseDisplayName", this);

     LPBC pbc;
     LPOLESTR lpszDisplayName;
     LPMONIKER pmk;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IParseDisplayName, ParseDisplayName));

     *ppmkOutA = NULL;

     hResult = WrapIBindCtxWFromA(pbcA, &pbc);
     if (FAILED(hResult))
           return (hResult);

     hResult = ConvertStringToW(lpszDisplayNameA, &lpszDisplayName);
     if (FAILED(hResult))
           goto Error;

     hResult = GetWide()->ParseDisplayName(pbc, lpszDisplayName, pchEaten,
&pmk);
     if (FAILED(hResult))
           goto Error1;

     hResult = WrapIMonikerAFromW(pmk, ppmkOutA);

     if (pmk)
           pmk->Release();
Error1:
     ConvertStringFree(lpszDisplayName);

Error:
     if (pbc)
           pbc->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleItemContainerA::EnumObjects, public
//
//  Synopsis:   Thunks EnumObjects to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleItemContainerA::EnumObjects(DWORD grfFlags,
           LPENUMUNKNOWN * ppenumUnknown)
{
     TraceMethodEnter("COleItemContainerA::EnumObjects", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleContainer, EnumObjects));

     return GetWide()->EnumObjects(grfFlags, ppenumUnknown);
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleItemContainerA::LockContainer, public
//
//  Synopsis:   Thunks LockContainer to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleItemContainerA::LockContainer(BOOL fLock)
{
     TraceMethodEnter("COleItemContainerA::LockContainer", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleContainer, LockContainer));

     return GetWide()->LockContainer(fLock);
}
```

```
//
+---------------------------------------------------------------------------
//
//  Member:     COleItemContainerA::GetObject, public
//
//  Synopsis:   Thunks GetObject to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleItemContainerA::GetObject(LPSTR lpszItemA,
           DWORD dwSpeedNeeded, LPBINDCTXA pbcA, REFIID riid,
           LPVOID * ppvObject)
{
     TraceMethodEnter("COleItemContainerA::GetObject", this);

     LPOLESTR lpszItem;
     LPBINDCTX   pbc;
     LPUNKNOWN   punk;
     IDINTERFACE idRef;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleItemContainer, GetObject));


     hResult = ConvertStringToW(lpszItemA, &lpszItem);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapIBindCtxWFromA(pbcA, &pbc);
     if (FAILED(hResult))
           goto Error;

     hResult = GetWide()->GetObject(lpszItem, dwSpeedNeeded, pbc, riid,
                 (LPVOID *)&punk);
     if (FAILED(hResult))
           goto Error1;

     idRef = WrapTranslateIID(riid);

     hResult = WrapInterfaceAFromW(idRef, punk, (LPUNKNOWN *)ppvObject);

     if (punk)
           punk->Release();

Error1:
     if (pbc)
           pbc->Release();

Error:
     ConvertStringFree(lpszItem);

     return hResult;
```

```
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleItemContainerA::GetObjectStorage, public
//
//  Synopsis:   Thunks GetObjectStorage to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleItemContainerA::GetObjectStorage(LPSTR lpszItemA,
           LPBINDCTXA pbcA, REFIID riid, LPVOID * ppvStorage)
{
     TraceMethodEnter("COleItemContainerA::GetObjectStorage", this);

     LPOLESTR lpszItem;
     LPBINDCTX   pbc;
     LPUNKNOWN   punk;
     IDINTERFACE idRef;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleItemContainer, GetObjectStorage));

     hResult = ConvertStringToW(lpszItemA, &lpszItem);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapIBindCtxWFromA(pbcA, &pbc);
     if (FAILED(hResult))
           goto Error;

     hResult = GetWide()->GetObjectStorage(lpszItem, pbc, riid, (LPVOID
*)&punk);
     if (FAILED(hResult))
           goto Error1;

     idRef = WrapTranslateIID(riid);

     hResult = WrapInterfaceAFromW(idRef, punk, (LPUNKNOWN *)ppvStorage);

     if (punk)
           punk->Release();

Error1:
     if (pbc)
           pbc->Release();

Error:
     ConvertStringFree(lpszItem);
     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleItemContainerA::IsRunning, public
//
//  Synopsis:   Thunks IsRunning to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleItemContainerA::IsRunning(LPSTR lpszItemA)
{
     TraceMethodEnter("COleItemContainerA::IsRunning", this);

     LPOLESTR lpszItem;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleItemContainer, IsRunning));

     hResult = ConvertStringToW(lpszItemA, &lpszItem);
     if (FAILED(hResult))
           return (hResult);

     hResult = GetWide()->IsRunning(lpszItem);

     ConvertStringFree(lpszItem);

     return hResult;
}
```

IOleAdviseHolderA Implementation (OLEANSI Sample)

```
//
**********************************************************************
//
//                    IOleAdviseHolderA Implementation
//
//
**********************************************************************

//
+---------------------------------------------------------------------
//
//  Member:     COleAdviseHolderA::Advise, public
//
//  Synopsis:   Thunks Advise to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------
-
STDMETHODIMP COleAdviseHolderA::Advise(LPADVISESINKA pAdviseA,
           DWORD * pdwConnection)
{
     TraceMethodEnter("COleAdviseHolderA::Advise", this);

     LPADVISESINK pAdvise;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleAdviseHolder, Advise));

     hResult = WrapIAdviseSinkWFromA(pAdviseA, &pAdvise);
     if (FAILED(hResult))
           return hResult;

     hResult =  GetWide()->Advise(pAdvise, pdwConnection);

     if (pAdvise)
           pAdvise->Release();

     return hResult;
}


//
+---------------------------------------------------------------------
//
//  Member:     COleAdviseHolderA::Unadvise, public
//
//  Synopsis:   Thunks Unadvise to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
```

```
//----------------------------------------------------------------------------
-
STDMETHODIMP COleAdviseHolderA::Unadvise(DWORD dwConnection)
{
     TraceMethodEnter("COleAdviseHolderA::Unadvise", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleAdviseHolder, Unadvise));

     return GetWide()->Unadvise(dwConnection);
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleAdviseHolderA::EnumAdvise, public
//
//  Synopsis:   Thunks EnumAdvise to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleAdviseHolderA::EnumAdvise(LPENUMSTATDATAA * ppenumAdviseA)
{
     TraceMethodEnter("COleAdviseHolderA::EnumAdvise", this);

     LPENUMSTATDATA penumAdvise = NULL;
     HRESULT hResult;



     _DebugHook(GetWide(), MEMBER_PTR(IOleAdviseHolder, EnumAdvise));

     hResult = GetWide()->EnumAdvise(&penumAdvise);
     if (FAILED(hResult))
           return (hResult);

     if (penumAdvise)
     {
           hResult = WrapIEnumSTATDATAAFromW(penumAdvise, ppenumAdviseA);
           penumAdvise->Release();
     }
     else
           *ppenumAdviseA = NULL;

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleAdviseHolderA::SendOnRename, public
//
```

```
//  Synopsis:   Thunks SendOnRename to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP COleAdviseHolderA::SendOnRename(LPMONIKERA pmkA)
{
     TraceMethodEnter("COleAdviseHolderA::SendOnRename", this);

     LPMONIKER pmk;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleAdviseHolder, SendOnRename));

     hResult = WrapIMonikerWFromA(pmkA, &pmk);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->SendOnRename(pmk);

     if (pmk)
           pmk->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     COleAdviseHolderA::SendOnSave, public
//
//  Synopsis:   Thunks SendOnSave to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP COleAdviseHolderA::SendOnSave(VOID)
{
     TraceMethodEnter("COleAdviseHolderA::SendOnSave", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleAdviseHolder, SendOnSave));

     return GetWide()->SendOnSave();
}


//
+--------------------------------------------------------------------------
//
//  Member:     COleAdviseHolderA::SendOnClose, public
```

```
//
//  Synopsis:   Thunks SendOnClose to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//-------------------------------------------------------------------------
-
STDMETHODIMP COleAdviseHolderA::SendOnClose(VOID)
{
     TraceMethodEnter("COleAdviseHolderA::SendOnClose", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleAdviseHolder, SendOnClose));

     return GetWide()->SendOnClose();
}
```

IOleLinkA Implementation (OLEANSI Sample)

```
//
***************************************************************************
//
//                    IOleLinkA Implementation
//
//
***************************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     COleLinkA::SetUpdateOptions, public
//
//  Synopsis:   Thunks SetUpdateOptions to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//-------------------------------------------------------------------------
-
STDMETHODIMP COleLinkA::SetUpdateOptions(DWORD dwUpdateOpt)
{
     TraceMethodEnter("COleLinkA::SetUpdateOptions", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleLink, SetUpdateOptions));

     return GetWide()->SetUpdateOptions(dwUpdateOpt);
}


//
+--------------------------------------------------------------------------
//
//  Member:     COleLinkA::GetUpdateOptions, public
//
//  Synopsis:   Thunks GetUpdateOptions to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//-------------------------------------------------------------------------
-
STDMETHODIMP COleLinkA::GetUpdateOptions(LPDWORD pdwUpdateOpt)
{
     TraceMethodEnter("COleLinkA::GetUpdateOptions", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleLink, GetUpdateOptions));

     return GetWide()->GetUpdateOptions(pdwUpdateOpt);
}


//
+--------------------------------------------------------------------------
```

```
//
//  Member:     COleLinkA::SetSourceMoniker, public
//
//  Synopsis:   Thunks SetSourceMoniker to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP COleLinkA::SetSourceMoniker(LPMONIKERA pmkA, REFCLSID rclsid)
{
     TraceMethodEnter("COleLinkA::SetSourceMoniker", this);

     LPMONIKER pmk;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleLink, SetSourceMoniker));

     hResult = WrapIMonikerWFromA(pmkA, &pmk);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->SetSourceMoniker(pmk, rclsid);

     if (pmk)
           pmk->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     COleLinkA::GetSourceMoniker, public
//
//  Synopsis:   Thunks GetSourceMoniker to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP COleLinkA::GetSourceMoniker(LPMONIKERA * ppmkA)
{
     TraceMethodEnter("COleLinkA::GetSourceMoniker", this);

     LPMONIKER pmk;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleLink, GetSourceMoniker));

     *ppmkA = NULL;
     hResult = GetWide()->GetSourceMoniker(&pmk);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapIMonikerAFromW(pmk, ppmkA);

     if (pmk)
           pmk->Release();

     return hResult;
}


//
+-------------------------------------------------------------------------
//
//  Member:     COleLinkA::SetSourceDisplayName, public
//
//  Synopsis:   Thunks SetSourceDisplayName to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP COleLinkA::SetSourceDisplayName(LPCSTR lpszDisplayNameA)
{
     TraceMethodEnter("COleLinkA::SetSourceDisplayName", this);


     LPOLESTR lpszDisplayName;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleLink, SetSourceDisplayName));

     hResult = ConvertStringToW(lpszDisplayNameA, &lpszDisplayName);
     if (FAILED(hResult))
           return (hResult);

     hResult = GetWide()->SetSourceDisplayName(lpszDisplayName);

     ConvertStringFree(lpszDisplayName);

     return hResult;
}


//
+-------------------------------------------------------------------------
//
//  Member:     COleLinkA::GetSourceDisplayName, public
//
//  Synopsis:   Thunks GetSourceDisplayName to Unicode method.
//
//  Returns:    OLE2 Result Code.
```

```
//
//---------------------------------------------------------------------------
-
STDMETHODIMP COleLinkA::GetSourceDisplayName(LPSTR * lplpszDisplayNameA)
{
     TraceMethodEnter("COleLinkA::GetSourceDisplayName", this);

     LPOLESTR lpszDisplayName;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleLink, GetSourceDisplayName));

     hResult = GetWide()->GetSourceDisplayName(&lpszDisplayName);
     if (FAILED(hResult))
           return (hResult);

     hResult = ConvertStringToA(lpszDisplayName, lplpszDisplayNameA);

     ConvertStringFree(lpszDisplayName);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleLinkA::BindToSource, public
//
//  Synopsis:   Thunks BindToSource to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP COleLinkA::BindToSource(DWORD bindflags, LPBINDCTXA pbcA)
{
     TraceMethodEnter("COleLinkA::BindToSource", this);

     LPBINDCTX pbc;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleLink, BindToSource));

     hResult = WrapIBindCtxWFromA(pbcA, &pbc);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->BindToSource(bindflags, pbc);

     if (pbc)
           pbc->Release();
     TraceMethodExit("COleLinkA::BindToSource", this, hResult);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleLinkA::BindIfRunning, public
//
//  Synopsis:   Thunks BindIfRunning to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleLinkA::BindIfRunning(VOID)
{
     TraceMethodEnter("COleLinkA::BindIfRunning", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleLink, BindIfRunning));

     return GetWide()->BindIfRunning();
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleLinkA::GetBoundSource, public
//
//  Synopsis:   Thunks GetBoundSource to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleLinkA::GetBoundSource(LPUNKNOWN * ppUnk)
{
     TraceMethodEnter("COleLinkA::GetBoundSource", this);

     LPUNKNOWN punk;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleLink, GetBoundSource));

     hResult = GetWide()->GetBoundSource(&punk);
     if (FAILED(hResult))
             return hResult;

     hResult = WrapIUnknownAFromW(punk, ppUnk);
     punk->Release();
     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleLinkA::UnbindSource, public
//
//  Synopsis:   Thunks UnbindSource to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleLinkA::UnbindSource(VOID)
{
     TraceMethodEnter("COleLinkA::UnbindSource", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleLink, UnbindSource));

     return GetWide()->UnbindSource();
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleLinkA::Update, public
//
//  Synopsis:   Thunks Update to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleLinkA::Update(LPBINDCTXA pbcA)
{
     TraceMethodEnter("COleLinkA::Update", this);

     LPBINDCTX pbc;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleLink, Update));

     hResult = WrapIBindCtxWFromA(pbcA, &pbc);
     if (FAILED(hResult))
          return hResult;

     hResult = GetWide()->Update(pbc);

     if (pbc)
          pbc->Release();
```

```
    return hResult;
}
```

IOleInPlaceObjectA Implementation   (OLEANSI Sample)

```
//
**********************************************************************
//
//                    IOleInPlaceObjectA Implementation
//
//
**********************************************************************

//
+---------------------------------------------------------------------
//
//  Member:     COleInPlaceObjectA::GetWindow, public
//
//  Synopsis:   Thunks GetWindow to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceObjectA::GetWindow(HWND * lphwnd)
{
     TraceMethodEnter("COleInPlaceObjectA::GetWindow", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleWindow, GetWindow));

     return GetWide()->GetWindow(lphwnd);
}


//
+---------------------------------------------------------------------
//
//  Member:     COleInPlaceObjectA::ContextSensitiveHelp, public
//
//  Synopsis:   Thunks ContextSensitiveHelp to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceObjectA::ContextSensitiveHelp(BOOL fEnterMode)
{
     TraceMethodEnter("COleInPlaceObjectA::ContextSensitiveHelp", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleWindow, ContextSensitiveHelp));

     return GetWide()->ContextSensitiveHelp(fEnterMode);
}


//
+---------------------------------------------------------------------
```

```
//
//  Member:     COleInPlaceObjectA::InPlaceDeactivate, public
//
//  Synopsis:   Thunks InPlaceDeactivate to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceObjectA::InPlaceDeactivate(VOID)
{
     TraceMethodEnter("COleInPlaceObjectA::InPlaceDeactivate", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceObject,
InPlaceDeactivate));

     return GetWide()->InPlaceDeactivate();
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleInPlaceObjectA::UIDeactivate, public
//
//  Synopsis:   Thunks UIDeactivate to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceObjectA::UIDeactivate(VOID)
{
     TraceMethodEnter("COleInPlaceObjectA::UIDeactivate", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceObject, UIDeactivate));

     return GetWide()->UIDeactivate();
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleInPlaceObjectA::SetObjectRects, public
//
//  Synopsis:   Thunks SetObjectRects to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceObjectA::SetObjectRects(LPCRECT lprcPosRect,
           LPCRECT lprcClipRect)
{
     TraceMethodEnter("COleInPlaceObjectA::SetObjectRects", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceObject, SetObjectRects));

     return GetWide()->SetObjectRects(lprcPosRect, lprcClipRect);
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleInPlaceObjectA::ReactivateAndUndo, public
//
//  Synopsis:   Thunks ReactivateAndUndo to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceObjectA::ReactivateAndUndo(VOID)
{
     TraceMethodEnter("COleInPlaceObjectA::ReactivateAndUndo", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceObject,
ReactivateAndUndo));

     return GetWide()->ReactivateAndUndo();
}
```

IOleInPlaceActiveObjectA Implementation   (OLEANSI Sample)

```
//
**********************************************************************
//
//                    IOleInPlaceActiveObjectA Implementation
//
//
**********************************************************************

//
+---------------------------------------------------------------------
//
//  Member:     COleInPlaceActiveObjectA::GetWindow, public
//
//  Synopsis:   Thunks GetWindow to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceActiveObjectA::GetWindow(HWND * lphwnd)
{
     TraceMethodEnter("COleInPlaceActiveObjectA::GetWindow", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleWindow, GetWindow));

     return GetWide()->GetWindow(lphwnd);
}


//
+---------------------------------------------------------------------
//
//  Member:     COleInPlaceActiveObjectA::ContextSensitiveHelp, public
//
//  Synopsis:   Thunks ContextSensitiveHelp to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceActiveObjectA::ContextSensitiveHelp(BOOL fEnterMode)
{
     TraceMethodEnter("COleInPlaceActiveObjectA::ContextSensitiveHelp",
this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleWindow, ContextSensitiveHelp));

     return GetWide()->ContextSensitiveHelp(fEnterMode);
}
```

```
//
+---------------------------------------------------------------------------
//
//  Member:     COleInPlaceActiveObjectA::TranslateAccelerator, public
//
//  Synopsis:   Thunks TranslateAccelerator to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceActiveObjectA::TranslateAccelerator(LPMSG lpmsg)
{
     TraceMethodEnter("COleInPlaceActiveObjectA::TranslateAccelerator",
this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceActiveObject,
TranslateAccelerator));

     return GetWide()->TranslateAccelerator(lpmsg);
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleInPlaceActiveObjectA::OnFrameWindowActivate, public
//
//  Synopsis:   Thunks OnFrameWindowActivate to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceActiveObjectA::OnFrameWindowActivate(BOOL fActivate)
{
     TraceMethodEnter("COleInPlaceActiveObjectA::OnFrameWindowActivate",
this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceActiveObject,
OnFrameWindowActivate));

     return GetWide()->OnFrameWindowActivate(fActivate);
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleInPlaceActiveObjectA::OnDocWindowActivate, public
//
//  Synopsis:   Thunks OnDocWindowActivate to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
```

```
//----------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceActiveObjectA::OnDocWindowActivate(BOOL fActivate)
{
     TraceMethodEnter("COleInPlaceActiveObjectA::OnDocWindowActivate",
this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceActiveObject,
OnDocWindowActivate));

     return GetWide()->OnDocWindowActivate(fActivate);
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleInPlaceActiveObjectA::ResizeBorder, public
//
//  Synopsis:   Thunks ResizeBorder to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceActiveObjectA::ResizeBorder(LPCRECT lprectBorder,
           LPOLEINPLACEUIWINDOWA lpUIWindowA, BOOL fFrameWindow)
{
     TraceMethodEnter("COleInPlaceActiveObjectA::ResizeBorder", this);

     LPOLEINPLACEUIWINDOW lpUIWindow;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceActiveObject,
ResizeBorder));

     hResult = WrapIOleInPlaceUIWindowWFromA(lpUIWindowA, &lpUIWindow);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->ResizeBorder(lprectBorder, lpUIWindow,
fFrameWindow);

     if (lpUIWindow)
           lpUIWindow->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
```

```
//  Member:     COleInPlaceActiveObjectA::EnableModeless, public
//
//  Synopsis:   Thunks EnableModeless to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceActiveObjectA::EnableModeless(BOOL fEnable)
{
     TraceMethodEnter("COleInPlaceActiveObjectA::EnableModeless", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceActiveObject,
EnableModeless));

     return GetWide()->EnableModeless(fEnable);
}
```

IOleInPlaceFrameA Implementation (OLEANSI Sample)

```
//
**********************************************************************
//
//                    IOleInPlaceFrameA Implementation
//
//
**********************************************************************

//
+---------------------------------------------------------------------
//
//  Member:     COleInPlaceFrameA::GetWindow, public
//
//  Synopsis:   Thunks GetWindow to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceFrameA::GetWindow(HWND * lphwnd)
{
     TraceMethodEnter("COleInPlaceFrameA::GetWindow", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleWindow, GetWindow));

     return GetWide()->GetWindow(lphwnd);
}


//
+---------------------------------------------------------------------
//
//  Member:     COleInPlaceFrameA::ContextSensitiveHelp, public
//
//  Synopsis:   Thunks ContextSensitiveHelp to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceFrameA::ContextSensitiveHelp(BOOL fEnterMode)
{
     TraceMethodEnter("COleInPlaceFrameA::ContextSensitiveHelp", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleWindow, ContextSensitiveHelp));

     return GetWide()->ContextSensitiveHelp(fEnterMode);
}


//
+---------------------------------------------------------------------
```

```
//
//  Member:     COleInPlaceFrameA::GetBorder, public
//
//  Synopsis:   Thunks GetBorder to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceFrameA::GetBorder(LPRECT lprectBorder)
{
     TraceMethodEnter("COleInPlaceFrameA::GetBorder", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceUIWindow, GetBorder));

     return GetWide()->GetBorder(lprectBorder);
}


//
+--------------------------------------------------------------------------
//
//  Member:     COleInPlaceFrameA::RequestBorderSpace, public
//
//  Synopsis:   Thunks RequestBorderSpace to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceFrameA::RequestBorderSpace(
           LPCBORDERWIDTHS lpborderwidths)
{
     TraceMethodEnter("COleInPlaceFrameA::RequestBorderSpace", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceUIWindow,
RequestBorderSpace));

     return GetWide()->RequestBorderSpace(lpborderwidths);
}


//
+--------------------------------------------------------------------------
//
//  Member:     COleInPlaceFrameA::SetBorderSpace, public
//
//  Synopsis:   Thunks SetBorderSpace to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceFrameA::SetBorderSpace(
           LPCBORDERWIDTHS lpborderwidths)
```

```
{
     TraceMethodEnter("COleInPlaceFrameA::SetBorderSpace", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceUIWindow, SetBorderSpace));

     return GetWide()->SetBorderSpace(lpborderwidths);
}


//
+--------------------------------------------------------------------------
//
//  Member:     COleInPlaceFrameA::SetActiveObject, public
//
//  Synopsis:   Thunks SetActiveObject to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceFrameA::SetActiveObject(
           LPOLEINPLACEACTIVEOBJECTA lpActiveObjectA, LPCSTR lpszObjNameA)
{
     TraceMethodEnter("COleInPlaceFrameA::SetActiveObject", this);


     LPOLEINPLACEACTIVEOBJECT lpActiveObject;
     LPOLESTR lpszObjName;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceUIWindow,
SetActiveObject));

     hResult = WrapIOleInPlaceActiveObjectWFromA(lpActiveObjectA,
&lpActiveObject);
     if (FAILED(hResult))
           return hResult;

     hResult = ConvertStringToW(lpszObjNameA, &lpszObjName);
     if (FAILED(hResult))
           goto Error;

     hResult = GetWide()->SetActiveObject(lpActiveObject, lpszObjName);

     ConvertStringFree(lpszObjName);

Error:
     if (lpActiveObject)
           lpActiveObject->Release();

     return hResult;
}
```

```
//
+-------------------------------------------------------------------------
//
//  Member:     COleInPlaceFrameA::InsertMenus, public
//
//  Synopsis:   Thunks InsertMenus to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceFrameA::InsertMenus(HMENU hmenuShared,
           LPOLEMENUGROUPWIDTHS lpMenuWidths)
{
     TraceMethodEnter("COleInPlaceFrameA::InsertMenus", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceFrame, InsertMenus));

     return GetWide()->InsertMenus(hmenuShared, lpMenuWidths);
}


//
+-------------------------------------------------------------------------
//
//  Member:     COleInPlaceFrameA::SetMenu, public
//
//  Synopsis:   Thunks SetMenu to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceFrameA::SetMenu(HMENU hmenuShared, HOLEMENU
holemenu,
           HWND hwndActiveObject)
{
     TraceMethodEnter("COleInPlaceFrameA::SetMenu", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceFrame, SetMenu));

     return GetWide()->SetMenu(hmenuShared, holemenu, hwndActiveObject);
}


//
+-------------------------------------------------------------------------
//
//  Member:     COleInPlaceFrameA::RemoveMenus, public
//
//  Synopsis:   Thunks RemoveMenus to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
```

```
//--------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceFrameA::RemoveMenus(HMENU hmenuShared)
{
     TraceMethodEnter("COleInPlaceFrameA::RemoveMenus", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceFrame, RemoveMenus));

     return GetWide()->RemoveMenus(hmenuShared);
}


//
+--------------------------------------------------------------------------
//
//  Member:     COleInPlaceFrameA::SetStatusText, public
//
//  Synopsis:   Thunks SetStatusText to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceFrameA::SetStatusText(LPCSTR lpszStatusTextA)
{
     TraceMethodEnter("COleInPlaceFrameA::SetStatusText", this);

     LPOLESTR lpszStatusText;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceFrame, SetStatusText));

     hResult = ConvertStringToW(lpszStatusTextA, &lpszStatusText);
     if (FAILED(hResult))
           return (hResult);

     hResult = GetWide()->SetStatusText(lpszStatusText);

     ConvertStringFree(lpszStatusText);

     return hResult;
}


//
+--------------------------------------------------------------------------

//
//  Member:     COleInPlaceFrameA::EnableModeless, public
//
//  Synopsis:   Thunks EnableModeless to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
```

```
//--------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceFrameA::EnableModeless(BOOL fEnable)
{
     TraceMethodEnter("COleInPlaceFrameA::EnableModeless", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceFrame, EnableModeless));

     return GetWide()->EnableModeless(fEnable);
}


//
+--------------------------------------------------------------------------
//
//  Member:     COleInPlaceFrameA::TranslateAccelerator, public
//
//  Synopsis:   Thunks TranslateAccelerator to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceFrameA::TranslateAccelerator(LPMSG lpmsg, WORD wID)
{
     TraceMethodEnter("COleInPlaceFrameA::TranslateAccelerator", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceFrame,
TranslateAccelerator));

     return GetWide()->TranslateAccelerator(lpmsg, wID);
}
```

IOleInPlaceSiteA Implementation   (OLEANSI Sample)

```
//
***************************************************************************
//
//                   IOleInPlaceSiteA Implementation
//
//
***************************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     COleInPlaceSiteA::GetWindow, public
//
//  Synopsis:   Thunks GetWindow to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceSiteA::GetWindow(HWND * lphwnd)
{
     TraceMethodEnter("COleInPlaceSiteA::GetWindow", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleWindow, GetWindow));

     return GetWide()->GetWindow(lphwnd);
}


//
+--------------------------------------------------------------------------
//
//  Member:     COleInPlaceSiteA::ContextSensitiveHelp, public
//
//  Synopsis:   Thunks ContextSensitiveHelp to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceSiteA::ContextSensitiveHelp(BOOL fEnterMode)
{
     TraceMethodEnter("COleInPlaceSiteA::ContextSensitiveHelp", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleWindow, ContextSensitiveHelp));

     return GetWide()->ContextSensitiveHelp(fEnterMode);
}


//
+--------------------------------------------------------------------------
```

```
//
//  Member:     COleInPlaceSiteA::CanInPlaceActivate, public
//
//  Synopsis:   Thunks CanInPlaceActivate to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceSiteA::CanInPlaceActivate(VOID)
{
     TraceMethodEnter("COleInPlaceSiteA::CanInPlaceActivate", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceSite, CanInPlaceActivate));

     return GetWide()->CanInPlaceActivate();
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleInPlaceSiteA::OnInPlaceActivate, public
//
//  Synopsis:   Thunks OnInPlaceActivate to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceSiteA::OnInPlaceActivate(VOID)
{
     TraceMethodEnter("COleInPlaceSiteA::OnInPlaceActivate", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceSite, OnInPlaceActivate));

     return GetWide()->OnInPlaceActivate();
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleInPlaceSiteA::OnUIActivate, public
//
//  Synopsis:   Thunks OnUIActivate to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceSiteA::OnUIActivate(VOID)
{
     TraceMethodEnter("COleInPlaceSiteA::OnUIActivate", this);
     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceSite, OnUIActivate));

     return GetWide()->OnUIActivate();
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleInPlaceSiteA::GetWindowContext, public
//
//  Synopsis:   Thunks GetWindowContext to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceSiteA::GetWindowContext(
           LPOLEINPLACEFRAMEA * lplpFrameA, LPOLEINPLACEUIWINDOWA *
lplpDocA,
           LPRECT lprcPosRect, LPRECT lprcClipRect,
           LPOLEINPLACEFRAMEINFO lpFrameInfo)
{
     TraceMethodEnter("COleInPlaceSiteA::GetWindowContext", this);


     LPOLEINPLACEFRAME lpFrame;
     LPOLEINPLACEUIWINDOW lpDoc;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceSite, GetWindowContext));

     *lplpFrameA = NULL;
     *lplpDocA = NULL;

     hResult = GetWide()->GetWindowContext(&lpFrame, &lpDoc, lprcPosRect,
                 lprcClipRect, lpFrameInfo);
     if (FAILED(hResult))
           goto Error;

     hResult = WrapIOleInPlaceFrameAFromW(lpFrame, lplpFrameA);
     if (FAILED(hResult))
           goto Error1;

     hResult = WrapIOleInPlaceUIWindowAFromW(lpDoc, lplpDocA);

Error:
     if (lpFrame)
           lpFrame->Release();
Error1:
     if (lpDoc)
           lpDoc->Release();

     return hResult;
```

```
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleInPlaceSiteA::Scroll, public
//
//  Synopsis:   Thunks Scroll to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceSiteA::Scroll(SIZE scrollExtent)
{
     TraceMethodEnter("COleInPlaceSiteA::Scroll", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceSite, Scroll));

     return GetWide()->Scroll(scrollExtent);
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleInPlaceSiteA::OnUIDeactivate, public
//
//  Synopsis:   Thunks OnUIDeactivate to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceSiteA::OnUIDeactivate(BOOL fUndoable)
{
     TraceMethodEnter("COleInPlaceSiteA::OnUIDeactivate", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceSite, OnUIDeactivate));

     return GetWide()->OnUIDeactivate(fUndoable);
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleInPlaceSiteA::OnInPlaceDeactivate, public
//
//  Synopsis:   Thunks OnInPlaceDeactivate to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
```

```
//---------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceSiteA::OnInPlaceDeactivate(VOID)
{
     TraceMethodEnter("COleInPlaceSiteA::OnInPlaceDeactivate", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceSite,
OnInPlaceDeactivate));

     return GetWide()->OnInPlaceDeactivate();
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleInPlaceSiteA::DiscardUndoState, public
//
//  Synopsis:   Thunks DiscardUndoState to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceSiteA::DiscardUndoState(VOID)
{
     TraceMethodEnter("COleInPlaceSiteA::DiscardUndoState", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceSite, DiscardUndoState));

     return GetWide()->DiscardUndoState();
}


//
+---------------------------------------------------------------------------
//
//  Member:     COleInPlaceSiteA::DeactivateAndUndo, public
//
//  Synopsis:   Thunks DeactivateAndUndo to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceSiteA::DeactivateAndUndo(VOID)
{
     TraceMethodEnter("COleInPlaceSiteA::DeactivateAndUndo", this);


     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceSite, DeactivateAndUndo));

     return GetWide()->DeactivateAndUndo();
}
```

```
//
+-------------------------------------------------------------------------
//
//  Member:     COleInPlaceSiteA::OnPosRectChange, public
//
//  Synopsis:   Thunks OnPosRectChange to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP COleInPlaceSiteA::OnPosRectChange(LPCRECT lprcPosRect)
{
     TraceMethodEnter("COleInPlaceSiteA::OnPosRectChange", this);

     _DebugHook(GetWide(), MEMBER_PTR(IOleInPlaceSite, OnPosRectChange));

     return GetWide()->OnPosRectChange(lprcPosRect);
}
```

ANSIOLE3.CPP (OLEANSI Sample)

```
//
+---------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       ansiole3.cpp
//
//  Contents:   ANSI Wrappers for Unicode Ole2 Interfaces and APIs.
//
//  Functions:  ReadClassStgA
//
//  History:    01-Nov-93   v-kentc     Created.
//
//---------------------------------------------------------------------------
-

#include "Ole2Ansi.h"



//
+---------------------------------------------------------------------------
//
//  Routine:    CreateDataCacheA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI CreateDataCacheA(LPUNKNOWN pUnkOuterA, REFCLSID rclsid,
                             REFIID iid, LPVOID FAR* ppv)
{
     TraceSTDAPIEnter("CreateDataCacheA");
     LPUNKNOWN pUnkOuter;
     LPUNKNOWN pUnk;
     IDINTERFACE      idRef;
     HRESULT hResult;


     hResult = WrapIUnknownWFromA(pUnkOuterA, &pUnkOuter);
     if (FAILED(hResult))
            return hResult;

     hResult = CreateDataCache(pUnkOuter, rclsid, iid, (LPVOID *)&pUnk);
     if (FAILED(hResult))
            goto Error;

     //
     //  Convert the 16 bytes GUID into an internal integer for speed.
     //
     idRef = WrapTranslateIID(iid);

     hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)ppv);

     if (pUnk)
          pUnk->Release();

Error:
     if (pUnkOuter)
          pUnkOuter->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleCreateDefaultHandlerA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI OleCreateDefaultHandlerA(REFCLSID clsid, LPUNKNOWN pUnkOuter,
                             REFIID riid, LPVOID FAR* lplpObj)
{
     TraceSTDAPIEnter("OleCreateDefaultHandlerA");
     LPUNKNOWN pUnk;
     IDINTERFACE      idRef;
     HRESULT hResult;

     // Note: Since OleCreateDefaultHandler doesn't increase pUnkOuter's
     // reference count, we can't create a wrapper for it here and then
     // release it, because we'd leave the default handler with a pointer
     // to garbage.  Instead, we insist on having pUnkOuter "pre-wrapped"
     // by the caller.

     hResult = OleCreateDefaultHandler(clsid, pUnkOuter, riid, (LPVOID
*)&pUnk);
     if (FAILED(hResult))
          goto Error;

     //
     //  Convert the 16 bytes GUID into an internal integer for speed.
     //
     idRef = WrapTranslateIID(riid);

     hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)lplpObj);

     if (pUnk)
           pUnk->Release();
```

```
Error:
     return hResult;
}



//
+---------------------------------------------------------------------------
//
//  Routine:    OleDrawA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI OleDrawA(LPUNKNOWN pUnknownA, DWORD dwAspect, HDC hdcDraw,
                             LPCRECT lprcBounds)
{
     TraceSTDAPIEnter("OleDrawA");
     LPUNKNOWN pUnk;
     HRESULT hResult;


     hResult = WrapIUnknownWFromA(pUnknownA, &pUnk);
     if (FAILED(hResult))
     return hResult;

     hResult = OleDraw(pUnk, dwAspect, hdcDraw, lprcBounds);

     if (pUnk)
           pUnk->Release();

     return hResult;
}



//
+---------------------------------------------------------------------------
//
//  Routine:    OleNoteObjectVisibleA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI OleNoteObjectVisibleA(LPUNKNOWN pUnknown, BOOL fVisible)
{
     TraceSTDAPIEnter("OleNoteObjectVisibleA");
     LPUNKNOWN pUnk;
```

```
    HRESULT hResult;


    hResult = WrapIUnknownWFromA(pUnknown, &pUnk);
    if (FAILED(hResult))
         return hResult;

    hResult = OleNoteObjectVisible(pUnk, fVisible);

    if (pUnk)
         pUnk->Release();

    return hResult;
}



//
+---------------------------------------------------------------------------
//
//  Routine:    OleRunA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI OleRunA(LPUNKNOWN pUnknown)
{
    TraceSTDAPIEnter("OleRunA");
    LPUNKNOWN pUnk;
    HRESULT hResult;


    hResult = WrapIUnknownWFromA(pUnknown, &pUnk);
    if (FAILED(hResult))
         return hResult;

    hResult = OleRun(pUnk);

    if (pUnk)
         pUnk->Release();

    return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleSetContainedObjectA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
```

```
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI OleSetContainedObjectA(LPUNKNOWN pUnknown, BOOL fContained)
{
     TraceSTDAPIEnter("OleSetContainedObjectA");
     LPUNKNOWN pUnk;
     HRESULT hResult;


     hResult = WrapIUnknownWFromA(pUnknown, &pUnk);
     if (FAILED(hResult))
           return hResult;

     hResult = OleSetContainedObject(pUnk, fContained);

     if (pUnk)
           pUnk->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    ReadClassStgA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI ReadClassStgA(LPSTORAGEA pStgA, CLSID * pclsid)
{
     TraceSTDAPIEnter("ReadClassStgA");
     LPSTORAGE pStg;
     HRESULT hResult;


     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
           return hResult;

     hResult = ReadClassStg(pStg, pclsid);

     if (pStg)
           pStg->Release();

     return hResult;
}
```

```
//
+--------------------------------------------------------------------------
//
//  Routine:    WriteClassStgA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
STDAPI WriteClassStgA(LPSTORAGEA pStgA, REFCLSID rclsid)
{
     TraceSTDAPIEnter("WriteClassStgA");
     LPSTORAGE pStg;
     HRESULT hResult;


     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
           return hResult;

     hResult = WriteClassStg(pStg, rclsid);

     if (pStg)
           pStg->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Routine:    ReadClassStmA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
STDAPI ReadClassStmA(LPSTREAMA pStmA, CLSID * pclsid)
{
     TraceSTDAPIEnter("ReadClassStmA");
     LPSTREAM pStm;
     HRESULT hResult;


     hResult = WrapIStreamWFromA(pStmA, &pStm);
     if (FAILED(hResult))
           return hResult;

     hResult = ReadClassStm(pStm, pclsid);
     if (pStm)
          pStm->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    WriteClassStmA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI WriteClassStmA(LPSTREAMA pStmA, REFCLSID rclsid)
{
     TraceSTDAPIEnter("WriteClassStmA");
     LPSTREAM pStm;
     HRESULT hResult;


     hResult = WrapIStreamWFromA(pStmA, &pStm);
     if (FAILED(hResult))
          return hResult;

     hResult = WriteClassStm(pStm, rclsid);

     if (pStm)
          pStm->Release();

     return hResult;
}



//
+---------------------------------------------------------------------------
//
//  Routine:    WriteFmtUserTypeStgA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI WriteFmtUserTypeStgA(LPSTORAGEA pStgA, CLIPFORMAT cf,
           LPSTR lpszUserTypeA)
{
     TraceSTDAPIEnter("WriteFmtUserTypeStgA");
```

```
	LPSTORAGE pStg;
	OLECHAR szUserType[MAX_STRING], * pszUserType;
	HRESULT hResult;


	hResult = WrapIStorageWFromA(pStgA, &pStg);
	if (FAILED(hResult))
		return hResult;

	if (lpszUserTypeA)
	{
		ConvertStringToW(lpszUserTypeA, szUserType);
		pszUserType = szUserType;
	}
	else
		pszUserType = NULL;

	hResult = WriteFmtUserTypeStg(pStg, cf, pszUserType);

	if (pStg)
		pStg->Release();

	return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    ReadFmtUserTypeStgA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI ReadFmtUserTypeStgA(LPSTORAGEA pStgA, CLIPFORMAT * pcf,
		LPSTR * lplpszUserTypeA)
{
	TraceSTDAPIEnter("ReadFmtUserTypeStgA");
	LPSTORAGE pStg;
	LPOLESTR *lplpszUserType;
	LPOLESTR lpszUserType;
	HRESULT hResult;


	hResult = WrapIStorageWFromA(pStgA, &pStg);
	if (FAILED(hResult))
		return hResult;

	if (lplpszUserTypeA)
		lplpszUserType = &lpszUserType;
```

```
    else
          lplpszUserType = NULL;

    hResult = ReadFmtUserTypeStg(pStg, pcf, lplpszUserType);
    if (FAILED(hResult))
          goto Error;

    if (lplpszUserType)
    {
          hResult = ConvertStringToA(lpszUserType, lplpszUserTypeA);
          ConvertStringFree(lpszUserType);
    }

Error:
    if (pStg)
          pStg->Release();

    return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleQueryLinkFromDataA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI OleQueryLinkFromDataA(LPDATAOBJECTA pSrcDataObjectA)
{
    TraceSTDAPIEnter("OleQueryLinkFromDataA");
    LPDATAOBJECT pSrcDataObject;
    HRESULT hResult;


    hResult = WrapIDataObjectWFromA(pSrcDataObjectA, &pSrcDataObject);
    if (FAILED(hResult))
          return (hResult);

    hResult = OleQueryLinkFromData(pSrcDataObject);

    if (pSrcDataObject)
          pSrcDataObject->Release();

    return hResult;
}


//
+---------------------------------------------------------------------------
//
```

```
//  Routine:    OleQueryCreateFromDataA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI OleQueryCreateFromDataA(LPDATAOBJECTA pSrcDataObjectA)
{
     TraceSTDAPIEnter("OleQueryCreateFromDataA");
     LPDATAOBJECT pSrcDataObject;
     HRESULT hResult;


     hResult = WrapIDataObjectWFromA(pSrcDataObjectA, &pSrcDataObject);
     if (FAILED(hResult))
           return (hResult);

     hResult = OleQueryCreateFromData(pSrcDataObject);

     if (pSrcDataObject)
           pSrcDataObject->Release();

     return hResult;
}



//
+--------------------------------------------------------------------------
//
//  Routine:    OleCreateA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI OleCreateA(REFCLSID rclsid, REFIID riid, DWORD renderopt,
           LPFORMATETC pFormatEtc, LPOLECLIENTSITEA pClientSiteA,
           LPSTORAGEA pStgA, LPVOID * ppvObj)
{
     TraceSTDAPIEnter("OleCreateA");
     LPOLECLIENTSITE pClientSite;
     LPSTORAGE pStg;
     LPUNKNOWN pUnk;
     IDINTERFACE      idRef;
     HRESULT hResult;


     hResult = WrapIOleClientSiteWFromA(pClientSiteA, &pClientSite);
     if (FAILED(hResult))
           return (hResult);
     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
           goto Error;

     hResult = OleCreate(rclsid, riid, renderopt, pFormatEtc, pClientSite,
                 pStg, (LPVOID *)&pUnk);
     if (FAILED(hResult))
           goto Error;

     //
     //  Convert the 16 bytes GUID into an internal integer for speed.
     //
     idRef = WrapTranslateIID(riid);

     hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)ppvObj);

     if (pStg)
           pStg->Release();

     if (pUnk)
           pUnk->Release();

Error:
     if (pClientSite)
           pClientSite->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleCreateFromDataA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI OleCreateFromDataA(LPDATAOBJECTA pSrcDataObjA, REFIID riid,
           DWORD renderopt, LPFORMATETC pFormatEtc,
           LPOLECLIENTSITEA pClientSiteA, LPSTORAGEA pStgA, LPVOID * ppvObj)
{
     TraceSTDAPIEnter("OleCreateFromDataA");
     LPDATAOBJECT pSrcDataObj;
     LPOLECLIENTSITE pClientSite;
     LPSTORAGE pStg;
     LPUNKNOWN pUnk;
     IDINTERFACE      idRef;
     HRESULT hResult;
```

```
	hResult = WrapIDataObjectWFromA(pSrcDataObjA, &pSrcDataObj);
	if (FAILED(hResult))
		return (hResult);

	hResult = WrapIOleClientSiteWFromA(pClientSiteA, &pClientSite);
	if (FAILED(hResult))
		goto Error;

	hResult = WrapIStorageWFromA(pStgA, &pStg);
	if (FAILED(hResult))
		goto Error1;

	hResult = OleCreateFromData(pSrcDataObj, riid, renderopt, pFormatEtc,
			pClientSite, pStg, (LPVOID *)&pUnk);
	if (FAILED(hResult))
		goto Error1;

	//
	//  Convert the 16 bytes GUID into an internal integer for speed.
	//
	idRef = WrapTranslateIID(riid);

	hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)ppvObj);

	if (pStg)
		pStg->Release();

	if (pUnk)
		pUnk->Release();

Error1:
	if (pClientSite)
		pClientSite->Release();

Error:
	if (pSrcDataObj)
		pSrcDataObj->Release();

	return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleCreateLinkFromDataA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI OleCreateLinkFromDataA(LPDATAOBJECTA pSrcDataObjA, REFIID riid,
           DWORD renderopt, LPFORMATETC pFormatEtc,
           LPOLECLIENTSITEA pClientSiteA, LPSTORAGEA pStgA, LPVOID * ppvObj)
{
     TraceSTDAPIEnter("OleCreateLinkFromDataA");
     LPDATAOBJECT pSrcDataObj;
     LPOLECLIENTSITE pClientSite;
     LPSTORAGE pStg;
     LPUNKNOWN pUnk;
     IDINTERFACE      idRef;
     HRESULT hResult;


     hResult = WrapIDataObjectWFromA(pSrcDataObjA, &pSrcDataObj);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapIOleClientSiteWFromA(pClientSiteA, &pClientSite);
     if (FAILED(hResult))
           goto Error;

     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
           goto Error1;

     hResult = OleCreateLinkFromData(pSrcDataObj, riid, renderopt,
pFormatEtc,
                 pClientSite, pStg, (LPVOID *)&pUnk);
     if (FAILED(hResult))
           goto Error1;

     //
     //  Convert the 16 bytes GUID into an internal integer for speed.
     //
     idRef = WrapTranslateIID(riid);


     hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)ppvObj);

     if (pStg)
           pStg->Release();

     if (pUnk)
           pUnk->Release();

Error1:
     if (pClientSite)
           pClientSite->Release();

Error:
     if (pSrcDataObj)
           pSrcDataObj->Release();

     return hResult;
}
```

```
//
+--------------------------------------------------------------------------
//
//  Routine:    OleCreateStaticFromDataA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
STDAPI OleCreateStaticFromDataA(LPDATAOBJECTA pSrcDataObjA, REFIID iid,
           DWORD renderopt, LPFORMATETC pFormatEtc,
           LPOLECLIENTSITEA pClientSiteA, LPSTORAGEA pStgA, LPVOID * ppvObj)
{
     TraceSTDAPIEnter("OleCreateStaticFromDataA");
     LPDATAOBJECT pSrcDataObj;
     LPOLECLIENTSITE pClientSite;
     LPSTORAGE pStg;
     LPUNKNOWN pUnk;
     IDINTERFACE      idRef;
     HRESULT hResult;


     hResult = WrapIDataObjectWFromA(pSrcDataObjA, &pSrcDataObj);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapIOleClientSiteWFromA(pClientSiteA, &pClientSite);
     if (FAILED(hResult))
           goto Error;

     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
           goto Error1;

     hResult = OleCreateStaticFromData(pSrcDataObj, iid, renderopt,
pFormatEtc,
                 pClientSite, pStg, (LPVOID *)&pUnk);
     if (FAILED(hResult))
           goto Error1;

     //
     //  Convert the 16 bytes GUID into an internal integer for speed.
     //
     idRef = WrapTranslateIID(iid);

     hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)ppvObj);

     if (pStg)
           pStg->Release();

     if (pUnk)
           pUnk->Release();
```

```
Error1:
     if (pClientSite)
           pClientSite->Release();

Error:
     if (pSrcDataObj)
           pSrcDataObj->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleCreateLinkA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI OleCreateLinkA(LPMONIKERA pmkLinkSrcA, REFIID riid, DWORD renderopt,
           LPFORMATETC lpFormatEtc, LPOLECLIENTSITEA pClientSiteA,
           LPSTORAGEA pStgA, LPVOID * ppvObj)
{
     TraceSTDAPIEnter("OleCreateLinkA");
     LPMONIKER pmkLinkSrc;
     LPOLECLIENTSITE pClientSite;
     LPSTORAGE pStg;
     LPUNKNOWN pUnk;
     IDINTERFACE      idRef;
     HRESULT hResult;


     hResult = WrapIMonikerWFromA(pmkLinkSrcA, &pmkLinkSrc);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapIOleClientSiteWFromA(pClientSiteA, &pClientSite);
     if (FAILED(hResult))
           goto Error;

     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
           goto Error1;

     hResult = OleCreateLink(pmkLinkSrc, riid, renderopt, lpFormatEtc,
                 pClientSite, pStg, (LPVOID *)&pUnk);
     if (FAILED(hResult))
           goto Error1;

     //
     //  Convert the 16 bytes GUID into an internal integer for speed.
     //
     idRef = WrapTranslateIID(riid);

     hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)ppvObj);


     if (pStg)
           pStg->Release();

     if (pUnk)
           pUnk->Release();

Error1:
     if (pClientSite)
           pClientSite->Release();

Error:
     if (pmkLinkSrc)
           pmkLinkSrc->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleCreateLinkToFileA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI OleCreateLinkToFileA(LPCSTR lpszFileNameA, REFIID riid,
           DWORD renderopt, LPFORMATETC lpFormatEtc,
           LPOLECLIENTSITEA pClientSiteA, LPSTORAGEA pStgA, LPVOID * ppvObj)
{
     TraceSTDAPIEnter("OleCreateLinkToFileA");
     OLECHAR szFileName[MAX_STRING], * pszFileName;
     LPOLECLIENTSITE pClientSite;
     LPSTORAGE pStg;
     LPUNKNOWN pUnk;
     IDINTERFACE      idRef;
     HRESULT hResult;


     if (lpszFileNameA)
     {
           ConvertStringToW(lpszFileNameA, szFileName);
           pszFileName = szFileName;
     }
     else
```

```
           pszFileName = NULL;

     hResult = WrapIOleClientSiteWFromA(pClientSiteA, &pClientSite);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
           goto Error;

     hResult = OleCreateLinkToFile(pszFileName, riid, renderopt,
lpFormatEtc,
                 pClientSite, pStg, (LPVOID *)&pUnk);
     if (FAILED(hResult))
           goto Error;

     //
     //  Convert the 16 bytes GUID into an internal integer for speed.
     //
     idRef = WrapTranslateIID(riid);

     hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)ppvObj);

     if (pStg)
           pStg->Release();

     if (pUnk)
           pUnk->Release();

Error:
     if (pClientSite)
           pClientSite->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleCreateFromFileA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI OleCreateFromFileA(REFCLSID rclsid, LPCSTR lpszFileNameA, REFIID
riid,
           DWORD renderopt, LPFORMATETC lpFormatEtc,
           LPOLECLIENTSITEA pClientSiteA, LPSTORAGEA pStgA, LPVOID * ppvObj)
{
     TraceSTDAPIEnter("OleCreateFromFileA");
     OLECHAR szFileName[MAX_PATH], *pszFileName;
     LPOLECLIENTSITE pClientSite;
     LPSTORAGE pStg;
     LPUNKNOWN pUnk;
     IDINTERFACE      idRef;
     HRESULT hResult;


     if (lpszFileNameA != NULL)
     {
           pszFileName = szFileName;
           ConvertStringToW(lpszFileNameA, pszFileName);
     }
     else
           pszFileName = NULL;

     hResult = WrapIOleClientSiteWFromA(pClientSiteA, &pClientSite);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
           goto Error;

     hResult = OleCreateFromFile(rclsid, pszFileName, riid, renderopt,
                 lpFormatEtc, pClientSite, pStg, (LPVOID *)&pUnk);
     if (FAILED(hResult))
           goto Error;

     //
     //  Convert the 16 bytes GUID into an internal integer for speed.
     //
     idRef = WrapTranslateIID(riid);

     hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)ppvObj);


     if (pStg)
           pStg->Release();

     if (pUnk)
           pUnk->Release();

Error:
     if (pClientSite)
           pClientSite->Release();

     TraceSTDAPIExit("OleCreateFromFileA", hResult);

     return hResult;
}


//
+--------------------------------------------------------------------------
//
```

```
//  Routine:    OleLoadA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI OleLoadA(LPSTORAGEA pStgA, REFIID riid, LPOLECLIENTSITEA
pClientSiteA,
           LPVOID * ppvObj)
{
     TraceSTDAPIEnter("OleLoadA");
     LPSTORAGE pStg;
     LPOLECLIENTSITE pClientSite;
     LPUNKNOWN pUnk;
     IDINTERFACE      idRef;
     HRESULT hResult;


     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIOleClientSiteWFromA(pClientSiteA, &pClientSite);
     if (FAILED(hResult))
           goto Error;

     hResult = OleLoad(pStg, riid, pClientSite, (LPVOID *)&pUnk);
     if (FAILED(hResult))
           goto Error;

     //
     //  Convert the 16 bytes GUID into an internal integer for speed.
     //
     idRef = WrapTranslateIID(riid);

     hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)ppvObj);

     if (pUnk)
           pUnk->Release();

     if (pClientSite)
           pClientSite->Release();

Error:
     if (pStg)
           pStg->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
```

```
//
//  Routine:    OleSaveA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI OleSaveA(LPPERSISTSTORAGEA pPSA, LPSTORAGEA pStgA, BOOL fSameAsLoad)
{
     TraceSTDAPIEnter("OleSaveA");
     LPPERSISTSTORAGE pPS;
     LPSTORAGE pStg;
     HRESULT hResult;


     hResult = WrapIPersistStorageWFromA(pPSA, &pPS);
     if (FAILED(hResult))
            return hResult;

     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
            goto Error;

     hResult = OleSave(pPS, pStg, fSameAsLoad);

     if (pStg)
            pStg->Release();

Error:
     if (pPS)
            pPS->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleLoadFromStreamA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI OleLoadFromStreamA(LPSTREAMA pStmA, REFIID iidInterface,
            LPVOID * ppvObj)
{
     TraceSTDAPIEnter("OleLoadFromStreamA");
     LPSTREAM pStm;
     LPUNKNOWN pUnk;
     IDINTERFACE      idRef;
     HRESULT hResult;


     hResult = WrapIStreamWFromA(pStmA, &pStm);
     if (FAILED(hResult))
           return hResult;

     hResult = OleLoadFromStream(pStm, iidInterface, (LPVOID *)&pUnk);
     if (FAILED(hResult))

           return hResult;

     //
     //  Convert the 16 bytes GUID into an internal integer for speed.
     //
     idRef = WrapTranslateIID(iidInterface);

     hResult = WrapInterfaceAFromW(idRef, pUnk, (LPUNKNOWN *)ppvObj);

     if (pStm)
           pStm->Release();

     if (pUnk)
           pUnk->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleSaveToStreamA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI OleSaveToStreamA(LPPERSISTSTREAMA pPStmA, LPSTREAMA pStmA)
{
     TraceSTDAPIEnter("OleSaveToStreamA");
     LPPERSISTSTREAM pPStm;
     LPSTREAM pStm;
     HRESULT hResult;


     hResult = WrapIPersistStreamWFromA(pPStmA, &pPStm);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIStreamWFromA(pStmA, &pStm);
     if (FAILED(hResult))
```

```
            goto Error;

      hResult = OleSaveToStream(pPStm, pStm);

      if (pStm)
            pStm->Release();

Error:
      if (pPStm)
            pPStm->Release();

      return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    RegisterDragDropA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI RegisterDragDropA(HWND hwnd, LPDROPTARGETA pDropTargetA)
{
      TraceSTDAPIEnter("RegisterDragDropA");
      LPDROPTARGET pDropTarget;
      HRESULT hResult;


      hResult = WrapIDropTargetWFromA(pDropTargetA, &pDropTarget);
      if (FAILED(hResult))
            return hResult;

      hResult = RegisterDragDrop(hwnd, pDropTarget);
      if (FAILED(hResult))
      {
            if (pDropTarget)
                  pDropTarget->Release();
            return hResult;
      }

      if (!SetProp(hwnd, "Ole2AnsiProp", (HANDLE)pDropTarget))
      {
            RevokeDragDrop(hwnd);
            if (pDropTarget)
                  pDropTarget->Release();
            return ResultFromScode(E_FAIL);
      }

      return hResult;
}
```

```
//
+---------------------------------------------------------------------------
//
//  Routine:    DoDragDropA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI DoDragDropA(LPDATAOBJECTA pDataObjA, LPDROPSOURCE pDropSource,
           DWORD dwOKEffects, LPDWORD pdwEffect)
{
     TraceSTDAPIEnter("DoDragDropA");
     LPDATAOBJECT pDataObj;
     HRESULT hResult;


     hResult = WrapIDataObjectWFromA(pDataObjA, &pDataObj);
     if (FAILED(hResult))
           return hResult;

     hResult = DoDragDrop(pDataObj, pDropSource, dwOKEffects, pdwEffect);

     if (pDataObj)
           pDataObj->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    RevokeDragDropA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI RevokeDragDropA(HWND hwnd)
{
     TraceSTDAPIEnter("RevokeDragDropA");
     LPDROPTARGET pDropTarget;
     HRESULT hResult;


     hResult = RevokeDragDrop(hwnd);
     if (FAILED(hResult))
```

```
           return hResult;

     pDropTarget = (LPDROPTARGET)GetProp(hwnd, "Ole2AnsiProp");
     if (pDropTarget)
           pDropTarget->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleSetClipboardA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI OleSetClipboardA(LPDATAOBJECTA pDataObjA)
{
     TraceSTDAPIEnter("OleSetClipboardA");
     LPDATAOBJECT pDataObj;
     HRESULT hResult;


     hResult = WrapIDataObjectWFromA(pDataObjA, &pDataObj);
     if (FAILED(hResult))
           return hResult;

     hResult = OleSetClipboard(pDataObj);

     if (pDataObj)
           pDataObj->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleGetClipboardA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI OleGetClipboardA(LPDATAOBJECTA * ppDataObjA)
{
     TraceSTDAPIEnter("OleGetClipboardA");
     LPDATAOBJECT pDataObj;
     HRESULT hReturn;
     HRESULT hResult;


     *ppDataObjA = NULL;

     hReturn = OleGetClipboard(&pDataObj);
     if (FAILED(hReturn))
          return hReturn;

     hResult = WrapIDataObjectAFromW(pDataObj, ppDataObjA);
     if (FAILED(hResult))
          hReturn = hResult;

     if (pDataObj)
          pDataObj->Release();

     return hReturn;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleIsCurrentClipboardA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI OleIsCurrentClipboardA(LPDATAOBJECTA pDataObjA)
{
     TraceSTDAPIEnter("OleIsCurrentClipboardA");
     LPDATAOBJECT pDataObj;
     HRESULT hResult;


     hResult = WrapIDataObjectWFromA(pDataObjA, &pDataObj);
     if (FAILED(hResult))
          return hResult;

     hResult = OleIsCurrentClipboard(pDataObj);

     if (pDataObj)
          pDataObj->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
```

```
//
//  Routine:    OleSetMenuDescriptorA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI OleSetMenuDescriptorA(HOLEMENU holemenu, HWND hwndFrame,
           HWND hwndActiveObject, LPOLEINPLACEFRAMEA lpFrameA,
     LPOLEINPLACEACTIVEOBJECTA lpActiveObjectA)
{
     TraceSTDAPIEnter("OleSetMenuDescriptorA");
     LPOLEINPLACEFRAME lpFrame;
     LPOLEINPLACEACTIVEOBJECT lpActiveObject;
     HRESULT hResult;



     hResult = WrapIOleInPlaceFrameWFromA(lpFrameA, &lpFrame);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIOleInPlaceActiveObjectWFromA(lpActiveObjectA,
&lpActiveObject);
     if (FAILED(hResult))
     goto Error;

     hResult = OleSetMenuDescriptor(holemenu, hwndFrame, hwndActiveObject,
           lpFrame, lpActiveObject);

     if (lpActiveObject)
           lpActiveObject->Release();

Error:
     if (lpFrame)
           lpFrame->Release();

     return hResult;
}


//
+----------------------------------------------------------------------------
//
//  Routine:    OleTranslateAcceleratorA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI OleTranslateAcceleratorA(LPOLEINPLACEFRAMEA lpFrameA,
           LPOLEINPLACEFRAMEINFO lpFrameInfo, LPMSG lpmsg)
{
     TraceSTDAPIEnter("OleTranslateAcceleratorA");
     LPOLEINPLACEFRAME lpFrame;
     HRESULT hResult;


     hResult = WrapIOleInPlaceFrameWFromA(lpFrameA, &lpFrame);
     if (FAILED(hResult))
           return hResult;

     hResult = OleTranslateAccelerator(lpFrame, lpFrameInfo, lpmsg);

     if (lpFrame)
           lpFrame->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleIsRunningA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
STDAPI_(BOOL) OleIsRunningA(LPOLEOBJECTA pObjectA)
{
     TraceSTDAPIEnter("OleIsRunningA");

     LPOLEOBJECT pObject;
     HRESULT hResult;
   BOOL    fResult;


     hResult = WrapIOleObjectWFromA(pObjectA, &pObject);
     if (FAILED(hResult))
           return FALSE;

     fResult = OleIsRunning(pObject);

     if (pObject)
           pObject->Release();

     return fResult;
}


//
+---------------------------------------------------------------------------
```

```
//
//  Routine:    ReleaseStgMediumA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
STDAPI_(void) ReleaseStgMediumA(LPSTGMEDIUMA pStgMediumA)
{
     TraceSTDAPIEnter("ReleaseStgMediumA");

     ConvertSTGMEDIUMFree(0, pStgMediumA);
}


//
+---------------------------------------------------------------------------
//
//  Routine:    CreateOleAdviseHolderA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
STDAPI CreateOleAdviseHolderA(LPOLEADVISEHOLDERA * ppOAHolderA)
{
     TraceSTDAPIEnter("CreateOleAdviseHolderA");
     LPOLEADVISEHOLDER pOAHolder;
     HRESULT hResult;


     *ppOAHolderA = NULL;

     hResult = CreateOleAdviseHolder(&pOAHolder);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIOleAdviseHolderAFromW(pOAHolder, ppOAHolderA);

     if (pOAHolder)
           pOAHolder->Release();


     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleGetIconOfFileA
```

```
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI_(HGLOBAL) OleGetIconOfFileA(LPSTR lpszPathA, BOOL fUseFileAsLabel)
{
     TraceSTDAPIEnter("OleGetIconOfFileA");

     OLECHAR szPath[MAX_PATH];
     OLECHAR *pPath;


     if (lpszPathA)
     {
           ConvertStringToW(lpszPathA, szPath);
           pPath = szPath;
     }
     else
           pPath = NULL;

     return OleGetIconOfFile(pPath, fUseFileAsLabel);
}


//
+--------------------------------------------------------------------------
//
//  Routine:    OleGetIconOfClassA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI_(HGLOBAL) OleGetIconOfClassA(REFCLSID rclsid, LPSTR lpszLabelA,
           BOOL fUseTypeAsLabel)
{
     TraceSTDAPIEnter("OleGetIconOfClassA");

     OLECHAR szLabel[MAX_PATH];
     OLECHAR *pLabel;

     if (lpszLabelA != NULL)
     {
           ConvertStringToW(lpszLabelA, szLabel);
           pLabel = szLabel;
     }
     else
           pLabel = NULL;

     return OleGetIconOfClass(rclsid, pLabel, fUseTypeAsLabel);
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleMetafilePictFromIconAndLabelA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI_(HGLOBAL) OleMetafilePictFromIconAndLabelA(HICON hIcon,
          LPSTR lpszLabelA, LPSTR lpszSourceFileA, UINT iIconIndex)
{
     TraceSTDAPIEnter("OleMetafilePictFromIconAndLabelA");

     LPOLESTR lpszLabel;
     LPOLESTR lpszSourceFile;
     HGLOBAL hGlobal;
     HRESULT hResult;


     hGlobal = NULL;

     hResult = ConvertStringToW(lpszLabelA, &lpszLabel);
     if (FAILED(hResult))
          return NULL;

     hResult = ConvertStringToW(lpszSourceFileA, &lpszSourceFile);
     if (FAILED(hResult))
          goto Error;

     hGlobal = OleMetafilePictFromIconAndLabel(hIcon, lpszLabel,
                lpszSourceFile, iIconIndex);

     ConvertStringFree(lpszSourceFile);

Error:
     ConvertStringFree(lpszLabel);

     return hGlobal;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleRegGetUserTypeA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
```

```
//
//--------------------------------------------------------------------------
-
STDAPI OleRegGetUserTypeA(REFCLSID clsid, DWORD dwFormOfType,
           LPSTR * pszUserTypeA)
{
     TraceSTDAPIEnter("OleRegGetUserTypeA");
     LPOLESTR pszUserType;
   HRESULT hResult;


     hResult = OleRegGetUserType(clsid, dwFormOfType, &pszUserType);
     if (FAILED(hResult))
           return hResult;

     hResult = ConvertStringToA(pszUserType, pszUserTypeA);

     ConvertStringFree(pszUserType);


     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleRegEnumFormatEtcA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
STDAPI OleRegEnumFormatEtcA(REFCLSID clsid, DWORD dwDirection,
           LPENUMFORMATETCA * ppenumA)
{
     TraceSTDAPIEnter("OleRegEnumFormatEtcA");
     LPENUMFORMATETC penum;
     HRESULT hResult;


     *ppenumA = NULL;

     hResult = OleRegEnumFormatEtc(clsid, dwDirection, &penum);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIEnumFORMATETCAFromW(penum, ppenumA);

     if (penum)
           penum->Release();

     return hResult;
```

```
}


//
+--------------------------------------------------------------------------
//
//  Routine:    OleRegEnumVerbsA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
STDAPI OleRegEnumVerbsA(REFCLSID clsid, LPENUMOLEVERBA * ppenumA)
{
     TraceSTDAPIEnter("OleRegEnumVerbsA");
     LPENUMOLEVERB penum;
     HRESULT hResult;


     *ppenumA = NULL;

     hResult = OleRegEnumVerbs(clsid, &penum);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIEnumOLEVERBAFromW(penum, ppenumA);

     if (penum)
           penum->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Routine:    GetHGlobalFromILockBytesA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//--------------------------------------------------------------------------
-
STDAPI GetHGlobalFromILockBytesA(LPLOCKBYTESA plkbytA, HGLOBAL * phglobal)
{
     TraceSTDAPIEnter("GetHGlobalFromILockBytesA");
     LPLOCKBYTES plkbyt;
     HRESULT hResult;


     hResult = WrapILockBytesWFromA(plkbytA, &plkbyt);
     if (FAILED(hResult))
          return hResult;

     hResult = GetHGlobalFromILockBytes(plkbyt, phglobal);

     if (plkbyt)
          plkbyt->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    CreateILockBytesOnHGlobalA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI CreateILockBytesOnHGlobalA(HGLOBAL hGlobal, BOOL fDeleteOnRelease,
           LPLOCKBYTESA * pplkbytA)
{
     TraceSTDAPIEnter("CreateILockBytesOnHGlobalA");
     LPLOCKBYTES plkbyt;
     HRESULT hResult;


     *pplkbytA = NULL;

     hResult = CreateILockBytesOnHGlobal(hGlobal, fDeleteOnRelease,
&plkbyt);
     if (FAILED(hResult))
          return hResult;

     hResult = WrapILockBytesAFromW(plkbyt, pplkbytA);

     if (plkbyt)
          plkbyt->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    GetHGlobalFromStreamA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
```

```
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI GetHGlobalFromStreamA(LPSTREAMA pStmA, HGLOBAL * phglobal)
{
     TraceSTDAPIEnter("GetHGlobalFromStreamA");
     LPSTREAM pStm;
     HRESULT hResult;


     hResult = WrapIStreamWFromA(pStmA, &pStm);
     if (FAILED(hResult))
           return hResult;

     hResult = GetHGlobalFromStream(pStm, phglobal);

     if (pStm)
           pStm->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    CreateStreamOnHGlobalA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI CreateStreamOnHGlobalA(HGLOBAL hGlobal, BOOL fDeleteOnRelease,
           LPSTREAMA * ppstmA)
{
     TraceSTDAPIEnter("CreateStreamOnHGlobalA");
     LPSTREAM pstm;
     HRESULT hResult;


     *ppstmA = NULL;

     hResult = CreateStreamOnHGlobal(hGlobal, fDeleteOnRelease, &pstm);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIStreamAFromW(pstm, ppstmA);

     if (pstm)
           pstm->Release();

     return hResult;
```

```
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleDoAutoConvertA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI OleDoAutoConvertA(LPSTORAGEA pStgA, LPCLSID pClsidNew)
{
     TraceSTDAPIEnter("OleDoAutoConvertA");
     LPSTORAGE pStg;
     HRESULT hResult;


     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
           return hResult;

     hResult = OleDoAutoConvert(pStg, pClsidNew);

     if (pStg)
           pStg->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    GetConvertStgA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI GetConvertStgA(LPSTORAGEA pStgA)
{
     TraceSTDAPIEnter("GetConvertStgA");
     LPSTORAGE pStg;
     HRESULT hResult;


     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
           return hResult;
     hResult = GetConvertStg(pStg);

     if (pStg)
          pStg->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    SetConvertStgA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI SetConvertStgA(LPSTORAGEA pStgA, BOOL fConvert)
{
     TraceSTDAPIEnter("SetConvertStgA");
     LPSTORAGE pStg;
     HRESULT hResult;


     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
          return hResult;

     hResult = SetConvertStg(pStg, fConvert);

     if (pStg)
          pStg->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleConvertOLESTREAMToIStorageA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI OleConvertOLESTREAMToIStorageA(LPOLESTREAM lpolestream,
           LPSTORAGEA pStgA, const DVTARGETDEVICE * ptd)
```

```
{
     TraceSTDAPIEnter("OleConvertOLESTREAMToIStorageA");
     LPSTORAGE pStg;
     HRESULT hResult;


     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
           return hResult;

     hResult = OleConvertOLESTREAMToIStorage(lpolestream, pStg, ptd);

     if (pStg)
           pStg->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    OleConvertIStorageToOLESTREAMA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI OleConvertIStorageToOLESTREAMA(LPSTORAGEA pStgA,
           LPOLESTREAM lpolestream)
{
     TraceSTDAPIEnter("OleConvertIStorageToOLESTREAMA");
     LPSTORAGE pStg;
     HRESULT hResult;


     hResult = WrapIStorageWFromA(pStgA, &pStg);
     if (FAILED(hResult))
           return hResult;

     hResult = OleConvertIStorageToOLESTREAM(pStg, lpolestream);

     if (pStg)
           pStg->Release();

     return hResult;
}
```

ANSISTOR.H (OLEANSI Sample)

```
//
+--------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       ansistor.h
//
//  Contents:   ANSI Wrappers for Unicode Storage Interfaces and APIs.
//
//  Classes:    CEnumSTATSTGA - ANSI wrapper object for IEnumSTATSTG.
//              CLockBytesA   - ANSI wrapper object for ILockBytes.
//              CStreamA      - ANSI wrapper object for IStream.
//              CStorageA     - ANSI wrapper object for IStorage.
//              CRootStorageA - ANSI wrapper object for IRootStorage.
//
//  Functions:  StgCreateDocfileA
//              StgCreateDocfileOnILockBytesA
//              StgOpenStorageA
//              StgOpenStorageOnILockBytesA
//              StgIsStorageFileA
//              StgIsStorageILockBytesA
//              StgSetTimesA
//
//  History:    01-Nov-93   v-kentc     Created.
//
//--------------------------------------------------------------------------
-


//
//  Forward declarations
//
class CEnumSTATSTGA;
class CLockBytesA;
class CStreamA;
class CStorageA;
class CRootStorageA;


typedef LPSTR * SNBA;

typedef struct tagSTATSTGA
{
     LPSTR pwcsName;
     DWORD type;
     ULARGE_INTEGER cbSize;
     FILETIME mtime;
     FILETIME ctime;
     FILETIME atime;
     DWORD grfMode;
     DWORD grfLocksSupported;
     CLSID clsid;
     DWORD grfStateBits;
     DWORD reserved;
} STATSTGA;


/*---------------------------------------------------------------------*/
/*                            IEnumSTATSTGA                            */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IEnumSTATSTGA

DECLARE_INTERFACE_(IEnumSTATSTGA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IENUMSTATSTGA methods ***
     STDMETHOD(Next) (THIS_ ULONG celt, STATSTGA * rgelt, ULONG
*pceltFetched) PURE;
     STDMETHOD(Skip) (THIS_ ULONG celt) PURE;
     STDMETHOD(Reset) (THIS) PURE;
     STDMETHOD(Clone) (THIS_ IEnumSTATSTGA **ppenm) PURE;
};

typedef IEnumSTATSTGA * LPENUMSTATSTGA;


/*---------------------------------------------------------------------*/
/*                             ILockBytesA                             */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   ILockBytesA

DECLARE_INTERFACE_(ILockBytesA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** ILockBytes methods ***
     STDMETHOD(ReadAt) (THIS_ ULARGE_INTEGER ulOffset,
                  VOID HUGEP *pv,
                  ULONG cb,
                  ULONG *pcbRead) PURE;
     STDMETHOD(WriteAt) (THIS_ ULARGE_INTEGER ulOffset,
                   VOID const HUGEP *pv,
                   ULONG cb,
                   ULONG *pcbWritten) PURE;
     STDMETHOD(Flush) (THIS) PURE;
     STDMETHOD(SetSize) (THIS_ ULARGE_INTEGER cb) PURE;
     STDMETHOD(LockRegion) (THIS_ ULARGE_INTEGER libOffset,
                        ULARGE_INTEGER cb,
                        DWORD dwLockType) PURE;
     STDMETHOD(UnlockRegion) (THIS_ ULARGE_INTEGER libOffset,
                          ULARGE_INTEGER cb,
                        DWORD dwLockType) PURE;
     STDMETHOD(Stat) (THIS_ STATSTGA *pstatstg, DWORD grfStatFlag) PURE;
};

typedef ILockBytesA * LPLOCKBYTESA;


/*----------------------------------------------------------------------*/
/*                              IStreamA                                */
/*----------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IStreamA

DECLARE_INTERFACE_(IStreamA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IStreamA methods ***
     STDMETHOD(Read) (THIS_ VOID HUGEP *pv,
                  ULONG cb, ULONG *pcbRead) PURE;
     STDMETHOD(Write) (THIS_ VOID const HUGEP *pv,
                   ULONG cb,
                   ULONG *pcbWritten) PURE;
     STDMETHOD(Seek) (THIS_ LARGE_INTEGER dlibMove,
                    DWORD dwOrigin,
                    ULARGE_INTEGER *plibNewPosition) PURE;
     STDMETHOD(SetSize) (THIS_ ULARGE_INTEGER libNewSize) PURE;
     STDMETHOD(CopyTo) (THIS_ IStreamA *pstm,
                  ULARGE_INTEGER cb,
                  ULARGE_INTEGER *pcbRead,
                  ULARGE_INTEGER *pcbWritten) PURE;
     STDMETHOD(Commit) (THIS_ DWORD grfCommitFlags) PURE;
     STDMETHOD(Revert) (THIS) PURE;
     STDMETHOD(LockRegion) (THIS_ ULARGE_INTEGER libOffset,
                        ULARGE_INTEGER cb,
                        DWORD dwLockType) PURE;
     STDMETHOD(UnlockRegion) (THIS_ ULARGE_INTEGER libOffset,
                        ULARGE_INTEGER cb,
                        DWORD dwLockType) PURE;
     STDMETHOD(Stat) (THIS_ STATSTGA *pstatstg, DWORD grfStatFlag) PURE;
     STDMETHOD(Clone)(THIS_ IStreamA * *ppstm) PURE;
};

typedef IStreamA * LPSTREAMA;
```

```
/*---------------------------------------------------------------------*/
/*                              IStorageA                              */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IStorageA

DECLARE_INTERFACE_(IStorageA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IStorage methods ***
     STDMETHOD(CreateStream) (THIS_ LPCSTR pwcsName,
                          DWORD grfMode,
                          DWORD reserved1,
                          DWORD reserved2,
                          IStreamA **ppstm) PURE;
     STDMETHOD(OpenStream) (THIS_ LPCSTR pwcsName,
            void *reserved1,
                        DWORD grfMode,
                        DWORD reserved2,
                        IStreamA **ppstm) PURE;
     STDMETHOD(CreateStorage) (THIS_ LPCSTR pwcsName,
                       DWORD grfMode,
                       DWORD reserved1,
                       DWORD reserved2,
                       IStorageA **ppstg) PURE;
     STDMETHOD(OpenStorage) (THIS_ LPCSTR pwcsName,
                         IStorageA *pstgPriority,
                         DWORD grfMode,
                         SNBA snbExclude,
                         DWORD reserved,
                         IStorageA **ppstg) PURE;
     STDMETHOD(CopyTo) (THIS_ DWORD ciidExclude,
                    IID const *rgiidExclude,
                    SNBA snbExclude,
                    IStorageA *pstgDest) PURE;
     STDMETHOD(MoveElementTo) (THIS_ LPCSTR lpszName,
                               IStorageA *pstgDest,
                                           LPCSTR lpszNewName,
                                           DWORD grfFlags) PURE;
     STDMETHOD(Commit) (THIS_ DWORD grfCommitFlags) PURE;
     STDMETHOD(Revert) (THIS) PURE;
     STDMETHOD(EnumElements) (THIS_ DWORD reserved1,
                        void *reserved2,
                        DWORD reserved3,
                        IEnumSTATSTGA **ppenm) PURE;
     STDMETHOD(DestroyElement) (THIS_ LPCSTR pwcsName) PURE;
     STDMETHOD(RenameElement) (THIS_ LPCSTR pcsOldName,
                       LPCSTR pcsNewName) PURE;
     STDMETHOD(SetElementTimes) (THIS_ LPCSTR lpszName,
                                   FILETIME const *pctime,
                                               FILETIME const *patime,
                                               FILETIME const *pmtime) PURE;
     STDMETHOD(SetClass) (THIS_ REFCLSID clsid) PURE;
     STDMETHOD(SetStateBits) (THIS_ DWORD grfStateBits, DWORD grfMask) PURE;
     STDMETHOD(Stat) (THIS_ STATSTGA *pstatstgA, DWORD grfStatFlag) PURE;
};

typedef IStorageA * LPSTORAGEA;


/*---------------------------------------------------------------------*/
/*                            IRootStorageA                            */
/*---------------------------------------------------------------------*/

#undef  INTERFACE
#define INTERFACE   IRootStorageA

DECLARE_INTERFACE_(IRootStorageA, IUnknown)
{
     // *** IUnknown methods ***
     STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
     STDMETHOD_(ULONG,AddRef) (THIS)  PURE;
     STDMETHOD_(ULONG,Release) (THIS) PURE;

     // *** IRootStorage methods ***
     STDMETHOD(SwitchToFile) (THIS_ LPSTR lpstrFile) PURE;
};

typedef IRootStorageA * LPROOTSTORAGEA;


/*---------------------------------------------------------------------*/
/*                         ANSI Storage API's                          */
/*---------------------------------------------------------------------*/

STDAPI StgCreateDocfileA(LPCSTR pwcsName,
                 DWORD grfMode,
                 DWORD reserved,
                 IStorageA * *ppstgOpenA);
STDAPI StgCreateDocfileOnILockBytesA(ILockBytesA *plkbyt,
                             DWORD grfMode,
                             DWORD reserved,
                             IStorageA * *ppstgOpen);

STDAPI StgOpenStorageA(LPCSTR pwcsName,
                   IStorageA *pstgPriority,
                   DWORD grfMode,
                   SNBA snbExclude,
                   DWORD reserved,
                   IStorageA * *ppstgOpenA);
STDAPI StgOpenStorageOnILockBytesA(ILockBytesA *plkbyt,
                         IStorageA *pstgPriority,
                         DWORD grfMode,
                         SNBA snbExclude,
                         DWORD reserved,
                         IStorageA * *ppstgOpen);
STDAPI StgIsStorageFileA(LPCSTR pwcsName);
STDAPI StgIsStorageILockBytesA(ILockBytesA * plkbyt);
STDAPI StgSetTimesA(LPCSTR lpszName,
             FILETIME const * pctime,
                         FILETIME const * patime,
                         FILETIME const * pmtime);


//
+--------------------------------------------------------------------------
//
//  Class:      CEnumSTATSTGA
//
//  Synopsis:   Class definition of IEnumSTATSTGA
//
//---------------------------------------------------------------------------
-
class CEnumSTATSTGA : CAnsiInterface
{
public:
     // *** IEnumSTATSTGA methods ***
     STDMETHOD(Next) (ULONG celt,
                                STATSTGA * rgelt,
                                ULONG * pceltFetched);
     STDMETHOD(Skip) (ULONG celt);
     STDMETHOD(Reset) (VOID);
     STDMETHOD(Clone) (IEnumSTATSTGA * * ppenm);

     inline CEnumSTATSTGA(LPUNKNOWN pUnk, IEnumSTATSTG * pObj) :
                 CAnsiInterface(ID_IEnumSTATSTG, pUnk, (LPUNKNOWN)pObj) {};

     inline IEnumSTATSTG * GetWide() const
                 { return (IEnumSTATSTG *)m_pObj; };
};


//
+--------------------------------------------------------------------------
//
//  Class:      CLockBytesA
//
//  Synopsis:   Class definition of ILockBytesA
//
//---------------------------------------------------------------------------
-

class CLockBytesA : CAnsiInterface
{
public:
     // *** ILockBytes methods ***
     STDMETHOD(ReadAt) (ULARGE_INTEGER ulOffset,
                  VOID HUGEP *pv,
                  ULONG cb,
                  ULONG *pcbRead);
     STDMETHOD(WriteAt) (ULARGE_INTEGER ulOffset,
                   VOID const HUGEP *pv,
                   ULONG cb,
                   ULONG *pcbWritten);
     STDMETHOD(Flush) (VOID);
     STDMETHOD(SetSize) (ULARGE_INTEGER cb);
     STDMETHOD(LockRegion) (ULARGE_INTEGER libOffset,
                        ULARGE_INTEGER cb,
                        DWORD dwLockType);
     STDMETHOD(UnlockRegion) (ULARGE_INTEGER libOffset,
                          ULARGE_INTEGER cb,
                        DWORD dwLockType);
     STDMETHOD(Stat) (STATSTGA *pstatstg, DWORD grfStatFlag);

     inline CLockBytesA(LPUNKNOWN pUnk, ILockBytes * pObj) :
                 CAnsiInterface(ID_ILockBytes, pUnk, (LPUNKNOWN)pObj) {};

     inline ILockBytes * GetWide() const
                 { return (ILockBytes *)m_pObj; };
};


//
+--------------------------------------------------------------------------
//
//  Class:      CStreamA
//
//  Synopsis:   Class definition of IStreamA
//
//--------------------------------------------------------------------------
-

class CStreamA : public CAnsiInterface
{
public:
     // *** IStream methods ***
     STDMETHOD(Read) (VOID HUGEP *pv,
                  ULONG cb, ULONG *pcbRead);
     STDMETHOD(Write) (VOID const HUGEP *pv,
                 ULONG cb,
                 ULONG *pcbWritten);
     STDMETHOD(Seek) (LARGE_INTEGER dlibMove,
                    DWORD dwOrigin,
                    ULARGE_INTEGER *plibNewPosition);
     STDMETHOD(SetSize) (ULARGE_INTEGER libNewSize);
     STDMETHOD(CopyTo) (IStreamA *pstm,
                  ULARGE_INTEGER cb,
                  ULARGE_INTEGER *pcbRead,
                  ULARGE_INTEGER *pcbWritten);
     STDMETHOD(Commit) (DWORD grfCommitFlags);
     STDMETHOD(Revert) (VOID);
     STDMETHOD(LockRegion) (ULARGE_INTEGER libOffset,
                        ULARGE_INTEGER cb,
                        DWORD dwLockType);
     STDMETHOD(UnlockRegion) (ULARGE_INTEGER libOffset,
                        ULARGE_INTEGER cb,
                        DWORD dwLockType);
     STDMETHOD(Stat) (STATSTGA *pstatstg, DWORD grfStatFlag);
     STDMETHOD(Clone)(IStreamA * *ppstm);

     inline CStreamA(LPUNKNOWN pUnk, IStream * pObj) :
                CAnsiInterface(ID_IStream, pUnk, (LPUNKNOWN)pObj) {};

     inline IStream * GetWide() const
                { return (IStream *)m_pObj; };
};


//
+---------------------------------------------------------------------------
//
//  Class:      CStorageA
//
//  Synopsis:   Class definition of IStorageA
//
//----------------------------------------------------------------------------
-


class CStorageA : public CAnsiInterface
{
public:
     // *** IStorage methods ***
     STDMETHOD(CreateStream) (LPCSTR psName,
                          DWORD grfMode,
                          DWORD reserved1,
                          DWORD reserved2,
                          IStreamA **ppstm);
     STDMETHOD(OpenStream) (LPCSTR psName,
            void *reserved1,
                        DWORD grfMode,
                        DWORD reserved2,
                        IStreamA **ppstm);
     STDMETHOD(CreateStorage) (LPCSTR psName,
                       DWORD grfMode,
                       DWORD reserved1,
                       DWORD reserved2,
                       IStorageA **ppstg);
     STDMETHOD(OpenStorage) (LPCSTR psName,
                         IStorageA *pstgPriority,
                         DWORD grfMode,
                         SNBA snbExclude,
                         DWORD reserved,
                         IStorageA **ppstg);
     STDMETHOD(CopyTo) (DWORD ciidExclude,
                    IID const *rgiidExclude,
                    SNBA snbExclude,
                    IStorageA *pstgDest);
     STDMETHOD(MoveElementTo) (LPCSTR lpszName,
                               IStorageA *pstgDest,
                                           LPCSTR lpszNewName,
                                           DWORD grfFlags);
     STDMETHOD(Commit) (DWORD grfCommitFlags);
     STDMETHOD(Revert) (VOID);
     STDMETHOD(EnumElements) (DWORD reserved1,
                        void *reserved2,
                        DWORD reserved3,
                        IEnumSTATSTGA **ppenm);
     STDMETHOD(DestroyElement) (LPCSTR psName);
     STDMETHOD(RenameElement) (LPCSTR pcsOldName,
                       LPCSTR pcsNewName);
     STDMETHOD(SetElementTimes) (LPCSTR lpszName,
                                   FILETIME const *pctime,
                                               FILETIME const *patime,
                                               FILETIME const *pmtime);
     STDMETHOD(SetClass) (REFCLSID clsid);
     STDMETHOD(SetStateBits) (DWORD grfStateBits, DWORD grfMask);
     STDMETHOD(Stat) (STATSTGA *pstatstg, DWORD grfStatFlag);

     inline CStorageA(LPUNKNOWN pUnk, IStorage * pObj) :
                 CAnsiInterface(ID_IStorage, pUnk, (LPUNKNOWN)pObj) {};

     inline IStorage * GetWide() const
                 { return (IStorage *)m_pObj; };
};


//
+--------------------------------------------------------------------------
//
//  Class:      CRootStorageA
//
//  Synopsis:   Class definition of IRootStorageA
//
//--------------------------------------------------------------------------
-

class CRootStorageA : public CAnsiInterface
{
public:
     // *** IRootStorage methods ***
     STDMETHOD(SwitchToFile) (LPSTR lpstrFile);

     inline CRootStorageA(LPUNKNOWN pUnk, IRootStorage * pObj) :
                 CAnsiInterface(ID_IRootStorage, pUnk, (LPUNKNOWN)pObj) {};

     inline IRootStorage * GetWide() const
                 { return (IRootStorage *)m_pObj; };
};
```

ANSISTOR.CPP (OLEANSI Sample)

```
//
+---------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       ansistor.cpp
//
//  Contents:   ANSI Wrappers for Unicode Storage Interfaces and APIs.
//
//  Classes:    CEnumSTATSTGA - ANSI wrapper object for IEnumSTATSTG.
//              CLockBytesA   - ANSI wrapper object for ILockBytes.
//              CStreamA      - ANSI wrapper object for IStream.
//              CStorageA     - ANSI wrapper object for IStorage.
//              CRootStorageA - ANSI wrapper object for IRootStorage.
//
//  Functions:  StgCreateDocfileA
//              StgCreateDocfileOnILockBytesA
//              StgOpenStorageA
//              StgOpenStorageOnILockBytesA
//              StgIsStorageFileA
//              StgIsStorageILockBytesA
//              StgSetTimesA
//
//  History:    01-Nov-93   v-kentc     Created.
//
//----------------------------------------------------------------------------
-

#include "Ole2Ansi.h"
```

IEnumSTATSTGA Implementation   (OLEANSI Sample)

```
//
**********************************************************************
//
//                    IEnumSTATSTGA Implementation
//
//
**********************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     CEnumSTATSTGA::Next, public
//
//  Synopsis:   Thunks Next to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CEnumSTATSTGA::Next(
           ULONG celt,
           STATSTGA * rgelt,
           ULONG * pceltFetched)
{
     TraceMethodEnter("CEnumSTATSTGA::Next", this);

     ULONG   celtFetched;
     HRESULT hReturn;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IEnumSTATSTG, Next));

     if (pceltFetched == NULL)
           pceltFetched = &celtFetched;

     hReturn = GetWide()->Next(celt, (STATSTG *)rgelt, pceltFetched);
     if (FAILED(hReturn))
           return (hReturn);

     hResult = ConvertSTATSTGArrayToA(rgelt, *pceltFetched);
     if (FAILED(hResult))
           return (hResult);

     return hReturn;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CEnumSTATSTGA::Skip, public
```

```
//
//  Synopsis:   Thunks Skip to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CEnumSTATSTGA::Skip(ULONG celt)
{
     _DebugHook(GetWide(), MEMBER_PTR(IEnumSTATSTG, Skip));

     return GetWide()->Skip(celt);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CEnumSTATSTGA::Reset, public
//
//  Synopsis:   Thunks Reset to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CEnumSTATSTGA::Reset(VOID)
{
     _DebugHook(GetWide(), MEMBER_PTR(IEnumSTATSTG, Reset));

     return GetWide()->Reset();
}


//
+--------------------------------------------------------------------------
//
//  Member:     CEnumSTATSTGA::Clone, public
//
//  Synopsis:   Thunks Clone to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CEnumSTATSTGA::Clone(IEnumSTATSTGA * * ppstatA)
{
     TraceMethodEnter("CEnumSTATSTGA::Clone", this);

     IEnumSTATSTG * pstat;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IEnumSTATSTG, Clone));
```

```
    *ppstatA = NULL;

    hResult = GetWide()->Clone(&pstat);
    if (FAILED(hResult))
          return (hResult);

    hResult = WrapIEnumSTATSTGAFromW(pstat, ppstatA);

    if (pstat)
          pstat->Release();

    return hResult;
}
```

ILockBytesA Implementation (OLEANSI Sample)

```
//
***************************************************************************
//
//                   ILockBytesA Implementation
//
//
***************************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     CLockBytesA::ReadAt, public
//
//  Synopsis:   Thunks ReadAt to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CLockBytesA::ReadAt(
           ULARGE_INTEGER ulOffset,
           VOID HUGEP *pv,
           ULONG cb,
           ULONG *pcbRead)
{
     _DebugHook(GetWide(), MEMBER_PTR(ILockBytes, ReadAt));

     return GetWide()->ReadAt(ulOffset, pv, cb, pcbRead);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CLockBytesA::WriteAt, public
//
//  Synopsis:   Thunks WriteAt to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CLockBytesA::WriteAt(
           ULARGE_INTEGER ulOffset,
           VOID const HUGEP *pv,
           ULONG cb,
           ULONG *pcbWritten)
{
     _DebugHook(GetWide(), MEMBER_PTR(ILockBytes, WriteAt));

     return GetWide()->WriteAt(ulOffset, pv, cb, pcbWritten);
}
```

```
//
+---------------------------------------------------------------------------
//
//  Member:     CLockBytesA::Flush, public
//
//  Synopsis:   Thunks Flush to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CLockBytesA::Flush(VOID)
{
     _DebugHook(GetWide(), MEMBER_PTR(ILockBytes, Flush));

     return GetWide()->Flush();
}


//
+---------------------------------------------------------------------------
//
//  Member:     CLockBytesA::SetSize, public
//
//  Synopsis:   Thunks SetSize to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CLockBytesA::SetSize(ULARGE_INTEGER cb)
{
     _DebugHook(GetWide(), MEMBER_PTR(ILockBytes, SetSize));

     return GetWide()->SetSize(cb);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CLockBytesA::LockRegion, public
//
//  Synopsis:   Thunks LockRegion to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CLockBytesA::LockRegion(
           ULARGE_INTEGER libOffset,
           ULARGE_INTEGER cb,
           DWORD dwLockType)
```

```
{
     _DebugHook(GetWide(), MEMBER_PTR(ILockBytes, LockRegion));

     return GetWide()->LockRegion(libOffset, cb, dwLockType);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CLockBytesA::UnlockRegion, public
//
//  Synopsis:   Thunks UnlockRegion to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CLockBytesA::UnlockRegion(
           ULARGE_INTEGER libOffset,
           ULARGE_INTEGER cb,
           DWORD dwLockType)
{
     _DebugHook(GetWide(), MEMBER_PTR(ILockBytes, UnlockRegion));

     return GetWide()->UnlockRegion(libOffset, cb, dwLockType);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CLockBytesA::Stat, public
//
//  Synopsis:   Thunks Stat to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CLockBytesA::Stat(STATSTGA *pstatstg, DWORD grfStatFlag)
{
     TraceMethodEnter("CLockBytesA::Stat", this);

     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(ILockBytes, Stat));

     hResult = GetWide()->Stat((STATSTG *)pstatstg, grfStatFlag);
     if (FAILED(hResult))
           return (hResult);

     return ConvertSTATSTGToA(pstatstg);
}
```

IStreamA Implementation   (OLEANSI Sample)

```
//
***************************************************************************
//
//                   IStreamA Implementation
//
//
***************************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     CStreamA::Read, public
//
//  Synopsis:   Thunks Read to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CStreamA::Read(
           VOID HUGEP *pv,
           ULONG cb,
           ULONG *pcbRead)
{
     _DebugHook(GetWide(), MEMBER_PTR(IStream, Read));

     return GetWide()->Read(pv, cb, pcbRead);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CStreamA::Write, public
//
//  Synopsis:   Thunks Write to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CStreamA::Write(
           VOID const HUGEP *pv,
           ULONG cb,
           ULONG *pcbWritten)
{
     _DebugHook(GetWide(), MEMBER_PTR(IStream, Write));

     return GetWide()->Write(pv, cb, pcbWritten);
}
```

```
//
+---------------------------------------------------------------------------
//
//  Member:     CStreamA::Seek, public
//
//  Synopsis:   Thunks Seek to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CStreamA::Seek(
           LARGE_INTEGER dlibMove,
           DWORD dwOrigin,
           ULARGE_INTEGER *plibNewPosition)
{
     _DebugHook(GetWide(), MEMBER_PTR(IStream, Seek));

     return GetWide()->Seek(dlibMove, dwOrigin, plibNewPosition);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CStreamA::SetSize, public
//
//  Synopsis:   Thunks SetSize to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CStreamA::SetSize(ULARGE_INTEGER libNewSize)
{
     _DebugHook(GetWide(), MEMBER_PTR(IStream, SetSize));

     return GetWide()->SetSize(libNewSize);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CStreamA::CopyTo, public
//
//  Synopsis:   Thunks CopyTo to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CStreamA::CopyTo(
           IStreamA *pstrmA,
           ULARGE_INTEGER cb,
           ULARGE_INTEGER *pcbRead,
           ULARGE_INTEGER *pcbWritten)
{
     TraceMethodEnter("CStreamA::CopyTo", this);

     IStream * pstrm;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IStream, CopyTo));

     hResult = WrapIStreamWFromA(pstrmA, &pstrm);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->CopyTo(pstrm, cb, pcbRead, pcbWritten);

     if (pstrm)
           pstrm->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CStreamA::Commit, public
//
//  Synopsis:   Thunks Commit to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CStreamA::Commit(DWORD grfCommitFlags)
{
     _DebugHook(GetWide(), MEMBER_PTR(IStream, Commit));

     return GetWide()->Commit(grfCommitFlags);

}


//
+---------------------------------------------------------------------------
//
//  Member:     CStreamA::Revert, public
//
//  Synopsis:   Thunks Revert to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
```

```
STDMETHODIMP CStreamA::Revert(VOID)
{
     _DebugHook(GetWide(), MEMBER_PTR(IStream, Revert));

     return GetWide()->Revert();
}


//
+--------------------------------------------------------------------------
//
//  Member:     CStreamA::LockRegion, public
//
//  Synopsis:   Thunks LockRegion to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CStreamA::LockRegion(
           ULARGE_INTEGER libOffset,
           ULARGE_INTEGER cb,
           DWORD dwLockType)
{
     _DebugHook(GetWide(), MEMBER_PTR(IStream, LockRegion));

     return GetWide()->LockRegion(libOffset, cb, dwLockType);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CStreamA::UnlockRegion, public
//
//  Synopsis:   Thunks UnlockRegion to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CStreamA::UnlockRegion(
           ULARGE_INTEGER libOffset,
           ULARGE_INTEGER cb,
           DWORD dwLockType)
{
     _DebugHook(GetWide(), MEMBER_PTR(IStream, UnlockRegion));

     return GetWide()->UnlockRegion(libOffset, cb, dwLockType);
}


//
+--------------------------------------------------------------------------
//
```

```
//  Member:     CStreamA::Stat, public
//
//  Synopsis:   Thunks Stat to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//-----------------------------------------------------------------------------
-
STDMETHODIMP CStreamA::Stat(STATSTGA *pstatstg, DWORD grfStatFlag)
{
     TraceMethodEnter("CStreamA::Stat", this);

     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IStream, Stat));

     hResult = GetWide()->Stat((STATSTG *)pstatstg, grfStatFlag);
     if (FAILED(hResult))
           return (hResult);

     return ConvertSTATSTGToA(pstatstg);
}


//
+----------------------------------------------------------------------------
//
//  Member:     CStreamA::Clone, public
//
//  Synopsis:   Thunks Clone to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//-----------------------------------------------------------------------------
-
STDMETHODIMP CStreamA::Clone(IStreamA * * ppstrmA)
{
     TraceMethodEnter("CStreamA::Clone", this);

     IStream * pstrm;


     _DebugHook(GetWide(), MEMBER_PTR(IStream, Clone));

     *ppstrmA = NULL;

     HRESULT hResult = GetWide()->Clone(&pstrm);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapIStreamAFromW(pstrm, ppstrmA);

     if (pstrm)
           pstrm->Release();
```

```
        return hResult;
}
```

IStorageA Implementation (OLEANSI Sample)

```
//
**********************************************************************
//
//                    IStorageA Implementation
//
//
**********************************************************************

//
+---------------------------------------------------------------------
//
//  Member:     CStorageA::CreateStream, public
//
//  Synopsis:   Thunks CreateStream to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------
-
STDMETHODIMP CStorageA::CreateStream(
           LPCSTR pstrNameA,
           DWORD grfMode,
           DWORD reserved1,
           DWORD reserved2,
           IStreamA **ppstrmA)
{
     TraceMethodEnter("CStorageA::CreateStream", this);

     OLECHAR strName[MAX_PATH];
     IStream * pstrm;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IStorage, CreateStream));

     *ppstrmA = NULL;

     ConvertStringToW(pstrNameA, strName);

     hResult = GetWide()->CreateStream(strName, grfMode, reserved1,
reserved2, &pstrm);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapIStreamAFromW(pstrm, ppstrmA);

     if (pstrm)
           pstrm->Release();

     return hResult;
}
```

```
//
+--------------------------------------------------------------------------
//
//  Member:     CStorageA::OpenStream, public
//
//  Synopsis:   Thunks OpenStream to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CStorageA::OpenStream(
            LPCSTR pstrNameA,
            void *reserved1,
            DWORD grfMode,
            DWORD reserved2,
            IStreamA **ppstrmA)
{
     TraceMethodEnter("CStorageA::OpenStream", this);

     OLECHAR strName[MAX_PATH];
     LPSTREAM pstrm;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IStorage, OpenStream));

     *ppstrmA = NULL;

     ConvertStringToW(pstrNameA, strName);

     hResult = GetWide()->OpenStream(strName, reserved1, grfMode, reserved2,
&pstrm);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapIStreamAFromW(pstrm, ppstrmA);

     if (pstrm)
           pstrm->Release();

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CStorageA::CreateStorage, public
//
//  Synopsis:   Thunks CreateStorage to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
```

```
//--------------------------------------------------------------------------
-
STDMETHODIMP CStorageA::CreateStorage(
           LPCSTR pstrNameA,
           DWORD grfMode,
           DWORD reserved1,
           DWORD reserved2,
           IStorageA **ppstgA)
{
     TraceMethodEnter("CStorageA::CreateStorage", this);

     OLECHAR strName[MAX_PATH];
     IStorage * pstg;
     HRESULT hResult;
     HRESULT hReturn;


     _DebugHook(GetWide(), MEMBER_PTR(IStorage, CreateStorage));

     *ppstgA = NULL;

     ConvertStringToW(pstrNameA, strName);

     hReturn = GetWide()->CreateStorage(strName, grfMode, reserved1,
reserved2, &pstg);
     if (FAILED(hReturn))
           return (hReturn);

     hResult = WrapIStorageAFromW(pstg, ppstgA);
     if (SUCCEEDED(hResult))
           hResult = hReturn;

     if (pstg)
           pstg->Release();

     return hResult;
}



//
+-------------------------------------------------------------------------
//
//  Member:     CStorageA::OpenStorage, public
//
//  Synopsis:   Thunks OpenStorage to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CStorageA::OpenStorage(
           LPCSTR pstrNameA,
           IStorageA *pstgPriorityA,
           DWORD grfMode,
            SNBA snbExcludeA,
            DWORD reserved,
            IStorageA **ppstgA)
{
     TraceMethodEnter("CStorageA::OpenStorage", this);

     OLECHAR strName[MAX_PATH];
     SNB      snbExclude;
     IStorage * pstgPriority;
     IStorage * pstg;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IStorage, OpenStorage));

     *ppstgA = NULL;

     ConvertStringToW(pstrNameA, strName);

     hResult = ConvertSNBToW(snbExcludeA, &snbExclude);
     if (FAILED(hResult))
           return hResult;

     hResult = WrapIStorageWFromA(pstgPriorityA, &pstgPriority);
     if (FAILED(hResult))
           goto Error;

     hResult = GetWide()->OpenStorage(strName, pstgPriority, grfMode,
snbExclude, reserved, &pstg);
     if (FAILED(hResult))
           goto Error1;

     hResult = WrapIStorageAFromW(pstg, ppstgA);

     if (pstg)
           pstg->Release();

Error1:
     if (pstgPriority)
           pstgPriority->Release();
Error:
     ConvertSNBFree(snbExclude);

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CStorageA::CopyTo, public
//
//  Synopsis:   Thunks CopyTo to Unicode method.
//
//  Returns:    OLE2 Result Code.
```

```
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CStorageA::CopyTo(
           DWORD ciidExclude,
           IID const *rgiidExclude,
           SNBA snbExcludeA,
           IStorageA *pstgDestA)
{
     TraceMethodEnter("CStorageA::CopyTo", this);

     SNB      snbExclude;
     IStorage * pstgDest;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IStorage, CopyTo));

     hResult = ConvertSNBToW(snbExcludeA, &snbExclude);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapIStorageWFromA(pstgDestA, &pstgDest);
     if (FAILED(hResult))
           goto Error;

     hResult = GetWide()->CopyTo(ciidExclude, rgiidExclude, snbExclude,
pstgDest);

     if (pstgDest)
           pstgDest->Release();

Error:
     ConvertSNBFree(snbExclude);

     return hResult;
}


//
+--------------------------------------------------------------------------
//
//  Member:     CStorageA::MoveElementTo, public
//
//  Synopsis:   Thunks MoveElementTo to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CStorageA::MoveElementTo(
           LPCSTR pstrNameA,
           IStorageA *pstgDestA,
           LPCSTR pstrNewNameA,
           DWORD grfFlags)
```

```
{
     TraceMethodEnter("CStorageA::MoveElementTo", this);

     OLECHAR strName[MAX_PATH];
     OLECHAR strNewName[MAX_PATH];
     IStorage * pstgDest;
     HRESULT hResult;


     _DebugHook(GetWide(), MEMBER_PTR(IStorage, MoveElementTo));

     ConvertStringToW(pstrNameA, strName);

     ConvertStringToW(pstrNewNameA, strNewName);

     hResult = WrapIStorageWFromA(pstgDestA, &pstgDest);
     if (FAILED(hResult))
           return hResult;

     hResult = GetWide()->MoveElementTo(strName, pstgDest, strNewName,
grfFlags);


     if (pstgDest)
           pstgDest->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Member:     CStorageA::Commit, public
//
//  Synopsis:   Thunks Commit to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CStorageA::Commit(DWORD grfCommitFlags)
{
     _DebugHook(GetWide(), MEMBER_PTR(IStorage, Commit));

     return GetWide()->Commit(grfCommitFlags);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CStorageA::Revert, public
//
//  Synopsis:   Thunks Revert to Unicode method.
```

```
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CStorageA::Revert(VOID)
{
     _DebugHook(GetWide(), MEMBER_PTR(IStorage, Revert));

     return GetWide()->Revert();
}


//
+---------------------------------------------------------------------------
//
//  Member:     CStorageA::EnumElements, public
//
//  Synopsis:   Thunks EnumElements to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//----------------------------------------------------------------------------
-
STDMETHODIMP CStorageA::EnumElements(
           DWORD reserved1,
           void *reserved2,
           DWORD reserved3,
           IEnumSTATSTGA **ppenmA)
{
     TraceMethodEnter("CStorageA::EnumElements", this);

     IEnumSTATSTG * penm = NULL;
     HRESULT hResult;



     _DebugHook(GetWide(), MEMBER_PTR(IStorage, EnumElements));

     hResult = GetWide()->EnumElements(reserved1, reserved2, reserved3,
&penm);
     if (FAILED(hResult))
           return (hResult);

     if (penm)
     {
           hResult = WrapIEnumSTATSTGAFromW(penm, ppenmA);
           penm->Release();
     }
     else
           *ppenmA = NULL;

     return hResult;
}
```

```
//
+---------------------------------------------------------------------------
//
//  Member:     CStorageA::DestroyElement, public
//
//  Synopsis:   Thunks DestroyElement to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CStorageA::DestroyElement(LPCSTR pstrNameA)
{
     TraceMethodEnter("CStorageA::DestroyElement", this);

     OLECHAR strName[MAX_PATH];


     _DebugHook(GetWide(), MEMBER_PTR(IStorage, DestroyElement));

     ConvertStringToW(pstrNameA, strName);

     return GetWide()->DestroyElement(strName);
}


//
+---------------------------------------------------------------------------
//
//  Member:     CStorageA::RenameElement, public
//
//  Synopsis:   Thunks RenameElement to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//---------------------------------------------------------------------------
-
STDMETHODIMP CStorageA::RenameElement(
           LPCSTR pstrOldNameA,
           LPCSTR pstrNewNameA)
{
     TraceMethodEnter("CStorageA::RenameElement", this);

     OLECHAR strOldName[MAX_PATH];
     OLECHAR strNewName[MAX_PATH];


     _DebugHook(GetWide(), MEMBER_PTR(IStorage, RenameElement));

     ConvertStringToW(pstrOldNameA, strOldName);

     ConvertStringToW(pstrNewNameA, strNewName);

     return GetWide()->RenameElement(strOldName, strNewName);
}


//
+-------------------------------------------------------------------------
//
//  Member:     CStorageA::SetElementTimes, public
//
//  Synopsis:   Thunks SetElementTimes to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CStorageA::SetElementTimes(
           LPCSTR pstrNameA,
           FILETIME const *pctime,
           FILETIME const *patime,
           FILETIME const *pmtime)
{
     TraceMethodEnter("CStorageA::SetElementTimes", this);

     OLECHAR strName[MAX_PATH];


     _DebugHook(GetWide(), MEMBER_PTR(IStorage, SetElementTimes));

     ConvertStringToW(pstrNameA, strName);

     return GetWide()->SetElementTimes(strName, pctime, patime, pmtime);
}


//
+-------------------------------------------------------------------------
//
//  Member:     CStorageA::SetClass, public
//
//  Synopsis:   Thunks SetClass to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CStorageA::SetClass(REFCLSID clsid)
{
     _DebugHook(GetWide(), MEMBER_PTR(IStorage, SetClass));

     return GetWide()->SetClass(clsid);
}


//
+-------------------------------------------------------------------------
```

```
//
//  Member:     CStorageA::SetStateBits, public
//
//  Synopsis:   Thunks SetStateBits to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CStorageA::SetStateBits(DWORD grfCommitFlags, DWORD grfMask)
{
     _DebugHook(GetWide(), MEMBER_PTR(IStorage, SetStateBits));

     return GetWide()->SetStateBits(grfCommitFlags, grfMask);
}


//
+--------------------------------------------------------------------------
//
//  Member:     CStorageA::Stat, public
//
//  Synopsis:   Thunks Stat to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CStorageA::Stat(STATSTGA *pstatstg, DWORD grfStatFlag)
{
     TraceMethodEnter("CStorageA::Stat", this);

     _DebugHook(GetWide(), MEMBER_PTR(IStorage, Stat));

     HRESULT hResult = GetWide()->Stat((STATSTG *)pstatstg, grfStatFlag);
     if (FAILED(hResult))
           return (hResult);

     return ConvertSTATSTGToA(pstatstg);
}
```

IRootStorageA Implementation (OLEANSI Sample)

```
//
***************************************************************************
//
//                   IRootStorageA Implementation
//
//
***************************************************************************

//
+--------------------------------------------------------------------------
//
//  Member:     CRootStorageA::SwitchToFile, public
//
//  Synopsis:   Thunks SwitchToFile to Unicode method.
//
//  Returns:    OLE2 Result Code.
//
//--------------------------------------------------------------------------
-
STDMETHODIMP CRootStorageA::SwitchToFile(LPSTR pstrFileA)
{
     TraceMethodEnter("CRootStorageA::SwitchToFile", this);

     OLECHAR strFile[MAX_PATH];


     _DebugHook(GetWide(), MEMBER_PTR(IRootStorage, SwitchToFile));

     ConvertStringToW(pstrFileA, strFile);

     return GetWide()->SwitchToFile(strFile);
}
```

Storage API Thunks. (OLEANSI Sample)

```
//
***************************************************************************
//
//                          Storage API Thunks.
//
//
***************************************************************************


//
+--------------------------------------------------------------------------
//
//  Routine:    StgCreateDocfileA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//-------------------------------------------------------------------------
-
STDAPI StgCreateDocfileA(LPCSTR pcsNameA,
                 DWORD grfMode,
                 DWORD reserved,
                 IStorageA * *ppstgOpenA)
{
     TraceSTDAPIEnter("StgCreateDocfileA");

     OLECHAR  csName[MAX_PATH];
     LPOLESTR pcsName;
     IStorage * pstg;
     HRESULT  hResult;
     HRESULT  hReturn;


     *ppstgOpenA = NULL;

     if (pcsNameA)
     {
           ConvertStringToW(pcsNameA, csName);
           pcsName = csName;
     }
     else
           pcsName = NULL;

     hReturn = StgCreateDocfile(pcsName, grfMode, reserved, &pstg);
     if (FAILED(hReturn))
           return (hReturn);

     hResult = WrapIStorageAFromW(pstg, ppstgOpenA);
     if (SUCCEEDED(hResult))
           hResult = hReturn;
     if (pstg)
          pstg->Release();

     return hResult;
}


//
+------------------------------------------------------------------------
//
//  Routine:    StgCreateDocfileOnILockBytesA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//------------------------------------------------------------------------
-
STDAPI StgCreateDocfileOnILockBytesA(ILockBytesA *plckbytA,
                              DWORD grfMode,
                              DWORD reserved,
                              IStorageA * *ppstgOpenA)
{
     TraceSTDAPIEnter("StgCreateDocfileOnILockBytesA");
     ILockBytes * plckbyt;
     IStorage * pstg;
     HRESULT hResult;
     HRESULT hReturn;


     *ppstgOpenA = NULL;

     hReturn = WrapILockBytesWFromA(plckbytA, &plckbyt);
     if (FAILED(hReturn))
          return (hReturn);

     hReturn = StgCreateDocfileOnILockBytes(plckbyt, grfMode, reserved,
&pstg);
     if (FAILED(hReturn))
          goto Error;

     hResult = WrapIStorageAFromW(pstg, ppstgOpenA);
     if (FAILED(hResult))
          hReturn = hResult;

     if (pstg)
          pstg->Release();

Error:
     if (plckbyt)
          plckbyt->Release();

     return hReturn;
}
```

```
//
+---------------------------------------------------------------------------
//
//  Routine:    StgOpenStorageA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI StgOpenStorageA(LPCSTR pcsNameA,
                   IStorageA *pstgPriorityA,
                   DWORD grfMode,
                   SNBA snbExcludeA,
                   DWORD reserved,
                   IStorageA * *ppstgOpenA)
{
     TraceSTDAPIEnter("StgOpenStorageA");

     OLECHAR   csName[MAX_PATH];
     LPOLESTR  pcsName;
     LPSTORAGE pstgPriority;
     SNB       snbExclude;
     LPSTORAGE pstg;
     HRESULT   hResult;


     *ppstgOpenA = NULL;

     if (pcsNameA)
     {
           ConvertStringToW(pcsNameA, csName);
           pcsName= csName;
     }
     else
           pcsName = NULL;


     hResult = WrapIStorageWFromA(pstgPriorityA, &pstgPriority);
     if (FAILED(hResult))
           return hResult;

     hResult = ConvertSNBToW(snbExcludeA, &snbExclude);
     if (FAILED(hResult))
           goto Error;

     hResult = StgOpenStorage(pcsName, pstgPriority, grfMode, snbExclude,
reserved, &pstg);
     if (FAILED(hResult))
           goto Error1;

     hResult = WrapIStorageAFromW(pstg, ppstgOpenA);
     if (pstg)
           pstg->Release();

Error1:
     ConvertSNBFree(snbExclude);

Error:
     if (pstgPriority)
           pstgPriority->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    StgOpenStorageOnILockBytesA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI StgOpenStorageOnILockBytesA(ILockBytesA *plckbytA,
                         IStorageA *pstgPriorityA,
                         DWORD grfMode,
                         SNBA snbExcludeA,
                         DWORD reserved,
                         IStorageA * *ppstgOpenA)
{
     TraceSTDAPIEnter("StgOpenStorageOnILockBytesA");
     ILockBytes * plckbyt;
     IStorage * pstgPriority;
     IStorage * pstg;
     SNB        snbExclude;
     HRESULT hResult;


     *ppstgOpenA = NULL;

     hResult = WrapILockBytesWFromA(plckbytA, &plckbyt);
     if (FAILED(hResult))
           return (hResult);

     hResult = WrapIStorageWFromA(pstgPriorityA, &pstgPriority);
     if (FAILED(hResult))
           goto Error;

     hResult = ConvertSNBToW(snbExcludeA, &snbExclude);
     if (FAILED(hResult))
           goto Error1;
     hResult = StgOpenStorageOnILockBytes(plckbyt, pstgPriority, grfMode,
snbExclude, reserved, &pstg);
     if (FAILED(hResult))
          goto Error2;

     hResult = WrapIStorageAFromW(pstg, ppstgOpenA);

     if (pstg)
          pstg->Release();

Error2:
     ConvertSNBFree(snbExclude);

Error1:
     if (pstgPriority)
          pstgPriority->Release();

Error:
     if (plckbyt)
          plckbyt->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    StgIsStorageFileA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//---------------------------------------------------------------------------
-
STDAPI StgIsStorageFileA(LPCSTR pcsNameA)
{
     TraceSTDAPIEnter("StgIsStorageFileA");

     OLECHAR csName[MAX_PATH];


     ConvertStringToW(pcsNameA, csName);

     return StgIsStorageFile(csName);
}


//
+---------------------------------------------------------------------------
//
//  Routine:    StgIsStorageILockBytesA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
```

```
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI StgIsStorageILockBytesA(ILockBytesA * plckbytA)
{
     TraceSTDAPIEnter("StgIsStorageILockBytesA");
     ILockBytes * plckbyt;
     HRESULT hResult;


     hResult = WrapILockBytesWFromA(plckbytA, &plckbyt);
     if (FAILED(hResult))
           return (hResult);

     hResult = StgIsStorageILockBytes(plckbyt);

     if (plckbyt)
           plckbyt->Release();

     return hResult;
}


//
+---------------------------------------------------------------------------
//
//  Routine:    StgSetTimesA
//
//  Synopsis:   Creates an ANSI wrapper of an Unicode OLE2 routine.
//
//  Note:       See OLE2 docs for details on this API.
//
//----------------------------------------------------------------------------
-
STDAPI StgSetTimesA(LPCSTR lpszNameA,
             FILETIME const FAR* pctime,
                         FILETIME const FAR* patime,
                         FILETIME const FAR* pmtime)
{
     TraceSTDAPIEnter("StgSetTimesA");

     OLECHAR szName[MAX_PATH];


     ConvertStringToW(lpszNameA, szName);

     return StgSetTimes(szName, pctime, patime, pmtime);
}
```

CONVCTL.H   (OLEANSI Sample)

```
//
+---------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       ConvCtl.h
//
//  Contents:   ANSI Wrappers for OLE Control Interfaces and APIs.
//
//  History:    01-Jun-94   johnels     Created.
//
//---------------------------------------------------------------------------
-

HRESULT ConvertPROPPAGEINFOToA(LPPROPPAGEINFO, LPPROPPAGEINFOA);
HRESULT ConvertPROPPAGEINFOToW(LPPROPPAGEINFOA, LPPROPPAGEINFO);
void ConvertPROPPAGEINFOFree(LPPROPPAGEINFO);
void ConvertPROPPAGEINFOFree(LPPROPPAGEINFOA);
```

CONVCTL.CPP   (OLEANSI Sample)

```
//
+---------------------------------------------------------------------------
//
//  Copyright (C) 1994, Microsoft Corporation.  All Rights Reserved.
//
//  File:       ConvCtl.cpp
//
//  Contents:   ANSI Wrappers for OLE Control Interfaces and APIs.
//
//  History:    01-Jun-94   johnels     Created.
//
//---------------------------------------------------------------------------
-

#include "Ole2Ansi.h"


HRESULT ConvertPROPPAGEINFOToA(LPPROPPAGEINFO pPageInfo,
     LPPROPPAGEINFOA pPageInfoA)
{
     HRESULT hResult;

     memcpy((LPVOID)pPageInfoA, (LPVOID)pPageInfo, sizeof(PROPPAGEINFO));

     hResult = ConvertStringToA(pPageInfo->pszTitle, &pPageInfoA->pszTitle);
     if (FAILED(hResult))
           return hResult;

     hResult = ConvertStringToA(pPageInfo->pszDocString, &pPageInfoA-
>pszDocString);
     if (FAILED(hResult))
           goto Error1;

     hResult = ConvertStringToA(pPageInfo->pszHelpFile, &pPageInfoA-
>pszHelpFile);
     if (FAILED(hResult))
           goto Error2;

     return hResult;

Error2:
     ConvertStringFree(pPageInfoA->pszDocString);
     pPageInfoA->pszDocString = NULL;

Error1:
     ConvertStringFree(pPageInfoA->pszTitle);
     pPageInfoA->pszTitle = NULL;

     return hResult;
}

HRESULT ConvertPROPPAGEINFOToW(LPPROPPAGEINFOA pPageInfoA,
     LPPROPPAGEINFO pPageInfo)
{
     HRESULT hResult;

     memcpy((LPVOID)pPageInfo, (LPVOID)pPageInfoA, sizeof(PROPPAGEINFO));

     hResult = ConvertStringToW(pPageInfoA->pszTitle, &pPageInfo->pszTitle);
     if (FAILED(hResult))
           return hResult;

     hResult = ConvertStringToW(pPageInfoA->pszDocString, &pPageInfo-
>pszDocString);
     if (FAILED(hResult))
           goto Error1;

     hResult = ConvertStringToW(pPageInfoA->pszHelpFile, &pPageInfo-
>pszHelpFile);
     if (FAILED(hResult))
           goto Error2;

     return hResult;

Error2:
     ConvertStringFree(pPageInfo->pszDocString);
     pPageInfo->pszDocString = NULL;

Error1:
     ConvertStringFree(pPageInfo->pszTitle);
     pPageInfo->pszTitle = NULL;

     return hResult;
}

void ConvertPROPPAGEINFOFree(LPPROPPAGEINFO pPageInfo)
{
     ConvertStringFree(pPageInfo->pszTitle);
     ConvertStringFree(pPageInfo->pszDocString);
     ConvertStringFree(pPageInfo->pszHelpFile);
}

void ConvertPROPPAGEINFOFree(LPPROPPAGEINFOA pPageInfoA)
{
     ConvertStringFree(pPageInfoA->pszTitle);
     ConvertStringFree(pPageInfoA->pszDocString);
     ConvertStringFree(pPageInfoA->pszHelpFile);
}
```

OUTLINE

TOPIC: OLE 2.0 Outline Sample Code FILE: readme.txt DATE: 11/11/93

OUTLINE SERIES --------------

This sample application set demonstrates taking a 'base' application (in this case OUTLINE.EXE) and extending it into an OLE 2 server and container (SVROUTL and CNTROUTL respectively) and into OLE 2 in-place (aka visual editing) container (ICNTROTL).

These applications attempt to implement the complete OLE 2 functionality and recommended user model. For example, these applications implement all of the OLE 2 User Interface dialogs. As such this is not a trivial sample application set.

All OUTLINE applications support loading/saving files; they all use docfiles for storage.

DIRECTORY STRUCTURE -------------------

The OUTLINE sample uses the following directory structure:

```
                        OUTLINE
                           |
      +-------------+-----------+--------+--------+
      |             |           |        |        |
   OUTLINE       OLE2UI       WRAPUI  BTTNCUR   GIZMOBAR
```

OUTLINE -- source directory for OUTLINE sample app series
OLE2UI -- source directory for common OLE 2 UI library using Unicode OLE.
WRAPUI -- source directory for common OLE 2 UI library using OLE2ANSI wrappers for the OLE 2 calls.

All five variations of the OUTLINE applications are built from common sources. All of the common source code (*.C and *.H files) is in the OUTLINE directory. Each variant of the application builds in the same directory for this release.

Files that begin with OUTL comprise the base version of the application. They are also used by OLE versions of the applicaton. Files that begin with CNTR are specific to the container version and files that begin with SVR are specific to the server version. Files that begin with OLE are common to both the container and server versions. This series of sample applications highlights the changes necessary to implement OLE into an existing application. The container version has the same functionality as the base (OUTLINE) and also allows you to embed objects.

Some of the major OLE features are organized into special files in order to group related code across the application variants:

DRAGDROP.C -- Drag/Drop implementation
CLIPBRD.C -- OLE 2 style copy/paste implementation (data transfer)
LINKING.C -- Linking and Moniker support implementation
CNTRINPL.C -- In-place container implementation
SVRINPL.C -- In-place server implementation
CLASSFAC.C -- IClassFactory implementation

Conditional compilation is used to control what code is used with which version. The following defines are used to control the compilation:

OLE_VERSION -- code common to all OLE versions
OLE_SERVER -- code used by both SVROUTL and ISVROTL
OLE_CNTR -- code used by both CNTROUTL and ICNTROTL

INPLACE_CNTR -- code used by ICNTROTL only

The following defines are used to identify code specific for a particular feature (NOTE: it is not really intended that these symbols should be turned of):

USE_DRAGDROP	-- drag/drop code
USE_FRAMETOOLS	-- formula bar and tool bar
USE_HEADING	-- row/column headings code
USE_STATUSBAR	-- status bar code
USE_CTL3D	-- use CTL3D.DLL to have 3D effect for dialogs
USE_MSGFILTER	-- IMessageFilter code
_DEBUG	-- debug code

HOW TO BUILD ------------

To build, simple do an nmake -a -i from the root of the outline directory.

This will first build a version of the OLE2UI library, using it's own ANSI to UNICODE translations. This library is a private version of the OLE2UI library to be used exclusively by the OUTLINE samples applications.

NO OTHER APPLICATION MAY SHIP A DLL CALLED "OUTLUI.DLL".

Every application that builds the OLE2UI library as a DLL must build it with a unique name for that application. See the OLE2UI\README.TXT for instructions on how to build the OLE2UI library.

NOTES ON WRAPUI ---------------

WRAPUI builds a dll with the same functionality as OLE2UI, except that it uses the OLE2ANSI wrappers provided in the ole2\ole2ansi directory instead of doing its own ANSI to UNICODE translation. In general we recommend that ISVs actually provide their own translation since it will improve the performance and reduce the size of your application. However we realize that for a large project this may not be a feasible solution, so we have provided WRAPUI to illustrate how you may use the OLE2ANSI wrappers in your own application. Please see the ole2\ole2ansi\readme.txt for further info- rmation regarding the wrappers.

MAKEFILE   (OUTLINE Sample)

```
#
# (c) Copyright Microsoft Corp. 1992 All Rights Reserved
#
# Makefile : Builds the OLE 2.0 Outline series sample apps
#        The Outline series includes:
#           Outline.exe -- base version of app (w/o OLE functionality)
#           SvrOutl.exe -- OLE 2.0 Server sample app
#           CntrOutl.exe -- OLE 2.0 Containter (Container) sample app
#        NOTE: apps are built in a subdir of same name as app.
#
# Usage:     NMAKE              (builds all executables)
#    or:     NMAKE APP=outline  (build outline.exe--non-OLE version)
#    or:     NMAKE APP=svroutl  (build svroutl.exe--server-only version)
#    or:     NMAKE APP=cntroutl (build cntroutl.exe--container-only version)
#    or:     NMAKE APP=icntrotl (build icntrotl.exe--InPlace container ver.)
#    or:     NMAKE APP=isvrotl  (build isvrotl.exe--InPlace server ver.)
#    or:     NMAKE clean        (erase all compiled files)
#

!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

!if "$(APP)" == ""
APPS = outline.exe svroutl.exe cntroutl.exe icntrotl.exe isvrotl.exe
!else
APPS = $(APP).exe
!endif

all: $(APPS)

$(APPS):
    nmake -a -f outline.mk APP=$*

##########################################################################
#
# clean (erase) generated files

clean:
    -del *.obj
    -del *.res
    -del *.exe
    -del *.map
    -del *.pch
```

ANSIAPI.H (OUTLINE Sample)

```
/*
 * ansiapi.h
 * This file contains prototypes of ANSI version of OLE apis
 * and mapping fooA to foo
 */

#ifndef _ANSIAPI_INCL
#define _ANSIAPI_INCL

#define UNICODEOLE32

#ifdef WIN32S
#if !defined(UNICODEOLE32)
#pragma message("Compiling for 32bit ANSI Ole!\n");
#endif
#endif

#define WASANSI() {
\
                  OutputDebugString("\n\r\t\tUnicode string was ANSI!
\r\n"); \
                  Assert(0);
\
                  _asm { int 3 }
\
                 }



#ifndef UNICODEOLE32

#define W2A(w, a, cb)     lstrcpy (a, w)
#define A2W(a, w, cb)     lstrcpy (w, a)

#define FREELOCALSTRING(p)
#define MAKE_STR_LOCAL_COPYW2A(s, l)  l = s
#define MAKE_STR_LOCAL_COPYA2W(s, l)  l = s

#define OLESTRCPY                lstrcpy
#define OLESTRCAT                lstrcat
#define OLESTRLEN                lstrlen
#define OLESTRCMP                lstrcmp

#define DeleteFile_AW            DeleteFile
#define _lopen_AW                _lopen
#define _lcreat_AW               _lcreat
#define GlobalAddAtom_AW         GlobalAddAtom
#define GlobalGetAtomName_AW     GlobalGetAtomName
#define RegOpenKey_AW            RegOpenKey

#define CoLoadLibraryA   CoLoadLibrary
```

```
#define StringFromCLSID2A    StringFromCLSID2
#define StringFromIID2A      StringFromIID2
#define StringFromGUID2A     StringFromGUID2
#define CLSIDFromProgIDA     CLSIDFromProgID
#define CLSIDFromStringA     CLSIDFromString
#define ProgIDFromCLSIDA     ProgIDFromCLSID
#define StringFromCLSIDA     StringFromCLSID

#define UtDupStringA2W       UtDupString

// Storage APIs
#define StgOpenStorageA      StgOpenStorage

// IPersistFile ansi translation

#define IPersistFile_LoadA(pf, file, mode) (pf)->Load(file, mode)

// IMoniker ansi translation

#define IMoniker_GetDisplayNameA(pm, p1, p2, p3) (pm)-
>GetDisplayName(p1,p2,p3)

#define CreateFileMonikerA   CreateFileMoniker

#else  // UNICODEOLE32


#define W2A(w, a, cb)
WideCharToMultiByte(                                   \
                                               CP_ACP,
\
                                               0,
\
                                               w,
\
                                               -1,
\
                                               a,
\
                                               cb,
\
                                               NULL,
\
                                               NULL)

#define A2W(a, w, cb)
MultiByteToWideChar(                                   \
                                               CP_ACP,
\
                                               0,
\
                                               a,
\
                                               -1,
\
                                                w,
\
                                                cb)


#define FREELOCALSTRING(p)      delete (p)
#define MAKE_STR_LOCAL_COPYW2A(s, l)  {                             \
                                        l = UtDupStringW2A(s);        \
                                        if (!l) {                     \
                                           return ResultFromScode(S_OOM);   \
                                        }                             \
                                     }

#define MAKE_STR_LOCAL_COPYA2W(s, l)  {                             \
                                        l = UtDupStringA2W(s);        \
                                        if (!l) {                     \
                                           return ResultFromScode(S_OOM);   \
                                        }                             \
                                     }

#define OLESTRCPY                lstrcpyW
#define OLESTRCAT                lstrcatW
#define OLESTRLEN                lstrlenW
#define OLESTRCMP                lstrcmpW

#define DeleteFile_AW            DeleteFileW
#define _lopen_AW                _lopenW
#define _lcreat_AW               _lcreatW
#define GlobalAddAtom_AW         GlobalAddAtomW
#define GlobalGetAtomName_AW     GlobalGetAtomNameW
#define RegOpenKey_AW            RegOpenKeyW

//STDAPI_(HINSTANCE) CoLoadLibraryA(LPSTR lpszLibName, BOOL bAutoFree);

#define StringFromCLSID2A(rclsid, lpsz, cbMax) \
    StringFromGUID2A(rclsid, lpsz, cbMax)

#define StringFromIID2A(riid, lpsz, cbMax) \
    StringFromGUID2A(riid, lpsz, cbMax)


//STDAPI_(int)  StringFromGUID2A(REFGUID rguid, LPSTR lpsz, int cbMax);
//STDAPI        CLSIDFromProgIDA(LPCSTR szProgID, LPCLSID pclsid);
//STDAPI        CLSIDFromStringA(LPSTR lpsz, LPCLSID lpclsid);
//STDAPI        StringFromCLSIDA(REFCLSID rclsid, LPSTR FAR* lplpsz);
//STDAPI        ProgIDFromCLSIDA (REFCLSID clsid, LPSTR FAR* lplpszProgID);

//LPWSTR UtDupStringA2W(LPCSTR pSrc);
//LPSTR  UtDupStringW2A(LPCWSTR pSrc);

// Storage APIs
//STDAPI StgOpenStorageA(LPCSTR pwcsName,IStorage FAR *pstgPriority, DWORD
grfMode, SNB snbExclude, DWORD reserved, IStorage FAR * FAR *ppstgOpen);

// IPersistFile ansi translation
```

```
//HRESULT IPersistFile_LoadA(LPPERSISTFILE pIPF, LPSTR szFile, DWORD
dwMode);

// IMoniker ansi translation

//HRESULT IMoniker_GetDisplayNameA(LPMONIKER pm, LPBC p1, LPMONIKER p2,
LPSTR FAR *p3);

//OLEAPI CreateFileMonikerA ( LPSTR lpszPathName, LPMONIKER FAR * ppmk );

#endif // !UNICODEOLE32

#endif //  _ANSIAPI_INCL
_
```

CLASSFAC.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    classfac.c
**
**    This file contains the implementation for IClassFactory for both the
**    server and the client version of the OUTLINE app.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"

OLEDBGDATA

extern LPOUTLINEAPP             g_lpApp;


/* OLE2NOTE: this object illustrates the manner in which to statically
**    (compile-time) initialize an interface VTBL.
*/
static IClassFactoryVtbl g_AppClassFactoryVtbl = {
     AppClassFactory_QueryInterface,
     AppClassFactory_AddRef,
     AppClassFactory_Release,
     AppClassFactory_CreateInstance,
     AppClassFactory_LockServer
};


/* AppClassFactory_Create
** ----------------------
**    create an instance of APPCLASSFACTORY.
**    NOTE: type of pointer returned is an IClassFactory* interface ptr.
**          the returned pointer can be directly passed to
**          CoRegisterClassObject and released later by calling the
**          Release method of the interface.
*/
LPCLASSFACTORY WINAPI AppClassFactory_Create(void)
{
     LPAPPCLASSFACTORY lpAppClassFactory;
     LPMALLOC lpMalloc;

     if (CoGetMalloc(MEMCTX_TASK, (LPMALLOC FAR*)&lpMalloc) != NOERROR)
           return NULL;

     lpAppClassFactory = (LPAPPCLASSFACTORY)lpMalloc->lpVtbl->Alloc(
                 lpMalloc, (sizeof(APPCLASSFACTORY)));
     lpMalloc->lpVtbl->Release(lpMalloc);
     if (! lpAppClassFactory) return NULL;
     lpAppClassFactory->m_lpVtbl = &g_AppClassFactoryVtbl;
     lpAppClassFactory->m_cRef   = 1;
#if defined( _DEBUG )
     lpAppClassFactory->m_cSvrLock = 0;
#endif
     return (LPCLASSFACTORY)lpAppClassFactory;
}


/*************************************************************************
** OleApp::IClassFactory interface implementation
*************************************************************************/

STDMETHODIMP AppClassFactory_QueryInterface(
           LPCLASSFACTORY lpThis, REFIID riid, LPVOID FAR* ppvObj)
{
     LPAPPCLASSFACTORY lpAppClassFactory = (LPAPPCLASSFACTORY)lpThis;
     SCODE scode;

     // Two interfaces supported: IUnknown, IClassFactory

     if (IsEqualIID(riid, &IID_IClassFactory) ||
                 IsEqualIID(riid, &IID_IUnknown)) {
           lpAppClassFactory->m_cRef++;   // A pointer to this object is
returned
           *ppvObj = lpThis;
           scode = S_OK;
     }
     else {                 // unsupported interface
           *ppvObj = NULL;
           scode = E_NOINTERFACE;
     }

     return ResultFromScode(scode);
}


STDMETHODIMP_(ULONG) AppClassFactory_AddRef(LPCLASSFACTORY lpThis)
{
     LPAPPCLASSFACTORY lpAppClassFactory = (LPAPPCLASSFACTORY)lpThis;
     return ++lpAppClassFactory->m_cRef;
}

STDMETHODIMP_(ULONG) AppClassFactory_Release(LPCLASSFACTORY lpThis)
{
     LPAPPCLASSFACTORY lpAppClassFactory = (LPAPPCLASSFACTORY)lpThis;
     LPMALLOC lpMalloc;

     if (--lpAppClassFactory->m_cRef != 0) // Still used by others
           return lpAppClassFactory->m_cRef;

     // Free storage
     if (CoGetMalloc(MEMCTX_TASK, (LPMALLOC FAR*)&lpMalloc) != NOERROR)
           return 0;
     lpMalloc->lpVtbl->Free(lpMalloc, lpAppClassFactory);
     lpMalloc->lpVtbl->Release(lpMalloc);
     return 0;
}


STDMETHODIMP AppClassFactory_CreateInstance (
           LPCLASSFACTORY      lpThis,
           LPUNKNOWN           lpUnkOuter,
           REFIID              riid,
           LPVOID FAR*         lplpvObj
)
{
     LPOUTLINEAPP    lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
     LPOLEDOC        lpOleDoc;
     HRESULT         hrErr;

     OLEDBG_BEGIN2("AppClassFactory_CreateInstance\r\n")

     /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
     *lplpvObj = NULL;

     /*******************************************************************
     ** OLE2NOTE: this is an SDI app; it can only create and support one
     **    instance. After the instance is created, the OLE libraries
     **    should not call CreateInstance again. it is a good practise
     **    to specifically guard against this.
     *******************************************************************/

     if (lpOutlineApp->m_lpDoc != NULL)
           return ResultFromScode(E_UNEXPECTED);

     /* OLE2NOTE: create a new document instance. by the time we return
     **    from this method the document's refcnt must be 1.
     */
     lpOutlineApp->m_lpDoc = OutlineApp_CreateDoc(lpOutlineApp, FALSE);
     lpOleDoc = (LPOLEDOC)lpOutlineApp->m_lpDoc;
     if (! lpOleDoc) {
           OLEDBG_END2
           return ResultFromScode(E_OUTOFMEMORY);
     }

     /* OLE2NOTE: retrieve pointer to requested interface. the ref cnt
     **    of the object after OutlineApp_CreateDoc is 0. this call to
     **    QueryInterface will increment the refcnt to 1. the object
     **    returned from IClassFactory::CreateInstance should have a
     **    refcnt of 1 and be controlled by the caller. If the caller
     **    releases the document, the document should be destroyed.
     */
     hrErr = OleDoc_QueryInterface(lpOleDoc, riid, lplpvObj);

     OLEDBG_END2
     return hrErr;
}
```

```
STDMETHODIMP AppClassFactory_LockServer (
           LPCLASSFACTORY      lpThis,
           BOOL                fLock
)
{
     LPAPPCLASSFACTORY lpAppClassFactory = (LPAPPCLASSFACTORY)lpThis;
     LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;
     HRESULT hrErr;
     OLEDBG_BEGIN2("AppClassFactory_LockServer\r\n")

#if defined( _DEBUG )
     if (fLock) {
           ++lpAppClassFactory->m_cSvrLock;
           OleDbgOutRefCnt3(
                       "AppClassFactory_LockServer: cLock++\r\n",
                       lpAppClassFactory, lpAppClassFactory->m_cSvrLock);
     } else {

           /* OLE2NOTE: when there are no open documents and the app is not
           **    under the control of the user and there are no outstanding
           **    locks on the app, then revoke our ClassFactory to enable
the
           **    app to shut down.
           */
           --lpAppClassFactory->m_cSvrLock;
           OleDbgAssertSz (lpAppClassFactory->m_cSvrLock >= 0,
                       "AppClassFactory_LockServer(FALSE) called with cLock
== 0"
           );

           if (lpAppClassFactory->m_cSvrLock == 0) {
                 OleDbgOutRefCnt2(
                             "AppClassFactory_LockServer: UNLOCKED\r\n",
                             lpAppClassFactory, lpAppClassFactory-
>m_cSvrLock);
           } else {
                 OleDbgOutRefCnt3(
                             "AppClassFactory_LockServer: cLock--\r\n",
                             lpAppClassFactory, lpAppClassFactory-
>m_cSvrLock);
           }
     }
#endif  // _DEBUG
     /* OLE2NOTE: in order to hold the application alive we call
     **    CoLockObjectExternal to add a strong reference to our app
     **    object. this will keep the app alive when all other external
     **    references release us. if the user issues File.Exit the
     **    application will shut down in any case ignoring any
     **    outstanding LockServer locks because CoDisconnectObject is
     **    called in OleApp_CloseAllDocsAndExitCommand. this will
     **    forceably break any existing strong reference counts
     **    including counts that we add ourselves by calling
     **    CoLockObjectExternal and guarantee that the App object gets
```

```
	**    its final release (ie. cRefs goes to 0).
	*/
	hrErr = OleApp_Lock(lpOleApp, fLock, TRUE /* fLastUnlockReleases */);

	OLEDBG_END2
	return hrErr;
}
```

CLIPBRD.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    clipbrd.c
**
**    This file contains the major interfaces, methods and related support
**    functions for implementing clipboard data transfer. The code
**    contained in this file is used by BOTH the Container and Server
**    (Object) versions of the Outline sample code.
**    (see file dragdrop.c for Drag/Drop support implementation)
**
**    OleDoc Object
**      exposed interfaces:
**          IDataObject
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"

OLEDBGDATA

extern LPOUTLINEAPP             g_lpApp;

// REVIEW: should use string resource for messages
OLECHAR ErrMsgPasting[] = OLESTR("Could not paste data from clipboard!");
OLECHAR ErrMsgBadFmt[] = OLESTR("Invalid format selected!");
OLECHAR ErrMsgPasteFailed[] = OLESTR("Could not paste data from
clipboard!");
OLECHAR ErrMsgClipboardChanged[] = OLESTR("Contents of clipboard have
changed!\r\nNo paste performed.");

#ifdef WIN32S
#define GALLOCFLG (GMEM_SHARE | GMEM_ZEROINIT | GMEM_MOVEABLE)
#else
#define GALLOCFLG (GMEM_SHARE | GMEM_ZEROINIT)
#endif

/*************************************************************************
** OleDoc::IDataObject interface implementation
*************************************************************************/

// IDataObject::QueryInterface
STDMETHODIMP OleDoc_DataObj_QueryInterface (
      LPDATAOBJECT        lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
)
{
   LPOLEDOC lpOleDoc = ((struct CDocDataObjectImpl FAR*)lpThis)->lpOleDoc;
   return OleDoc_QueryInterface((LPOLEDOC)lpOleDoc, riid, lplpvObj);
}


// IDataObject::AddRef
STDMETHODIMP_(ULONG) OleDoc_DataObj_AddRef(LPDATAOBJECT lpThis)
{
   LPOLEDOC lpOleDoc = ((struct CDocDataObjectImpl FAR*)lpThis)->lpOleDoc;

   OleDbgAddRefMethod(lpThis, "IDataObject");

   return OleDoc_AddRef((LPOLEDOC)lpOleDoc);
}


// IDataObject::Release
STDMETHODIMP_(ULONG) OleDoc_DataObj_Release (LPDATAOBJECT lpThis)
{
   LPOLEDOC lpOleDoc = ((struct CDocDataObjectImpl FAR*)lpThis)->lpOleDoc;

   OleDbgReleaseMethod(lpThis, "IDataObject");

   return OleDoc_Release((LPOLEDOC)lpOleDoc);
}


// IDataObject::GetData
STDMETHODIMP OleDoc_DataObj_GetData (
      LPDATAOBJECT        lpThis,
      LPFORMATETC         lpFormatetc,
      LPSTGMEDIUM         lpMedium
)
{
   LPOLEDOC lpOleDoc = ((struct CDocDataObjectImpl FAR*)lpThis)->lpOleDoc;
   HRESULT hrErr;

   OLEDBG_BEGIN2("OleDoc_DataObj_GetData\r\n")

#if defined( OLE_SERVER )
   // Call OLE Server specific version of this function
   hrErr = ServerDoc_GetData((LPSERVERDOC)lpOleDoc, lpFormatetc, lpMedium);
#endif
#if defined( OLE_CNTR )
   // Call OLE Container specific version of this function
   hrErr = ContainerDoc_GetData(
         (LPCONTAINERDOC)lpOleDoc,
         lpFormatetc,
         lpMedium
   );
#endif

   OLEDBG_END2
   return hrErr;
}
```

```
// IDataObject::GetDataHere
STDMETHODIMP OleDoc_DataObj_GetDataHere (
      LPDATAOBJECT        lpThis,
      LPFORMATETC         lpFormatetc,
      LPSTGMEDIUM         lpMedium
)
{
   LPOLEDOC lpOleDoc = ((struct CDocDataObjectImpl FAR*)lpThis)->lpOleDoc;
   HRESULT hrErr;

   OLEDBG_BEGIN2("OleDoc_DataObj_GetDataHere\r\n")

#if defined( OLE_SERVER )
   // Call OLE Server specific version of this function
   hrErr = ServerDoc_GetDataHere(
         (LPSERVERDOC)lpOleDoc,
         lpFormatetc,
         lpMedium
   );
#endif
#if defined( OLE_CNTR )
   // Call OLE Container specific version of this function
   hrErr = ContainerDoc_GetDataHere(
         (LPCONTAINERDOC)lpOleDoc,
         lpFormatetc,
         lpMedium
   );
#endif

   OLEDBG_END2
   return hrErr;
}


// IDataObject::QueryGetData
STDMETHODIMP OleDoc_DataObj_QueryGetData (
      LPDATAOBJECT        lpThis,
      LPFORMATETC         lpFormatetc
)
{
   LPOLEDOC lpOleDoc = ((struct CDocDataObjectImpl FAR*)lpThis)->lpOleDoc;
   HRESULT hrErr;
   OLEDBG_BEGIN2("OleDoc_DataObj_QueryGetData\r\n");

#if defined( OLE_SERVER )
   // Call OLE Server specific version of this function
   hrErr = ServerDoc_QueryGetData((LPSERVERDOC)lpOleDoc, lpFormatetc);
#endif
#if defined( OLE_CNTR )
   // Call OLE Container specific version of this function
   hrErr = ContainerDoc_QueryGetData((LPCONTAINERDOC)lpOleDoc, lpFormatetc);
#endif
```

```
   OLEDBG_END2
   return hrErr;
}


// IDataObject::GetCanonicalFormatEtc
STDMETHODIMP OleDoc_DataObj_GetCanonicalFormatEtc(
      LPDATAOBJECT        lpThis,
      LPFORMATETC         lpformatetc,
      LPFORMATETC         lpformatetcOut
)
{
   HRESULT hrErr;
   OleDbgOut2("OleDoc_DataObj_GetCanonicalFormatEtc\r\n");

   if (!lpformatetcOut)
      return ResultFromScode(E_INVALIDARG);

   /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
   lpformatetcOut->ptd = NULL;

   if (!lpformatetc)
      return ResultFromScode(E_INVALIDARG);

   // OLE2NOTE: we must validate that the format requested is supported
   if ((hrErr=lpThis->lpVtbl->QueryGetData(lpThis,lpformatetc)) != NOERROR)
      return hrErr;

   /* OLE2NOTE: an app that is insensitive to target device (as the
   **    Outline Sample is) should fill in the lpformatOut parameter
   **    but NULL out the "ptd" field; it should return NOERROR if the
   **    input formatetc->ptd what non-NULL. this tells the caller
   **    that it is NOT necessary to maintain a separate screen
   **    rendering and printer rendering. if should return
   **    DATA_S_SAMEFORMATETC if the input and output formatetc's are
   **    identical.
   */

   *lpformatetcOut = *lpformatetc;
   if (lpformatetc->ptd == NULL)
      return ResultFromScode(DATA_S_SAMEFORMATETC);
   else {
      lpformatetcOut->ptd = NULL;
      return NOERROR;
   }
}


// IDataObject::SetData
STDMETHODIMP OleDoc_DataObj_SetData (
      LPDATAOBJECT    lpThis,
      LPFORMATETC     lpFormatetc,
      LPSTGMEDIUM     lpMedium,
      BOOL            fRelease
)
```

```
{
   LPOLEDOC lpOleDoc = ((struct CDocDataObjectImpl FAR*)lpThis)->lpOleDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpOleDoc;
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   SCODE sc = S_OK;
   OLEDBG_BEGIN2("OleDoc_DataObj_SetData\r\n")

   /* OLE2NOTE: a document that is used to transfer data (either via
   **    the clipboard or drag/drop) does NOT accept SetData on ANY
   **    format!
   */
   if (lpOutlineDoc->m_fDataTransferDoc) {
      sc = E_FAIL;
      goto error;
   }

#if defined( OLE_SERVER )
   if (lpFormatetc->cfFormat == lpOutlineApp->m_cfOutline) {
      OLEDBG_BEGIN2("ServerDoc_SetData: CF_OUTLINE\r\n")
      OutlineDoc_SetRedraw ( lpOutlineDoc, FALSE );
      OutlineDoc_ClearAllLines(lpOutlineDoc);
      OutlineDoc_PasteOutlineData(lpOutlineDoc,lpMedium->u.hGlobal,-1);
      OutlineDoc_SetRedraw ( lpOutlineDoc, TRUE );
      OLEDBG_END3
   } else if (lpFormatetc->cfFormat == CF_TEXT) {
      OLEDBG_BEGIN2("ServerDoc_SetData: CF_TEXT\r\n")
      OutlineDoc_SetRedraw ( lpOutlineDoc, FALSE );
      OutlineDoc_ClearAllLines(lpOutlineDoc);
      OutlineDoc_PasteTextData(lpOutlineDoc,lpMedium->u.hGlobal,-1);
      OutlineDoc_SetRedraw ( lpOutlineDoc, TRUE );
      OLEDBG_END3
   } else {
      sc = DV_E_FORMATETC;
   }
#endif  // OLE_SERVER
#if defined( OLE_CNTR )
   /* the Container-Only version of Outline does NOT offer
   **    IDataObject interface from its User documents. this is
   **    required by objects which can be embedded or linked. the
   **    Container-only app only allows linking to its contained
   **    objects, NOT the data of the container itself.
   */
   OleDbgAssertSz(0, "User documents do NOT support IDataObject\r\n");
   sc = E_NOTIMPL;
#endif  // OLE_CNTR

error:

   /* OLE2NOTE: if fRelease==TRUE, then we must take
   **    responsibility to release the lpMedium. we should only do
   **    this if we are going to return NOERROR. if we do NOT
   **    accept the data, then we should NOT release the lpMedium.
   **    if fRelease==FALSE, then the caller retains ownership of
   **    the data.
   */
   if (sc == S_OK && fRelease)
      ReleaseStgMedium(lpMedium);

   OLEDBG_END2
   return ResultFromScode(sc);

}


// IDataObject::EnumFormatEtc
STDMETHODIMP OleDoc_DataObj_EnumFormatEtc(
      LPDATAOBJECT            lpThis,
      DWORD                   dwDirection,
      LPENUMFORMATETC FAR*    lplpenumFormatEtc
)
{
   LPOLEDOC lpOleDoc=((struct CDocDataObjectImpl FAR*)lpThis)->lpOleDoc;
   HRESULT hrErr;

   OLEDBG_BEGIN2("OleDoc_DataObj_EnumFormatEtc\r\n")

   /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
   *lplpenumFormatEtc = NULL;

#if defined( OLE_SERVER )
   /* OLE2NOTE: a user document only needs to enumerate the static list
   **    of formats that are registered for our app in the
   **    registration database. OLE provides a default enumerator
   **    which enumerates from the registration database. this default
   **    enumerator is requested by returning OLE_S_USEREG. it is NOT
   **    required that a user document (ie. non-DataTransferDoc)
   **    enumerate the OLE formats: CF_LINKSOURCE, CF_EMBEDSOURCE, or
   **    CF_EMBEDDEDOBJECT.
   **
   **    An object implemented as a server EXE (as this sample
   **    is) may simply return OLE_S_USEREG to instruct the OLE
   **    DefHandler to call the OleReg* helper API which uses info in
   **    the registration database. Alternatively, the OleRegEnumFormatEtc
   **    API may be called directly. Objects implemented as a server
   **    DLL may NOT return OLE_S_USEREG; they must call the OleReg*
   **    API or provide their own implementation. For EXE based
   **    objects it is more efficient to return OLE_S_USEREG, because
   **    in then the enumerator is instantiated in the callers
   **    process space and no LRPC remoting is required.
   */
   if (! ((LPOUTLINEDOC)lpOleDoc)->m_fDataTransferDoc)
      return ResultFromScode(OLE_S_USEREG);

   // Call OLE Server specific version of this function
   hrErr = ServerDoc_EnumFormatEtc(
         (LPSERVERDOC)lpOleDoc,
         dwDirection,
         lplpenumFormatEtc
   );
#endif
```

```
#if defined( OLE_CNTR )
   // Call OLE Container specific version of this function
   hrErr = ContainerDoc_EnumFormatEtc(
         (LPCONTAINERDOC)lpOleDoc,
         dwDirection,
         lplpenumFormatEtc
   );
#endif

   OLEDBG_END2
   return hrErr;
}


// IDataObject::DAdvise
STDMETHODIMP OleDoc_DataObj_DAdvise(
      LPDATAOBJECT        lpThis,
      FORMATETC FAR*      lpFormatetc,
      DWORD               advf,
      LPADVISESINK        lpAdvSink,
      DWORD FAR*          lpdwConnection
)
{
   LPOLEDOC lpOleDoc=((struct CDocDataObjectImpl FAR*)lpThis)->lpOleDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpOleDoc;
   SCODE sc;

   OLEDBG_BEGIN2("OleDoc_DataObj_DAdvise\r\n")

   /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
   *lpdwConnection = 0;

   /* OLE2NOTE: a document that is used to transfer data (either via
   **    the clipboard or drag/drop) does NOT support Advise notifications.
   */
   if (lpOutlineDoc->m_fDataTransferDoc) {
      sc = OLE_E_ADVISENOTSUPPORTED;
      goto error;
   }

#if defined( OLE_SERVER )
   {
      HRESULT hrErr;
      LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOleDoc;

      /* OLE2NOTE: we should validate if the caller is setting up an
      **    Advise for a data type that we support. we must
      **    explicitly allow an advise for the "wildcard" advise.
      */
#ifdef WIN32
      if ( !( lpFormatetc->cfFormat == 0 &&
            lpFormatetc->ptd == 0 &&
#else
      if ( !( lpFormatetc->cfFormat == NULL &&
            lpFormatetc->ptd == NULL &&
#endif
            lpFormatetc->dwAspect == -1L &&
            lpFormatetc->lindex == -1L &&
            lpFormatetc->tymed == -1L) &&
          (hrErr = OleDoc_DataObj_QueryGetData(lpThis, lpFormatetc))
                                                != NOERROR) {
         sc = GetScode(hrErr);
         goto error;
      }

      if (lpServerDoc->m_OleDoc.m_fObjIsClosing)
      {
          // We don't accept any more Advies's once we're closing
          sc = OLE_E_ADVISENOTSUPPORTED;
          goto error;
      }

      if (lpServerDoc->m_lpDataAdviseHldr == NULL &&
         CreateDataAdviseHolder(&lpServerDoc->m_lpDataAdviseHldr)
                                                != NOERROR) {
            sc = E_OUTOFMEMORY;
            goto error;
      }
      // artificial AddRef in case someone releases this object
      // during the next call
      OleDoc_DataObj_AddRef(lpThis);

      OLEDBG_BEGIN2("IDataAdviseHolder::Advise called\r\n");
      hrErr = lpServerDoc->m_lpDataAdviseHldr->lpVtbl->Advise(
            lpServerDoc->m_lpDataAdviseHldr,
            (LPDATAOBJECT)&lpOleDoc->m_DataObject,
            lpFormatetc,
            advf,
            lpAdvSink,
            lpdwConnection
      );
      OLEDBG_END2

      // release artificial AddRef above
      OleDoc_DataObj_Release(lpThis);

      OLEDBG_END2
      return hrErr;
   }
#endif  // OLE_SVR
#if defined( OLE_CNTR )
   {
      /* the Container-Only version of Outline does NOT offer
      **    IDataObject interface from its User documents. this is
      **    required by objects which can be embedded or linked. the
      **    Container-only app only allows linking to its contained
      **    objects, NOT the data of the container itself.
      */
      OleDbgAssertSz(0, "User documents do NOT support IDataObject\r\n");
      sc = E_NOTIMPL;
      goto error;
   }
#endif  // OLE_CNTR

error:
   OLEDBG_END2
   return ResultFromScode(sc);
}



// IDataObject::DUnadvise
STDMETHODIMP OleDoc_DataObj_DUnadvise(LPDATAOBJECT lpThis, DWORD
dwConnection)
{
   LPOLEDOC lpOleDoc=((struct CDocDataObjectImpl FAR*)lpThis)->lpOleDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpOleDoc;
   SCODE sc;

   OLEDBG_BEGIN2("OleDoc_DataObj_DUnadvise\r\n")

   /* OLE2NOTE: a document that is used to transfer data (either via
   **    the clipboard or drag/drop) does NOT support Advise notifications.
   */
   if (lpOutlineDoc->m_fDataTransferDoc) {
      sc = OLE_E_ADVISENOTSUPPORTED;
      goto error;
   }

#if defined( OLE_SERVER )
   {
      LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOleDoc;
      HRESULT hrErr;

      if (lpServerDoc->m_lpDataAdviseHldr == NULL) {
         sc = E_FAIL;
         goto error;
      }

      // artificial AddRef in case someone releases this object
      // during the next call
      OleDoc_DataObj_AddRef(lpThis);

      OLEDBG_BEGIN2("IDataAdviseHolder::Unadvise called\r\n");
      hrErr = lpServerDoc->m_lpDataAdviseHldr->lpVtbl->Unadvise(
            lpServerDoc->m_lpDataAdviseHldr,
            dwConnection
      );
      OLEDBG_END2

      // release artifical AddRef
      OleDoc_DataObj_Release(lpThis);

      OLEDBG_END2
      return hrErr;
   }
#endif
#if defined( OLE_CNTR )
   {
      /* the Container-Only version of Outline does NOT offer
      **    IDataObject interface from its User documents. this is
      **    required by objects which can be embedded or linked. the
      **    Container-only app only allows linking to its contained
      **    objects, NOT the data of the container itself.
      */
      OleDbgAssertSz(0, "User documents do NOT support IDataObject\r\n");
      sc = E_NOTIMPL;
      goto error;
   }
#endif

error:
   OLEDBG_END2
   return ResultFromScode(sc);
}


// IDataObject::EnumDAdvise
STDMETHODIMP OleDoc_DataObj_EnumDAdvise(
      LPDATAOBJECT        lpThis,
      LPENUMSTATDATA FAR* lplpenumAdvise
)
{
   LPOLEDOC lpOleDoc=((struct CDocDataObjectImpl FAR*)lpThis)->lpOleDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpOleDoc;
   SCODE sc;

   OLEDBG_BEGIN2("OleDoc_DataObj_EnumDAdvise\r\n")

   /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
   *lplpenumAdvise = NULL;

   /* OLE2NOTE: a document that is used to transfer data (either via
   **    the clipboard or drag/drop) does NOT support Advise notifications.
   */
   if (lpOutlineDoc->m_fDataTransferDoc) {
      sc = OLE_E_ADVISENOTSUPPORTED;
      goto error;
   }

#if defined( OLE_SERVER )
   {
      LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOleDoc;
      HRESULT hrErr;

      if (lpServerDoc->m_lpDataAdviseHldr == NULL) {
         sc = E_FAIL;
         goto error;
      }
      // artificial AddRef in case someone releases this
      // object during the next call
      OleDoc_DataObj_AddRef(lpThis);

      OLEDBG_BEGIN2("IDataAdviseHolder::EnumAdvise called\r\n");
      hrErr = lpServerDoc->m_lpDataAdviseHldr->lpVtbl->EnumAdvise(
            lpServerDoc->m_lpDataAdviseHldr,
            lplpenumAdvise
      );
      OLEDBG_END2

      // release artifical AddRef
      OleDoc_DataObj_Release(lpThis);

      OLEDBG_END2
      return hrErr;
   }
#endif
#if defined( OLE_CNTR )
   {
      /* the Container-Only version of Outline does NOT offer
      **    IDataObject interface from its User documents. this is
      **    required by objects which can be embedded or linked. the
      **    Container-only app only allows linking to its contained
      **    objects, NOT the data of the container itself.
      */
      OleDbgAssertSz(0, "User documents do NOT support IDataObject\r\n");
      sc = E_NOTIMPL;
      goto error;
   }
#endif

error:
   OLEDBG_END2
   return ResultFromScode(sc);
}



/*************************************************************************
** OleDoc Supprt Functions common to both Container and Server versions
*************************************************************************/


/* OleDoc_CopyCommand
 * ------------------
 *  Copy selection to clipboard.
 *  Post to the clipboard the formats that the app can render.
 *  the actual data is not rendered at this time. using the
 *  delayed rendering technique, Windows will send the clipboard
 *  owner window either a WM_RENDERALLFORMATS or a WM_RENDERFORMAT
 *  message when the actual data is requested.
 *
 *    OLE2NOTE: the normal delayed rendering technique where Windows
 *    sends the clipboard owner window either a WM_RENDERALLFORMATS or
 *    a WM_RENDERFORMAT message when the actual data is requested is
 *    NOT exposed to the app calling OleSetClipboard. OLE internally
 *    creates its own window as the clipboard owner and thus our app
 *    will NOT get these WM_RENDER messages.
 */
void OleDoc_CopyCommand(LPOLEDOC lpSrcOleDoc)
{
   LPOUTLINEDOC lpSrcOutlineDoc = (LPOUTLINEDOC)lpSrcOleDoc;
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPOUTLINEDOC lpClipboardDoc;

   /* squirrel away a copy of the current selection to the ClipboardDoc */
   lpClipboardDoc = OutlineDoc_CreateDataTransferDoc(lpSrcOutlineDoc);

   if (! lpClipboardDoc)
      return;     // Error: could not create DataTransferDoc

   lpOutlineApp->m_lpClipboardDoc = (LPOUTLINEDOC)lpClipboardDoc;

   /* OLE2NOTE: initially the Doc object is created with a 0 ref
   **    count. in order to have a stable Doc object during the
   **    process of initializing the Doc instance and transfering it
   **    to the clipboard, we intially AddRef the Doc ref cnt and later
   **    Release it. This initial AddRef is artificial; it is simply
   **    done to guarantee that a harmless QueryInterface followed by
   **    a Release does not inadvertantly force our object to destroy
   **    itself prematurely.
   */
   OleDoc_AddRef((LPOLEDOC)lpClipboardDoc);

   /* OLE2NOTE: the OLE 2.0 style to put data onto the clipboard is to
   **    give the clipboard a pointer to an IDataObject interface that
   **    is able to statisfy IDataObject::GetData calls to render
   **    data. in our case we give the pointer to the ClipboardDoc
   **    which holds a cloned copy of the current user's selection.
   */
   OLEDBG_BEGIN2("OleSetClipboard called\r\n")
   OleSetClipboard((LPDATAOBJECT)&((LPOLEDOC)lpClipboardDoc)->m_DataObject);
   OLEDBG_END2

   OleDoc_Release((LPOLEDOC)lpClipboardDoc);   // rel artificial AddRef
above
}


/* OleDoc_PasteCommand
** -------------------
**    Paste default format data from the clipboard.
**    In this function we choose the highest fidelity format that the
**    source clipboard IDataObject* offers that we understand.
**
**    OLE2NOTE: clipboard handling in an OLE 2.0 application is
**    different than normal Windows clipboard handling. Data from the
**    clipboard is retieved by getting the IDataObject* pointer
**    returned by calling OleGetClipboard.
```

```
*/
void OleDoc_PasteCommand(LPOLEDOC lpOleDoc)
{
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpOleDoc;
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPDATAOBJECT lpClipboardDataObj = NULL;
   BOOL fLink = FALSE;
   BOOL fLocalDataObj = FALSE;
   BOOL fStatus;
   HRESULT hrErr;

   hrErr = OleGetClipboard((LPDATAOBJECT FAR*)&lpClipboardDataObj);
   if (hrErr != NOERROR)
      return;     // Clipboard seems to be empty or can't be accessed

   OutlineDoc_SetRedraw ( lpOutlineDoc, FALSE );

   /* check if the data on the clipboard is local to our application
   **    instance.
   */
   if (lpOutlineApp->m_lpClipboardDoc) {
      LPOLEDOC lpOleDoc = (LPOLEDOC)lpOutlineApp->m_lpClipboardDoc;
      if (lpClipboardDataObj == (LPDATAOBJECT)&lpOleDoc->m_DataObject)
         fLocalDataObj = TRUE;
   }

   fStatus = OleDoc_PasteFromData(
         lpOleDoc,
         lpClipboardDataObj,
         fLocalDataObj,
         fLink
   );

   OutlineDoc_SetRedraw ( lpOutlineDoc, TRUE );

   if (! fStatus)
      OutlineApp_ErrorMessage(g_lpApp,OLESTR("Could not paste data from
clipboard!"));

   if (lpClipboardDataObj)
      OleStdRelease((LPUNKNOWN)lpClipboardDataObj);
}


/* OleDoc_PasteSpecialCommand
** --------------------------
**    Allow the user to paste data in a particular format from the
**    clipboard. The paste special command displays a dialog to the
**    user that allows him to choose the format to be pasted from the
**    list of formats available.
**
**    OLE2NOTE: the PasteSpecial dialog is one of the standard OLE 2.0
**    UI dialogs for which the dialog is implemented and in the OLE2UI
**    library.
**
```

```
**    OLE2NOTE: clipboard handling in an OLE 2.0 application is
**    different than normal Windows clipboard handling. Data from the
**    clipboard is retieved by getting the IDataObject* pointer
**    returned by calling OleGetClipboard.
*/
void OleDoc_PasteSpecialCommand(LPOLEDOC lpOleDoc)
{
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpOleDoc;
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;
   LPDATAOBJECT lpClipboardDataObj = NULL;
   CLIPFORMAT cfFormat;
   int nFmtEtc;
   UINT uInt;
   BOOL fLink = FALSE;
   BOOL fLocalDataObj = FALSE;
   BOOL fStatus;
   HRESULT hrErr;
   OLEUIPASTESPECIAL ouiPasteSpl;
   BOOL fDisplayAsIcon;

   hrErr = OleGetClipboard((LPDATAOBJECT FAR*)&lpClipboardDataObj);
   if (hrErr != NOERROR)
      return;     // Clipboard seems to be empty or can't be accessed

   /* check if the data on the clipboard is local to our application
   **    instance.
   */
   if (lpOutlineApp->m_lpClipboardDoc) {
      LPOLEDOC lpOleDoc = (LPOLEDOC)lpOutlineApp->m_lpClipboardDoc;
      if (lpClipboardDataObj == (LPDATAOBJECT)&lpOleDoc->m_DataObject)
         fLocalDataObj = TRUE;
   }

   /* Display the PasteSpecial dialog and allow the user to select the
   **    format to paste.
   */
   _fmemset((LPOLEUIPASTESPECIAL)&ouiPasteSpl, 0,
sizeof(OLEUIPASTESPECIAL));
   ouiPasteSpl.cbStruct = sizeof(OLEUIPASTESPECIAL);       //Structure Size
   ouiPasteSpl.dwFlags =  PSF_SELECTPASTE | PSF_SHOWHELP;  //IN-OUT:  Flags
   ouiPasteSpl.hWndOwner = lpOutlineApp->m_lpDoc->m_hWndDoc; //Owning window
   ouiPasteSpl.lpszCaption = OLESTR("Paste Special");    //Dialog caption
bar contents
   ouiPasteSpl.lpfnHook = NULL;       //Hook callback
   ouiPasteSpl.lCustData = 0;         //Custom data to pass to hook
   ouiPasteSpl.hInstance = NULL;      //Instance for customized template
name
   ouiPasteSpl.lpszTemplate = NULL;   //Customized template name
   ouiPasteSpl.hResource = NULL;      //Customized template handle

   ouiPasteSpl.arrPasteEntries = lpOleApp->m_arrPasteEntries;
   ouiPasteSpl.cPasteEntries = lpOleApp->m_nPasteEntries;
   ouiPasteSpl.lpSrcDataObj = lpClipboardDataObj;
   ouiPasteSpl.arrLinkTypes = lpOleApp->m_arrLinkTypes;
   ouiPasteSpl.cLinkTypes = lpOleApp->m_nLinkTypes;
   ouiPasteSpl.cClsidExclude = 0;

   OLEDBG_BEGIN3("OleUIPasteSpecial called\r\n")
   uInt = OleUIPasteSpecial(&ouiPasteSpl);
   OLEDBG_END3

   fDisplayAsIcon =
         (ouiPasteSpl.dwFlags & PSF_CHECKDISPLAYASICON ? TRUE : FALSE);

   if (uInt == OLEUI_OK) {
      nFmtEtc = ouiPasteSpl.nSelectedIndex;
      fLink =  ouiPasteSpl.fLink;

      if (nFmtEtc < 0 || nFmtEtc >= lpOleApp->m_nPasteEntries) {
         OutlineApp_ErrorMessage(lpOutlineApp, ErrMsgBadFmt);
         goto error;
      }

      OutlineDoc_SetRedraw ( lpOutlineDoc, FALSE );

      cfFormat = lpOleApp->m_arrPasteEntries[nFmtEtc].fmtetc.cfFormat;

      fStatus = OleDoc_PasteFormatFromData(
            lpOleDoc,
            cfFormat,
            lpClipboardDataObj,
            fLocalDataObj,
            fLink,
            fDisplayAsIcon,
            ouiPasteSpl.hMetaPict,
            (LPSIZEL)&ouiPasteSpl.sizel
      );

      OutlineDoc_SetRedraw ( lpOutlineDoc, TRUE );

      if (! fStatus) {
         OutlineApp_ErrorMessage(lpOutlineApp, ErrMsgPasteFailed);
         goto error;
      }

   } else if (uInt == OLEUI_PSERR_CLIPBOARDCHANGED) {
      /* OLE2NOTE: this error code is returned when the contents of
      **    the clipboard change while the PasteSpecial dialog is up.
      **    in this situation the PasteSpecial dialog automatically
      **    brings itself down and NO paste operation should be performed.
      */
      OutlineApp_ErrorMessage(lpOutlineApp, ErrMsgClipboardChanged);
   }

error:

   if (lpClipboardDataObj)
      OleStdRelease((LPUNKNOWN)lpClipboardDataObj);
   if (uInt == OLEUI_OK && ouiPasteSpl.hMetaPict)
      // clean up metafile
      OleUIMetafilePictIconFree(ouiPasteSpl.hMetaPict);
}



/* OleDoc_CreateDataTransferDoc
 * ----------------------------
 *
 *      Create a document to be use to transfer data (either via a
 *  drag/drop operation of the clipboard). Copy the selection of the
 *  source doc to the data transfer document. A data transfer document is
 *  the same as a document that is created by the user except that it is
 *  NOT made visible to the user. it is specially used to hold a copy of
 *  data that the user should not be able to change.
 *
 *  OLE2NOTE: in the OLE version the data transfer document is used
 *      specifically to provide an IDataObject* that renders the data
copied.
 */
LPOUTLINEDOC OleDoc_CreateDataTransferDoc(LPOLEDOC lpSrcOleDoc)
{
   LPOUTLINEDOC lpSrcOutlineDoc = (LPOUTLINEDOC)lpSrcOleDoc;
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPOUTLINEDOC lpDestOutlineDoc;
   LPLINELIST lpSrcLL = &lpSrcOutlineDoc->m_LineList;
   LINERANGE lrSel;
   int nCopied;

   lpDestOutlineDoc = OutlineApp_CreateDoc(lpOutlineApp, TRUE);
   if (! lpDestOutlineDoc) return NULL;

   // set the ClipboardDoc to an (Untitled) doc.
   if (! OutlineDoc_InitNewFile(lpDestOutlineDoc))
      goto error;

   LineList_GetSel(lpSrcLL, (LPLINERANGE)&lrSel);
   nCopied = LineList_CopySelToDoc(
         lpSrcLL,
         (LPLINERANGE)&lrSel,
         lpDestOutlineDoc
   );

   if (nCopied != (lrSel.m_nEndLine - lrSel.m_nStartLine + 1)) {
      OleDbgAssertSz(FALSE,"OleDoc_CreateDataTransferDoc: entire selection
NOT copied\r\n");
      goto error;     // ERROR: all lines could NOT be copied
   }

#if defined( OLE_SERVER )
   {
      LPOLEDOC lpSrcOleDoc = (LPOLEDOC)lpSrcOutlineDoc;
      LPOLEDOC lpDestOleDoc = (LPOLEDOC)lpDestOutlineDoc;
      LPSERVERDOC lpDestServerDoc = (LPSERVERDOC)lpDestOutlineDoc;
      LPMONIKER lpmkDoc = NULL;
      LPMONIKER lpmkItem = NULL;

      /* If source document is able to provide a moniker, then the
      **    destination document (lpDestOutlineDoc) should offer
      **    CF_LINKSOURCE via its IDataObject interface that it gives
      **    to the clipboard or the drag/drop operation.
      **
      **    OLE2NOTE: we want to ask the source document if it can
      **    produce a moniker, but we do NOT want to FORCE moniker
      **    assignment at this point. we only want to FORCE moniker
      **    assignment later if a Paste Link occurs (ie. GetData for
      **    CF_LINKSOURCE). if the source document is able to give
      **    a moniker, then we store a pointer to the source document
      **    so we can ask it at a later time to get the moniker. we
      **    also save the range of the current selection so we can
      **    generate a proper item name later when Paste Link occurs.
      **    Also we need to give a string which identifies the source
      **    of the copy in the CF_OBJECTDESCRIPTOR format. this
      **    string is used to display in the PasteSpecial dialog. we
      **    get and store a TEMPFORUSER moniker which identifies the
      **    source of copy.
      */
      lpDestOleDoc->m_lpSrcDocOfCopy = lpSrcOleDoc;
      lpmkDoc = OleDoc_GetFullMoniker(lpSrcOleDoc, GETMONIKER_TEMPFORUSER);
      if (lpmkDoc != NULL) {
         lpDestOleDoc->m_fLinkSourceAvail = TRUE;
         lpDestServerDoc->m_lrSrcSelOfCopy = lrSel;
         OleStdRelease((LPUNKNOWN)lpmkDoc);
      }
   }
#endif
#if defined( OLE_CNTR )
   {
      LPOLEDOC lpSrcOleDoc = (LPOLEDOC)lpSrcOutlineDoc;
      LPOLEDOC lpDestOleDoc = (LPOLEDOC)lpDestOutlineDoc;
      LPCONTAINERDOC lpDestContainerDoc = (LPCONTAINERDOC)lpDestOutlineDoc;

      /* If one line was copied from the source document, and it was a
      **    single OLE object, then the destination document should
      **    offer additional data formats to allow the transfer of
      **    the OLE object via IDataObject::GetData. Specifically, the
      **    following additional data formats are offered if a single
      **    OLE object is copied:
      **          CF_EMBEDDEDOBJECT
      **          CF_OBJECTDESCRIPTOR     (should be given even w/o object)
      **          CF_METAFILEPICT         (note: dwAspect depends on object)
      **          CF_LINKSOURCE           -- if linking is possible
      **          CF_LINKSOURCEDESCRIPTOR -- if linking is possible
      **
      **    optionally the container may give
      **          <data format available in OLE object's cache>
      */

      if (nCopied == 1) {
```

```
LPOLEOBJECT lpSrcOleObj;
LPCONTAINERLINE lpSrcContainerLine;
DWORD dwStatus;

lpSrcContainerLine = (LPCONTAINERLINE)LineList_GetLine(
      lpSrcLL,
      lrSel.m_nStartLine
);

if (! lpSrcContainerLine)
   goto error;

lpDestOleDoc->m_lpSrcDocOfCopy = lpSrcOleDoc;

if ((((LPLINE)lpSrcContainerLine)->m_lineType==CONTAINERLINETYPE)
      && ((lpSrcOleObj=lpSrcContainerLine->m_lpOleObj)!=NULL)) {

   lpDestContainerDoc->m_fEmbeddedObjectAvail = TRUE;
   lpSrcOleObj->lpVtbl->GetUserClassID(
         lpSrcOleObj,
         &lpDestContainerDoc->m_clsidOleObjCopied
   );
   lpDestContainerDoc->m_dwAspectOleObjCopied =
            lpSrcContainerLine->m_dwDrawAspect;

   /* OLE2NOTE: if the object is allowed to be linked
   **    to from the inside (ie. we are allowed to
   **    give out a moniker which binds to the running
   **    OLE object), then we want to offer
   **    CF_LINKSOURCE format. if the object is an OLE
   **    2.0 embedded object then it is allowed to be
   **    linked to from the inside. if the object is
   **    either an OleLink or an OLE 1.0 embedding
   **    then it can not be linked to from the inside.
   **    if we were a container/server app then we
   **    could offer linking to the outside of the
   **    object (ie. a pseudo object within our
   **    document). we are a container only app that
   **    does not support linking to ranges of its data.
   */

   lpSrcOleObj->lpVtbl->GetMiscStatus(
         lpSrcOleObj,
         DVASPECT_CONTENT, /* aspect is not important */
         (LPDWORD)&dwStatus
   );
   if (! (dwStatus & OLEMISC_CANTLINKINSIDE)) {
      /* Our container supports linking to an embedded
      **    object. We want the lpDestContainerDoc to
      **    offer CF_LINKSOURCE via the IDataObject
      **    interface that it gives to the clipboard or
      **    the drag/drop operation. The link source will
      **    be identified by a composite moniker
      **    comprised of the FileMoniker of the source
      **    document and an ItemMoniker which identifies
```

```
                **    the OLE object inside the container. we do
                **    NOT want to force moniker assignment to the
                **    OLE object now (at copy time); we only want
                **    to FORCE moniker assignment later if a Paste
                **    Link occurs (ie. GetData for CF_LINKSOURCE).
                **    thus we store a pointer to the source document
                **    and the source ContainerLine so we can
                **    generate a proper ItemMoniker later when
                **    Paste Link occurs.
                */
                lpDestOleDoc->m_fLinkSourceAvail = TRUE;
                lpDestContainerDoc->m_lpSrcContainerLine =
                      lpSrcContainerLine;
             }
          }
       }
    }

#endif  // OLE_CNTR

   return lpDestOutlineDoc;

error:
   if (lpDestOutlineDoc)
      OutlineDoc_Destroy(lpDestOutlineDoc);

   return NULL;
}


/* OleDoc_PasteFromData
** --------------------
**
**    Paste data from an IDataObject*. The IDataObject* may come from
**    the clipboard (GetClipboard) or from a drag/drop operation.
**    In this function we choose the best format that we prefer.
**
**    Returns TRUE if data was successfully pasted.
**            FALSE if data could not be pasted.
*/

BOOL OleDoc_PasteFromData(
      LPOLEDOC            lpOleDoc,
      LPDATAOBJECT        lpSrcDataObj,
      BOOL                fLocalDataObj,
      BOOL                fLink
)
{
   LPOLEAPP        lpOleApp = (LPOLEAPP)g_lpApp;
   CLIPFORMAT      cfFormat;
   BOOL            fDisplayAsIcon = FALSE;
   SIZEL           sizelInSrc = {0, 0};
   HGLOBAL         hMem = NULL;
   HGLOBAL         hMetaPict = NULL;
   STGMEDIUM       medium;
   if (fLink) {
#if defined( OLE_SERVER )
      return FALSE;       // server version of app does NOT support links
#endif
#if defined( OLE_CNTR )
      // container version of app only supports OLE object type links
      cfFormat = lpOleApp->m_cfLinkSource;
#endif

   } else {

      int nFmtEtc;

      nFmtEtc = OleStdGetPriorityClipboardFormat(
            lpSrcDataObj,
            lpOleApp->m_arrPasteEntries,
            lpOleApp->m_nPasteEntries
      );

      if (nFmtEtc < 0)
         return FALSE;   // there is no format we like

      cfFormat = lpOleApp->m_arrPasteEntries[nFmtEtc].fmtetc.cfFormat;
   }

   /* OLE2NOTE: we need to check what dwDrawAspect is being
   **    transfered. if the data is an object that is displayed as an
   **    icon in the source, then we want to keep it as an icon. the
   **    aspect the object is displayed in at the source is transfered
   **    via the CF_OBJECTDESCRIPTOR format for a Paste operation.
   */
   if (hMem = OleStdGetData(
         lpSrcDataObj,
         (CLIPFORMAT)lpOleApp->m_cfObjectDescriptor,
         NULL,
         DVASPECT_CONTENT,
         (LPSTGMEDIUM)&medium)) {
      LPOBJECTDESCRIPTOR lpOD = GlobalLock(hMem);
      fDisplayAsIcon = (lpOD->dwDrawAspect == DVASPECT_ICON ? TRUE : FALSE);
      sizelInSrc = lpOD->sizel;   // size of object/picture in source (opt.)
      GlobalUnlock(hMem);
      ReleaseStgMedium((LPSTGMEDIUM)&medium);     // equiv to GlobalFree

      if (fDisplayAsIcon) {
         hMetaPict = OleStdGetData(
               lpSrcDataObj,
               CF_METAFILEPICT,
               NULL,
               DVASPECT_ICON,
               (LPSTGMEDIUM)&medium
         );
         if (hMetaPict == NULL)
            fDisplayAsIcon = FALSE; // give up; failed to get icon MFP
      }
   }

   return OleDoc_PasteFormatFromData(
         lpOleDoc,
         cfFormat,
         lpSrcDataObj,
         fLocalDataObj,
         fLink,
         fDisplayAsIcon,
         hMetaPict,
         (LPSIZEL)&sizelInSrc
   );

   if (hMetaPict)
      ReleaseStgMedium((LPSTGMEDIUM)&medium);  // properly free METAFILEPICT
}


/* OleDoc_PasteFormatFromData
** --------------------------
**
**    Paste a particular data format from a IDataObject*. The
**    IDataObject* may come from the clipboard (GetClipboard) or from a
**    drag/drop operation.
**
**    Returns TRUE if data was successfully pasted.
**            FALSE if data could not be pasted.
*/

BOOL OleDoc_PasteFormatFromData(
      LPOLEDOC            lpOleDoc,
      CLIPFORMAT          cfFormat,
      LPDATAOBJECT        lpSrcDataObj,
      BOOL                fLocalDataObj,
      BOOL                fLink,
      BOOL                fDisplayAsIcon,
      HGLOBAL             hMetaPict,
      LPSIZEL             lpSizelInSrc
)
{
#if defined( OLE_SERVER )
   /* call server specific version of the function. */
   return ServerDoc_PasteFormatFromData(
         (LPSERVERDOC)lpOleDoc,
         cfFormat,
         lpSrcDataObj,
         fLocalDataObj,
         fLink
   );
#endif
#if defined( OLE_CNTR )

   /* call container specific version of the function. */
   return ContainerDoc_PasteFormatFromData(
         (LPCONTAINERDOC)lpOleDoc,
         cfFormat,
         lpSrcDataObj,
         fLocalDataObj,
         fLink,
         fDisplayAsIcon,
         hMetaPict,
         lpSizelInSrc
   );
#endif
}


/* OleDoc_QueryPasteFromData
** -------------------------
**
**    Check if the IDataObject* offers data in a format that we can
**    paste. The IDataObject* may come from the clipboard
**    (GetClipboard) or from a drag/drop operation.
**
**    Returns TRUE if paste can be performed
**            FALSE if paste is not possible.
*/

BOOL OleDoc_QueryPasteFromData(
      LPOLEDOC            lpOleDoc,
      LPDATAOBJECT        lpSrcDataObj,
      BOOL                fLink
)
{
#if defined( OLE_SERVER )
   return ServerDoc_QueryPasteFromData(
         (LPSERVERDOC) lpOleDoc,
         lpSrcDataObj,
         fLink
   );
#endif
#if defined( OLE_CNTR )

   return ContainerDoc_QueryPasteFromData(
         (LPCONTAINERDOC) lpOleDoc,
         lpSrcDataObj,
         fLink
   );
#endif
}


/* OleDoc_GetExtent
 * ----------------
 *
 *      Get the extent (width, height) of the entire document in Himetric.
 */
void OleDoc_GetExtent(LPOLEDOC lpOleDoc, LPSIZEL lpsizel)
{
   LPLINELIST lpLL = (LPLINELIST)&((LPOUTLINEDOC)lpOleDoc)->m_LineList;
   LineList_CalcSelExtentInHimetric(lpLL, NULL, lpsizel);
}


/* OleDoc_GetObjectDescriptorData
 * ------------------------------
 *
 * Return a handle to an object's data in CF_OBJECTDESCRIPTOR form
 *
 */
HGLOBAL OleDoc_GetObjectDescriptorData(LPOLEDOC lpOleDoc, LPLINERANGE
lplrSel)
{
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpOleDoc;

   /* Only our data transfer doc renders CF_OBJECTDESCRIPTOR */
   OleDbgAssert(lpOutlineDoc->m_fDataTransferDoc);

#if defined( OLE_SERVER )
   {
      LPSERVERDOC   lpServerDoc = (LPSERVERDOC)lpOleDoc;
      SIZEL         sizel;
      POINTL        pointl;
      LPOLESTR      lpszSrcOfCopy = NULL;
      IBindCtx  FAR *pbc = NULL;
      HGLOBAL       hObjDesc;
      DWORD         dwStatus = 0;
      LPOUTLINEDOC  lpSrcDocOfCopy=(LPOUTLINEDOC)lpOleDoc->m_lpSrcDocOfCopy;
      LPMONIKER lpSrcMonikerOfCopy = ServerDoc_GetSelFullMoniker(
            (LPSERVERDOC)lpOleDoc->m_lpSrcDocOfCopy,
            &lpServerDoc->m_lrSrcSelOfCopy,
            GETMONIKER_TEMPFORUSER
      );

      SvrDoc_OleObj_GetMiscStatus(
            (LPOLEOBJECT)&lpServerDoc->m_OleObject,
            DVASPECT_CONTENT,
            &dwStatus
      );

      OleDoc_GetExtent(lpOleDoc, &sizel);
      pointl.x = pointl.y = 0;

      if (lpSrcMonikerOfCopy) {
         CreateBindCtx(0, (LPBC FAR*)&pbc);
         lpSrcMonikerOfCopy->lpVtbl->GetDisplayName(
            lpSrcMonikerOfCopy, pbc, NULL, &lpszSrcOfCopy);
         pbc->lpVtbl->Release(pbc);
         lpSrcMonikerOfCopy->lpVtbl->Release(lpSrcMonikerOfCopy);
      } else {
         /* this document has no moniker; use our FullUserTypeName
         **    as the description of the source of copy.
         */
         lpszSrcOfCopy = FULLUSERTYPENAME;
```

```
            }

            hObjDesc =  OleStdGetObjectDescriptorData(
                  CLSID_APP,
                  DVASPECT_CONTENT,
                  sizel,
                  pointl,
                  dwStatus,
                  FULLUSERTYPENAME,
                  lpszSrcOfCopy
            );

            if (lpSrcMonikerOfCopy && lpszSrcOfCopy)
               OleStdFreeString(lpszSrcOfCopy, NULL);
            return hObjDesc;

         }
      #endif
      #if defined( OLE_CNTR )
         {
            LPCONTAINERDOC lpContainerDoc = (LPCONTAINERDOC)lpOleDoc;
            LPLINELIST lpLL = (LPLINELIST)&((LPOUTLINEDOC)lpOleDoc)->m_LineList;
            LPCONTAINERLINE lpContainerLine;
            HGLOBAL hObjDesc;
            BOOL fSelIsOleObject = FALSE;
            LPOLEOBJECT lpOleObj;
            SIZEL sizel;
            POINTL pointl;

            if ( lpLL->m_nNumLines == 1 ) {
               fSelIsOleObject = ContainerDoc_IsSelAnOleObject(
                     lpContainerDoc,
                     &IID_IOleObject,
                     (LPUNKNOWN FAR*)&lpOleObj,
                     NULL,    /* we don't need the line index */
                     (LPCONTAINERLINE FAR*)&lpContainerLine
               );
            }

            pointl.x = pointl.y = 0;

            if (fSelIsOleObject) {
               /* OLE2NOTE: a single OLE object is being transfered via
               **    this DataTransferDoc. we need to generate the
               **    CF_ObjectDescrioptor which describes the OLE object.
               */

               LPOUTLINEDOC lpSrcOutlineDoc =
                     (LPOUTLINEDOC)lpOleDoc->m_lpSrcDocOfCopy;
               LPOLESTR lpszSrcOfCopy = lpSrcOutlineDoc->m_szFileName;
               BOOL fFreeSrcOfCopy = FALSE;
               SIZEL sizelOleObject;
               LPLINE lpLine = (LPLINE)lpContainerLine;

               /* if the object copied can be linked to then get a
               **    TEMPFORUSER form of the moniker which identifies the
               **    source of copy. we do not want to force the
               **    assignment of the moniker until CF_LINKSOURCE is
               **    rendered.
               **    if the object copied can not be a link source then use
               **    the source filename to identify the source of copy.
               **    there is no need to generate a moniker for the object
               **    copied.
               */
               if (lpOleDoc->m_fLinkSourceAvail &&
                     lpContainerDoc->m_lpSrcContainerLine) {
                  LPBINDCTX pbc = NULL;
                  LPMONIKER lpSrcMonikerOfCopy = ContainerLine_GetFullMoniker(
                        lpContainerDoc->m_lpSrcContainerLine,
                        GETMONIKER_TEMPFORUSER
                  );
                  if (lpSrcMonikerOfCopy) {
                     CreateBindCtx(0, (LPBC FAR*)&pbc);
                     if (pbc != NULL) {
                        lpSrcMonikerOfCopy->lpVtbl->GetDisplayName(
                              lpSrcMonikerOfCopy, pbc, NULL,&lpszSrcOfCopy);
                        pbc->lpVtbl->Release(pbc);
                        fFreeSrcOfCopy = TRUE;
                     }
                     lpSrcMonikerOfCopy->lpVtbl->Release(lpSrcMonikerOfCopy);
                  }
               }

               /* OLE2NOTE: Get size that object is being drawn. If the
               **    object has been scaled because the user resized the
               **    object, then we want to pass the scaled size of the
               **    object in the ObjectDescriptor rather than the size
               **    that the object would return via
               **    IOleObject::GetExtent and IViewObject2::GetExtent. in
               **    this way if the object is transfered to another container
               **    (via clipboard or drag/drop), then the object will
               **    remain the scaled size.
               */
               sizelOleObject.cx = lpLine->m_nWidthInHimetric;
               sizelOleObject.cy = lpLine->m_nHeightInHimetric;

               hObjDesc = OleStdGetObjectDescriptorDataFromOleObject(
                     lpOleObj,
                     lpszSrcOfCopy,
                     lpContainerLine->m_dwDrawAspect,
                     pointl,
                     (LPSIZEL)&sizelOleObject
               );

               if (fFreeSrcOfCopy && lpszSrcOfCopy)
                  OleStdFreeString(lpszSrcOfCopy, NULL);
               OleStdRelease((LPUNKNOWN)lpOleObj);
               return hObjDesc;
            } else {
               /* OLE2NOTE: the data being transfered via this
```

```
        **    DataTransferDoc is NOT a single OLE object. thus in
        **    this case the CF_ObjectDescriptor data should
        **    describe our container app itself.
        */
        OleDoc_GetExtent(lpOleDoc, &sizel);
        return OleStdGetObjectDescriptorData(
              CLSID_NULL, /* not used if no object formats */
              DVASPECT_CONTENT,
              sizel,
              pointl,
              0,
              NULL,       /* UserTypeName not used if no obj fmt's */
              FULLUSERTYPENAME   /* string to identify source of copy */
        );

      }
   }
#endif  // OLE_CNTR
}


#if defined( OLE_SERVER )

/*************************************************************************
** ServerDoc Supprt Functions Used by Server versions
*************************************************************************/


/* ServerDoc_PasteFormatFromData
** ----------------------------
**
**    Paste a particular data format from a IDataObject*. The
**    IDataObject* may come from the clipboard (GetClipboard) or from a
**    drag/drop operation.
**
**    NOTE: fLink is specified then FALSE if returned because the
**    Server only version of the app can not support links.
**
**    Returns TRUE if data was successfully pasted.
**            FALSE if data could not be pasted.
*/
BOOL ServerDoc_PasteFormatFromData(
      LPSERVERDOC            lpServerDoc,
      CLIPFORMAT             cfFormat,
      LPDATAOBJECT           lpSrcDataObj,
      BOOL                   fLocalDataObj,
      BOOL                   fLink
)
{
   LPLINELIST   lpLL = (LPLINELIST)&((LPOUTLINEDOC)lpServerDoc)->m_LineList;
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPOLEAPP     lpOleApp = (LPOLEAPP)g_lpApp;
   int          nIndex;
   int          nCount = 0;
   HGLOBAL      hData;
```

```
STGMEDIUM    medium;
LINERANGE    lrSel;

if (LineList_GetCount(lpLL) == 0)
   nIndex = -1;    // pasting to empty list
else
   nIndex=LineList_GetFocusLineIndex(lpLL);

if (fLink) {
   /* We should paste a Link to the data, but we do not support links */
   return FALSE;

} else {

   if (cfFormat == lpOutlineApp->m_cfOutline) {

      hData = OleStdGetData(
            lpSrcDataObj,
            (CLIPFORMAT)lpOutlineApp->m_cfOutline,
            NULL,
            DVASPECT_CONTENT,
            (LPSTGMEDIUM)&medium
      );
      if (hData == NULL)
         return FALSE;

      nCount = OutlineDoc_PasteOutlineData(
            (LPOUTLINEDOC)lpServerDoc,
            hData,
            nIndex
      );
      // OLE2NOTE: we must free data handle by releasing the medium
      ReleaseStgMedium((LPSTGMEDIUM)&medium);

   } else if(cfFormat == CF_TEXT) {

      hData = OleStdGetData(
            lpSrcDataObj,
            CF_TEXT,
            NULL,
            DVASPECT_CONTENT,
            (LPSTGMEDIUM)&medium
      );
      if (hData == NULL)
         return FALSE;

      nCount = OutlineDoc_PasteTextData(
            (LPOUTLINEDOC)lpServerDoc,
            hData,
            nIndex
      );
      // OLE2NOTE: we must free data handle by releasing the medium
      ReleaseStgMedium((LPSTGMEDIUM)&medium);
   }
}
```

```
   lrSel.m_nEndLine   = nIndex + 1;
   lrSel.m_nStartLine = nIndex + nCount;
   LineList_SetSel(lpLL, &lrSel);
   return TRUE;
}


/* ServerDoc_QueryPasteFromData
** ----------------------------
**
**    Check if the IDataObject* offers data in a format that we can
**    paste. The IDataObject* may come from the clipboard
**    (GetClipboard) or from a drag/drop operation.
**    In this function we look if one of the following formats is
**    offered:
**              CF_OUTLINE
**              CF_TEXT
**
**    NOTE: fLink is specified then FALSE if returned because the
**    Server only version of the app can not support links.
**
**    Returns TRUE if paste can be performed
**            FALSE if paste is not possible.
*/
BOOL ServerDoc_QueryPasteFromData(
      LPSERVERDOC             lpServerDoc,
      LPDATAOBJECT            lpSrcDataObj,
      BOOL                    fLink
)
{
   LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;

   if (fLink) {
      /* we do not support links */
      return FALSE;

   } else {

      int nFmtEtc;

      nFmtEtc = OleStdGetPriorityClipboardFormat(
            lpSrcDataObj,
            lpOleApp->m_arrPasteEntries,
            lpOleApp->m_nPasteEntries
         );

      if (nFmtEtc < 0)
         return FALSE;   // there is no format we like
   }

   return TRUE;
}
```

```
/* ServerDoc_GetData
 * -----------------
 *
 * Render data from the document on a CALLEE allocated STGMEDIUM.
 *      This routine is called via IDataObject::GetData.
 */

HRESULT ServerDoc_GetData (
      LPSERVERDOC             lpServerDoc,
      LPFORMATETC             lpformatetc,
      LPSTGMEDIUM             lpMedium
)
{
   LPOLEDOC  lpOleDoc = (LPOLEDOC)lpServerDoc;
   LPOUTLINEDOC  lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPSERVERAPP lpServerApp = (LPSERVERAPP)g_lpApp;
   LPOLEAPP  lpOleApp = (LPOLEAPP)lpServerApp;
   LPOUTLINEAPP  lpOutlineApp = (LPOUTLINEAPP)lpServerApp;
   HRESULT hrErr;
   SCODE sc;

   // OLE2NOTE: we must set out pointer parameters to NULL
   lpMedium->pUnkForRelease = NULL;

   /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
   lpMedium->tymed = TYMED_NULL;
   lpMedium->pUnkForRelease = NULL;    // we transfer ownership to caller
   lpMedium->u.hGlobal = NULL;

   if(lpformatetc->cfFormat == lpOutlineApp->m_cfOutline) {
      // Verify caller asked for correct medium
      if (!(lpformatetc->tymed & TYMED_HGLOBAL)) {
         sc = DV_E_FORMATETC;
         goto error;
      }
      lpMedium->u.hGlobal = OutlineDoc_GetOutlineData (lpOutlineDoc,NULL);
      if (! lpMedium->u.hGlobal) {
         sc = E_OUTOFMEMORY;
         goto error;
      }

      lpMedium->tymed = TYMED_HGLOBAL;
      OleDbgOut3("ServerDoc_GetData: rendered CF_OUTLINE\r\n");
      return NOERROR;

   } else if (lpformatetc->cfFormat == CF_METAFILEPICT &&
      (lpformatetc->dwAspect & DVASPECT_CONTENT) ) {
      // Verify caller asked for correct medium
      if (!(lpformatetc->tymed & TYMED_MFPICT)) {
         sc = DV_E_FORMATETC;
         goto error;
      }

      lpMedium->u.hGlobal = ServerDoc_GetMetafilePictData(lpServerDoc,NULL);
      if (! lpMedium->u.hGlobal) {
         sc = E_OUTOFMEMORY;
         goto error;
      }

      lpMedium->tymed = TYMED_MFPICT;
      OleDbgOut3("ServerDoc_GetData: rendered CF_METAFILEPICT\r\n");
      return NOERROR;

   } else if (lpformatetc->cfFormat == CF_METAFILEPICT &&
      (lpformatetc->dwAspect & DVASPECT_ICON) ) {
      CLSID clsid;
      // Verify caller asked for correct medium
      if (!(lpformatetc->tymed & TYMED_MFPICT)) {
         sc = DV_E_FORMATETC;
         goto error;
      }

      /* OLE2NOTE: we should return the default icon for our class.
      **    we must be carefull to use the correct CLSID here.
      **    if we are currently preforming a "TreatAs (aka. ActivateAs)"
      **    operation then we need to use the class of the object
      **    written in the storage of the object. otherwise we would
      **    use our own class id.
      */
      if (ServerDoc_GetClassID(lpServerDoc, (LPCLSID)&clsid) != NOERROR) {
         sc = DV_E_FORMATETC;
         goto error;
      }

      lpMedium->u.hGlobal=GetIconOfClass(g_lpApp->m_hInst,(REFCLSID)&clsid,
NULL, FALSE);
      if (! lpMedium->u.hGlobal) {
         sc = E_OUTOFMEMORY;
         goto error;
      }

      lpMedium->tymed = TYMED_MFPICT;
      OleDbgOut3("ServerDoc_GetData: rendered CF_METAFILEPICT (icon)\r\n");
      return NOERROR;

   } else if (lpformatetc->cfFormat == CF_TEXT) {
      // Verify caller asked for correct medium
      if (!(lpformatetc->tymed & TYMED_HGLOBAL)) {
         sc = DV_E_FORMATETC;
         goto error;
      }

      lpMedium->u.hGlobal = OutlineDoc_GetTextData (
            (LPOUTLINEDOC)lpServerDoc,
            NULL
      );
      if (! lpMedium->u.hGlobal) {
         sc = E_OUTOFMEMORY;
         goto error;
      }
      lpMedium->tymed = TYMED_HGLOBAL;
      OleDbgOut3("ServerDoc_GetData: rendered CF_TEXT\r\n");
      return NOERROR;
   }

   /* the above are the only formats supports by a user document (ie.
   **    a non-data transfer doc). if the document is used for
   **    purposes of data transfer, then additional formats are offered.
   */
   if (! lpOutlineDoc->m_fDataTransferDoc) {
      sc = DV_E_FORMATETC;
      goto error;
   }

   /* OLE2NOTE: ObjectDescriptor and LinkSrcDescriptor will
   **    contain the same data for the pure container and pure server
   **    type applications. only a container/server application may
   **    have different content for ObjectDescriptor and
   **    LinkSrcDescriptor. if a container/server copies a link for
   **    example, then the ObjectDescriptor would give the class
   **    of the link source but the LinkSrcDescriptor would give the
   **    class of the container/server itself. in this situation if a
   **    paste operation occurs, an equivalent link is pasted, but if
   **    a pastelink operation occurs, then a link to a pseudo object
   **    in the container/server is created.
   */
   if (lpformatetc->cfFormat == lpOleApp->m_cfObjectDescriptor ||
      (lpformatetc->cfFormat == lpOleApp->m_cfLinkSrcDescriptor &&
            lpOleDoc->m_fLinkSourceAvail)) {
      // Verify caller asked for correct medium
      if (!(lpformatetc->tymed & TYMED_HGLOBAL)) {
         sc = DV_E_FORMATETC;
         goto error;
      }

      lpMedium->u.hGlobal = OleDoc_GetObjectDescriptorData (
            (LPOLEDOC)lpServerDoc,
            NULL
      );
      if (! lpMedium->u.hGlobal) {
         sc = E_OUTOFMEMORY;
         goto error;
      }

      lpMedium->tymed = TYMED_HGLOBAL;
      OleDbgOut3("ServerDoc_GetData: rendered CF_OBJECTDESCRIPTOR\r\n");
      return NOERROR;

   } else if (lpformatetc->cfFormat == lpOleApp->m_cfEmbedSource) {
      hrErr = OleStdGetOleObjectData(
            (LPPERSISTSTORAGE)&lpServerDoc->m_PersistStorage,
            lpformatetc,
            lpMedium,
            FALSE   /* fUseMemory -- (use file-base stg) */
      );
      if (hrErr != NOERROR) {
         sc = GetScode(hrErr);
         goto error;
      }
      OleDbgOut3("ServerDoc_GetData: rendered CF_EMBEDSOURCE\r\n");
      return NOERROR;

   } else if (lpformatetc->cfFormat == lpOleApp->m_cfLinkSource) {
      if (lpOleDoc->m_fLinkSourceAvail) {
         LPMONIKER lpmk;

         lpmk = ServerDoc_GetSelFullMoniker(
               (LPSERVERDOC)lpOleDoc->m_lpSrcDocOfCopy,
               &lpServerDoc->m_lrSrcSelOfCopy,
               GETMONIKER_FORCEASSIGN
         );
         if (lpmk) {
            hrErr = OleStdGetLinkSourceData(
                  lpmk,
                  (LPCLSID)&CLSID_APP,
                  lpformatetc,
                  lpMedium
            );
            OleStdRelease((LPUNKNOWN)lpmk);
            if (hrErr != NOERROR) {
               sc = GetScode(hrErr);
               goto error;
            }
            OleDbgOut3("ServerDoc_GetData: rendered CF_LINKSOURCE\r\n");
            return NOERROR;

         } else {
            sc = E_FAIL;
            goto error;
         }
      } else {
         sc = DV_E_FORMATETC;
         goto error;
      }

   } else {
      sc = DV_E_FORMATETC;
      goto error;
   }

   return NOERROR;

error:
   return ResultFromScode(sc);
}


/* ServerDoc_GetDataHere
```

```
 * ---------------------
 *
 * Render data from the document on a CALLER allocated STGMEDIUM.
 *      This routine is called via IDataObject::GetDataHere.
 */
HRESULT ServerDoc_GetDataHere (
      LPSERVERDOC             lpServerDoc,
      LPFORMATETC             lpformatetc,
      LPSTGMEDIUM             lpMedium
)
{
   LPOLEDOC        lpOleDoc = (LPOLEDOC)lpServerDoc;
   LPOUTLINEDOC    lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPSERVERAPP     lpServerApp = (LPSERVERAPP)g_lpApp;
   LPOLEAPP        lpOleApp = (LPOLEAPP)lpServerApp;
   LPOUTLINEAPP    lpOutlineApp = (LPOUTLINEAPP)lpServerApp;
   HRESULT         hrErr;
   SCODE           sc;

   // OLE2NOTE: lpMedium is an IN parameter. we should NOT set
   //           lpMedium->pUnkForRelease to NULL

   /* our user document does not support any formats for GetDataHere.
   **    if the document is used for
   **    purposes of data transfer, then additional formats are offered.
   */
   if (! lpOutlineDoc->m_fDataTransferDoc) {
      sc = DV_E_FORMATETC;
      goto error;
   }

   if (lpformatetc->cfFormat == lpOleApp->m_cfEmbedSource) {
      hrErr = OleStdGetOleObjectData(
            (LPPERSISTSTORAGE)&lpServerDoc->m_PersistStorage,
            lpformatetc,
            lpMedium,
            FALSE   /* fUseMemory -- (use file-base stg) */
      );
      if (hrErr != NOERROR) {
         sc = GetScode(hrErr);
         goto error;
      }
      OleDbgOut3("ServerDoc_GetDataHere: rendered CF_EMBEDSOURCE\r\n");
      return NOERROR;

   } else if (lpformatetc->cfFormat == lpOleApp->m_cfLinkSource) {
      if (lpOleDoc->m_fLinkSourceAvail) {
         LPMONIKER lpmk;

         lpmk = ServerDoc_GetSelFullMoniker(
               (LPSERVERDOC)lpOleDoc->m_lpSrcDocOfCopy,
               &lpServerDoc->m_lrSrcSelOfCopy,
               GETMONIKER_FORCEASSIGN
         );
         if (lpmk) {
            hrErr = OleStdGetLinkSourceData(
                  lpmk,
                  (LPCLSID)&CLSID_APP,
                  lpformatetc,
                  lpMedium
            );
            OleStdRelease((LPUNKNOWN)lpmk);
            if (hrErr != NOERROR) {
               sc = GetScode(hrErr);
               goto error;
            }

            OleDbgOut3("ServerDoc_GetDataHere: rendered CF_LINKSOURCE\r\n");
            return NOERROR;

         } else {
            sc = E_FAIL;
            goto error;
         }
      } else {
         sc = DV_E_FORMATETC;
         goto error;
      }
   } else {

      /* Caller is requesting data to be returned in Caller allocated
      **    medium, but we do NOT support this. we only support
      **    global memory blocks that WE allocate for the caller.
      */
      sc = DV_E_FORMATETC;
      goto error;
   }

   return NOERROR;

error:
   return ResultFromScode(sc);
}


/* ServerDoc_QueryGetData
 * ----------------------
 *
 * Answer if a particular data format is supported via GetData/GetDataHere.
 *      This routine is called via IDataObject::QueryGetData.
 */

HRESULT ServerDoc_QueryGetData (LPSERVERDOC lpServerDoc,LPFORMATETC
lpformatetc)
{
   LPOLEDOC        lpOleDoc = (LPOLEDOC)lpServerDoc;
   LPOUTLINEDOC    lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPSERVERAPP     lpServerApp = (LPSERVERAPP)g_lpApp;
   LPOLEAPP        lpOleApp = (LPOLEAPP)lpServerApp;
   LPOUTLINEAPP    lpOutlineApp = (LPOUTLINEAPP)lpServerApp;
   /* Caller is querying if we support certain format but does not
   **    want any data actually returned.
   */
   if (lpformatetc->cfFormat == lpOutlineApp->m_cfOutline ||
         lpformatetc->cfFormat == CF_TEXT) {
      // we only support HGLOBAL
      return OleStdQueryFormatMedium(lpformatetc, TYMED_HGLOBAL);
   } else if (lpformatetc->cfFormat == CF_METAFILEPICT &&
      (lpformatetc->dwAspect &
         (DVASPECT_CONTENT | DVASPECT_ICON)) ) {
      return OleStdQueryFormatMedium(lpformatetc, TYMED_MFPICT);
   }

   /* the above are the only formats supports by a user document (ie.
   **    a non-data transfer doc). if the document is used for
   **    purposes of data transfer, then additional formats are offered.
   */
   if (! lpOutlineDoc->m_fDataTransferDoc)
      return ResultFromScode(DV_E_FORMATETC);

   if (lpformatetc->cfFormat == lpOleApp->m_cfEmbedSource) {
      return OleStdQueryOleObjectData(lpformatetc);

   } else if (lpformatetc->cfFormat == lpOleApp->m_cfLinkSource &&
      lpOleDoc->m_fLinkSourceAvail) {
      return OleStdQueryLinkSourceData(lpformatetc);

   } else if (lpformatetc->cfFormat == lpOleApp->m_cfObjectDescriptor) {
      return OleStdQueryObjectDescriptorData(lpformatetc);

   } else if (lpformatetc->cfFormat == lpOleApp->m_cfLinkSrcDescriptor &&
            lpOleDoc->m_fLinkSourceAvail) {
      return OleStdQueryObjectDescriptorData(lpformatetc);
   }

   return ResultFromScode(DV_E_FORMATETC);
}


/* ServerDoc_EnumFormatEtc
 * -----------------------
 *
 * Return an enumerator which enumerates the data accepted/offered by
 *      the document.
 *      This routine is called via IDataObject::EnumFormatEtc.
 */
HRESULT ServerDoc_EnumFormatEtc(
      LPSERVERDOC             lpServerDoc,
      DWORD                   dwDirection,
      LPENUMFORMATETC FAR*    lplpenumFormatEtc
)
{
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpServerDoc;
   LPOLEAPP  lpOleApp = (LPOLEAPP)g_lpApp;
   int nActualFmts;
   SCODE sc = S_OK;

   /* OLE2NOTE: the enumeration of formats for a data transfer
   **    document is not a static list. the list of formats offered
   **    may or may not include CF_LINKSOURCE depending on whether a
   **    moniker is available for our document. thus we can NOT use
   **    the default OLE enumerator which enumerates the formats that
   **    are registered for our app in the registration database.
   */
   if (dwDirection == DATADIR_GET) {
      nActualFmts = lpOleApp->m_nDocGetFmts;

      /* If the document does not have a Moniker, then exclude
      **    CF_LINKSOURCE and CF_LINKSRCDESCRIPTOR from the list of
      **    formats available. these formats are deliberately listed
      **    last in the array of possible "Get" formats.
      */
      if (! lpOleDoc->m_fLinkSourceAvail)
         nActualFmts -= 2;

      *lplpenumFormatEtc = OleStdEnumFmtEtc_Create(
            nActualFmts, lpOleApp->m_arrDocGetFmts);
      if (*lplpenumFormatEtc == NULL)
         sc = E_OUTOFMEMORY;

   } else if (dwDirection == DATADIR_SET) {
      /* OLE2NOTE: a document that is used to transfer data
      **    (either via the clipboard or drag/drop does NOT
      **    accept SetData on ANY format!
      */
      sc = E_NOTIMPL;
      goto error;
   } else {
      sc = E_INVALIDARG;
      goto error;
   }

error:
   return ResultFromScode(sc);
}


/* ServerDoc_GetMetafilePictData
 * -----------------------------
 *
 * Return a handle to an object's picture data in metafile format.
 *
 *
 * RETURNS: A handle to the object's data in metafile format.
 *
 */
HGLOBAL ServerDoc_GetMetafilePictData(
      LPSERVERDOC         lpServerDoc,
      LPLINERANGE         lplrSel
```

```
)
{
   LPOUTLINEAPP    lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPOUTLINEDOC    lpOutlineDoc=(LPOUTLINEDOC)lpServerDoc;
   LPLINELIST      lpLL=(LPLINELIST)&lpOutlineDoc->m_LineList;
   LPLINE          lpLine;
   LPMETAFILEPICT  lppict = NULL;
   HGLOBAL         hMFPict = NULL;
   HMETAFILE       hMF = NULL;
   RECT            rect;
   RECT            rectWBounds;
   HDC             hDC;
   int             i;
   int             nWidth;
   int     nStart = (lplrSel ? lplrSel->m_nStartLine : 0);
   int     nEnd =(lplrSel ? lplrSel->m_nEndLine :
LineList_GetCount(lpLL)-1);
   int     nLines = nEnd - nStart + 1;
   UINT    fuAlign;
   POINT point;
   SIZE  size;

   hDC = CreateMetaFile(NULL);

   rect.left = 0;
   rect.right = 0;
   rect.bottom = 0;

   if (nLines > 0) {
   // calculate the total height/width of LineList in HIMETRIC
      for(i = nStart; i <= nEnd; i++) {
         lpLine = LineList_GetLine(lpLL,i);
         if (! lpLine)
            continue;

         nWidth = Line_GetTotalWidthInHimetric(lpLine);
         rect.right = max(rect.right, nWidth);
         rect.bottom -= Line_GetHeightInHimetric(lpLine);
      }


      SetMapMode(hDC, MM_ANISOTROPIC);

      SetWindowOrgEx(hDC, 0, 0, &point);
      SetWindowExtEx(hDC, rect.right, rect.bottom, &size);
      rectWBounds = rect;

      // Set the default font size, and font face name
      SelectObject(hDC, OutlineApp_GetActiveFont(lpOutlineApp));

      FillRect(hDC, (LPRECT) &rect, GetStockObject(WHITE_BRUSH));

      rect.bottom = 0;

      fuAlign = SetTextAlign(hDC, TA_LEFT | TA_TOP | TA_NOUPDATECP);
      /* While more lines print out the text */
      for(i = nStart; i <= nEnd; i++) {
         lpLine = LineList_GetLine(lpLL,i);
         if (! lpLine)
            continue;

         rect.top = rect.bottom;
         rect.bottom -= Line_GetHeightInHimetric(lpLine);

         /* Draw the line */
         Line_Draw(lpLine, hDC, &rect, &rectWBounds, FALSE /*fHighlight*/);
      }

      SetTextAlign(hDC, fuAlign);
   }

   // Get handle to the metafile.
   if (!(hMF = CloseMetaFile (hDC)))
      return NULL;

   if (!(hMFPict = GlobalAlloc (GALLOCFLG, sizeof (METAFILEPICT)))) {
      DeleteMetaFile (hMF);
      return NULL;
   }

   if (!(lppict = (LPMETAFILEPICT)GlobalLock(hMFPict))) {
      DeleteMetaFile (hMF);
      GlobalFree (hMFPict);
      return NULL;
   }

   lppict->mm   =  MM_ANISOTROPIC;
   lppict->hMF  =  hMF;
   lppict->xExt =  rect.right;
   lppict->yExt =  - rect.bottom;  // add minus sign to make it +ve
   GlobalUnlock (hMFPict);

   return hMFPict;
}

#endif  // OLE_SERVER



#if defined( OLE_CNTR )

/*************************************************************************
** ContainerDoc Supprt Functions Used by Container versions
*************************************************************************/


/* Paste OLE Link from clipboard */
void ContainerDoc_PasteLinkCommand(LPCONTAINERDOC lpContainerDoc)
{
```

```
LPOUTLINEAPP    lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
LPOLEAPP        lpOleApp = (LPOLEAPP)g_lpApp;
LPDATAOBJECT    lpClipboardDataObj = NULL;
BOOL            fLink = TRUE;
BOOL            fLocalDataObj = FALSE;
BOOL            fDisplayAsIcon = FALSE;
SIZEL           sizelInSrc;
HCURSOR         hPrevCursor;
HGLOBAL         hMem = NULL;
HGLOBAL         hMetaPict = NULL;
STGMEDIUM       medium;
BOOL            fStatus;
HRESULT         hrErr;

hrErr = OleGetClipboard((LPDATAOBJECT FAR*)&lpClipboardDataObj);
if (hrErr != NOERROR)
   return;     // Clipboard seems to be empty or can't be accessed

// this may take a while, put up hourglass cursor
hPrevCursor = SetCursor(LoadCursor(NULL, IDC_WAIT));

/* check if the data on the clipboard is local to our application
**    instance.
*/
if (lpOutlineApp->m_lpClipboardDoc) {
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpOutlineApp->m_lpClipboardDoc;
   if (lpClipboardDataObj == (LPDATAOBJECT)&lpOleDoc->m_DataObject)
      fLocalDataObj = TRUE;
}

/* OLE2NOTE: we need to check what dwDrawAspect is being
**    transfered. if the data is an object that is displayed as an
**    icon in the source, then we want to keep it as an icon. the
**    aspect the object is displayed in at the source is transfered
**    via the CF_LINKSOURCEDESCRIPTOR format for a PasteLink
**    operation.
*/
if (hMem = OleStdGetData(
      lpClipboardDataObj,
      (CLIPFORMAT)lpOleApp->m_cfLinkSrcDescriptor,
      NULL,
      DVASPECT_CONTENT,
      (LPSTGMEDIUM)&medium)) {
   LPOBJECTDESCRIPTOR lpOD = GlobalLock(hMem);
   fDisplayAsIcon = (lpOD->dwDrawAspect == DVASPECT_ICON ? TRUE : FALSE);
   sizelInSrc = lpOD->sizel;   // size of object/picture in source (opt.)
   GlobalUnlock(hMem);
   ReleaseStgMedium((LPSTGMEDIUM)&medium);     // equiv to GlobalFree

   if (fDisplayAsIcon) {
      hMetaPict = OleStdGetData(
            lpClipboardDataObj,
            CF_METAFILEPICT,
            NULL,
            DVASPECT_ICON,
               (LPSTGMEDIUM)&medium
         );
         if (hMetaPict == NULL)
            fDisplayAsIcon = FALSE; // give up; failed to get icon MFP
      }
   }

   fStatus = ContainerDoc_PasteFormatFromData(
         lpContainerDoc,
         (CLIPFORMAT)lpOleApp->m_cfLinkSource,
         lpClipboardDataObj,
         fLocalDataObj,
         fLink,
         fDisplayAsIcon,
         hMetaPict,
         (LPSIZEL)&sizelInSrc
   );

   if (!fStatus)
      OutlineApp_ErrorMessage(g_lpApp, ErrMsgPasting);

   if (hMetaPict)
      ReleaseStgMedium((LPSTGMEDIUM)&medium);  // properly free METAFILEPICT

   if (lpClipboardDataObj)
      OleStdRelease((LPUNKNOWN)lpClipboardDataObj);

   SetCursor(hPrevCursor);     // restore original cursor
}


/* ContainerDoc_PasteFormatFromData
** --------------------------------
**
**    Paste a particular data format from a IDataObject*. The
**    IDataObject* may come from the clipboard (GetClipboard) or from a
**    drag/drop operation.
**
**    Returns TRUE if data was successfully pasted.
**             FALSE if data could not be pasted.
*/
BOOL ContainerDoc_PasteFormatFromData(
      LPCONTAINERDOC          lpContainerDoc,
      CLIPFORMAT              cfFormat,
      LPDATAOBJECT            lpSrcDataObj,
      BOOL                    fLocalDataObj,
      BOOL                    fLink,
      BOOL                    fDisplayAsIcon,
      HGLOBAL                 hMetaPict,
      LPSIZEL                 lpSizelInSrc
)
{
   LPLINELIST lpLL = (LPLINELIST)&((LPOUTLINEDOC)lpContainerDoc)-
>m_LineList;
   LPOUTLINEAPP    lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
```

```
LPOLEAPP        lpOleApp = (LPOLEAPP)g_lpApp;
LPCONTAINERAPP  lpContainerApp = (LPCONTAINERAPP)g_lpApp;
int             nIndex;
int             nCount = 0;
HGLOBAL         hData;
STGMEDIUM       medium;
FORMATETC       formatetc;
HRESULT         hrErr;
LINERANGE       lrSel;

if (LineList_GetCount(lpLL) == 0)
   nIndex = -1;    // pasting to empty list
else
   nIndex=LineList_GetFocusLineIndex(lpLL);

if (fLink) {

   /* We should paste a Link to the data */

   if (cfFormat != lpOleApp->m_cfLinkSource)
      return FALSE;   // we only support OLE object type links

   nCount = ContainerDoc_PasteOleObject(
         lpContainerDoc,
         lpSrcDataObj,
         OLECREATEFROMDATA_LINK,
         cfFormat,
         nIndex,
         fDisplayAsIcon,
         hMetaPict,
         lpSizelInSrc
      );
   return (nCount > 0 ? TRUE : FALSE);

} else {

   if (cfFormat == lpContainerApp->m_cfCntrOutl) {
      if (fLocalDataObj) {

         /* CASE I: IDataObject* is local to our app
         **
         **    if the source of the data is local to our
         **    application instance, then we can get direct
         **    access to the original OleDoc object that
         **    corresponds to the IDataObject* given.
         **    CF_CNTROUTL data is passed through a LPSTORAGE.
         **    if we call OleGetData asking for CF_CNTROUTL, we
         **    will be returned a copy of the existing open pStg
         **    of the original source document. we can NOT open
         **    streams and sub-storages again via this pStg
         **    since it is already open within our same
         **    application instance. we must copy the data from
         **    the original OleDoc source document.
         */
         LPLINELIST lpSrcLL;
         LPOLEDOC lpLocalSrcDoc =
            ((struct CDocDataObjectImpl FAR*)lpSrcDataObj)->lpOleDoc;

         /* copy all lines from SrcDoc to DestDoc. */
         lpSrcLL = &((LPOUTLINEDOC)lpLocalSrcDoc)->m_LineList;
         nCount = LineList_CopySelToDoc(
               lpSrcLL,
               NULL,
               (LPOUTLINEDOC)lpContainerDoc
         );

      } else {

         /* CASE II: IDataObject* is NOT local to our app
         **
         **    if the source of the data comes from another
         **    application instance. we can call GetDataHere to
         **    retrieve the CF_CNTROUTL data. CF_CNTROUTL data
         **    is passed through a LPSTORAGE. we MUST use
         **    IDataObject::GetDataHere. calling
         **    IDataObject::GetData does NOT work because OLE
         **    currently does NOT support remoting of a callee
         **    allocated root storage back to the caller. this
         **    hopefully will be supported in a future version.
         **    in order to call GetDataHere we must allocate an
         **    IStorage instance for the callee to write into.
         **    we will allocate an IStorage docfile that will
         **    delete-on-release. we could use either a
         **    memory-based storage or a file-based storage.
         */
         LPSTORAGE lpTmpStg = OleStdCreateTempStorage(
               FALSE /*fUseMemory*/,
               STGM_READWRITE | STGM_TRANSACTED |STGM_SHARE_EXCLUSIVE
         );
         if (! lpTmpStg)
            return FALSE;

         formatetc.cfFormat = cfFormat;
         formatetc.ptd = NULL;
         formatetc.dwAspect = DVASPECT_CONTENT;
         formatetc.tymed = TYMED_ISTORAGE;
         formatetc.lindex = -1;

         medium.tymed = TYMED_ISTORAGE;
         medium.u.pstg = lpTmpStg;
         medium.pUnkForRelease = NULL;

         OLEDBG_BEGIN2("IDataObject::GetDataHere called\r\n")
         hrErr = lpSrcDataObj->lpVtbl->GetDataHere(
               lpSrcDataObj,
               (LPFORMATETC)&formatetc,
               (LPSTGMEDIUM)&medium
         );
         OLEDBG_END2
```

```
      if (hrErr == NOERROR) {
         nCount = ContainerDoc_PasteCntrOutlData(
               lpContainerDoc,
               lpTmpStg,
               nIndex
         );
      }
      OleStdVerifyRelease(
         (LPUNKNOWN)lpTmpStg, OLESTR("Temp stg NOT released!\r\n"));
      return ((hrErr == NOERROR) ? TRUE : FALSE);
   }

} else if (cfFormat == lpOutlineApp->m_cfOutline) {

   hData = OleStdGetData(
         lpSrcDataObj,
         (CLIPFORMAT)lpOutlineApp->m_cfOutline,
         NULL,
         DVASPECT_CONTENT,
         (LPSTGMEDIUM)&medium
   );
   nCount = OutlineDoc_PasteOutlineData(
         (LPOUTLINEDOC)lpContainerDoc,
         hData,
         nIndex
      );
   // OLE2NOTE: we must free data handle by releasing the medium
   ReleaseStgMedium((LPSTGMEDIUM)&medium);

} else if (cfFormat == lpOleApp->m_cfEmbedSource ||
   cfFormat == lpOleApp->m_cfEmbeddedObject ||
   cfFormat == lpOleApp->m_cfFileName) {
   /* OLE2NOTE: OleCreateFromData API creates an OLE object if
   **    CF_EMBEDDEDOBJECT, CF_EMBEDSOURCE, or CF_FILENAME are
   **    available from the source data object. the
   **    CF_FILENAME case arises when a file is copied to the
   **    clipboard from the FileManager. if the file has an
   **    associated class (see GetClassFile API), then an
   **    object of that class is created. otherwise an OLE 1.0
   **    Packaged object is created.
   */
   nCount = ContainerDoc_PasteOleObject(
         lpContainerDoc,
         lpSrcDataObj,
         OLECREATEFROMDATA_OBJECT,
         0,   /* N/A -- cfFormat */
         nIndex,
         fDisplayAsIcon,
         hMetaPict,
         lpSizelInSrc
   );
   return (nCount > 0 ? TRUE : FALSE);

} else if (cfFormat == CF_METAFILEPICT
         || cfFormat == CF_DIB
```

```
                || cfFormat == CF_BITMAP) {

            /* OLE2NOTE: OleCreateStaticFromData API creates an static
            **    OLE object if CF_METAFILEPICT, CF_DIB, or CF_BITMAP is
            **    CF_EMBEDDEDOBJECT, CF_EMBEDSOURCE, or CF_FILENAME are
            **    available from the source data object.
            */
            nCount = ContainerDoc_PasteOleObject(
                    lpContainerDoc,
                    lpSrcDataObj,
                    OLECREATEFROMDATA_STATIC,
                    cfFormat,
                    nIndex,
                    fDisplayAsIcon,
                    hMetaPict,
                    lpSizelInSrc
            );
            return (nCount > 0 ? TRUE : FALSE);

        } else if(cfFormat == CF_TEXT) {

            hData = OleStdGetData(
                    lpSrcDataObj,
                    CF_TEXT,
                    NULL,
                    DVASPECT_CONTENT,
                    (LPSTGMEDIUM)&medium
            );
            nCount = OutlineDoc_PasteTextData(
                    (LPOUTLINEDOC)lpContainerDoc,
                    hData,
                    nIndex
                );
            // OLE2NOTE: we must free data handle by releasing the medium
            ReleaseStgMedium((LPSTGMEDIUM)&medium);

        } else {
            return FALSE;   // no acceptable format available to paste
        }
    }

    lrSel.m_nStartLine = nIndex + nCount;
    lrSel.m_nEndLine = nIndex + 1;
    LineList_SetSel(lpLL, &lrSel);
    return TRUE;
}


/* ContainerDoc_PasteCntrOutlData
 * -------------------------------
 *
 *      Load the lines stored in a lpSrcStg (stored in CF_CNTROUTL format)
 *  into the document.
 *
 * Return the number of items added
 */
int ContainerDoc_PasteCntrOutlData(
      LPCONTAINERDOC          lpDestContainerDoc,
      LPSTORAGE               lpSrcStg,
      int                     nStartIndex
)
{
   int nCount;
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPOUTLINEDOC lpDestOutlineDoc = (LPOUTLINEDOC)lpDestContainerDoc;
   LPOUTLINEDOC lpSrcOutlineDoc;
   LPLINELIST   lpSrcLL;

   // create a temp document that will be used to load the lpSrcStg data.
   lpSrcOutlineDoc = (LPOUTLINEDOC)OutlineApp_CreateDoc(lpOutlineApp,
FALSE);
   if ( ! lpSrcOutlineDoc )
      return 0;

   if (! OutlineDoc_LoadFromStg(lpSrcOutlineDoc, lpSrcStg))
      goto error;

   /* copy all lines from the SrcDoc to the DestDoc. */
   lpSrcLL = &lpSrcOutlineDoc->m_LineList;
   nCount = LineList_CopySelToDoc(lpSrcLL, NULL, lpDestOutlineDoc);

   if (lpSrcOutlineDoc)            // destroy temporary document.
      OutlineDoc_Close(lpSrcOutlineDoc, OLECLOSE_NOSAVE);

   return nCount;

error:
   if (lpSrcOutlineDoc)            // destroy temporary document.
      OutlineDoc_Close(lpSrcOutlineDoc, OLECLOSE_NOSAVE);

   return 0;
}


/* ContainerDoc_QueryPasteFromData
** -------------------------------
**
**    Check if the IDataObject* offers data in a format that we can
**    paste. The IDataObject* may come from the clipboard
**    (GetClipboard) or from a drag/drop operation.
**    In this function we look if one of the following formats is
**    offered:
**              CF_OUTLINE
**              <OLE object -- CF_EMBEDSOURCE or CF_EMBEDDEDOBJECT>
**              CF_TEXT
**
**    NOTE: fLink is specified and CF_LINKSOURCE is available then TRUE
**    is returned, else FALSE.
**
**    Returns TRUE if paste can be performed
**            FALSE if paste is not possible.
*/
BOOL ContainerDoc_QueryPasteFromData(
      LPCONTAINERDOC          lpContainerDoc,
      LPDATAOBJECT            lpSrcDataObj,
      BOOL                    fLink
)
{
   LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;

   if (fLink) {
      /* check if we can paste a Link to the data */
      if (OleQueryLinkFromData(lpSrcDataObj) != NOERROR)
         return FALSE;   // linking is NOT possible
   } else {

      int nFmtEtc;

      nFmtEtc = OleStdGetPriorityClipboardFormat(
            lpSrcDataObj,
            lpOleApp->m_arrPasteEntries,
            lpOleApp->m_nPasteEntries
         );

      if (nFmtEtc < 0)
         return FALSE;   // there is no format we like
   }

   return TRUE;
}


/* ContainerDoc_PasteOleObject
** ---------------------------
**
**    Embed or link an OLE object. the source of the data is a pointer
**    to an IDataObject. normally this lpSrcDataObj comes from the
**    clipboard after call OleGetClipboard.
**
**    dwCreateType controls what type of object will created:
**    OLECREATEFROMDATA_LINK -- OleCreateLinkFromData will be called
**    OLECREATEFROMDATA_OBJECT -- OleCreateFromData will be called
**    OLECREATEFROMDATA_STATIC -- OleCreateStaticFromData will be called
**                                  cfFormat controls the type of static
**    a CONTAINERLINE object is created to manage the OLE object. this
**    CONTAINERLINE is added to the ContainerDoc after line nIndex.
**
*/
int ContainerDoc_PasteOleObject(
      LPCONTAINERDOC          lpContainerDoc,
      LPDATAOBJECT            lpSrcDataObj,
      DWORD                   dwCreateType,
      CLIPFORMAT              cfFormat,
      int                     nIndex,
      BOOL                    fDisplayAsIcon,
      HGLOBAL                 hMetaPict,
      LPSIZEL                 lpSizelInSrc
)
{
   LPLINELIST          lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   LPLINE              lpLine = NULL;
   HDC                 hDC;
   int                 nTab = 0;
   OLECHAR             szStgName[CWCSTORAGENAME];
   LPCONTAINERLINE     lpContainerLine = NULL;

   ContainerDoc_GetNextStgName(lpContainerDoc, szStgName, CWCSTORAGENAME);

   /* default the new line to have the same indent as previous line */
   lpLine = LineList_GetLine(lpLL, nIndex);
   if (lpLine)
      nTab = Line_GetTabLevel(lpLine);

   hDC = LineList_GetDC(lpLL);

   lpContainerLine = ContainerLine_CreateFromData(
         hDC,
         nTab,
         lpContainerDoc,
         lpSrcDataObj,
         dwCreateType,
         cfFormat,
         fDisplayAsIcon,
         hMetaPict,
         szStgName
      );
   LineList_ReleaseDC(lpLL, hDC);

   if (! lpContainerLine)
      goto error;

   /* add a ContainerLine object to the document's LineList. The
   **    ContainerLine manages the rectangle on the screen occupied by
   **    the OLE object. later when the app is updated to support
   **    extended layout, there could be more than one Line associated
   **    with the OLE object.
   */

   LineList_AddLine(lpLL, (LPLINE)lpContainerLine, nIndex);

   /* OLE2NOTE: if the source of the OLE object just pasted, passed a
   **    non-zero sizel in the ObjectDescriptor, then we will try to
   **    keep the object the same size as it is in the source. this
   **    may be a scaled size if the object had been resized in the
   **    source container. if the source did not give a valid sizel,
   **    then we will retrieve the size of the object by calling
   **    IViewObject2::GetExtent.
   */
   if (lpSizelInSrc && (lpSizelInSrc->cx != 0 || lpSizelInSrc->cy != 0)) {
      ContainerLine_UpdateExtent(lpContainerLine, lpSizelInSrc);
   } else
      ContainerLine_UpdateExtent(lpContainerLine, NULL);

   OutlineDoc_SetModified((LPOUTLINEDOC)lpContainerDoc, TRUE, TRUE, TRUE);

   return 1;   // one line added to LineList

error:
   // NOTE: if ContainerLine_CreateFromClip failed
   OutlineApp_ErrorMessage(g_lpApp, OLESTR("Paste Object failed!"));
   return 0;       // no lines added to line list
}


/* ContainerDoc_GetData
 * --------------------
 *
 * Render data from the document on a CALLEE allocated STGMEDIUM.
 *      This routine is called via IDataObject::GetData.
 */
HRESULT ContainerDoc_GetData (
      LPCONTAINERDOC          lpContainerDoc,
      LPFORMATETC             lpformatetc,
      LPSTGMEDIUM             lpMedium
)
{
   LPOLEDOC  lpOleDoc = (LPOLEDOC)lpContainerDoc;
   LPOUTLINEDOC  lpOutlineDoc = (LPOUTLINEDOC)lpContainerDoc;
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)g_lpApp;
   LPOLEAPP  lpOleApp = (LPOLEAPP)lpContainerApp;
   LPOUTLINEAPP  lpOutlineApp = (LPOUTLINEAPP)lpContainerApp;
   HRESULT hrErr;
   SCODE sc;

   // OLE2NOTE: we must set out pointer parameters to NULL
   lpMedium->pUnkForRelease = NULL;

   /* OLE2NOTE: we must set all out pointer parameters to NULL. */
   lpMedium->tymed = TYMED_NULL;
   lpMedium->pUnkForRelease = NULL;    // we transfer ownership to caller
   lpMedium->u.hGlobal = NULL;

   if (lpformatetc->cfFormat == lpContainerApp->m_cfCntrOutl) {

      /* OLE2NOTE: currently OLE does NOT support remoting a root
      **    level IStorage (either memory or file based) as an OUT
      **    parameter. thus, we can NOT support GetData for this
      **    TYMED_ISTORAGE based format. the caller MUST call GetDataHere.
      */
      sc = DV_E_FORMATETC;
      goto error;

   } else if (!lpContainerDoc->m_fEmbeddedObjectAvail &&
         lpformatetc->cfFormat == lpOutlineApp->m_cfOutline) {
      // Verify caller asked for correct medium
```

```
        if (!(lpformatetc->tymed & TYMED_HGLOBAL)) {
            sc = DV_E_FORMATETC;
            goto error;
        }

        lpMedium->u.hGlobal = OutlineDoc_GetOutlineData(lpOutlineDoc, NULL);
        if (! lpMedium->u.hGlobal) {
            sc = E_OUTOFMEMORY;
            goto error;
        }

        lpMedium->tymed = TYMED_HGLOBAL;
        OleDbgOut3("ContainerDoc_GetData: rendered CF_OUTLINE\r\n");
        return NOERROR;

    } else if (!lpContainerDoc->m_fEmbeddedObjectAvail &&
            lpformatetc->cfFormat == CF_TEXT) {
        // Verify caller asked for correct medium
        if (!(lpformatetc->tymed & TYMED_HGLOBAL)) {
            sc = DV_E_FORMATETC;
            goto error;
        }

        lpMedium->u.hGlobal = OutlineDoc_GetTextData (
                (LPOUTLINEDOC)lpContainerDoc,
                NULL
        );
        if (! lpMedium->u.hGlobal) {
            sc = E_OUTOFMEMORY;
            goto error;
        }

        lpMedium->tymed = TYMED_HGLOBAL;
        OleDbgOut3("ContainerDoc_GetData: rendered CF_TEXT\r\n");
        return NOERROR;

    } else if ( lpformatetc->cfFormat == lpOleApp->m_cfObjectDescriptor ||
        (lpformatetc->cfFormat == lpOleApp->m_cfLinkSrcDescriptor &&
            lpOleDoc->m_fLinkSourceAvail) ) {
        // Verify caller asked for correct medium
        if (!(lpformatetc->tymed & TYMED_HGLOBAL)) {
            sc = DV_E_FORMATETC;
            goto error;
        }

        lpMedium->u.hGlobal = OleDoc_GetObjectDescriptorData (
                (LPOLEDOC)lpContainerDoc,
                NULL
        );
        if (! lpMedium->u.hGlobal) {
            sc = E_OUTOFMEMORY;
            goto error;
        }

        lpMedium->tymed = TYMED_HGLOBAL;
```

```
#if defined( _DEBUG )
      if (lpformatetc->cfFormat == lpOleApp->m_cfObjectDescriptor)
         OleDbgOut3(
            "ContainerDoc_GetData: rendered CF_OBJECTDESCRIPTOR\r\n");
      else
         OleDbgOut3(
            "ContainerDoc_GetData: rendered CF_LINKSRCDESCRIPTOR\r\n");
#endif
      return NOERROR;

   } else if (lpContainerDoc->m_fEmbeddedObjectAvail) {

      /* OLE2NOTE: if this document contains a single OLE object
      **    (ie. cfEmbeddedObject data format is available), then
      **    the formats offered via our IDataObject must include
      **    the formats available from the OLE object itself.
      **    thus, we delegate this call to the IDataObject* of the
      **    OLE object.
      */

      if (lpformatetc->cfFormat == lpOleApp->m_cfEmbeddedObject) {
         LPPERSISTSTORAGE lpPersistStg =
               (LPPERSISTSTORAGE)ContainerDoc_GetSingleOleObject(
                     lpContainerDoc,
                     &IID_IPersistStorage,
                     NULL
         );

         if (! lpPersistStg)
            return ResultFromScode(DV_E_FORMATETC);

         /* render CF_EMBEDDEDOBJECT by asking the object to save
         **    into a temporary, DELETEONRELEASE pStg allocated by us.
         */

         hrErr = OleStdGetOleObjectData(
               lpPersistStg,
               lpformatetc,
               lpMedium,
               FALSE   /* fUseMemory -- (use file-base stg) */
         );
         OleStdRelease((LPUNKNOWN)lpPersistStg);
         if (hrErr != NOERROR) {
            sc = GetScode(hrErr);
            goto error;
         }
         OleDbgOut3("ContainerDoc_GetData: rendered CF_EMBEDDEDOBJECT\r\n");
         return hrErr;

      } else if (lpformatetc->cfFormat == CF_METAFILEPICT) {

         /* OLE2NOTE: as a container which draws objects, when a single
         **    OLE object is copied, we can give the Metafile picture of
         **    the object.
         */
```

```
          LPCONTAINERLINE lpContainerLine;
          LPOLEOBJECT lpOleObj;
          SIZEL sizelOleObject;

          // Verify caller asked for correct medium
          if (!(lpformatetc->tymed & TYMED_MFPICT)) {
             sc = DV_E_FORMATETC;
             goto error;
          }

          lpOleObj = (LPOLEOBJECT)ContainerDoc_GetSingleOleObject(
                lpContainerDoc,
                &IID_IOleObject,
                (LPCONTAINERLINE FAR*)&lpContainerLine
          );

          if (! lpOleObj) {
             sc = E_OUTOFMEMORY;     // could not load object
             goto error;
          }
          if (lpformatetc->dwAspect & lpContainerLine->m_dwDrawAspect) {
             LPLINE lpLine = (LPLINE)lpContainerLine;

             /* render CF_METAFILEPICT by drawing the object into
             **    a metafile DC
             */

             /* OLE2NOTE: Get size that object is being drawn. If the
             **    object has been scaled because the user resized the
             **    object, then we want to render a metafile with the
             **    scaled size.
             */
             sizelOleObject.cx = lpLine->m_nWidthInHimetric;
             sizelOleObject.cy = lpLine->m_nHeightInHimetric;

             lpMedium->u.hGlobal = OleStdGetMetafilePictFromOleObject(
                   lpOleObj,
                   lpContainerLine->m_dwDrawAspect,
                   (LPSIZEL)&sizelOleObject,
                   lpformatetc->ptd
             );
             OleStdRelease((LPUNKNOWN)lpOleObj);
             if (! lpMedium->u.hGlobal) {
                sc = E_OUTOFMEMORY;
                goto error;
             }

             lpMedium->tymed = TYMED_MFPICT;
             OleDbgOut3("ContainerDoc_GetData: rendered
CF_METAFILEPICT\r\n");
             return NOERROR;
          } else {
             // improper aspect requested
             OleStdRelease((LPUNKNOWN)lpOleObj);
             return ResultFromScode(DV_E_FORMATETC);
```

```
         }

      } else if (lpformatetc->cfFormat == lpOleApp->m_cfLinkSource) {
         if (lpOleDoc->m_fLinkSourceAvail) {
            LPMONIKER lpmk;

            lpmk = ContainerLine_GetFullMoniker(
                  lpContainerDoc->m_lpSrcContainerLine,
                  GETMONIKER_FORCEASSIGN
            );
            if (lpmk) {
               hrErr = OleStdGetLinkSourceData(
                     lpmk,
                     &lpContainerDoc->m_clsidOleObjCopied,
                     lpformatetc,
                     lpMedium
               );
               OleStdRelease((LPUNKNOWN)lpmk);
               if (hrErr != NOERROR) {
                  sc = GetScode(hrErr);
                  goto error;
               }
               OleDbgOut3("ContainerDoc_GetData: rendered
CF_LINKSOURCE\r\n");
               return hrErr;
            } else {
               sc = DV_E_FORMATETC;
               goto error;
            }
         } else {
            sc = DV_E_FORMATETC;
            goto error;
         }

      }
#if defined( OPTIONAL_ADVANCED_DATA_TRANSFER )
      /* OLE2NOTE: optionally, a container that wants to have a
      **    potentially richer data transfer, can enumerate the data
      **    formats from the OLE object's cache and offer them too. if
      **    the object has a special handler, then it might be able to
      **    render additional data formats. in this case, the
      **    container must delegate the GetData call to the object if
      **    it does not directly support the format.
      **
      **    CNTROUTL does NOT enumerate the cache; it implements the
      **    simpler strategy of offering a static list of formats.
      **    thus the delegation is NOT required.
      */
     else {

         /* OLE2NOTE: we delegate this call to the IDataObject* of the
         **    OLE object.
         */
         LPDATAOBJECT lpDataObj;
```

```
            lpDataObj = (LPDATAOBJECT)ContainerDoc_GetSingleOleObject(
                  lpContainerDoc,
                  &IID_IDataObject,
                  NULL
            );

            if (! lpDataObj) {
               sc = DV_E_FORMATETC;
               goto error;
            }

            OLEDBG_BEGIN2("ContainerDoc_GetData: delegate to OLE obj\r\n")
            hrErr=lpDataObj->lpVtbl->GetData(lpDataObj,lpformatetc,lpMedium);
            OLEDBG_END2

            OleStdRelease((LPUNKNOWN)lpDataObj);
            return hrErr;
         }
#endif  // ! OPTIONAL_ADVANCED_DATA_TRANSFER

   }

   // if we get here then we do NOT support the requested format
   sc = DV_E_FORMATETC;

error:
   return ResultFromScode(sc);
}


/* ContainerDoc_GetDataHere
 * ------------------------
 *
 * Render data from the document on a CALLER allocated STGMEDIUM.
 *      This routine is called via IDataObject::GetDataHere.
 */
HRESULT ContainerDoc_GetDataHere (
      LPCONTAINERDOC          lpContainerDoc,
      LPFORMATETC             lpformatetc,
      LPSTGMEDIUM             lpMedium
)
{
   LPOLEDOC  lpOleDoc = (LPOLEDOC)lpContainerDoc;
   LPOUTLINEDOC  lpOutlineDoc = (LPOUTLINEDOC)lpContainerDoc;
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)g_lpApp;
   LPOLEAPP  lpOleApp = (LPOLEAPP)lpContainerApp;
   LPOUTLINEAPP  lpOutlineApp = (LPOUTLINEAPP)lpContainerApp;
   HRESULT hrErr;

   // OLE2NOTE: lpMedium is an IN parameter. we should NOT set
   //           lpMedium->pUnkForRelease to NULL

   // we only support  IStorage medium
   if (lpformatetc->cfFormat == lpContainerApp->m_cfCntrOutl) {
      if (!(lpformatetc->tymed & TYMED_ISTORAGE))
```

```
         return ResultFromScode(DV_E_FORMATETC);

      if (lpMedium->tymed == TYMED_ISTORAGE) {
         /* Caller has allocated the storage. we must copy all of our
         **    data into his storage.
         */

         /*  OLE2NOTE: we must be sure to write our class ID into our
         **    storage. this information is used by OLE to determine the
         **    class of the data stored in our storage.
         */
         if((hrErr=WriteClassStg(lpMedium->u.pstg,&CLSID_APP)) != NOERROR)
            return hrErr;

         OutlineDoc_SaveSelToStg(
               (LPOUTLINEDOC)lpContainerDoc,
               NULL,   /* entire doc */
               lpContainerApp->m_cfCntrOutl,
               lpMedium->u.pstg,
               FALSE,  /* fSameAsLoad */
               FALSE   /* fRemember */
         );
         OleStdCommitStorage(lpMedium->u.pstg);

         OleDbgOut3("ContainerDoc_GetDataHere: rendered CF_CNTROUTL\r\n");
         return NOERROR;
      } else {
         // we only support IStorage medium
         return ResultFromScode(DV_E_FORMATETC);
      }

   } else if (lpContainerDoc->m_fEmbeddedObjectAvail) {

      /* OLE2NOTE: if this document contains a single OLE object
      **    (ie. cfEmbeddedObject data format is available), then
      **    the formats offered via our IDataObject must include
      **    CF_EMBEDDEDOBJECT and the formats available from the OLE
      **    object itself.
      */

      if (lpformatetc->cfFormat == lpOleApp->m_cfEmbeddedObject) {
         LPPERSISTSTORAGE lpPersistStg =
               (LPPERSISTSTORAGE)ContainerDoc_GetSingleOleObject(
                     lpContainerDoc,
                     &IID_IPersistStorage,
                     NULL
         );

         if (! lpPersistStg) {
            return ResultFromScode(E_OUTOFMEMORY);
         }
         /* render CF_EMBEDDEDOBJECT by asking the object to save
         **    into the IStorage allocated by the caller.
         */
         hrErr = OleStdGetOleObjectData(
               lpPersistStg,
               lpformatetc,
               lpMedium,
               FALSE   /* fUseMemory -- N/A */
         );
         OleStdRelease((LPUNKNOWN)lpPersistStg);
         if (hrErr != NOERROR) {
            return hrErr;
         }
         OleDbgOut3("ContainerDoc_GetDataHere: rendered
CF_EMBEDDEDOBJECT\r\n");
         return hrErr;

      } else if (lpformatetc->cfFormat == lpOleApp->m_cfLinkSource) {
         if (lpOleDoc->m_fLinkSourceAvail) {
            LPMONIKER lpmk;

            lpmk = ContainerLine_GetFullMoniker(
                  lpContainerDoc->m_lpSrcContainerLine,
                  GETMONIKER_FORCEASSIGN
            );
            if (lpmk) {
               hrErr = OleStdGetLinkSourceData(
                     lpmk,
                     &lpContainerDoc->m_clsidOleObjCopied,
                     lpformatetc,
                     lpMedium
               );
               OleStdRelease((LPUNKNOWN)lpmk);
               OleDbgOut3("ContainerDoc_GetDataHere: rendered
CF_LINKSOURCE\r\n");
               return hrErr;
            } else {
               return ResultFromScode(E_FAIL);
            }
         } else {
            return ResultFromScode(DV_E_FORMATETC);
         }

      } else {
#if !defined( OPTIONAL_ADVANCED_DATA_TRANSFER )
         return ResultFromScode(DV_E_FORMATETC);
#endif
#if defined( OPTIONAL_ADVANCED_DATA_TRANSFER )
         /* OLE2NOTE: optionally, a container that wants to have a
         **    potentially richer data transfer, can enumerate the data
         **    formats from the OLE object's cache and offer them too. if
         **    the object has a special handler, then it might be able to
         **    render additional data formats. in this case, the
         **    container must delegate the GetData call to the object if
         **    it does not directly support the format.
         **
         **    CNTROUTL does NOT enumerate the cache; it implements the
         **    simpler strategy of offering a static list of formats.
         **    thus the delegation is NOT required.
         */
         /* OLE2NOTE: we delegate this call to the IDataObject* of the
         **    OLE object.
         */
         LPDATAOBJECT lpDataObj;

         lpDataObj = (LPDATAOBJECT)ContainerDoc_GetSingleOleObject(
               lpContainerDoc,
               &IID_IDataObject,
               NULL
         );

         if (! lpDataObj)
            return ResultFromScode(DV_E_FORMATETC);

         OLEDBG_BEGIN2("ContainerDoc_GetDataHere: delegate to OLE obj\r\n")
         hrErr = lpDataObj->lpVtbl->GetDataHere(
               lpDataObj,
               lpformatetc,
               lpMedium
         );
         OLEDBG_END2

         OleStdRelease((LPUNKNOWN)lpDataObj);
         return hrErr;
#endif  // OPTIONAL_ADVANCED_DATA_TRANSFER
      }
   } else {
      return ResultFromScode(DV_E_FORMATETC);
   }
}


/* ContainerDoc_QueryGetData
 * -------------------------
 *
 * Answer if a particular data format is supported via GetData/GetDataHere.
 *      This routine is called via IDataObject::QueryGetData.
 */
HRESULT ContainerDoc_QueryGetData (
      LPCONTAINERDOC          lpContainerDoc,
      LPFORMATETC             lpformatetc
)
{
   LPOLEDOC  lpOleDoc = (LPOLEDOC)lpContainerDoc;
   LPOUTLINEDOC  lpOutlineDoc = (LPOUTLINEDOC)lpContainerDoc;
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)g_lpApp;
   LPOLEAPP  lpOleApp = (LPOLEAPP)lpContainerApp;
   LPOUTLINEAPP  lpOutlineApp = (LPOUTLINEAPP)lpContainerApp;
   LPDATAOBJECT  lpDataObj = NULL;
   LPCONTAINERLINE lpContainerLine = NULL;
   SCODE sc;
   HRESULT hrErr;
```

```
if (lpContainerDoc->m_fEmbeddedObjectAvail) {
   lpDataObj = (LPDATAOBJECT)ContainerDoc_GetSingleOleObject(
            lpContainerDoc,
            &IID_IDataObject,
            (LPCONTAINERLINE FAR*)&lpContainerLine
   );
}

/* Caller is querying if we support certain format but does not
**    want any data actually returned.
*/
if (lpformatetc->cfFormat == lpContainerApp->m_cfCntrOutl) {
   // we only support ISTORAGE medium
   sc = GetScode( OleStdQueryFormatMedium(lpformatetc, TYMED_ISTORAGE) );

} else if (lpformatetc->cfFormat == lpOleApp->m_cfEmbeddedObject &&
      lpContainerDoc->m_fEmbeddedObjectAvail ) {
   sc = GetScode( OleStdQueryOleObjectData(lpformatetc) );

} else if (lpformatetc->cfFormat == lpOleApp->m_cfLinkSource &&
      lpOleDoc->m_fLinkSourceAvail) {
   sc = GetScode( OleStdQueryLinkSourceData(lpformatetc) );

// CF_TEXT and CF_OUTLINE are NOT supported when single object is copied
} else if (!lpContainerDoc->m_fEmbeddedObjectAvail &&
      (lpformatetc->cfFormat == (lpOutlineApp)->m_cfOutline ||
       lpformatetc->cfFormat == CF_TEXT) ) {
   // we only support HGLOBAL medium
   sc = GetScode( OleStdQueryFormatMedium(lpformatetc, TYMED_HGLOBAL) );

} else if ( lpformatetc->cfFormat == lpOleApp->m_cfObjectDescriptor ||
   (lpformatetc->cfFormat == lpOleApp->m_cfLinkSrcDescriptor &&
      lpOleDoc->m_fLinkSourceAvail) ) {
   sc = GetScode( OleStdQueryObjectDescriptorData(lpformatetc) );

} else if (lpformatetc->cfFormat == CF_METAFILEPICT &&
      lpContainerDoc->m_fEmbeddedObjectAvail && lpContainerLine &&
      (lpformatetc->dwAspect & lpContainerLine->m_dwDrawAspect)) {

   /* OLE2NOTE: as a container which draws objects, when a single
   **    OLE object is copied, we can give the Metafile picture of
   **    the object.
   */
   // we only support MFPICT medium
   sc = GetScode( OleStdQueryFormatMedium(lpformatetc, TYMED_MFPICT) );

} else if (lpDataObj) {

   /* OLE2NOTE: if this document contains a single OLE object
   **    (ie. cfEmbeddedObject data format is available), then
   **    the formats offered via our IDataObject must include
   **    the formats available from the OLE object itself.
   **    thus we delegate this call to the IDataObject* of the
   **    OLE object.
   */
```

```
      OLEDBG_BEGIN2("ContainerDoc_QueryGetData: delegate to OLE obj\r\n")
      hrErr = lpDataObj->lpVtbl->QueryGetData(lpDataObj, lpformatetc);
      OLEDBG_END2

      sc = GetScode(hrErr);

   } else {
      sc = DV_E_FORMATETC;
   }

   if (lpDataObj)
      OleStdRelease((LPUNKNOWN)lpDataObj);
   return ResultFromScode(sc);
}



/* ContainerDoc_SetData
 * --------------------
 *
 * Set (modify) data of the document.
 *      This routine is called via IDataObject::SetData.
 */
HRESULT ContainerDoc_SetData (
      LPCONTAINERDOC          lpContainerDoc,
      LPFORMATETC             lpformatetc,
      LPSTGMEDIUM             lpmedium,
      BOOL                    fRelease
)
{
   /* in the container version of Outline, only DataTransferDoc's support
   **    IDataObject interface; the user documents do not support
   **    IDataObject. DataTransferDoc's do not accept SetData calls.
   */
   return ResultFromScode(DV_E_FORMATETC);
}


/* ContainerDoc_EnumFormatEtc
 * --------------------------
 *
 * Return an enumerator which enumerates the data accepted/offered by
 *      the document.
 *      This routine is called via IDataObject::SetData.
 */
HRESULT ContainerDoc_EnumFormatEtc(
      LPCONTAINERDOC          lpContainerDoc,
      DWORD                   dwDirection,
      LPENUMFORMATETC FAR*    lplpenumFormatEtc
)
{
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpContainerDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpContainerDoc;
   LPOLEAPP  lpOleApp = (LPOLEAPP)g_lpApp;
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)lpOleApp;
```

```
int nActualFmts;
int i;
SCODE sc = S_OK;

/* the Container-Only version of Outline does NOT offer
**    IDataObject interface from its User documents.
*/
if (! lpOutlineDoc->m_fDataTransferDoc)
   return ResultFromScode(E_FAIL);

if (dwDirection == DATADIR_GET) {
   if (lpContainerDoc->m_fEmbeddedObjectAvail) {

      /* OLE2NOTE: if this document contains a single OLE object
      **    (ie. cfEmbeddedObject data format is available), then
      **    the formats offered via our enumerator must include
      **    the formats available from the OLE object itself. we
      **    have previously set up a special array of FORMATETC's
      **    in OutlineDoc_CreateDataTransferDoc routine which includes
      **    the combination of data we offer directly and data
      **    offered by the OLE object.
      */

      /* If the document does not have a Moniker, then exclude
      **    CF_LINKSOURCE CF_LINKSRCDESCRIPTOR from the list of
      **    formats available. these formats are deliberately
      **    listed last in the array of possible "Get" formats.
      */
      nActualFmts = lpContainerApp->m_nSingleObjGetFmts;
      if (! lpOleDoc->m_fLinkSourceAvail)
         nActualFmts -= 2;

      // set correct dwDrawAspect for METAFILEPICT of object copied
      for (i = 0; i < nActualFmts; i++) {
         if (lpContainerApp->m_arrSingleObjGetFmts[i].cfFormat ==
                                         CF_METAFILEPICT) {
            lpContainerApp->m_arrSingleObjGetFmts[i].dwAspect =
                  lpContainerDoc->m_dwAspectOleObjCopied;
            break;  // DONE
         }
      }
      *lplpenumFormatEtc = OleStdEnumFmtEtc_Create(
            nActualFmts, lpContainerApp->m_arrSingleObjGetFmts);
      if (*lplpenumFormatEtc == NULL)
         sc = E_OUTOFMEMORY;

   } else {

      /* This document does NOT offer cfEmbeddedObject,
      **    therefore we can simply enumerate the
      **    static list of formats that we handle directly.
      */
      *lplpenumFormatEtc = OleStdEnumFmtEtc_Create(
            lpOleApp->m_nDocGetFmts, lpOleApp->m_arrDocGetFmts);
      if (*lplpenumFormatEtc == NULL)
```

```
            sc = E_OUTOFMEMORY;
        }
    } else if (dwDirection == DATADIR_SET) {
        /* OLE2NOTE: a document that is used to transfer data
        **    (either via the clipboard or drag/drop does NOT
        **    accept SetData on ANY format!
        */
        sc = E_NOTIMPL;

    } else {
        sc = E_NOTIMPL;
    }

    return ResultFromScode(sc);
}


#if defined( OPTIONAL_ADVANCED_DATA_TRANSFER )
/* OLE2NOTE: optionally, a container that wants to have a
**    potentially richer data transfer, can enumerate the data
**    formats from the OLE object's cache and offer them too. if
**    the object has a special handler, then it might be able to
**    render additional data formats.
**
**    CNTROUTL does NOT enumerate the cache; it implements the simpler
**    strategy of offering a static list of formats. the following
**    function is included in order to illustrates how enumerating the
**    cache could be done. CNTROUTL does NOT call this function.
**
*/

/* ContainerDoc_SetupDocGetFmts
** ----------------------------
**    Setup the combined list of formats that this data transfer
**    ContainerDoc which contains a single OLE object should offer.
**
**    OLE2NOTE: The list of formats that should be offered when a
**    single OLE object is being transfered include the following:
**          * any formats the container app wants to give
**          * CF_EMBEDDEDOBJECT
**          * CF_METAFILEPICT
**          * any formats that the OLE object's cache can offer directly
**
**    We will offer the following formats in the order given:
**                  1. CF_CNTROUTL
**                  2. CF_EMBEDDEDOBJECT
**                  3. CF_OBJECTDESCRIPTOR
**                  4. CF_METAFILEPICT
**                  5. <data formats from OLE object's cache>
**                  6. CF_LINKSOURCE
**                  7. CF_LINKSRCDESCRIPTOR
*/
BOOL ContainerDoc_SetupDocGetFmts(
      LPCONTAINERDOC          lpContainerDoc,
      LPCONTAINERLINE         lpContainerLine
```

```
)
{
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpContainerDoc;
   LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)g_lpApp;
   LPOLECACHE lpOleCache;
   HRESULT hrErr;
   STATDATA        StatData;
   LPENUMSTATDATA  lpEnumStatData = NULL;
   LPFORMATETC lparrDocGetFmts = NULL;
   UINT nOleObjFmts = 0;
   UINT nTotalFmts;
   UINT i;
   UINT iFmt;

   lpOleCache = (LPOLECACHE)OleStdQueryInterface(
         (LPUNKNOWN)lpContainerLine->m_lpOleObj,
         &IID_IOleCache
   );
   if (lpOleCache) {
      OLEDBG_BEGIN2("IOleCache::EnumCache called\r\n")
      hrErr = lpOleCache->lpVtbl->EnumCache(
            lpOleCache,
            (LPENUMSTATDATA FAR*)&lpEnumStatData
      );
      OLEDBG_END2
   }

   if (lpEnumStatData) {
      /* Cache enumerator is available. count the number of
      **    formats that the OLE object's cache offers.
      */
      while(lpEnumStatData->lpVtbl->Next(
               lpEnumStatData,
               1,
               (LPSTATDATA)&StatData,
               NULL) == NOERROR) {
         nOleObjFmts++;
         // OLE2NOTE: we MUST free the TargetDevice
         OleStdFree(StatData.formatetc.ptd);
      }
      lpEnumStatData->lpVtbl->Reset(lpEnumStatData);  // reset for next loop
   }

   /* OLE2NOTE: the maximum total number of formats that our IDataObject
   **    could offer equals the sum of the following:
   **          n offered by the OLE object's cache
   **       +  n normally offered by our app
   **       +  1 CF_EMBEDDEDOBJECT
   **       +  1 CF_METAFILEPICT
   **       +  1 CF_LINKSOURCE
   **       +  1 CF_LINKSRCDESCRIPTOR
   **    the actual number of formats that we can offer could be less
   **    than this total if there is any clash between the formats
   **    that we offer directly and those offered by the cache. if
   **    there is a clash, the container's rendering overrides that of
   **    the object. eg.: as a container transfering an OLE object we
   **    should directly offer CF_METAFILEPICT to guarantee that this
   **    format is always available. thus, if the cache offers
   **    CF_METAFILEPICT then it is skipped.
   */
   nTotalFmts = nOleObjFmts + lpOleApp->m_nDocGetFmts + 4;
   lparrDocGetFmts = (LPFORMATETC)New (nTotalFmts * sizeof(FORMATETC));

   OleDbgAssertSz(lparrDocGetFmts != NULL,"Error allocating arrDocGetFmts");
   if (lparrDocGetFmts == NULL)
         return FALSE;

   for (i = 0, iFmt = 0; i < lpOleApp->m_nDocGetFmts; i++) {
      _fmemcpy((LPFORMATETC)&lparrDocGetFmts[iFmt++],
            (LPFORMATETC)&lpOleApp->m_arrDocGetFmts[i],
            sizeof(FORMATETC)
      );
      if (lpOleApp->m_arrDocGetFmts[i].cfFormat ==
         lpContainerApp->m_cfCntrOutl) {
         /* insert CF_EMBEDDEDOBJECT, CF_METAFILEPICT, and formats
         **    available from the OLE object's cache following
         **    CF_CNTROUTL.
         */
         lparrDocGetFmts[iFmt].cfFormat = lpOleApp->m_cfEmbeddedObject;
         lparrDocGetFmts[iFmt].ptd      = NULL;
         lparrDocGetFmts[iFmt].dwAspect = DVASPECT_CONTENT;
         lparrDocGetFmts[iFmt].tymed    = TYMED_ISTORAGE;
         lparrDocGetFmts[iFmt].lindex   = -1;
         iFmt++;
         lparrDocGetFmts[iFmt].cfFormat = CF_METAFILEPICT;
         lparrDocGetFmts[iFmt].ptd      = NULL;
         lparrDocGetFmts[iFmt].dwAspect = lpContainerLine->m_dwDrawAspect;
         lparrDocGetFmts[iFmt].tymed    = TYMED_MFPICT;
         lparrDocGetFmts[iFmt].lindex   = -1;
         iFmt++;

         if (lpEnumStatData) {
            /* Cache enumerator is available. enumerate all of
            **    the formats that the OLE object's cache offers.
            */
            while(lpEnumStatData->lpVtbl->Next(
                  lpEnumStatData,
                  1,
                  (LPSTATDATA)&StatData,
                  NULL) == NOERROR) {
               /* check if the format clashes with one of our fmts */
               if (StatData.formatetc.cfFormat != CF_METAFILEPICT
                  && ! OleStdIsDuplicateFormat(
                        (LPFORMATETC)&StatData.formatetc,
                        lpOleApp->m_arrDocGetFmts,
                        lpOleApp->m_nDocGetFmts)) {
                  OleStdCopyFormatEtc(
                        &(lparrDocGetFmts[iFmt]),&StatData.formatetc);
                  iFmt++;
```

```
                }
                // OLE2NOTE: we MUST free the TargetDevice
                OleStdFree(StatData.formatetc.ptd);
            }
        }
    }
}

    if (lpOleCache)
        OleStdRelease((LPUNKNOWN)lpOleCache);

    /* append CF_LINKSOURCE format */
    lparrDocGetFmts[iFmt].cfFormat = lpOleApp->m_cfLinkSource;
    lparrDocGetFmts[iFmt].ptd      = NULL;
    lparrDocGetFmts[iFmt].dwAspect = DVASPECT_CONTENT;
    lparrDocGetFmts[iFmt].tymed    = TYMED_ISTREAM;
    lparrDocGetFmts[iFmt].lindex   = -1;
    iFmt++;

    /* append CF_LINKSRCDESCRIPTOR format */
    lparrDocGetFmts[iFmt].cfFormat = lpOleApp->m_cfLinkSrcDescriptor;
    lparrDocGetFmts[iFmt].ptd      = NULL;
    lparrDocGetFmts[iFmt].dwAspect = DVASPECT_CONTENT;
    lparrDocGetFmts[iFmt].tymed    = TYMED_HGLOBAL;
    lparrDocGetFmts[iFmt].lindex   = -1;
    iFmt++;

    lpContainerDoc->m_lparrDocGetFmts = lparrDocGetFmts;
    lpContainerDoc->m_nDocGetFmts = iFmt;

    if (lpEnumStatData)
        OleStdVerifyRelease(
                (LPUNKNOWN)lpEnumStatData,
                "Cache enumerator not released properly"
        );

    return TRUE;
}
#endif  // OPTIONAL_ADVANCED_DATA_TRANSFER

#endif  // OLE_CNTR
```

CNTRBASE.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Container Sample Code
**
**    cntrbase.c
**
**    This file contains all interfaces, methods and related support
**    functions for the basic OLE Container application. The
**    basic OLE Container application supports being a container for
**    embedded and linked objects.
**    The basic Container application includes the following
**    implementation objects:
**
**    ContainerDoc Object
**      no required interfaces for basic functionality
**      (see linking.c for linking related support)
**      (see clipbrd.c for clipboard related support)
**      (see dragdrop.c for drag/drop related support)
**
**    ContainerLine Object
**    (see cntrline.c for all ContainerLine functions and interfaces)
**      exposed interfaces:
**          IOleClientSite
**          IAdviseSink
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"


OLEDBGDATA


extern LPOUTLINEAPP             g_lpApp;
extern IOleUILinkContainerVtbl  g_CntrDoc_OleUILinkContainerVtbl;

#if defined( INPLACE_CNTR )
extern BOOL g_fInsideOutContainer;
#endif  // INPLACE_CNTR

// REVIEW: should use string resource for messages
OLECHAR ErrMsgShowObj[] = OLESTR("Could not show object server!");
OLECHAR ErrMsgInsertObj[] = OLESTR("Insert Object failed!");
OLECHAR ErrMsgConvertObj[] = OLESTR("Convert Object failed!");
OLECHAR ErrMsgCantConvert[] = OLESTR("Unable to convert the selection!");
OLECHAR ErrMsgActivateAsObj[] = OLESTR("Activate As Object failed!");

extern OLECHAR ErrMsgSaving[];
extern OLECHAR ErrMsgOpening[];
```

```
/* ContainerDoc_Init
 * -----------------
 *
 *  Initialize the fields of a new ContainerDoc object. The doc is initially
 *  not associated with a file or an (Untitled) document. This function sets
 *  the docInitType to DOCTYPE_UNKNOWN. After calling this function the
 *  caller should call:
 *      1.) Doc_InitNewFile to set the ContainerDoc to (Untitled)
 *      2.) Doc_LoadFromFile to associate the ContainerDoc with a file.
 *  This function creates a new window for the document.
 *
 *  NOTE: the window is initially created with a NIL size. it must be
 *        sized and positioned by the caller. also the document is initially
 *        created invisible. the caller must call Doc_ShowWindow
 *        after sizing it to make the document window visible.
 */
BOOL ContainerDoc_Init(LPCONTAINERDOC lpContainerDoc, BOOL fDataTransferDoc)
{
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)g_lpApp;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpContainerDoc;

   lpOutlineDoc->m_cfSaveFormat             = lpContainerApp->m_cfCntrOutl;
   lpContainerDoc->m_nNextObjNo            = 0L;
   lpContainerDoc->m_lpNewStg              = NULL;
   lpContainerDoc->m_fEmbeddedObjectAvail  = FALSE;
   lpContainerDoc->m_clsidOleObjCopied     = CLSID_NULL;
   lpContainerDoc->m_dwAspectOleObjCopied  = DVASPECT_CONTENT;
   lpContainerDoc->m_lpSrcContainerLine    = NULL;
   lpContainerDoc->m_fShowObject           = TRUE;

#if defined( INPLACE_CNTR )
   lpContainerDoc->m_lpLastIpActiveLine    = NULL;
   lpContainerDoc->m_lpLastUIActiveLine    = NULL;
   lpContainerDoc->m_hWndUIActiveObj       = NULL;
   lpContainerDoc->m_fAddMyUI              = TRUE; // UI needs to be added
   lpContainerDoc->m_cIPActiveObjects      = 0;
   lpContainerApp->m_fMenuHelpMode         = FALSE; // F1 pressed in menu

#if defined( INPLACE_CNTRSVR )
   lpContainerDoc->m_lpTopIPFrame          =
            (LPOLEINPLACEUIWINDOW)&lpContainerDoc->m_OleInPlaceFrame;
   lpContainerDoc->m_lpTopIPDoc            =
            (LPOLEINPLACEUIWINDOW)&lpContainerDoc->m_OleInPlaceDoc;
   lpContainerDoc->m_hSharedMenu           = NULL;
   lpContainerDoc->m_hOleMenu              = NULL;

#endif  // INPLACE_CNTRSVR
#endif  // INPLACE_CNTR

   INIT_INTERFACEIMPL(
         &lpContainerDoc->m_OleUILinkContainer,
         &g_CntrDoc_OleUILinkContainerVtbl,
         lpContainerDoc
   );
   return TRUE;
}


/* ContainerDoc_GetNextLink
 * ------------------------
 *
 *  Update all links in the document. A dialog box will be popped up showing
 *  the progress of the update and allow the user to quit by pushing the
 *  stop button
 */
LPCONTAINERLINE ContainerDoc_GetNextLink(
      LPCONTAINERDOC lpContainerDoc,
      LPCONTAINERLINE lpContainerLine
)
{
   LPLINELIST lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   DWORD dwNextLink = 0;
   LPLINE lpLine;
   static int nIndex = 0;

   if (lpContainerLine==NULL)
      nIndex = 0;

   for ( ; nIndex < lpLL->m_nNumLines; nIndex++) {
      lpLine = LineList_GetLine(lpLL, nIndex);

      if (lpLine
         && (Line_GetLineType(lpLine) == CONTAINERLINETYPE)
         && ContainerLine_IsOleLink((LPCONTAINERLINE)lpLine)) {

         nIndex++;
         ContainerLine_LoadOleObject((LPCONTAINERLINE)lpLine);
         return (LPCONTAINERLINE)lpLine;
      }
   }

   return NULL;
}



/* ContainerDoc_UpdateLinks
 * ------------------------
 *
 *  Update all links in the document. A dialog box will be popped up showing
 *  the progress of the update and allow the user to quit by pushing the
 *  stop button
 */
void ContainerDoc_UpdateLinks(LPCONTAINERDOC lpContainerDoc)
{
   int             cLinks;
   BOOL            fAllLinksUpToDate = TRUE;
   HWND            hwndDoc = ((LPOUTLINEDOC)lpContainerDoc)->m_hWndDoc;
   HCURSOR          hCursor;
   LPCONTAINERLINE lpContainerLine = NULL;
   HRESULT          hrErr;
   DWORD            dwUpdateOpt;
   LPOLEAPP         lpOleApp = (LPOLEAPP)g_lpApp;
   BOOL             fPrevEnable1;
   BOOL             fPrevEnable2;

   hCursor = SetCursor(LoadCursor(NULL, IDC_WAIT));

   /* OLE2NOTE: we do not want to ever give the Busy/NotResponding
   **    dialogs when we are updating automatic links as part of
   **    opening a document.  even if the link source of data is busy,
   **    we do not want put up the busy dialog. thus we will disable
   **    the dialog and later re-enable them
   */
   OleApp_DisableBusyDialogs(lpOleApp, &fPrevEnable1, &fPrevEnable2);

   /* get total number of automatic links */
   cLinks = 0;
   while (lpContainerLine = ContainerDoc_GetNextLink(
                           lpContainerDoc,
                           lpContainerLine)) {
      hrErr = CntrDoc_LinkCont_GetLinkUpdateOptions(
            (LPOLEUILINKCONTAINER)&lpContainerDoc->m_OleUILinkContainer,
            (DWORD)lpContainerLine,
            (LPDWORD)&dwUpdateOpt
      );
      if (hrErr == NOERROR) {
         if (dwUpdateOpt==OLEUPDATE_ALWAYS) {
            cLinks++;
            if (fAllLinksUpToDate) {
               OLEDBG_BEGIN2("IOleObject::IsUpToDate called\r\n")
               hrErr = lpContainerLine->m_lpOleObj->lpVtbl->IsUpToDate(
                     lpContainerLine->m_lpOleObj);
               OLEDBG_END2
               if (hrErr != NOERROR)
                  fAllLinksUpToDate = FALSE;
            }
         }
      }
#if defined( _DEBUG )
      else
         OleDbgOutHResult("IOleUILinkContainer::GetLinkUpdateOptions
returned",hrErr);
#endif

   }

   if (fAllLinksUpToDate)
      goto done; // don't bother user if all links are up-to-date

   SetCursor(hCursor);

   if ((cLinks > 0) && !OleUIUpdateLinks(
         (LPOLEUILINKCONTAINER)&lpContainerDoc->m_OleUILinkContainer,
         hwndDoc,
         (LPOLESTR)APPNAME,
         cLinks))
      {
      char szTemp[256];
      wcstombs(szTemp, APPNAME, 255);
      if (ID_PU_LINKS == OleUIPromptUser(
            (WORD)IDD_CANNOTUPDATELINK,
            hwndDoc,
            szTemp)) {
         ContainerDoc_EditLinksCommand(lpContainerDoc);
      }
   }

done:
   // re-enable the Busy/NotResponding dialogs
   OleApp_EnableBusyDialogs(lpOleApp, fPrevEnable1, fPrevEnable2);
}



/* ContainerDoc_SetShowObjectFlag
 * ------------------------------
 *
 *  Set/Clear the ShowObject flag of ContainerDoc
 */
void ContainerDoc_SetShowObjectFlag(LPCONTAINERDOC lpContainerDoc, BOOL
fShow)
{
   if (!lpContainerDoc)
      return;

   lpContainerDoc->m_fShowObject = fShow;
}


/* ContainerDoc_GetShowObjectFlag
 * ------------------------------
 *
 *  Get the ShowObject flag of ContainerDoc
 */
BOOL ContainerDoc_GetShowObjectFlag(LPCONTAINERDOC lpContainerDoc)
{
   if (!lpContainerDoc)
      return FALSE;

   return lpContainerDoc->m_fShowObject;
}


/* ContainerDoc_InsertOleObjectCommand
 * -----------------------------------
 *
 * Insert a new OLE object in the ContainerDoc.
 */
void ContainerDoc_InsertOleObjectCommand(LPCONTAINERDOC lpContainerDoc)
{
   LPLINELIST               lpLL =&((LPOUTLINEDOC)lpContainerDoc)-
>m_LineList;
   LPLINE                   lpLine = NULL;
   HDC                      hDC;
   int                      nTab = 0;
   int                      nIndex = LineList_GetFocusLineIndex(lpLL);
   LPCONTAINERLINE          lpContainerLine=NULL;
   OLECHAR                  szStgName[CWCSTORAGENAME];
   UINT                     uRet;
   OLEUIINSERTOBJECT        io;
   OLECHAR                  szFile[OLEUI_CCHPATHMAX];
   DWORD                    dwOleCreateType;
   BOOL                     fDisplayAsIcon;
   HCURSOR                  hPrevCursor;

   _fmemset((LPOLEUIINSERTOBJECT)&io, 0, sizeof(OLEUIINSERTOBJECT));
   io.cbStruct=sizeof(OLEUIINSERTOBJECT);
   io.dwFlags=IOF_SELECTCREATENEW | IOF_SHOWHELP;
   io.hWndOwner=((LPOUTLINEDOC)lpContainerDoc)->m_hWndDoc;
   io.lpszFile=(LPOLESTR)szFile;
   io.cchFile=OLEUI_CCHPATHMAX;
//    _fmemset((LPOLESTR)szFile, 0, OLEUI_CCHPATHMAX);
   _fmemset((LPOLESTR)szFile, 0, OLEUI_CCHPATHMAX*sizeof(OLECHAR));

#if defined( OLE_VERSION )
   OleApp_PreModalDialog((LPOLEAPP)g_lpApp, (LPOLEDOC)lpContainerDoc);
#endif

   OLEDBG_BEGIN3("OleUIInsertObject called\r\n")
   uRet=OleUIInsertObject((LPOLEUIINSERTOBJECT)&io);
   OLEDBG_END3

#if defined( OLE_VERSION )
   OleApp_PostModalDialog((LPOLEAPP)g_lpApp, (LPOLEDOC)lpContainerDoc);
#endif

   if (OLEUI_OK != uRet)
      return;     // user canceled dialog

   // this may take a while, put up hourglass cursor
   hPrevCursor = SetCursor(LoadCursor(NULL, IDC_WAIT));

   fDisplayAsIcon = (io.dwFlags & IOF_CHECKDISPLAYASICON ? TRUE : FALSE);

   // make up a storage name for the OLE object
   ContainerDoc_GetNextStgName(lpContainerDoc, szStgName, CWCSTORAGENAME);

   /* default the new line to have the same indent as previous line */
   lpLine = LineList_GetLine(lpLL, nIndex);
   if (lpLine)
      nTab = Line_GetTabLevel(lpLine);
```

```
hDC = LineList_GetDC(lpLL);

if ((io.dwFlags & IOF_SELECTCREATENEW))
   dwOleCreateType = IOF_SELECTCREATENEW;
else if ((io.dwFlags & IOF_CHECKLINK))
   dwOleCreateType = IOF_CHECKLINK;
else
   dwOleCreateType = IOF_SELECTCREATEFROMFILE;

lpContainerLine = ContainerLine_Create(
      dwOleCreateType,
      hDC,
      nTab,
      lpContainerDoc,
      &io.clsid,
      (LPOLESTR)szFile,
      fDisplayAsIcon,
      io.hMetaPict,
      szStgName
);

if (!lpContainerLine)
   goto error;         // creation of OLE object FAILED

if (io.hMetaPict) {
   OleUIMetafilePictIconFree(io.hMetaPict);    // clean up metafile
}

/* add a ContainerLine object to the document's LineList. The
**    ContainerLine manages the rectangle on the screen occupied by
**    the OLE object.
*/

LineList_AddLine(lpLL, (LPLINE)lpContainerLine, nIndex);

/* before calling DoVerb(OLEIVERB_SHOW), check to see if the object
**    has any initial extents.
*/
ContainerLine_UpdateExtent(lpContainerLine, NULL);

/* If a new embedded object was created, tell the object server to
**    make itself visible (show itself).
**    OLE2NOTE: the standard OLE 2 User Model is to only call
**    IOleObject::DoVerb(OLEIVERB_SHOW...) if a new object is
**    created. specifically, it should NOT be calld if the object
**    is created from file or link to file.
*/
if (dwOleCreateType == IOF_SELECTCREATENEW) {
   if (! ContainerLine_DoVerb(
         lpContainerLine, OLEIVERB_SHOW, NULL, TRUE, TRUE)) {
      OutlineApp_ErrorMessage(g_lpApp, ErrMsgShowObj);
   }

   /* OLE2NOTE: we will immediately force a save of the object
   **    to guarantee that a valid initial object is saved
```

```
      **    with our document. if the object is a OLE 1.0 object,
      **    then it may exit without update. by forcing this
      **    initial save we consistently always have a valid
      **    object even if it is a OLE 1.0 object that exited
      **    without saving. if we did NOT do this save here, then
      **    we would have to worry about deleting the object if
      **    it was a OLE 1.0 object that closed without saving.
      **    the OLE 2.0 User Model dictates that the object
      **    should always be valid after CreateNew performed. the
      **    user must explicitly delete it.
      */
      ContainerLine_SaveOleObjectToStg(
            lpContainerLine,
            lpContainerLine->m_lpStg,
            lpContainerLine->m_lpStg,
            TRUE    /* fRemember */
      );
   }
#if defined( INPLACE_CNTR )
   else if (dwOleCreateType == IOF_SELECTCREATEFROMFILE) {
      /* OLE2NOTE: an inside-out container should check if the object
      **    created from file is an inside-out and prefers to be
      **    activated when visible type of object. if so, the object
      **    should be immediately activated in-place, BUT NOT UIActived.
      */
      if (g_fInsideOutContainer &&
            lpContainerLine->m_dwDrawAspect == DVASPECT_CONTENT &&
            lpContainerLine->m_fInsideOutObj ) {
         HWND hWndDoc = OutlineDoc_GetWindow((LPOUTLINEDOC)lpContainerDoc);

         ContainerLine_DoVerb(
               lpContainerLine,OLEIVERB_INPLACEACTIVATE,NULL,FALSE,FALSE);

         /* OLE2NOTE: following this DoVerb(INPLACEACTIVATE) the
         **    object may have taken focus. but because the
         **    object is NOT UIActive it should NOT have focus.
         **    we will make sure our document has focus.
         */
         SetFocus(hWndDoc);
      }
   }
#endif  // INPLACE_CNTR

   OutlineDoc_SetModified((LPOUTLINEDOC)lpContainerDoc, TRUE, TRUE, TRUE);

   LineList_ReleaseDC(lpLL, hDC);

   SetCursor(hPrevCursor);     // restore original cursor

   return;

error:
   // NOTE: if ContainerLine_Create failed
   LineList_ReleaseDC(lpLL, hDC);
   if (OLEUI_OK == uRet && io.hMetaPict)
      OleUIMetafilePictIconFree(io.hMetaPict);    // clean up metafile

   SetCursor(hPrevCursor);     // restore original cursor
   OutlineApp_ErrorMessage(g_lpApp, ErrMsgInsertObj);
}



void ContainerDoc_EditLinksCommand(LPCONTAINERDOC lpContainerDoc)
{
   UINT        uRet;
   OLEUIEDITLINKS      el;
   LPCONTAINERLINE lpContainerLine = NULL;
   LPLINELIST lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;

   _fmemset((LPOLEUIEDITLINKS)&el,0,sizeof(OLEUIEDITLINKS));
   el.cbStruct=sizeof(OLEUIEDITLINKS);
   el.dwFlags=ELF_SHOWHELP;
   el.hWndOwner=((LPOUTLINEDOC)lpContainerDoc)->m_hWndDoc;
   el.lpOleUILinkContainer =
         (LPOLEUILINKCONTAINER)&lpContainerDoc->m_OleUILinkContainer;

#if defined( OLE_VERSION )
   OleApp_PreModalDialog((LPOLEAPP)g_lpApp, (LPOLEDOC)lpContainerDoc);
#endif

   OLEDBG_BEGIN3("OleUIEditLinks called\r\n")
   uRet=OleUIEditLinks((LPOLEUIEDITLINKS)&el);
   OLEDBG_END3

#if defined( OLE_VERSION )
   OleApp_PostModalDialog((LPOLEAPP)g_lpApp, (LPOLEDOC)lpContainerDoc);
#endif

   OleDbgAssert((uRet==1) || (uRet==OLEUI_CANCEL));

}


/* Convert command - brings up the "Convert" dialog
 */
void ContainerDoc_ConvertCommand(
      LPCONTAINERDOC      lpContainerDoc,
      BOOL                fServerNotRegistered
)
{
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpContainerDoc;
   OLEUICONVERT ct;
   UINT         uRet;
   LPDATAOBJECT  lpDataObj;
   LPLINELIST lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   LPCONTAINERLINE lpContainerLine = NULL;
   BOOL         fSelIsOleObject;
   int          nIndex;
```

```
STGMEDIUM    medium;
LPOLESTR     lpErrMsg = NULL;
HRESULT      hrErr;
HCURSOR      hPrevCursor;
BOOL         fMustRun = FALSE;
BOOL         fMustClose = FALSE;
BOOL         fObjConverted = FALSE;
BOOL         fDisplayChanged = FALSE;
BOOL         fHaveCLSID = FALSE;
BOOL         fHaveFmtUserType = FALSE;
OLECHAR      szUserType[128];
BOOL         fMustActivate;

/* OLE2NOTE: if we came to the Convert dialog because the user
**    activated a non-registered object, then we should activate
**    the object after the user has converted it or setup an
**    ActivateAs server.
*/
fMustActivate = fServerNotRegistered;

_fmemset((LPOLEUICONVERT)&ct,0,sizeof(OLEUICONVERT));

fSelIsOleObject = ContainerDoc_IsSelAnOleObject(
      (LPCONTAINERDOC)lpContainerDoc,
      &IID_IDataObject,
      (LPUNKNOWN FAR*)&lpDataObj,
      &nIndex,
      (LPCONTAINERLINE FAR*)&lpContainerLine
);

lpErrMsg = ErrMsgCantConvert;

if (! fSelIsOleObject)
   goto error;     // can NOT do Convert.

if (! lpContainerLine) {
   OleStdRelease((LPUNKNOWN)lpDataObj);
   goto error;     // can NOT do Convert.
}

ct.cbStruct  = sizeof(OLEUICONVERT);
ct.dwFlags   = CF_SHOWHELPBUTTON;
ct.hWndOwner = lpContainerDoc->m_OleDoc.m_OutlineDoc.m_hWndDoc;
ct.lpszCaption = (LPOLESTR)NULL;
ct.lpfnHook  = NULL;
ct.lCustData = 0;
ct.hInstance = NULL;
ct.lpszTemplate = NULL;
ct.hResource = 0;
ct.fIsLinkedObject = ContainerLine_IsOleLink(lpContainerLine);
ct.dvAspect = lpContainerLine->m_dwDrawAspect;
ct.cClsidExclude = 0;
ct.lpClsidExclude = NULL;

if (! ct.fIsLinkedObject || !lpContainerLine->m_lpOleLink) {
   /* OLE2NOTE: the object is an embedded object. we should first
   **    attempt to read the actual object CLSID, file data
   **    format, and full user type name that is written inside of
   **    the object's storage as this should be the most
   **    definitive information. if this fails we will ask the
   **    object what its class is and attempt to get the rest of
   **    the information out of the REGDB.
   */
   hrErr=ReadClassStg(lpContainerLine->m_lpStg,(CLSID FAR*)&(ct.clsid));
   if (hrErr == NOERROR)
      fHaveCLSID = TRUE;
   else {
      OleDbgOutHResult("ReadClassStg returned", hrErr);
   }
   hrErr = ReadFmtUserTypeStg(
         lpContainerLine->m_lpStg,
         (CLIPFORMAT FAR*)&ct.wFormat,
         (LPOLESTR FAR*)&ct.lpszUserType);
   if (hrErr == NOERROR)
      fHaveFmtUserType = TRUE;
   else {
      OleDbgOutHResult("ReadFmtUserTypeStg returned", hrErr);
   }
} else {
   /* OLE2NOTE: the object is a linked object. we should give the
   **    DisplayName of the link source as the default icon label.
   */
   OLEDBG_BEGIN2("IOleLink::GetSourceDisplayName called\r\n")
   hrErr = lpContainerLine->m_lpOleLink->lpVtbl->GetSourceDisplayName(
         lpContainerLine->m_lpOleLink, &ct.lpszDefLabel);
   OLEDBG_END2
}

if (! fHaveCLSID) {
   hrErr = lpContainerLine->m_lpOleObj->lpVtbl->GetUserClassID(
         lpContainerLine->m_lpOleObj,
         (CLSID FAR*)&ct.clsid
   );
   if (hrErr != NOERROR)
      ct.clsid = CLSID_NULL;
}
if (! fHaveFmtUserType) {
   ct.wFormat = 0;
   if (OleStdGetUserTypeOfClass(
         (CLSID FAR*)&ct.clsid,szUserType,128,NULL)) {
      ct.lpszUserType = OleStdCopyString(szUserType, NULL);
   } else {
      ct.lpszUserType = NULL;
   }
}

if (lpContainerLine->m_dwDrawAspect == DVASPECT_ICON) {
   ct.hMetaPict = OleStdGetData(
         lpDataObj,
         CF_METAFILEPICT,
         NULL,
         DVASPECT_ICON,
         (LPSTGMEDIUM)&medium
   );
} else {
   ct.hMetaPict = NULL;
}
OleStdRelease((LPUNKNOWN)lpDataObj);

#if defined( OLE_VERSION )
OleApp_PreModalDialog((LPOLEAPP)g_lpApp, (LPOLEDOC)lpContainerDoc);
#endif

OLEDBG_BEGIN3("OleUIConvert called\r\n")
uRet = OleUIConvert(&ct);
OLEDBG_END3

#if defined( OLE_VERSION )
OleApp_PostModalDialog((LPOLEAPP)g_lpApp, (LPOLEDOC)lpContainerDoc);
#endif

// this may take a while, put up hourglass cursor
hPrevCursor = SetCursor(LoadCursor(NULL, IDC_WAIT));

if (uRet == OLEUI_OK) {

   /*****************************************************************
   **  OLE2NOTE: the convert dialog actually allows the user to
   **    change two orthogonal properties of the object: the
   **    object's type/server and the object's display aspect.
   **    first we will execute the ConvertTo/ActivateAs action and
   **    then we will deal with any display aspect change. we want
   **    to be careful to only call IOleObject::Update once
   **    because this is an expensive operation; it results in
   **    launching the object's server.
   *****************************************************************/

   if (ct.dwFlags & CF_SELECTCONVERTTO &&
         ! IsEqualCLSID(&ct.clsid, &ct.clsidNew)) {

      /* user selected CONVERT.
      **
      ** OLE2NOTE: to achieve the "Convert To" at this point we
      **    need to take the following steps:
      **    1. unload the object.
      **    2. write the NEW CLSID and NEW user type name
      **       string into the storage of the object,
      **       BUT write the OLD format tag.
      **    3. force an update to force the actual conversion of
      **       the data bits.
      */
      lpErrMsg = ErrMsgConvertObj; // setup correct msg in case of error

      ContainerLine_UnloadOleObject(lpContainerLine,
OLECLOSE_SAVEIFDIRTY);
```

```
   OLEDBG_BEGIN2("OleStdDoConvert called \r\n")
   hrErr = OleStdDoConvert(
         lpContainerLine->m_lpStg, (REFCLSID)&ct.clsidNew);
   OLEDBG_END2
   if (hrErr != NOERROR)
      goto error;

   // Reload the object
   ContainerLine_LoadOleObject(lpContainerLine);

   /* we need to force the object to run to complete the
   **    conversion. set flag to force OleRun to be called at
   **    end of function.
   */
   fMustRun = TRUE;
   fObjConverted = TRUE;

} else if (ct.dwFlags & CF_SELECTACTIVATEAS) {
   /* user selected ACTIVATE AS.
   **
   ** OLE2NOTE: to achieve the "Activate As" at this point we
   **    need to take the following steps:
   **    1. unload ALL objects of the OLD class that app knows about
   **    2. add the TreatAs tag in the registration database
   **    by calling CoTreatAsClass().
   **    3. lazily it can reload the objects; when the objects
   **    are reloaded the TreatAs will take effect.
   */
   lpErrMsg = ErrMsgActivateAsObj; // setup msg in case of error

   ContainerDoc_UnloadAllOleObjectsOfClass(
         lpContainerDoc,
         (REFCLSID)&ct.clsid,
         OLECLOSE_SAVEIFDIRTY
   );

   OLEDBG_BEGIN2("OleStdDoTreatAsClass called \r\n")
   hrErr = OleStdDoTreatAsClass(ct.lpszUserType, (REFCLSID)&ct.clsid,
         (REFCLSID)&ct.clsidNew);
   OLEDBG_END2

   // Reload the object
   ContainerLine_LoadOleObject(lpContainerLine);

   fMustActivate = TRUE;   // we should activate this object
}

/****************************************************************
**  OLE2NOTE: now we will try to change the display if
**    necessary.
****************************************************************/

if (lpContainerLine->m_lpOleObj &&
      ct.dvAspect != lpContainerLine->m_dwDrawAspect) {
         /* user has selected to change display aspect between icon
         **    aspect and content aspect.
         **
         ** OLE2NOTE: if we got here because the server was not
         **    registered, then we will NOT delete the object's
         **    original display aspect. because we do not have the
         **    original server, we can NEVER get it back. this is a
         **    safety precaution.
         */

         hrErr = OleStdSwitchDisplayAspect(
               lpContainerLine->m_lpOleObj,
               &lpContainerLine->m_dwDrawAspect,
               ct.dvAspect,
               ct.hMetaPict,
               !fServerNotRegistered,   /* fDeleteOldAspect */
               TRUE,                    /* fSetupViewAdvise */
               (LPADVISESINK)&lpContainerLine->m_AdviseSink,
               (BOOL FAR*)&fMustRun
         );

         if (hrErr == NOERROR)
            fDisplayChanged = TRUE;

#if defined( INPLACE_CNTR )
            ContainerDoc_UpdateInPlaceObjectRects(
                  lpContainerLine->m_lpDoc, nIndex);
#endif

      } else if (ct.dvAspect == DVASPECT_ICON && ct.fObjectsIconChanged) {
         hrErr = OleStdSetIconInCache(
               lpContainerLine->m_lpOleObj,
               ct.hMetaPict
         );

         if (hrErr == NOERROR)
            fDisplayChanged = TRUE;
      }

      /* we deliberately run the object so that the update won't shut
      ** the server down.
      */
      if (fMustActivate || fMustRun) {

         /* if we force the object to run, then shut it down after
         **    the update. do NOT force the object to close if we
         **    want to activate the object or if the object was
         **    already running.
         */
         if (!fMustActivate && !OleIsRunning(lpContainerLine->m_lpOleObj))
            fMustClose = TRUE;  // shutdown after update

         hrErr = ContainerLine_RunOleObject(lpContainerLine);

         if (fObjConverted &&
            FAILED(hrErr) && GetScode(hrErr)!=OLE_E_STATIC) {

            // ERROR: convert of the object failed.
            // revert the storage to restore the original link.
            // (OLE2NOTE: static object always return OLE_E_STATIC
            //        when told to run; this is NOT an error here.
            //        the OLE2 libraries have built in handlers for
            //        the static objects that do the conversion.
            ContainerLine_UnloadOleObject(
                  lpContainerLine, OLECLOSE_NOSAVE);
            lpContainerLine->m_lpStg->lpVtbl->Revert(
                  lpContainerLine->m_lpStg);
            goto error;

         } else if (fObjConverted) {
            FORMATETC  FmtEtc;
            DWORD      dwNewConnection;
            LPOLECACHE lpOleCache = (LPOLECACHE)OleStdQueryInterface
                 ((LPUNKNOWN)lpContainerLine->m_lpOleObj,&IID_IOleCache);

            /* OLE2NOTE: we need to force the converted object to
            **    setup a new OLERENDER_DRAW cache. it is possible
            **    that the new object needs to cache different data
            **    in order to support drawing than the old object.
            */
            if (lpOleCache &&
                  lpContainerLine->m_dwDrawAspect == DVASPECT_CONTENT) {
               FmtEtc.cfFormat = 0; //NULL; // whatever is needed for Draw
               FmtEtc.ptd      = NULL;
               FmtEtc.dwAspect = DVASPECT_CONTENT;
               FmtEtc.lindex   = -1;
               FmtEtc.tymed    = TYMED_NULL;

               OLEDBG_BEGIN2("IOleCache::Cache called\r\n")
               hrErr = lpOleCache->lpVtbl->Cache(
                     lpOleCache,
                     (LPFORMATETC)&FmtEtc,
                     ADVF_PRIMEFIRST,
                     (LPDWORD)&dwNewConnection
               );
               OLEDBG_END2
#if defined( _DEBUG )
               if (! SUCCEEDED(hrErr))
                  OleDbgOutHResult("IOleCache::Cache returned", hrErr);
#endif
               OleStdRelease((LPUNKNOWN)lpOleCache);
            }

            // Close and force object to save; this will commit the stg
            ContainerLine_CloseOleObject(
               lpContainerLine, OLECLOSE_SAVEIFDIRTY);
            fMustClose = FALSE;     // we already closed the object
         }
         if (fMustClose)
            ContainerLine_CloseOleObject(lpContainerLine,OLECLOSE_NOSAVE);
```

```
      }

      if (fDisplayChanged) {
         /* the Object's display was changed, force a repaint of
         **    the line. note the extents of the object may have
         **    changed.
         */
         ContainerLine_UpdateExtent(lpContainerLine, NULL);
         LineList_ForceLineRedraw(lpLL, nIndex, TRUE);
      }

      if (fDisplayChanged || fObjConverted) {
         /* mark ContainerDoc as now dirty. if display changed, then
         **    the extents of the object may have changed.
         */
         OutlineDoc_SetModified(lpOutlineDoc, TRUE, TRUE, fDisplayChanged);
      }

      if (fMustActivate) {
         ContainerLine_DoVerb(
               lpContainerLine, OLEIVERB_PRIMARY, NULL, FALSE,FALSE);
      }
   }


   if (ct.lpszUserType)
      OleStdFreeString(ct.lpszUserType, NULL);

   if (ct.lpszDefLabel)
      OleStdFreeString(ct.lpszDefLabel, NULL);

   if (ct.hMetaPict)
      OleUIMetafilePictIconFree(ct.hMetaPict);    // clean up metafile

   SetCursor(hPrevCursor);     // restore original cursor

   return;

error:
   if (ct.lpszUserType)
      OleStdFreeString(ct.lpszUserType, NULL);

   if (ct.hMetaPict)
      OleUIMetafilePictIconFree(ct.hMetaPict);    // clean up metafile

   SetCursor(hPrevCursor);     // restore original cursor
   if (lpErrMsg)
      OutlineApp_ErrorMessage(g_lpApp, lpErrMsg);
}


/* ContainerDoc_CloseAllOleObjects
** -------------------------------
**    Close all OLE objects. This forces all OLE objects to transition
**    from the running state to the loaded state.
```

```
**
**    Returns TRUE if all objects closed successfully
**            FALSE if any object could not be closed.
*/
BOOL ContainerDoc_CloseAllOleObjects(
      LPCONTAINERDOC          lpContainerDoc,
      DWORD                   dwSaveOption
)
{
   LPLINELIST  lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   int         i;
   LPLINE      lpLine;
   BOOL        fStatus = TRUE;

   for (i = 0; i < lpLL->m_nNumLines; i++) {
      lpLine=LineList_GetLine(lpLL, i);

      if (lpLine && (Line_GetLineType(lpLine)==CONTAINERLINETYPE))
         if (! ContainerLine_CloseOleObject(
                              (LPCONTAINERLINE)lpLine,dwSaveOption))
            fStatus = FALSE;
   }

   return fStatus;
}


/* ContainerDoc_UnloadAllOleObjectsOfClass
** ---------------------------------------
**    Unload all OLE objects of a particular class. this is necessary
**    when a class level "ActivateAs" (aka. TreatAs) is setup. the user
**    can do this with the Convert dialog. for the TreatAs to take
**    effect, all objects of the class have to loaded and reloaded.
*/
void ContainerDoc_UnloadAllOleObjectsOfClass(
      LPCONTAINERDOC      lpContainerDoc,
      REFCLSID            rClsid,
      DWORD               dwSaveOption
)
{
   LPLINELIST  lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   int         i;
   LPLINE      lpLine;
   CLSID       clsid;
   HRESULT     hrErr;


   for (i = 0; i < lpLL->m_nNumLines; i++) {
      lpLine=LineList_GetLine(lpLL, i);

      if (lpLine && (Line_GetLineType(lpLine)==CONTAINERLINETYPE)) {
         LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)lpLine;

         if (! lpContainerLine->m_lpOleObj)
            continue;       // this object is NOT loaded
         hrErr = lpContainerLine->m_lpOleObj->lpVtbl->GetUserClassID(
               lpContainerLine->m_lpOleObj,
               (CLSID FAR*)&clsid
         );
         if (hrErr == NOERROR &&
               ( IsEqualCLSID((CLSID FAR*)&clsid,rClsid)
                 || IsEqualCLSID(rClsid,&CLSID_NULL) ) ) {
            ContainerLine_UnloadOleObject(lpContainerLine, dwSaveOption);
         }
      }
   }
}


/* ContainerDoc_UpdateExtentOfAllOleObjects
** ----------------------------------------
**    Update the extents of any OLE object that is marked that its size
**    may  have changed. when an IAdviseSink::OnViewChange notification
**    is received, the corresponding ContainerLine is marked
**    (m_fDoGetExtent==TRUE) and a message (WM_U_UPDATEOBJECTEXTENT) is
**    posted to the document indicating that there are dirty objects.
**    when this message is received, this function is called.
*/
void ContainerDoc_UpdateExtentOfAllOleObjects(LPCONTAINERDOC lpContainerDoc)
{
   LPLINELIST  lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   int         i;
   LPLINE      lpLine;
   BOOL        fStatus = TRUE;
#if defined( INPLACE_CNTR )
   int         nFirstUpdate = -1;
#endif

   for (i = 0; i < lpLL->m_nNumLines; i++) {
      lpLine=LineList_GetLine(lpLL, i);

      if (lpLine && (Line_GetLineType(lpLine)==CONTAINERLINETYPE)) {
         LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)lpLine;

         if (lpContainerLine->m_fDoGetExtent) {
            ContainerLine_UpdateExtent(lpContainerLine, NULL);
#if defined( INPLACE_CNTR )
            if (nFirstUpdate == -1)
               nFirstUpdate = i;
#endif
         }
      }
   }

#if defined( INPLACE_CNTR )
   /* OLE2NOTE: after changing the extents of any line, we need to
   **    update the PosRect of the In-Place active
   **    objects (if any) that follow the first modified line.
   */
   if (nFirstUpdate != -1)
      ContainerDoc_UpdateInPlaceObjectRects(lpContainerDoc, nFirstUpdate+1);
#endif
}


BOOL ContainerDoc_SaveToFile(
      LPCONTAINERDOC          lpContainerDoc,
      LPCOLESTR               lpszFileName,
      UINT                    uFormat,
      BOOL                    fRemember
)
{
   LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpContainerDoc;
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpContainerDoc;
   LPSTORAGE lpDestStg;
   BOOL fStatus;
   BOOL fMustRelDestStg = FALSE;
   HRESULT hrErr;
#if defined( OPTIONAL )
   FILETIME filetimeBeforeSave;
#endif

   if (fRemember) {
      if (lpszFileName) {
         fStatus = OutlineDoc_SetFileName(
               lpOutlineDoc, (LPOLESTR)lpszFileName, NULL);
         if (! fStatus) goto error;
      }

      /* The ContainerDoc keeps its storage open at all times. it is not
      **    necessary to reopen the file.
      ** if SaveAs is pending, then lpNewStg is the new destination for
      **    the save operation, else the existing storage is the dest.
      */
      lpDestStg = (lpContainerDoc->m_lpNewStg ?
                  lpContainerDoc->m_lpNewStg : lpOleDoc->m_lpStg);

#if defined( OPTIONAL )
      /* OLE2NOTE: an automatic link to an embedded object within the
      **    same container document (that uses ItemMonikers) will
      **    always be considered "out-of-date' by OLE. if a container
      **    application REALLY wants to fix this it can do one of the
      **    following:
      **      1. implement a new moniker better than ItemMonikers
      **    that look into the objects storage to find the real last
      **    change time (rather then defaulting to that of the outer
      **    container file).
      ** or   2. using item monikers it is possible to fix the case
      **    where the container document is saved while the embedded
      **    object is running but it will NOT fix the case when the
      **    document is saved when the embedded object was only
      **    loaded. the fix is to:
      **      a. remember the time (T) before the save operation starts
      **      b. call IRunningObjectTable::NoteChangeTime(lpDoc, T)
      **      c. do the saving and commit the file
      **      d. call StgSetTimes to reset the file time to T
      **      e. remember time T in document structure and when the
      **         root storage is finally released reset the file time
      **         again to T (closing the file on DOS sets the time).
      */
      CoFileTimeNow( &filetimeBeforeSave );
      if (lpOleDoc->m_dwRegROT != 0) {
         LPRUNNINGOBJECTTABLE lprot;

         if (GetRunningObjectTable(0,&lprot) == NOERROR)
         {
            OleDbgOut2("IRunningObjectTable::NoteChangeTime called\r\n");
            lprot->lpVtbl->NoteChangeTime(
                  lprot, lpOleDoc->m_dwRegROT, &filetimeBeforeSave );
            lprot->lpVtbl->Release(lprot);
         }
      }
#endif
   } else {
      if (! lpszFileName)
         goto error;

      /* OLE2NOTE: since we are preforming a SaveCopyAs operation, we
      **    do not need to have the DocFile open in STGM_TRANSACTED mode.
      **    there is less overhead to use STGM_DIRECT mode.
      */
      hrErr = StgCreateDocfile(
            lpszFileName,
            STGM_READWRITE|STGM_DIRECT|STGM_SHARE_EXCLUSIVE|STGM_CREATE,
            0,
            &lpDestStg
      );
      OleDbgAssertSz(hrErr == NOERROR, "Could not create Docfile");
      if (hrErr != NOERROR) {
            OleDbgOutHResult("StgCreateDocfile returned", hrErr);
         goto error;
        }
      fMustRelDestStg = TRUE;
   }

   /*  OLE2NOTE: we must be sure to write our class ID into our
   **    storage. this information is used by OLE to determine the
   **    class of the data stored in our storage. Even for top
   **    "file-level" objects this information should be written to
   **    the file.
   */
   hrErr = WriteClassStg(lpDestStg, &CLSID_APP);
   if(hrErr != NOERROR) goto error;

   fStatus = OutlineDoc_SaveSelToStg(
         lpOutlineDoc,
         NULL,           // save all lines
         uFormat,
         lpDestStg,
         FALSE,          // fSameAsLoad
         TRUE            // remember this stg
      );

   if (fStatus)
      fStatus = OleStdCommitStorage(lpDestStg);

   if (fRemember) {
      /* if SaveAs was pending, then release the old storage and remember
      **    the new storage as the active current storage. all data from
      **    the old storage has been copied into the new storage.
      */
      if (lpContainerDoc->m_lpNewStg) {
         OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpStg);  // free old stg
         lpOleDoc->m_lpStg = lpContainerDoc->m_lpNewStg;   // save new stg
         lpContainerDoc->m_lpNewStg = NULL;
      }
      if (! fStatus) goto error;

      OutlineDoc_SetModified(lpOutlineDoc, FALSE, FALSE, FALSE);

#if defined( OPTIONAL )
      /* reset time of file on disk to be time just prior to saving.
      ** NOTE: it would also be necessary to remember
      **    filetimeBeforeSave in the document structure and when the
      **    root storage is finally released reset the file time
      **    again to this value (closing the file on DOS sets the time).
      */
      StgSetTimes(
            lpOutlineDoc->m_szFileName, NULL, NULL, &filetimeBeforeSave);
#endif
   }

   if (fMustRelDestStg)
      OleStdRelease((LPUNKNOWN)lpDestStg);
   return TRUE;

error:
   if (fMustRelDestStg)
      OleStdRelease((LPUNKNOWN)lpDestStg);
   OutlineApp_ErrorMessage(g_lpApp, ErrMsgSaving);
   return FALSE;
}


/* ContainerDoc_ContainerLineDoVerbCommand
** ---------------------------------------
**    Execute a verb of the OLE object in the current focus line.
*/
void ContainerDoc_ContainerLineDoVerbCommand(
      LPCONTAINERDOC          lpContainerDoc,
      LONG                    iVerb
)
{
   LPLINELIST lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   int nIndex = LineList_GetFocusLineIndex(lpLL);
   LPLINE lpLine = LineList_GetLine(lpLL, nIndex);
   HCURSOR                  hPrevCursor;

   if (! lpLine || (Line_GetLineType(lpLine) != CONTAINERLINETYPE) ) return;

   // this may take a while, put up hourglass cursor
   hPrevCursor = SetCursor(LoadCursor(NULL, IDC_WAIT));

   ContainerLine_DoVerb((LPCONTAINERLINE) lpLine, iVerb, NULL, TRUE, TRUE);

   SetCursor(hPrevCursor);     // restore original cursor
}


/* ContainerDoc_GetNextStgName
** ---------------------------
**    Generate the next unused name for a sub-storage to be used by an
**    OLE object. The ContainerDoc keeps a counter. The storages for
**    OLE objects are simply numbered (eg. Obj 0, Obj 1). A "long"
**    integer worth of storage names should be more than enough than we
**    will ever need.
**
**    NOTE: when an OLE object is transfered via drag/drop or the
**    clipboard, we attempt to keep the currently assigned name for the
**    object (if not currently in use). thus it is possible that an
**    object with a the next default name (eg. "Obj 5") already exists
**    in the current document if an object with this name was privously
**    transfered (pasted or dropped). we therefore loop until we find
**    the next lowest unused name.
*/
void ContainerDoc_GetNextStgName(
      LPCONTAINERDOC          lpContainerDoc,
      LPOLESTR                lpszStgName,
      int                     nLen
)
{
   char szAnsiStgName[256];

   wsprintf(szAnsiStgName, "%s %ld",
         (LPSTR)DEFOBJNAMEPREFIX,
         ++(lpContainerDoc->m_nNextObjNo)
   );

   while (ContainerDoc_IsStgNameUsed(lpContainerDoc, lpszStgName) == TRUE) {
      wsprintf(szAnsiStgName, "%s %ld",
            (LPSTR)DEFOBJNAMEPREFIX,
            ++(lpContainerDoc->m_nNextObjNo)
      );
   }
   A2W (szAnsiStgName, lpszStgName, nLen);
}
```

```
/* ContainerDoc_IsStgNameUsed
** --------------------------
**    Check if a given StgName is already in use.
*/
BOOL ContainerDoc_IsStgNameUsed(
      LPCONTAINERDOC          lpContainerDoc,
      LPOLESTR                lpszStgName
)
{
   LPLINELIST lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   int i;
   LPLINE lpLine;

   for (i = 0; i < lpLL->m_nNumLines; i++) {
      lpLine=LineList_GetLine(lpLL, i);

      if (lpLine && (Line_GetLineType(lpLine)==CONTAINERLINETYPE)) {
         if (OLESTRCMP(lpszStgName,
               ((LPCONTAINERLINE)lpLine)->m_szStgName) == 0) {
            return TRUE;    // Match FOUND!
         }
      }
   }
   return FALSE;   // if we get here, then NO match was found.
}


LPSTORAGE ContainerDoc_GetStg(LPCONTAINERDOC lpContainerDoc)
{
   return ((LPOLEDOC)lpContainerDoc)->m_lpStg;
}


/* ContainerDoc_GetSingleOleObject
** -------------------------------
**    If the entire document contains a single OLE object, then
**    return the desired interface of the object.
**
**    Returns NULL if there is are multiple lines in the document or
**    the single line is not a ContainerLine.
*/
LPUNKNOWN ContainerDoc_GetSingleOleObject(
      LPCONTAINERDOC          lpContainerDoc,
      REFIID                  riid,
      LPCONTAINERLINE FAR*    lplpContainerLine
)
{
   LPLINELIST lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   LPLINE lpLine;
   LPUNKNOWN lpObj = NULL;


   if (lplpContainerLine)
      *lplpContainerLine = NULL;
   if (lpLL->m_nNumLines != 1)
   {
      return NULL;    // doc does NOT contain a single line
   }

   lpLine=LineList_GetLine(lpLL, 0);

   if (lpLine && (Line_GetLineType(lpLine)==CONTAINERLINETYPE))
      lpObj = ContainerLine_GetOleObject((LPCONTAINERLINE)lpLine, riid);

   if (lplpContainerLine)
      *lplpContainerLine = (LPCONTAINERLINE)lpLine;

   return lpObj;
}


/* ContainerDoc_IsSelAnOleObject
** -----------------------------
**    Check if the selection is a single selection of an OLE object.
**    if so, then optionally return the desired interface of the object
**    and/or index of the ContainerLine containing the OLE object.
**
**    Returns FALSE if there is a multiple selection or the single
**    selection is not a ContainerLine.
*/
BOOL ContainerDoc_IsSelAnOleObject(
      LPCONTAINERDOC          lpContainerDoc,
      REFIID                  riid,
      LPUNKNOWN FAR*          lplpvObj,
      int FAR*                lpnIndex,
      LPCONTAINERLINE FAR*    lplpContainerLine
)
{
   LPLINELIST lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   LINERANGE lrSel;
   int nNumSel;
   LPLINE lpLine;


   if (lplpvObj) *lplpvObj = NULL;
   if (lpnIndex) *lpnIndex = -1;
   if (lplpContainerLine) *lplpContainerLine = NULL;

   nNumSel = LineList_GetSel(lpLL, (LPLINERANGE)&lrSel);
   if (nNumSel != 1)
   {
      return FALSE;   // selection is not a single line
   }

   lpLine = LineList_GetLine(lpLL, lrSel.m_nStartLine);

   if (lpLine && (Line_GetLineType(lpLine)==CONTAINERLINETYPE)) {
      if (lpnIndex)
         *lpnIndex = lrSel.m_nStartLine;
      if (lplpContainerLine)
         *lplpContainerLine = (LPCONTAINERLINE)lpLine;
      if (riid) {
         *lplpvObj = ContainerLine_GetOleObject(
               (LPCONTAINERLINE)lpLine,
               riid
         );
      }

      return (*lplpvObj ? TRUE : FALSE);
   }

   return FALSE;
}


/*************************************************************************
** ContainerDoc::IOleUILinkContainer interface implementation
*************************************************************************/

STDMETHODIMP CntrDoc_LinkCont_QueryInterface(
      LPOLEUILINKCONTAINER    lpThis,
      REFIID                  riid,
      LPVOID FAR*             lplpvObj
)
{
   LPOLEDOC lpOleDoc = (LPOLEDOC)
         ((struct CDocOleUILinkContainerImpl FAR*)lpThis)->lpContainerDoc;

   return OleDoc_QueryInterface(lpOleDoc, riid, lplpvObj);
}


STDMETHODIMP_(ULONG) CntrDoc_LinkCont_AddRef(LPOLEUILINKCONTAINER lpThis)
{
   LPOLEDOC lpOleDoc = (LPOLEDOC)
         ((struct CDocOleUILinkContainerImpl FAR*)lpThis)->lpContainerDoc;

   OleDbgAddRefMethod(lpThis, OLESTR("IOleUILinkContainer"));

   return OleDoc_AddRef(lpOleDoc);
}


STDMETHODIMP_(ULONG) CntrDoc_LinkCont_Release(LPOLEUILINKCONTAINER lpThis)
{
   LPOLEDOC lpOleDoc = (LPOLEDOC)
         ((struct CDocOleUILinkContainerImpl FAR*)lpThis)->lpContainerDoc;

   OleDbgReleaseMethod(lpThis, "IOleUILinkContainer");

   return OleDoc_Release(lpOleDoc);
}
```

```
STDMETHODIMP_(DWORD) CntrDoc_LinkCont_GetNextLink(
      LPOLEUILINKCONTAINER    lpThis,
      DWORD                   dwLink
)
{
   LPCONTAINERDOC lpContainerDoc =
         ((struct CDocOleUILinkContainerImpl FAR*)lpThis)->lpContainerDoc;
   LPCONTAINERLINE lpContainerLine = NULL;

    // artificial AddRef in case object is released
    // during this call
    CntrDoc_LinkCont_AddRef(lpThis);

   OLEDBG_BEGIN2("CntrDoc_LinkCont_GetNextLink\r\n")

   lpContainerLine = ContainerDoc_GetNextLink(
         lpContainerDoc,
         (LPCONTAINERLINE)dwLink
   );

   OLEDBG_END2

   // release artificial AddRef
   CntrDoc_LinkCont_Release(lpThis);

   return (DWORD)lpContainerLine;
}


STDMETHODIMP CntrDoc_LinkCont_SetLinkUpdateOptions(
      LPOLEUILINKCONTAINER    lpThis,
      DWORD                   dwLink,
      DWORD                   dwUpdateOpt
)
{
   LPCONTAINERDOC lpContainerDoc =
         ((struct CDocOleUILinkContainerImpl FAR*)lpThis)->lpContainerDoc;
   LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)dwLink;
   LPOLELINK lpOleLink = lpContainerLine->m_lpOleLink;
   SCODE sc = S_OK;
   HRESULT hrErr;

   // artificial AddRef in case object is released during call
   CntrDoc_LinkCont_AddRef(lpThis);

   OLEDBG_BEGIN2("CntrDoc_LinkCont_SetLinkUpdateOptions\r\n")

   OleDbgAssert(lpContainerLine);

   if (! lpOleLink) {
      sc = E_FAIL;
      goto error;
   }

   OLEDBG_BEGIN2("IOleLink::SetUpdateOptions called\r\n")
   hrErr = lpOleLink->lpVtbl->SetUpdateOptions(
         lpOleLink,
         dwUpdateOpt
   );
   OLEDBG_END2

   // save new link type update option
   lpContainerLine->m_dwLinkType = dwUpdateOpt;

   if (hrErr != NOERROR) {
      OleDbgOutHResult("IOleLink::SetUpdateOptions returned", hrErr);
      sc = GetScode(hrErr);
      goto error;
   }

error:
   // release artificial AddRef
   CntrDoc_LinkCont_Release(lpThis);

   OLEDBG_END2
   return ResultFromScode(sc);
}


STDMETHODIMP CntrDoc_LinkCont_GetLinkUpdateOptions(
      LPOLEUILINKCONTAINER    lpThis,
      DWORD                   dwLink,
      DWORD FAR*              lpdwUpdateOpt
)
{
   LPCONTAINERDOC lpContainerDoc =
         ((struct CDocOleUILinkContainerImpl FAR*)lpThis)->lpContainerDoc;
   LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)dwLink;
   LPOLELINK lpOleLink = lpContainerLine->m_lpOleLink;
   SCODE sc = S_OK;
   HRESULT hrErr;

   // artificial AddRef in case object is released during call
   CntrDoc_LinkCont_AddRef(lpThis);

   OLEDBG_BEGIN2("CntrDoc_LinkCont_GetLinkUpdateOptions\r\n")

   OleDbgAssert(lpContainerLine);

   if (! lpOleLink) {
      sc = E_FAIL;
      goto error;
   }

   OLEDBG_BEGIN2("IOleLink::GetUpdateOptions called\r\n")
   hrErr = lpOleLink->lpVtbl->GetUpdateOptions(
         lpOleLink,
         lpdwUpdateOpt
   );
   OLEDBG_END2
```

```
    // reset saved link type to ensure it is correct
    lpContainerLine->m_dwLinkType = *lpdwUpdateOpt;

    if (hrErr != NOERROR) {
       OleDbgOutHResult("IOleLink::GetUpdateOptions returned", hrErr);
       sc = GetScode(hrErr);
       goto error;
    }

error:
    OLEDBG_END2

    // release artificial AddRef
    CntrDoc_LinkCont_Release(lpThis);

    return ResultFromScode(sc);
}


STDMETHODIMP CntrDoc_LinkCont_SetLinkSource(
      LPOLEUILINKCONTAINER    lpThis,
      DWORD                   dwLink,
      LPOLESTR                lpszDisplayName,
      ULONG                   lenFileName,
      ULONG FAR*              lpchEaten,
      BOOL                    fValidateSource
)
{
    LPCONTAINERDOC lpContainerDoc =
          ((struct CDocOleUILinkContainerImpl FAR*)lpThis)->lpContainerDoc;
    LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)dwLink;
    SCODE       sc = S_OK;
    HRESULT     hrErr;
    LPOLELINK   lpOleLink = lpContainerLine->m_lpOleLink;
    LPBC        lpbc = NULL;
    LPMONIKER   lpmk = NULL;
    LPOLEOBJECT lpLinkSrcOleObj = NULL;
    CLSID       clsid = CLSID_NULL;
    CLSID       clsidOld = CLSID_NULL;


    // artificial AddRef in case object is released during call
    CntrDoc_LinkCont_AddRef(lpThis);

    OLEDBG_BEGIN2("CntrDoc_LinkCont_SetLinkSource\r\n")

    OleDbgAssert(lpContainerLine);

    lpContainerLine->m_fLinkUnavailable = TRUE;

    if (fValidateSource) {

       /* OLE2NOTE: validate the link source by parsing the string
       **    into a Moniker. if this is successful, then the string is
       **    valid.
       */
       hrErr = CreateBindCtx(0, (LPBC FAR*)&lpbc);
       if (hrErr != NOERROR) {
          sc = GetScode(hrErr);   // ERROR: OOM
          goto cleanup;
       }

       // Get class of orignial link source if it is available
       if (lpContainerLine->m_lpOleObj) {

          OLEDBG_BEGIN2("IOleObject::GetUserClassID called\r\n")
          hrErr = lpContainerLine->m_lpOleObj->lpVtbl->GetUserClassID(
                lpContainerLine->m_lpOleObj, (CLSID FAR*)&clsidOld);
          OLEDBG_END2
          if (hrErr != NOERROR) clsidOld = CLSID_NULL;
       }

       hrErr = OleStdMkParseDisplayName(
            &clsidOld,lpbc,lpszDisplayName,lpchEaten,(LPMONIKER FAR*)&lpmk);

       if (hrErr != NOERROR) {
          sc = GetScode(hrErr);   // ERROR in parsing moniker!
          goto cleanup;
       }
       /* OLE2NOTE: the link source was validated; it successfully
       **    parsed into a Moniker. we can set the source of the link
       **    directly with this Moniker. if we want the link to be
       **    able to know the correct class for the new link source,
       **    we must bind to the moniker and get the CLSID. if we do
       **    not do this then methods like IOleObject::GetUserType
       **    will return nothing (NULL strings).
       */

       hrErr = lpmk->lpVtbl->BindToObject(
             lpmk,lpbc,NULL,&IID_IOleObject,(LPVOID FAR*)&lpLinkSrcOleObj);
       if (hrErr == NOERROR) {
          OLEDBG_BEGIN2("IOleObject::GetUserClassID called\r\n")
          hrErr = lpLinkSrcOleObj->lpVtbl->GetUserClassID(
                      lpLinkSrcOleObj, (CLSID FAR*)&clsid);
          OLEDBG_END2
          lpContainerLine->m_fLinkUnavailable = FALSE;

          /* get the short user type name of the link because it may
          **    have changed. we cache this name and must update our
          **    cache. this name is used all the time when we have to
          **    build the object verb menu. we cache this information
          **    to make it quicker to build the verb menu.
          */
          if (lpContainerLine->m_lpszShortType) {
             OleStdFree(lpContainerLine->m_lpszShortType);
             lpContainerLine->m_lpszShortType = NULL;
          }
          OLEDBG_BEGIN2("IOleObject::GetUserType called\r\n")
          lpContainerLine->m_lpOleObj->lpVtbl->GetUserType(
                lpContainerLine->m_lpOleObj,
                USERCLASSTYPE_SHORT,
                (LPOLESTR FAR*)&lpContainerLine->m_lpszShortType
          );
          OLEDBG_END2
       }
       else
          lpContainerLine->m_fLinkUnavailable = TRUE;
   }
   else {
       LPMONIKER   lpmkFile = NULL;
       LPMONIKER   lpmkItem = NULL;
       OLECHAR     szDelim[2];
       LPOLESTR    lpszName;

       szDelim[0] = lpszDisplayName[(int)lenFileName];
       szDelim[1] = '\0';
       lpszDisplayName[(int)lenFileName] = '\0';

       OLEDBG_BEGIN2("CreateFileMoniker called\r\n")
       CreateFileMoniker((LPOLESTR)lpszDisplayName, (LPMONIKER
FAR*)&lpmkFile);
       OLEDBG_END2

       lpszDisplayName[(int)lenFileName] = szDelim[0];

       if (!lpmkFile)
          goto cleanup;

       if (OLESTRLEN(lpszDisplayName) > (int)lenFileName) {  // have item
name
          lpszName = lpszDisplayName + lenFileName + 1;

          OLEDBG_BEGIN2("CreateItemMoniker called\r\n")
          CreateItemMoniker(
                szDelim, lpszName, (LPMONIKER FAR*)&lpmkItem);
          OLEDBG_END2

          if (!lpmkItem) {
             OleStdRelease((LPUNKNOWN)lpmkFile);
             goto cleanup;
          }

          OLEDBG_BEGIN2("CreateGenericComposite called\r\n")
          CreateGenericComposite(lpmkFile, lpmkItem, (LPMONIKER FAR*)&lpmk);
          OLEDBG_END2

          if (lpmkFile)
             OleStdRelease((LPUNKNOWN)lpmkFile);
          if (lpmkItem)
             OleStdRelease((LPUNKNOWN)lpmkItem);

          if (!lpmk)
             goto cleanup;
       }
```

```
      else
         lpmk = lpmkFile;
   }

   if (! lpOleLink) {
      OleDbgAssert(lpOleLink != NULL);
      sc = E_FAIL;
      goto cleanup;
   }

   if (lpmk) {

      OLEDBG_BEGIN2("IOleLink::SetSourceMoniker called\r\n")
      hrErr = lpOleLink->lpVtbl->SetSourceMoniker(
            lpOleLink, lpmk, (REFCLSID)&clsid);
      OLEDBG_END2

      if (FAILED(GetScode(hrErr))) {
         OleDbgOutHResult("IOleLink::SetSourceMoniker returned",hrErr);
         sc = GetScode(hrErr);
         goto cleanup;
      }

      /* OLE2NOTE: above we forced the link source moniker to bind.
      **    because we deliberately hold on to the bind context
      **    (lpbc) the link source object will not shut down. during
      **    the call to IOleLink::SetSourceMoniker, the link will
      **    connect to the running link source (the link internally
      **    calls BindIfRunning). it is important to initially allow
      **    the link to bind to the running object so that it can get
      **    an update of the presentation for its cache. we do not
      **    want the connection from our link to the link source be
      **    the only reason the link source stays running. thus we
      **    deliberately for the link to release (unbind) the source
      **    object, we then release the bind context, and then we
      **    allow the link to rebind to the link source if it is
      **    running anyway.
      */
      if (lpbc && lpmk->lpVtbl->IsRunning(lpmk,lpbc,NULL,NULL) == NOERROR) {

         OLEDBG_BEGIN2("IOleLink::Update called\r\n")
         hrErr = lpOleLink->lpVtbl->Update(lpOleLink, NULL);
         OLEDBG_END2

#if defined( _DEBUG )
         if (FAILED(GetScode(hrErr)))
            OleDbgOutHResult("IOleLink::Update returned",hrErr);
#endif

         OLEDBG_BEGIN2("IOleLink::UnbindSource called\r\n")
         hrErr = lpOleLink->lpVtbl->UnbindSource(lpOleLink);
         OLEDBG_END2

#if defined( _DEBUG )
         if (FAILED(GetScode(hrErr)))
```

```
            OleDbgOutHResult("IOleLink::UnbindSource returned",hrErr);
#endif

         if (lpLinkSrcOleObj) {
            OleStdRelease((LPUNKNOWN)lpLinkSrcOleObj);
            lpLinkSrcOleObj = NULL;
         }

         if (lpbc) {
            OleStdRelease((LPUNKNOWN)lpbc);
            lpbc = NULL;
         }

         OLEDBG_BEGIN2("IOleLink::BindIfRunning called\r\n")
         hrErr = lpOleLink->lpVtbl->BindIfRunning(lpOleLink);
         OLEDBG_END2

#if defined( _DEBUG )
         if (FAILED(GetScode(hrErr)))
            OleDbgOutHResult("IOleLink::BindIfRunning returned",hrErr);
#endif
      }
   } else {
      /* OLE2NOTE: the link source was NOT validated; it was NOT
      **    successfully parsed into a Moniker. we can only set the
      **    display name string as the source of the link. this link
      **    is not able to bind.
      */
      OLEDBG_BEGIN2("IOleLink::SetSourceDisplayName called\r\n")
      hrErr = lpOleLink->lpVtbl->SetSourceDisplayName(
            lpOleLink, (LPCOLESTR)lpszDisplayName);
      OLEDBG_END2

      if (hrErr != NOERROR) {
         OleDbgOutHResult("IOleLink::SetSourceDisplayName returned",hrErr);
         sc = GetScode(hrErr);
         goto cleanup;
      }
   }

cleanup:
   if (lpLinkSrcOleObj)
      OleStdRelease((LPUNKNOWN)lpLinkSrcOleObj);
   if (lpmk)
      OleStdRelease((LPUNKNOWN)lpmk);
   if (lpbc)
      OleStdRelease((LPUNKNOWN)lpbc);

   OLEDBG_END2
   // release artificial AddRef
   CntrDoc_LinkCont_Release(lpThis);

   return ResultFromScode(sc);
}
```

```
STDMETHODIMP CntrDoc_LinkCont_GetLinkSource(
      LPOLEUILINKCONTAINER    lpThis,
      DWORD                   dwLink,
      LPOLESTR FAR*           lplpszDisplayName,
      ULONG FAR*              lplenFileName,
      LPOLESTR FAR*           lplpszFullLinkType,
      LPOLESTR FAR*           lplpszShortLinkType,
      BOOL FAR*               lpfSourceAvailable,
      BOOL FAR*               lpfIsSelected
)
{
   LPCONTAINERDOC lpContainerDoc =
         ((struct CDocOleUILinkContainerImpl FAR*)lpThis)->lpContainerDoc;
   LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)dwLink;
   LPOLELINK       lpOleLink = lpContainerLine->m_lpOleLink;
   LPOLEOBJECT     lpOleObj = NULL;
   LPMONIKER       lpmk = NULL;
   LPMONIKER       lpmkFirst = NULL;
   LPBC            lpbc = NULL;
   SCODE           sc = S_OK;
   HRESULT         hrErr;

   // artificial AddRef in case object is released during call
   CntrDoc_LinkCont_AddRef(lpThis);

   OLEDBG_BEGIN2("CntrDoc_LinkCont_GetLinkSource\r\n")

   /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
   *lplpszDisplayName  = NULL;
   *lplpszFullLinkType = NULL;
   *lplpszShortLinkType= NULL;
   *lplenFileName      = 0;
   *lpfSourceAvailable = !lpContainerLine->m_fLinkUnavailable;

   OleDbgAssert(lpContainerLine);

   if (! lpOleLink) {
      OLEDBG_END2
      sc = E_FAIL;
      goto cleanup;
   }

   OLEDBG_BEGIN2("IOleLink::GetSourceMoniker called\r\n")
   hrErr = lpOleLink->lpVtbl->GetSourceMoniker(
         lpOleLink,
         (LPMONIKER FAR*)&lpmk
   );
   OLEDBG_END2

   if (hrErr == NOERROR) {
      /* OLE2NOTE: the link has the Moniker form of the link source;
      **    this is therefore a validated link source. if the first
      **    part of the Moniker is a FileMoniker, then we need to
      **    return the length of the filename string. we need to
      **    return the ProgID associated with the link source as the
      **    "lpszShortLinkType". we need to return the
      **    FullUserTypeName associated with the link source as the
      **    "lpszFullLinkType".
      */

      lpOleObj = (LPOLEOBJECT)OleStdQueryInterface(
            (LPUNKNOWN)lpOleLink, &IID_IOleObject);
      if (lpOleObj) {
         lpOleObj->lpVtbl->GetUserType(
               lpOleObj,
               USERCLASSTYPE_FULL,
               lplpszFullLinkType
         );
         lpOleObj->lpVtbl->GetUserType(
               lpOleObj,
               USERCLASSTYPE_SHORT,
               lplpszShortLinkType
         );
         OleStdRelease((LPUNKNOWN)lpOleObj);
      }
      *lplenFileName = OleStdGetLenFilePrefixOfMoniker(lpmk);
      lpmk->lpVtbl->Release(lpmk);
   }

   OLEDBG_BEGIN2("IOleLink::GetSourceDisplayName called\r\n")
   hrErr = lpOleLink->lpVtbl->GetSourceDisplayName(
         lpOleLink,
         lplpszDisplayName
   );
   OLEDBG_END2

   if (hrErr != NOERROR) {
      OleDbgOutHResult("IOleLink::GetSourceDisplayName returned", hrErr);
      OLEDBG_END2
      sc = hrErr;
      goto cleanup;
   }

   OLEDBG_END2

   if (lpfIsSelected)
      *lpfIsSelected = Line_IsSelected((LPLINE)lpContainerLine);

   sc = NOERROR;

cleanup:
   // release artificial AddRef
   CntrDoc_LinkCont_Release(lpThis);

   return sc;
}


STDMETHODIMP CntrDoc_LinkCont_OpenLinkSource(
      LPOLEUILINKCONTAINER    lpThis,
      DWORD                   dwLink
)
{
   LPCONTAINERDOC lpContainerDoc =
         ((struct CDocOleUILinkContainerImpl FAR*)lpThis)->lpContainerDoc;
   LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)dwLink;
   SCODE sc = S_OK;

   // artificial AddRef in case object is destroyed during call
   CntrDoc_LinkCont_AddRef(lpThis);

   OLEDBG_BEGIN2("CntrDoc_LinkCont_OpenLinkSource\r\n")

   OleDbgAssert(lpContainerLine);

   if (! ContainerLine_DoVerb(
         lpContainerLine, OLEIVERB_SHOW, NULL, TRUE, FALSE)) {
      sc = E_FAIL;
   }

   lpContainerLine->m_fLinkUnavailable = (sc != S_OK);

   OLEDBG_END2

   // release artificial AddRef
   CntrDoc_LinkCont_Release(lpThis);

   return ResultFromScode(sc);
}


STDMETHODIMP CntrDoc_LinkCont_UpdateLink(
      LPOLEUILINKCONTAINER    lpThis,
      DWORD                   dwLink,
      BOOL                    fErrorMessage,
      BOOL                    fErrorAction        // ignore if fErrorMessage
                                       //      is FALSE
)
{
   LPCONTAINERDOC lpContainerDoc =
         ((struct CDocOleUILinkContainerImpl FAR*)lpThis)->lpContainerDoc;
   LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)dwLink;
   SCODE sc = S_OK;
   // default to update of the link
   HRESULT hrErr = S_FALSE;

   // artificial AddRef in case object is destroyed during call
   CntrDoc_LinkCont_AddRef(lpThis);

   OLEDBG_BEGIN2("CntrDoc_LinkCont_UpdateLink\r\n")

   OleDbgAssert(lpContainerLine);

   if (! lpContainerLine->m_lpOleObj)
```

```
      ContainerLine_LoadOleObject(lpContainerLine);

   if (!fErrorMessage) {
      OLEDBG_BEGIN2("IOleObject::IsUpToDate called\r\n")
      hrErr = lpContainerLine->m_lpOleObj->lpVtbl->IsUpToDate(
            lpContainerLine->m_lpOleObj
      );
      OLEDBG_END2
   }

   if (hrErr != NOERROR) {
      OLEDBG_BEGIN2("IOleObject::Update called\r\n")
      hrErr = lpContainerLine->m_lpOleObj->lpVtbl->Update(
            lpContainerLine->m_lpOleObj
      );
      OLEDBG_END2
   }

   /* OLE2NOTE: If IOleObject::Update on the Link object returned
   **    OLE_E_CLASSDIFF because the link source is no longer
   **    the expected class, then the link should be re-created with
   **    the new link source. thus the link will be updated with the
   **    new link source.
   */
   if (GetScode(hrErr) == OLE_E_CLASSDIFF)
      hrErr = ContainerLine_ReCreateLinkBecauseClassDiff(lpContainerLine);

   lpContainerLine->m_fLinkUnavailable = (hrErr != NOERROR);

   if (hrErr != NOERROR) {
      OleDbgOutHResult("IOleObject::Update returned", hrErr);
      sc = GetScode(hrErr);
      if (fErrorMessage) {
         ContainerLine_ProcessOleRunError(
               lpContainerLine,hrErr,fErrorAction,FALSE/*fMenuInvoked*/);
      }
   }
   /* OLE2NOTE: if the update of the object requires us to update our
   **    display, then we will automatically be sent a OnViewChange
   **    advise. thus we do not need to take any action here to force
   **    a repaint.
   */

   OLEDBG_END2

   // release artificial AddRef
   CntrDoc_LinkCont_Release(lpThis);

   return ResultFromScode(sc);
}


/* CntrDoc_LinkCont_CancelLink
** ---------------------------
**    Convert the link to a static picture.
```

```
**
**    OLE2NOTE: OleCreateStaticFromData can be used to create a static
**    picture object.
*/
STDMETHODIMP CntrDoc_LinkCont_CancelLink(
      LPOLEUILINKCONTAINER    lpThis,
      DWORD                   dwLink
)
{
   LPCONTAINERDOC lpContainerDoc =
         ((struct CDocOleUILinkContainerImpl FAR*)lpThis)->lpContainerDoc;
   LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)dwLink;
   LPLINELIST          lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   LPLINE              lpLine = NULL;
   HDC                 hDC;
   int                 nTab = 0;
   OLECHAR             szStgName[CWCSTORAGENAME];
   LPCONTAINERLINE     lpNewContainerLine = NULL;
   LPDATAOBJECT        lpSrcDataObj;
   LPOLELINK           lpOleLink = lpContainerLine->m_lpOleLink;
   int nIndex = LineList_GetLineIndex(lpLL, (LPLINE)lpContainerLine);

   // artificial AddRef in case object is destroyed during call
   CntrDoc_LinkCont_AddRef(lpThis);

   OLEDBG_BEGIN2("CntrDoc_LinkCont_CancelLink\r\n")

   /* we will first break the connection of the link to its source. */
   if (lpOleLink) {
      lpContainerLine->m_dwLinkType = 0;
      OLEDBG_BEGIN2("IOleLink::SetSourceMoniker called\r\n")
      lpOleLink->lpVtbl->SetSourceMoniker(
            lpOleLink, NULL, (REFCLSID)&CLSID_NULL);
      OLEDBG_END2
   }

   lpSrcDataObj = (LPDATAOBJECT)ContainerLine_GetOleObject(
         lpContainerLine,&IID_IDataObject);
   if (! lpSrcDataObj)
      goto error;

   ContainerDoc_GetNextStgName(lpContainerDoc, szStgName, CWCSTORAGENAME);
   nTab = Line_GetTabLevel((LPLINE)lpContainerLine);
   hDC = LineList_GetDC(lpLL);

   lpNewContainerLine = ContainerLine_CreateFromData(
         hDC,
         nTab,
         lpContainerDoc,
         lpSrcDataObj,
         OLECREATEFROMDATA_STATIC,
         0,   /* no special cfFormat required */
         (lpContainerLine->m_dwDrawAspect == DVASPECT_ICON),
         NULL,   /* hMetaPict */
         szStgName
   );
   LineList_ReleaseDC(lpLL, hDC);

   OleStdRelease((LPUNKNOWN)lpSrcDataObj);

   if (! lpNewContainerLine)
      goto error;

   OutlineDoc_SetModified((LPOUTLINEDOC)lpContainerDoc, TRUE, TRUE, FALSE);

   LineList_ReplaceLine(lpLL, (LPLINE)lpNewContainerLine, nIndex);

   OLEDBG_END2
   // release artificial AddRef
   CntrDoc_LinkCont_Release(lpThis);

   return ResultFromScode(NOERROR);

error:
   OutlineApp_ErrorMessage(g_lpApp, OLESTR("Could not break the link."));
   OLEDBG_END2
   // release artificial AddRef
   CntrDoc_LinkCont_Release(lpThis);

   return ResultFromScode(E_FAIL);
}
```

CNTRINPL.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Container Sample Code
**
**    cntrinpl.c
**
**    This file contains all interfaces, methods and related support
**    functions for an In-Place Container application (aka. Visual
**    Editing). The in-place Container application includes the following
**    implementation objects:
**
**    ContainerApp Object
**      exposed interfaces:
**          IOleInPlaceFrame
**
**    ContainerDoc Object
**      support functions only
**      (ICntrOtl is an SDI app; it doesn't support a Doc level
IOleUIWindow)
**
**    ContainerLin Object
**      exposed interfaces:
**          IOleInPlaceSite
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"
#if defined( USE_STATUSBAR )
#include "status.h"
#endif

OLEDBGDATA

extern LPOUTLINEAPP g_lpApp;
extern BOOL g_fInsideOutContainer;
extern RECT g_rectNull;

/*************************************************************************
** ContainerApp::IOleInPlaceFrame interface implementation
*************************************************************************/

// IOleInPlaceFrame::QueryInterface
STDMETHODIMP CntrApp_IPFrame_QueryInterface(
      LPOLEINPLACEFRAME   lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
)
{
   SCODE sc = E_NOINTERFACE;
   LPCONTAINERAPP lpContainerApp =
         ((struct COleInPlaceFrameImpl FAR*)lpThis)->lpContainerApp;

   /* The object should not be able to access the other interfaces
   ** of our App object by doing QI on this interface.
   */
   *lplpvObj = NULL;
   if (IsEqualIID(riid, &IID_IUnknown) ||
      IsEqualIID(riid, &IID_IOleWindow) ||
      IsEqualIID(riid, &IID_IOleInPlaceUIWindow) ||
      IsEqualIID(riid, &IID_IOleInPlaceFrame)) {
      OleDbgOut4("CntrApp_IPFrame_QueryInterface: IOleInPlaceFrame* 
RETURNED\r\n");
      *lplpvObj = (LPVOID) &lpContainerApp->m_OleInPlaceFrame;
      OleApp_AddRef((LPOLEAPP)lpContainerApp);
      sc = S_OK;
   }

   OleDbgQueryInterfaceMethod(*lplpvObj);

   return ResultFromScode(sc);
}


// IOleInPlaceFrame::AddRef
STDMETHODIMP_(ULONG) CntrApp_IPFrame_AddRef(LPOLEINPLACEFRAME lpThis)
{
   OleDbgAddRefMethod(lpThis, "IOleInPlaceFrame");

   return OleApp_AddRef((LPOLEAPP)g_lpApp);
}


// IOleInPlaceFrame::Release
STDMETHODIMP_(ULONG) CntrApp_IPFrame_Release(LPOLEINPLACEFRAME lpThis)
{
   OleDbgReleaseMethod(lpThis, "IOleInPlaceFrame");

   return OleApp_Release((LPOLEAPP)g_lpApp);
}


// IOleInPlaceFrame::GetWindow
STDMETHODIMP CntrApp_IPFrame_GetWindow(
   LPOLEINPLACEFRAME   lpThis,
   HWND FAR*           lphwnd
)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;

   OLEDBG_BEGIN2("CntrApp_IPFrame_GetWindow\r\n")
   *lphwnd = lpOutlineApp->m_hWndApp;
   OLEDBG_END2
   return NOERROR;
}
```

```
// IOleInPlaceFrame::ContextSensitiveHelp
STDMETHODIMP CntrApp_IPFrame_ContextSensitiveHelp(
   LPOLEINPLACEFRAME   lpThis,
   BOOL                fEnterMode
)
{
   LPCONTAINERAPP lpContainerApp =
         ((struct COleInPlaceFrameImpl FAR*)lpThis)->lpContainerApp;

   // artificial AddRef in case object is destroyed during call
   CntrApp_IPFrame_AddRef(lpThis);

   OleDbgOut("CntrApp_IPFrame_ContextSensitiveHelp\r\n");
   /* OLE2NOTE: see context sensitive help technote (CSHELP.DOC)
   **    This method is called when F1 is pressed when a menu item is
   **    selected. We set the frame's m_fMenuMode flag here. later,
   **    in WM_COMMAND processing in the AppWndProc, if this flag is
   **    set then the command is NOT executed and help is given
   **    instead.
   */
   lpContainerApp->m_fMenuHelpMode = fEnterMode;

   // release artificial AddRef
   CntrApp_IPFrame_Release(lpThis);

   return NOERROR;
}


// IOleInPlaceFrame::GetBorder
STDMETHODIMP CntrApp_IPFrame_GetBorder(
   LPOLEINPLACEFRAME   lpThis,
   LPRECT              lprectBorder
)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;

   // artificial AddRef in case object is destroyed during call
   CntrApp_IPFrame_AddRef(lpThis);

   OLEDBG_BEGIN2("CntrApp_IPFrame_GetBorder\r\n")

   OutlineApp_GetFrameRect(lpOutlineApp, lprectBorder);

   OLEDBG_END2
   // release artificial AddRef
   CntrApp_IPFrame_Release(lpThis);
   return NOERROR;
}


// IOleInPlaceFrame::RequestBorderSpace
STDMETHODIMP CntrApp_IPFrame_RequestBorderSpace(
   LPOLEINPLACEFRAME   lpThis,
   LPCBORDERWIDTHS     lpWidths
)
{
#if defined( _DEBUG )
   OleDbgOut2("CntrApp_IPFrame_RequestBorderSpace\r\n");

   {
      /* FOR DEBUGING PURPOSES ONLY -- we will fail to allow to an
      **    object to get any frame border space for frame tools if
      **    our own frame tools are poped up in the tool pallet. this
      **    is NOT recommended UI behavior but it allows us to test
      **    in the condition when the frame does not give border
      **    space for the object. an object in this situation must
      **    then either popup its tools in a floating pallet, do
      **    without the tools, or fail to in-place activate.
      */
      LPCONTAINERAPP lpContainerApp =
            ((struct COleInPlaceFrameImpl FAR*)lpThis)->lpContainerApp;
      LPFRAMETOOLS lpft = OutlineApp_GetFrameTools(
            (LPOUTLINEAPP)lpContainerApp);

      if (lpft->m_ButtonBar.m_nState == BARSTATE_POPUP &&
         lpft->m_FormulaBar.m_nState == BARSTATE_POPUP) {
         OleDbgOut3(
               "CntrApp_IPFrame_RequestBorderSpace: allow NO SPACE\r\n");
         return ResultFromScode(E_FAIL);
      }
   }
#endif  // _DEBUG

   /* OLE2NOTE: we allow the object to have as much border space as it
   **    wants.
   */
   return NOERROR;
}


// IOleInPlaceFrame::SetBorderSpace
STDMETHODIMP CntrApp_IPFrame_SetBorderSpace(
   LPOLEINPLACEFRAME   lpThis,
   LPCBORDERWIDTHS     lpWidths
)
{
   LPCONTAINERAPP lpContainerApp =
         ((struct COleInPlaceFrameImpl FAR*)lpThis)->lpContainerApp;

   // artificial AddRef in case object is destroyed during call
   CntrApp_IPFrame_AddRef(lpThis);

   OLEDBG_BEGIN2("CntrApp_IPFrame_SetBorderSpace\r\n")

   /* OLE2NOTE: this fMustResizeClientArea flag is used as part of our
   **    defensive programming for frame window resizing. when the
   **    frame window is resized,IOleInPlaceActiveObject::ResizeBorder
   **    is called the object should normally call back to renegotiate
   **    for frame-level tools space. if SetBorderSpace is called then
   **    our client area windows are properly resized. if the in-place
   **    active object does NOT call SetBorderSpace, then the
   **    container must take care to resize its client area windows
   **    itself (see ContainerDoc_FrameWindowResized)
   */
   if (lpContainerApp->m_fMustResizeClientArea)
      lpContainerApp->m_fMustResizeClientArea = FALSE;

   if (lpWidths == NULL) {

      /* OLE2NOTE: IOleInPlaceSite::SetBorderSpace(NULL) is called
      **    when the in-place active object does NOT want any tool
      **    space. in this situation the in-place container should
      **    put up its tools.
      */
      LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)lpContainerApp;
      LPCONTAINERDOC lpContainerDoc;

      lpContainerDoc =(LPCONTAINERDOC)OutlineApp_GetActiveDoc(lpOutlineApp);
      ContainerDoc_AddFrameLevelTools(lpContainerDoc);
   } else {

      // OLE2NOTE: you could do validation of borderwidths here
#if defined( _DEBUG )
      /* FOR DEBUGING PURPOSES ONLY -- we will fail to allow to an
      **    object to get any frame border space for frame tools if
      **    our own frame tools are poped up in the tool pallet. this
      **    is NOT recommended UI behavior but it allows us to test
      **    in the condition when the frame does not give border
      **    space for the object. an object in this situation must
      **    then either popup its tools in a floating pallet, do
      **    without the tools, or fail to in-place activate.
      */
      LPFRAMETOOLS lpft = OutlineApp_GetFrameTools(
            (LPOUTLINEAPP)lpContainerApp);

      if ((lpft->m_ButtonBar.m_nState == BARSTATE_POPUP &&
         lpft->m_FormulaBar.m_nState == BARSTATE_POPUP) &&
         (lpWidths->top || lpWidths->bottom ||
            lpWidths->left || lpWidths->right) ) {
         OleDbgOut3("CntrApp_IPFrame_SetBorderSpace: allow NO SPACE\r\n");
         OLEDBG_END2

         OutlineApp_SetBorderSpace(
               (LPOUTLINEAPP) lpContainerApp,
               (LPBORDERWIDTHS)&g_rectNull
         );
         OLEDBG_END2
         // release artificial AddRef
         CntrApp_IPFrame_Release(lpThis);

         return ResultFromScode(E_FAIL);
      }
#endif  // _DEBUG
```

```
      OutlineApp_SetBorderSpace(
            (LPOUTLINEAPP) lpContainerApp,
            (LPBORDERWIDTHS)lpWidths
      );
   }
   OLEDBG_END2
   // release artificial AddRef
   CntrApp_IPFrame_Release(lpThis);

   return NOERROR;
}


// IOleInPlaceFrame::SetActiveObject
STDMETHODIMP CntrApp_IPFrame_SetActiveObject(
   LPOLEINPLACEFRAME           lpThis,
   LPOLEINPLACEACTIVEOBJECT    lpActiveObject,
   LPCOLESTR                   lpszObjName
)
{
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)g_lpApp;
   // artificial AddRef in case object is destroyed during call
   CntrApp_IPFrame_AddRef(lpThis);

   OLEDBG_BEGIN2("CntrApp_IPFrame_SetActiveObject\r\n")

   lpContainerApp->m_hWndUIActiveObj = NULL;

   if (lpContainerApp->m_lpIPActiveObj)
      lpContainerApp->m_lpIPActiveObj->lpVtbl->Release(lpContainerApp-
>m_lpIPActiveObj);

   if ((lpContainerApp->m_lpIPActiveObj = lpActiveObject) != NULL) {
      lpContainerApp->m_lpIPActiveObj->lpVtbl->AddRef(
            lpContainerApp->m_lpIPActiveObj);

      OLEDBG_BEGIN2("IOleInPlaceActiveObject::GetWindow called\r\n")
      lpActiveObject->lpVtbl->GetWindow(
            lpActiveObject,
            (HWND FAR*)&lpContainerApp->m_hWndUIActiveObj
      );
      OLEDBG_END2

      /* OLE2NOTE: see comment for ContainerDoc_ForwardPaletteChangedMsg */
      /* No need to do this if you don't allow object to own the palette */
      OleApp_QueryNewPalette((LPOLEAPP)lpContainerApp);
   }

   /* OLE2NOTE: the new UI Guidelines recommend that in-place
   **    containers do NOT change their window titles when an object
   **    becomes in-place (UI) active.
   */

   OLEDBG_END2
   // release artificial AddRef
   CntrApp_IPFrame_Release(lpThis);
   return NOERROR;
}


// IOleInPlaceFrame::InsertMenus
STDMETHODIMP CntrApp_IPFrame_InsertMenus(
   LPOLEINPLACEFRAME       lpThis,
   HMENU                   hMenu,
   LPOLEMENUGROUPWIDTHS    lpMenuWidths
)
{
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)g_lpApp;
   BOOL    fNoError = TRUE;

   OLEDBG_BEGIN2("CntrApp_IPFrame_InsertMenus\r\n")

   fNoError &= AppendMenu(hMenu, MF_POPUP, (UINT)lpContainerApp-
>m_hMenuFile,
               "&File");
   fNoError &= AppendMenu(hMenu, MF_POPUP, (UINT)lpContainerApp-
>m_hMenuView,
               "O&utline");
   fNoError &= AppendMenu(hMenu, MF_POPUP,(UINT)lpContainerApp-
>m_hMenuDebug,
               "DbgI&Cntr");
   lpMenuWidths->width[0] = 1;
   lpMenuWidths->width[2] = 1;
   lpMenuWidths->width[4] = 1;

   OLEDBG_END2

   return (fNoError ? NOERROR : ResultFromScode(E_FAIL));
}


// IOleInPlaceFrame::SetMenu
STDMETHODIMP CntrApp_IPFrame_SetMenu(
   LPOLEINPLACEFRAME   lpThis,
   HMENU               hMenuShared,
   HOLEMENU            hOleMenu,
   HWND                hwndActiveObject
)
{
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)g_lpApp;
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   HMENU   hMenu;
   HRESULT hrErr;

   OLEDBG_BEGIN2("CntrApp_IPFrame_InsertMenus\r\n")


   /* OLE2NOTE: either put up the shared menu (combined menu from
   **    in-place server and in-place container) or our container's
   **    normal menu as directed.
   */
   if (hOleMenu && hMenuShared)
      hMenu = hMenuShared;
   else
      hMenu = lpOutlineApp->m_hMenuApp;

   /* OLE2NOTE: SDI apps put menu on frame by calling SetMenu.
   **    MDI apps would send WM_MDISETMENU message instead.
   */
   SetMenu (lpOutlineApp->m_hWndApp, hMenu);
   OLEDBG_BEGIN2("OleSetMenuDescriptor called\r\n")
   hrErr = OleSetMenuDescriptor (hOleMenu, lpOutlineApp->m_hWndApp,
               hwndActiveObject, NULL, NULL);
   OLEDBG_END2

   OLEDBG_END2
   return hrErr;
}


// IOleInPlaceFrame::RemoveMenus
STDMETHODIMP CntrApp_IPFrame_RemoveMenus(
   LPOLEINPLACEFRAME   lpThis,
   HMENU               hMenu
)
{
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)g_lpApp;
   BOOL fNoError = TRUE;

   OLEDBG_BEGIN2("CntrApp_IPFrame_RemoveMenus\r\n")

   /* Remove container group menus */
   while (GetMenuItemCount(hMenu))
      fNoError &= RemoveMenu(hMenu, 0, MF_BYPOSITION);

   OleDbgAssert(fNoError == TRUE);

   OLEDBG_END2

   return (fNoError ? NOERROR : ResultFromScode(E_FAIL));
}


// IOleInPlaceFrame::SetStatusText
STDMETHODIMP CntrApp_IPFrame_SetStatusText(
   LPOLEINPLACEFRAME   lpThis,
   LPCOLESTR           lpszStatusText
)
{
#if defined( USE_STATUSBAR )
   LPOUTLINEAPP   lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   OLECHAR         szMessageHold[128];
   OleDbgOut2("CntrApp_IPFrame_SetStatusText\r\n");
   /* OLE2NOTE: it is important to save a private copy of status text.
   **    lpszStatusText is only valid for the duration of this call.
   */
   OLESTRCPY(szMessageHold, lpszStatusText);
   OutlineApp_SetStatusText(lpOutlineApp, (LPOLESTR)szMessageHold);

   return ResultFromScode(S_OK);
#else
   return ResultFromScode(E_NOTIMPL);
#endif  // USE_STATUSBAR
}


// IOleInPlaceFrame::EnableModeless
STDMETHODIMP CntrApp_IPFrame_EnableModeless(
   LPOLEINPLACEFRAME   lpThis,
   BOOL                fEnable
)
{
   LPOLEAPP lpOleApp = (LPOLEAPP)
         ((struct COleInPlaceFrameImpl FAR*)lpThis)->lpContainerApp;
#if defined( _DEBUG )
   if (fEnable)
      OleDbgOut2("CntrApp_IPFrame_EnableModeless(TRUE)\r\n");
   else
      OleDbgOut2("CntrApp_IPFrame_EnableModeless(FALSE)\r\n");
#endif  // _DEBUG

   /* OLE2NOTE: this method is called when an object puts up a modal
   **    dialog. it tells the top-level in-place container to disable
   **    it modeless dialogs for the duration that the object is
   **    displaying a modal dialog.
   **
   **    ICNTROTL does not use any modeless dialogs, thus we can
   **    ignore this method.
   */
   return NOERROR;
}


// IOleInPlaceFrame::TranslateAccelerator
STDMETHODIMP CntrApp_IPFrame_TranslateAccelerator(
   LPOLEINPLACEFRAME   lpThis,
   LPMSG               lpmsg,
   WORD                wID
)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)g_lpApp;
   SCODE sc;

   if (TranslateAccelerator (lpOutlineApp->m_hWndApp,
               lpContainerApp->m_hAccelIPCntr, lpmsg))
      sc = S_OK;
```

```
#if defined (MDI_VERSION)
   else if (TranslateMDISysAccel(lpOutlineApp->m_hWndMDIClient, lpmsg))
      sc = S_OK;
#endif  // MDI_VERSION

   else
      sc = S_FALSE;

   return ResultFromScode(sc);
}



/*************************************************************************
** ContainerDoc Support Functions
*************************************************************************/


/* ContainerDoc_UpdateInPlaceObjectRects
** -------------------------------------
**    Update the PosRect and ClipRect of the currently in-place active
**    object. if there is no object active in-place, then do nothing.
**
**    OLE2NOTE: this function should be called when an action occurs
**    that changes either the position of the object in the document
**    (eg. changing document margins changes PosRect) or the clipRect
**    changes (eg. resizing the document window changes the ClipRect).
*/
void ContainerDoc_UpdateInPlaceObjectRects(LPCONTAINERDOC lpContainerDoc,
int nIndex)
{
   LPLINELIST lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   int i;
   LPLINE lpLine;
   RECT rcClipRect;

   if (g_fInsideOutContainer) {

      if (lpContainerDoc->m_cIPActiveObjects) {

         /* OLE2NOTE: (INSIDE-OUT CONTAINER) we must update the
         **    PosRect/ClipRect for all in-place active objects
         **    starting from line "nIndex".
         */
         ContainerDoc_GetClipRect(lpContainerDoc, (LPRECT)&rcClipRect);

#if defined( _DEBUG )
         OleDbgOutRect3(
               "ContainerDoc_UpdateInPlaceObjectRects (ClipRect)",
               (LPRECT)&rcClipRect
         );
#endif
         for (i = nIndex; i < lpLL->m_nNumLines; i++) {
            lpLine=LineList_GetLine(lpLL, i);
            if (lpLine && (Line_GetLineType(lpLine)==CONTAINERLINETYPE)) {
               LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)lpLine;
               ContainerLine_UpdateInPlaceObjectRects(
                     lpContainerLine, &rcClipRect);
            }
         }
      }
   }
   else {
      /* OLE2NOTE: (OUTSIDE-IN CONTAINER) if there is a currently
      **    UIActive object, we must inform it that the
      **    PosRect/ClipRect has now changed.
      */
      LPCONTAINERLINE lpLastUIActiveLine =
            lpContainerDoc->m_lpLastUIActiveLine;
      if (lpLastUIActiveLine && lpLastUIActiveLine->m_fUIActive) {
         ContainerDoc_GetClipRect(lpContainerDoc, (LPRECT)&rcClipRect);

         OleDbgOutRect3("OutlineDoc_Resize (ClipRect)",(LPRECT)&rcClipRect);
         ContainerLine_UpdateInPlaceObjectRects(
               lpLastUIActiveLine, &rcClipRect);
      }
   }
}

/* ContainerDoc_IsUIDeactivateNeeded
** ---------------------------------
**    Check if it is necessary to UIDeactivate an in-place active
**    object upon a mouse LBUTTONDOWN event. The position of the button
**    down click is given by "pt".
**    If there is not currently an in-place active line, then
**    UIDeactivate is NOT needed.
**    If there is a current in-place active line, then check if the
**    point position is outside of the object extents on the screen. If
**    so, then the object should be UIDeactivated, otherwise not.
*/
BOOL ContainerDoc_IsUIDeactivateNeeded(
      LPCONTAINERDOC  lpContainerDoc,
      POINT           pt
)
{
   LPCONTAINERLINE lpUIActiveLine=lpContainerDoc->m_lpLastUIActiveLine;
   RECT rect;

   if (! lpUIActiveLine || ! lpUIActiveLine->m_fUIActive)
      return FALSE;

   ContainerLine_GetPosRect(
         lpUIActiveLine,
         (LPRECT) &rect
   );

   if (! PtInRect((LPRECT) &rect, pt))
      return TRUE;
   return FALSE;
}


/* ContainerDoc_ShutDownLastInPlaceServerIfNotNeeded
** -------------------------------------------------
**    OLE2NOTE: this function ONLY applies for OUTSIDE-IN containers
**
**    If there is a previous in-place active server still running and
**    this server will not be needed to support the next OLE object
**    about to be activated, then shut it down.
**    in this way we manage a policy of having at most one in-place
**    server running at a time. we do not imediately shut down the
**    in-place server when the object is UIDeactivated because we want
**    it to be fast if the server decides to re-activate the object
**    in-place.
**
**    shutting down the server is achieved by forcing the object to
**    transition from running to loaded by calling IOleObject::Close.
*/
void ContainerDoc_ShutDownLastInPlaceServerIfNotNeeded(
      LPCONTAINERDOC          lpContainerDoc,
      LPCONTAINERLINE         lpNextActiveLine
)
{
   LPCONTAINERLINE lpLastIpActiveLine = lpContainerDoc-
>m_lpLastIpActiveLine;
   BOOL fEnableServerShutDown = TRUE;
   LPMONIKER lpmkLinkSrc;
   LPMONIKER lpmkLastActiveObj;
   LPMONIKER lpmkCommonPrefix;
   LPOLELINK lpOleLink;
   HRESULT hrErr;

   /* OLE2NOTE: an inside-out style container can NOT use this scheme
   **    to shut down in-place servers. it would have to have a more
   **    sophistocated mechanism by which it keeps track of which
   **    objects are on screen and which were the last recently used.
   */
   if (g_fInsideOutContainer)
      return;


   if (lpLastIpActiveLine != lpNextActiveLine) {
      if (lpLastIpActiveLine) {

         /* OLE2NOTE: if the object which is about to be activated is
         **    actually a link to the OLE object in last activated line,
         **    then we do NOT want to shut down the last activated
         **    server because it is about to be used. when activating a
         **    linked object, the source of the link gets activated.
         */
         lpOleLink = (LPOLELINK)ContainerLine_GetOleObject(
               lpNextActiveLine,
               &IID_IOleLink
```

```
);
if (lpOleLink) {
   OLEDBG_BEGIN2("IOleObject::GetSourceMoniker called\r\n")
   lpOleLink->lpVtbl->GetSourceMoniker(
         lpOleLink,
         (LPMONIKER FAR*)&lpmkLinkSrc
   );
   OLEDBG_END2

   if (lpmkLinkSrc) {
      lpmkLastActiveObj = ContainerLine_GetFullMoniker(
            lpLastIpActiveLine,
            GETMONIKER_ONLYIFTHERE
      );
      if (lpmkLastActiveObj) {
         hrErr = lpmkLinkSrc->lpVtbl->CommonPrefixWith(
               lpmkLinkSrc,
               lpmkLastActiveObj,
               &lpmkCommonPrefix

         );
         if (GetScode(hrErr) == MK_S_HIM ||
                       hrErr == NOERROR ||
               GetScode(hrErr) == MK_S_US) {
            /* the link source IS to the object
            **    contained in the last activated
            **    line of the document; disable the
            **    attempt to shut down the last
            **    running in-place server.
            */
            fEnableServerShutDown = FALSE;
         }
         if (lpmkCommonPrefix)
            OleStdRelease((LPUNKNOWN)lpmkCommonPrefix);
         OleStdRelease((LPUNKNOWN)lpmkLastActiveObj);
      }
      OleStdRelease((LPUNKNOWN)lpmkLinkSrc);
   }
   OleStdRelease((LPUNKNOWN)lpOleLink);
}

/* if it is OK to shut down the previous in-place server
**    and one is still running, then shut it down. shutting
**    down the server is accomplished by forcing the OLE
**    object to close. this forces the object to transition
**    from running to loaded. if the object is actually
**    only loaded then this is a NOP.
*/
if (fEnableServerShutDown &&
      lpLastIpActiveLine->m_fIpServerRunning) {

   OleDbgOut1("@@@ previous in-place server SHUT DOWN\r\n");
   ContainerLine_CloseOleObject(
         lpLastIpActiveLine, OLECLOSE_SAVEIFDIRTY);
```

```
            // we can now forget this last in-place active line.
            lpContainerDoc->m_lpLastIpActiveLine = NULL;
         }
      }
   }
}


/* ContainerDoc_GetUIActiveWindow
** ------------------------------
**    If there is an UIActive object, then return its HWND.
*/
HWND ContainerDoc_GetUIActiveWindow(LPCONTAINERDOC lpContainerDoc)
{
   return lpContainerDoc->m_hWndUIActiveObj;
}


/* ContainerDoc_GetClipRect
** ------------------------
**    Get the ClipRect in client coordinates.
**
** OLE2NOTE: the ClipRect is defined as the maximum window rectangle
**    that the in-place active object must be clipped to. this
**    rectangle MUST be described in Client Coordinates of the window
**    that is used as the Parent of the in-place active object's
**    window. in our case, the LineList ListBox window is both the
**    parent of the in-place active object AND defines precisely the
**    clipping rectangle.
*/
void ContainerDoc_GetClipRect(
      LPCONTAINERDOC      lpContainerDoc,
      LPRECT              lprcClipRect
)
{
   /* OLE2NOTE: the ClipRect can be used to ensure that the in-place
   **    server does not overwrite areas of the window that the
   **    container paints into but which should never be overwritten
   **    (eg. if an app were to paint row and col headings directly in
   **    the same window that is the parent of the in-place window.
   **    whenever the window size changes or gets scrolled, in-place
   **    active object must be informed of the new clip rect.
   **
   **    normally an app would pass the rect returned from GetClientRect.
   **    but because CntrOutl uses separate windows for row/column
   **    headings, status line, formula/tool bars, etc. it is NOT
   **    necessary to pass a constrained clip rect. Windows standard
   **    window clipping will automatically take care of all clipping
   **    that is necessary. thus we can take a short cut of passing an
   **    "infinite" clip rect, and then we do NOT need to call
   **    IOleInPlaceObject::SetObjectRects when our document is scrolled.
   */

   lprcClipRect->top = -32767;
   lprcClipRect->left = -32767;
   lprcClipRect->right = 32767;
   lprcClipRect->bottom = 32767;
}


/* ContainerDoc_GetTopInPlaceFrame
** -------------------------------
**    returns NON-AddRef'ed pointer to Top In-Place Frame interface
*/
LPOLEINPLACEFRAME ContainerDoc_GetTopInPlaceFrame(
      LPCONTAINERDOC      lpContainerDoc
)
{
#if defined( INPLACE_CNTRSVR )
   return lpContainerDoc->m_lpTopIPFrame;
#else
   return (LPOLEINPLACEFRAME)&((LPCONTAINERAPP)g_lpApp)->m_OleInPlaceFrame;
#endif
}

void ContainerDoc_GetSharedMenuHandles(
      LPCONTAINERDOC  lpContainerDoc,
      HMENU FAR*      lphSharedMenu,
      HOLEMENU FAR*   lphOleMenu
)
{
#if defined( INPLACE_CNTRSVR )
   if (lpContainerDoc->m_DocType == DOCTYPE_EMEBEDDEDOBJECT) {
      *lphSharedMenu = lpContainerDoc->m_hSharedMenu;
      *lphOleMenu = lpContainerDoc->m_hOleMenu;
      return;
   }
#endif

   *lphSharedMenu = NULL;
   *lphOleMenu = NULL;
}


#if defined( USE_FRAMETOOLS )
void ContainerDoc_RemoveFrameLevelTools(LPCONTAINERDOC lpContainerDoc)
{
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpContainerDoc;
   OleDbgAssert(lpOutlineDoc->m_lpFrameTools != NULL);

   FrameTools_Enable(lpOutlineDoc->m_lpFrameTools, FALSE);
}
#endif


void ContainerDoc_AddFrameLevelUI(LPCONTAINERDOC lpContainerDoc)
{
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpContainerDoc;
   LPOLEINPLACEFRAME lpTopIPFrame = ContainerDoc_GetTopInPlaceFrame(
         lpContainerDoc);
   HMENU           hSharedMenu;            // combined obj/cntr menu
   HOLEMENU        hOleMenu;               // returned by OleCreateMenuDesc.

   ContainerDoc_GetSharedMenuHandles(
         lpContainerDoc,
         &hSharedMenu,
         &hOleMenu
   );

   lpTopIPFrame->lpVtbl->SetMenu(
         lpTopIPFrame,
         hSharedMenu,
         hOleMenu,
         lpOutlineDoc->m_hWndDoc
   );

   /* OLE2NOTE: even if our app does NOT use FrameTools, we must still
   **    call IOleInPlaceFrame::SetBorderSpace.
   */
   ContainerDoc_AddFrameLevelTools(lpContainerDoc);
}


void ContainerDoc_AddFrameLevelTools(LPCONTAINERDOC lpContainerDoc)
{
   LPOLEINPLACEFRAME lpTopIPFrame = ContainerDoc_GetTopInPlaceFrame(
         lpContainerDoc);
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)g_lpApp;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpContainerDoc;


   OleDbgAssert(lpTopIPFrame != NULL);

#if defined( USE_FRAMETOOLS )

   FrameTools_Enable(lpOutlineDoc->m_lpFrameTools, TRUE);
   OutlineDoc_UpdateFrameToolButtons(lpOutlineDoc);

   FrameTools_NegotiateForSpaceAndShow(
         lpOutlineDoc->m_lpFrameTools,
         NULL,
         lpTopIPFrame
   );

#else   // ! USE_FRAMETOOLS

#if defined( INPLACE_CNTRSVR )
   if (lpContainerDoc->m_DocType == DOCTYPE_EMBEDDEDOBJECT) {
      /* this says i do not need space, so the top Frame should
      **    leave its tools behind
      */
      OLEDBG_BEGIN2("IOleInPlaceFrame::SetBorderSpace(NULL) called\r\n")
      lpTopIPFrame->lpVtbl->SetBorderSpace(lpTopIPFrame, NULL);
      OLEDBG_END2
      return;
   }
#else   // INPLACE_CNTR && ! USE_FRAMETOOLS

   OLEDBG_BEGIN2("IOleInPlaceFrame::SetBorderSpace(0,0,0,0) called\r\n")
   lpTopIPFrame->lpVtbl->SetBorderSpace(
         lpTopIPFrame,
         (LPCBORDERWIDTHS)&g_rectNull
   );
   OLEDBG_END2

#endif  // INPLACE_CNTR && ! USE_FRAMETOOLS
#endif  // ! USE_FRAMETOOLS

}


void ContainerDoc_FrameWindowResized(LPCONTAINERDOC lpContainerDoc)
{
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)g_lpApp;


   if (lpContainerApp->m_lpIPActiveObj) {
      RECT rcFrameRect;

      /* OLE2NOTE: this fMustResizeClientArea flag is used as part of
      **    our defensive programming for frame window resizing. when
      **    the frame window is
      **    resized, IOleInPlaceActiveObject::ResizeBorder is called
      **    the object should normally call back to renegotiate
      **    for frame-level tools space. if SetBorderSpace is called
      **    then our client area windows are properly resized.
      **    CntrApp_IPFrame_SetBorderSpace clears this flag. if the
      **    in-place active object does NOT call SetBorderSpace, then
      **    the container must take care to resize its client area
      **    windows itself.
      */
      lpContainerApp->m_fMustResizeClientArea = TRUE;

      OutlineApp_GetFrameRect(g_lpApp, (LPRECT)&rcFrameRect);

      OLEDBG_BEGIN2("IOleInPlaceActiveObject::ResizeBorder called\r\n")
      lpContainerApp->m_lpIPActiveObj->lpVtbl->ResizeBorder(
            lpContainerApp->m_lpIPActiveObj,
            (LPCRECT)&rcFrameRect,
            (LPOLEINPLACEUIWINDOW)&lpContainerApp->m_OleInPlaceFrame,
            TRUE    /* fFrameWindow */
      );
      OLEDBG_END2

      /* the object did NOT call IOleInPlaceUIWindow::SetBorderSpace
      **    therefore we must resize our client area windows ourself.
      */
      if (lpContainerApp->m_fMustResizeClientArea) {
         lpContainerApp->m_fMustResizeClientArea = FALSE;
         OutlineApp_ResizeClientArea(g_lpApp);
      }
   }

#if defined( USE_FRAMETOOLS )
   else {
      ContainerDoc_AddFrameLevelTools(lpContainerDoc);
   }
#endif
}


#if defined( INPLACE_CNTRSVR ) || defined( INPLACE_MDICNTR )

/* ContainerDoc_GetTopInPlaceDoc
**    returns NON-AddRef'ed pointer to Top In-Place Doc interface
*/
LPOLEINPLACEUIWINDOW ContainerDoc_GetTopInPlaceDoc(
      LPCONTAINERDOC      lpContainerDoc
)
{
#if defined( INPLACE_CNTRSVR )
   return lpContainerDoc->m_lpTopIPDoc;
#else
   return (LPOLEINPLACEUIWINDOW)&lpContainerDoc->m_OleInPlaceDoc;
#endif
}


void ContainerDoc_RemoveDocLevelTools(LPCONTAINERDOC lpContainerDoc);
{
   LPOLEINPLACEUIWINDOW lpTopIPDoc = ContainerDoc_GetTopInPlaceDoc(
         lpContainerDoc);

   if (lpTopIPDoc && lpContainerDoc->m_fMyToolsOnDoc) {
      lpContainerDoc->m_fMyToolsOnDoc = FALSE;

      // if we had doc tools we would HIDE them here;
      //   but NOT call SetBorderSpace(NULL)

   }
}

void ContainerDoc_AddDocLevelTools(LPCONTAINERDOC lpContainerDoc)
{
   LPOLEINPLACEUIWINDOW lpTopIPDoc = ContainerDoc_GetTopInPlaceDoc(
         lpContainerDoc);

   if (! lpTopIPDoc)
      return;


#if defined( USE_DOCTOOLS )
   if (lpTopIPDoc && ! lpContainerDoc->m_fMyToolsOnDoc) {

      /* if we had doc tools we would negotiate for toolbar space at
      **    doc level and SHOW them.
      */

      /* we do NOT have doc level tools, so we just call
      **    SetBorderSpace() to indicate to the top doc that
      **    our object does not need tool space.
      */

      lpContainerDoc->m_fMyToolsOnDoc = TRUE;
      return;
   }
#else   // ! USE_DOCTOOLS

#if defined( INPLACE_CNTRSVR )
   if (lpContainerDoc->m_DocType == DOCTYPE_EMBEDDEDOBJECT) {
      /* this says i do not need space, so the top doc should
      **    leave its tools behind
      */
      lpTopIPDoc->lpVtbl->SetBorderSpace(lpTopIPDoc, NULL);
      return;
   }
#else
   lpTopIPDoc->lpVtbl->SetBorderSpace(
         lpTopIPDoc,
         (LPCBORDERWIDTHS)&g_rectNull
   );

#endif
#endif  // ! USE_DOCTOOLS
}

#endif  // INPLACE_CNTRSVR || INPLACE_MDICNTR


/* ContainerDoc_ContextSensitiveHelp
** ---------------------------------
**    forward the ContextSensitiveHelp mode on to any in-place objects
**    that currently have their window visible. this informs the
**    objects whether to give help or take action on subsequent mouse
**    clicks and menu commands. this function is called from our
**    IOleInPlaceSite::ContextSensitiveHelp implementation.
**
** OLE2NOTE: see context sensitive help technote (CSHELP.DOC).
**    This function is called when SHIFT-F1 context sensitive help is
**    entered. the cursor should then change to a question mark
**    cursor and the app should enter a modal state where the next
**    mouse click does not perform its normal action but rather
**    gives help corresponding to the location clicked. if the app
**    does not implement a help system, it should at least eat the
**    click and do nothing.
*/
void ContainerDoc_ContextSensitiveHelp(
      LPCONTAINERDOC  lpContainerDoc,
      BOOL            fEnterMode,
      BOOL            fInitiatedByObj
```

```
)
{
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpContainerDoc;
   LPLINELIST lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   int i;
   LPLINE lpLine;

   lpOleDoc->m_fCSHelpMode = fEnterMode;

   if (g_fInsideOutContainer) {

      if (lpContainerDoc->m_cIPActiveObjects) {
         for (i = 0; i < lpLL->m_nNumLines; i++) {
            lpLine=LineList_GetLine(lpLL, i);

            if (lpLine && (Line_GetLineType(lpLine)==CONTAINERLINETYPE)) {
               LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)lpLine;
               ContainerLine_ContextSensitiveHelp(
                     lpContainerLine, fEnterMode);
            }
         }
      }
   }
   else if (! fInitiatedByObj) {
      /* OLE2NOTE: (OUTSIDE-IN CONTAINER) if there is a currently
      **    UIActive object (ie. its window is visible), we must
      **    forward the ContextSensitiveHelp mode on to the
      **    object--assuming it was not the object that initiated the
      **    context sensitve help mode.
      */
      LPCONTAINERLINE lpLastUIActiveLine =
            lpContainerDoc->m_lpLastUIActiveLine;
      if (lpLastUIActiveLine && lpLastUIActiveLine->m_fUIActive) {
         ContainerLine_ContextSensitiveHelp(lpLastUIActiveLine,fEnterMode);
      }
   }
}

/* ContainerDoc_ForwardPaletteChangedMsg
** -------------------------------------
**    forward the WM_PALETTECHANGED message (via SendMessage) to any
**    in-place objects that currently have their window visible. this
**    gives these objects the chance to select their palette as a
**    BACKGROUND palette.
**
**    SEE THE TECHNOTE FOR DETAILED INFORMATION ON PALETTE COORDINATION
**
**    OLE2NOTE: For containers and objects to manage palettes properly
**    (when objects are getting inplace edited) they should follow a
**    set of rules.
**
**    Rule 1: The Container can decide if it wants to own the palette or
**    it wants to allow its UIActive object to own the palette.
**    a) If the container wants to let its UIActive object own the
**    palette then it should forward WM_QUERYNEWPALETTE to the object
**    when it is received to the top frame window. also it should send
**    WM_QUERYNEWPALETTE to the object in its
**    IOleInPlaceFrame::SetActiveObject implementation. if the object
**    is given ownership of the palette, then it owns the palette until
**    it is UIDeactivated.
**    b) If the container wants to own the palette it should NOT send
**    WM_QUERYNEWPALETTE to the UIActive object.
**
**    Rule 2: Whether the container wants to own the palette or not, it
**    should forward the WM_PALETTECHANGED to the immediate child
**    inplace active objects in its documents. if it is an inside-out
**    style container then it must forward it to ALL objects that
**    currently have their windows visible. When the object recives the
**    WM_PALETTECHANGED message it must select its color palette as the
**    background palette. When the objects are in loaded state they will be
**    drawn by (OLE) by selecting their palettes as background palettes
**    anyway.
**
**    Note: IOleObject::SetColorScheme is not related to the palette
**    issue.
*/
void ContainerDoc_ForwardPaletteChangedMsg(
      LPCONTAINERDOC  lpContainerDoc,
      HWND            hwndPalChg
)
{
   LPLINELIST lpLL;
   int i;
   LPLINE lpLine;

   if (!lpContainerDoc)
      return;

   lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   if (g_fInsideOutContainer) {

      if (lpContainerDoc->m_cIPActiveObjects) {
         for (i = 0; i < lpLL->m_nNumLines; i++) {
            lpLine=LineList_GetLine(lpLL, i);

            if (lpLine && (Line_GetLineType(lpLine)==CONTAINERLINETYPE)) {
               LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)lpLine;
               ContainerLine_ForwardPaletteChangedMsg(
                     lpContainerLine, hwndPalChg);
            }
         }
      }
   }
   else {
      /* OLE2NOTE: (OUTSIDE-IN CONTAINER) if there is a currently
      **    UIActive object (ie. its window is visible), we must
      **    forward it the WM_PALETTECHANGED message.
      */
      LPCONTAINERLINE lpLastUIActiveLine =
            lpContainerDoc->m_lpLastUIActiveLine;
      if (lpLastUIActiveLine && lpLastUIActiveLine->m_fUIActive) {
         ContainerLine_ForwardPaletteChangedMsg(
               lpLastUIActiveLine, hwndPalChg);
      }
   }
}


/*************************************************************************
** ContainerLine Support Functions and Interfaces
*************************************************************************/


/* ContainerLine_UIDeactivate
** --------------------------
**    tell the OLE object associated with the ContainerLine to
**    UIDeactivate. this informs the in-place server to tear down
**    the UI and window that it put up for the object. it will remove
**    its active editor menus and any frame and object adornments
**    (eg. toolbars, rulers, etc.).
*/
void ContainerLine_UIDeactivate(LPCONTAINERLINE lpContainerLine)
{
   HRESULT hrErr;

   if (!lpContainerLine || !lpContainerLine->m_fUIActive)
      return;

   OLEDBG_BEGIN2("IOleInPlaceObject::UIDeactivate called\r\n")
   hrErr = lpContainerLine->m_lpOleIPObj->lpVtbl->UIDeactivate(
         lpContainerLine->m_lpOleIPObj);
   OLEDBG_END2

   if (hrErr != NOERROR) {
      OleDbgOutHResult("IOleInPlaceObject::UIDeactivate returned", hrErr);
   }
}



/* ContainerLine_UpdateInPlaceObjectRects
** -------------------------------------
**    Update the PosRect and ClipRect of the given line
**    currently in-place active
**    object. if there is no object active in-place, then do nothing.
**
**    OLE2NOTE: this function should be called when an action occurs
**    that changes either the position of the object in the document
**    (eg. changing document margins changes PosRect) or the clipRect
**    changes (eg. resizing the document window changes the ClipRect).
*/
void ContainerLine_UpdateInPlaceObjectRects(
      LPCONTAINERLINE lpContainerLine,
      LPRECT          lprcClipRect
)
```

```
{
   LPCONTAINERDOC lpContainerDoc = lpContainerLine->m_lpDoc;
   RECT rcClipRect;
   RECT rcPosRect;

   if (! lpContainerLine->m_fIpVisible)
      return;

   if (! lprcClipRect) {
      ContainerDoc_GetClipRect(lpContainerDoc, (LPRECT)&rcClipRect);
      lprcClipRect = (LPRECT)&rcClipRect;
   }

#if defined( _DEBUG )
   OleDbgOutRect3(
         "ContainerLine_UpdateInPlaceObjectRects (ClipRect)",
         (LPRECT)&rcClipRect
   );
#endif
   ContainerLine_GetPosRect(lpContainerLine,(LPRECT)&rcPosRect);

#if defined( _DEBUG )
   OleDbgOutRect3(
      "ContainerLine_UpdateInPlaceObjectRects (PosRect)",
(LPRECT)&rcPosRect);
#endif

   OLEDBG_BEGIN2("IOleInPlaceObject::SetObjectRects called\r\n")
   lpContainerLine->m_lpOleIPObj->lpVtbl->SetObjectRects(
         lpContainerLine->m_lpOleIPObj,
         (LPRECT)&rcPosRect,
         lprcClipRect
   );
   OLEDBG_END2
}


/* ContainerLine_ContextSensitveHelp
** ---------------------------------
**    forward the ContextSensitiveHelp mode on to the in-place object
**    if it currently has its window visible. this informs the
**    object whether to give help or take action on subsequent mouse
**    clicks and menu commands.
**
**    this function is called from ContainerDoc_ContextSensitiveHelp
**    function as a result of a call to
**    IOleInPlaceSite::ContextSensitiveHelp if the in-place container
**    is operating as an in-side out container.
*/
void ContainerLine_ContextSensitiveHelp(
      LPCONTAINERLINE lpContainerLine,
      BOOL            fEnterMode
)
{
   if (! lpContainerLine->m_fIpVisible)
```

```
      return;

   OLEDBG_BEGIN2("IOleInPlaceObject::ContextSensitiveHelp called\r\n")
   lpContainerLine->m_lpOleIPObj->lpVtbl->ContextSensitiveHelp(
         lpContainerLine->m_lpOleIPObj, fEnterMode);
   OLEDBG_END2
}


/* ContainerLine_ForwardPaletteChangedMsg
** --------------------------------------
**    forward it the WM_PALETTECHANGED message (via SendMessage) to the
**    in-place object if it currently has its window visible. this
**    gives the object the chance to select its palette as a BACKGROUND
**    palette if it doesn't own the palette--or as the
**    foreground palette if it currently DOES own the palette.
**
**    SEE THE TECHNOTE FOR DETAILED INFORMATION ON PALETTE COORDINATION
*/
void ContainerLine_ForwardPaletteChangedMsg(
      LPCONTAINERLINE lpContainerLine,
      HWND             hwndPalChg
)
{
   if (! lpContainerLine->m_fIpVisible)
      return;

   if (lpContainerLine->m_hWndIpObject)
   {
       SendMessage(
             lpContainerLine->m_hWndIpObject,
             WM_PALETTECHANGED,
             (WPARAM)hwndPalChg,
             0L
       );
   }
}



/*************************************************************************
** ContainerLine::IOleInPlaceSite interface implementation
*************************************************************************/

STDMETHODIMP CntrLine_IPSite_QueryInterface(
      LPOLEINPLACESITE    lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleInPlaceSiteImpl FAR*)lpThis)->lpContainerLine;

   return ContainerLine_QueryInterface(lpContainerLine, riid, lplpvObj);
}
```

```
STDMETHODIMP_(ULONG) CntrLine_IPSite_AddRef(LPOLEINPLACESITE lpThis)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleInPlaceSiteImpl FAR*)lpThis)->lpContainerLine;

   OleDbgAddRefMethod(lpThis, "IOleInPlaceSite");

   return ContainerLine_AddRef(lpContainerLine);
}


STDMETHODIMP_(ULONG) CntrLine_IPSite_Release(LPOLEINPLACESITE lpThis)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleInPlaceSiteImpl FAR*)lpThis)->lpContainerLine;

   OleDbgReleaseMethod(lpThis, "IOleInPlaceSite");

   return ContainerLine_Release(lpContainerLine);
}


STDMETHODIMP CntrLine_IPSite_GetWindow(
   LPOLEINPLACESITE    lpThis,
   HWND FAR*           lphwnd
)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleInPlaceSiteImpl FAR*)lpThis)->lpContainerLine;

   // artificial AddRef in case object is destroyed during call
   CntrLine_IPSite_AddRef(lpThis);

   OleDbgOut2("CntrLine_IPSite_GetWindow\r\n");

   *lphwnd = LineList_GetWindow(
         &((LPOUTLINEDOC)lpContainerLine->m_lpDoc)->m_LineList);

   // release artificial AddRef
   CntrLine_IPSite_Release(lpThis);

   return NOERROR;
}

// IOleInPlaceSite::ContextSensitiveHelp
STDMETHODIMP CntrLine_IPSite_ContextSensitiveHelp(
   LPOLEINPLACESITE    lpThis,
   BOOL                fEnterMode
)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleInPlaceSiteImpl FAR*)lpThis)->lpContainerLine;
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpContainerLine->m_lpDoc;
   // artificial AddRef in case object is destroyed during call
   CntrLine_IPSite_AddRef(lpThis);

   OleDbgOut2("CntrLine_IPSite_ContextSensitiveHelp\r\n");

   /* OLE2NOTE: see context sensitive help technote (CSHELP.DOC).
   **    This method is called when SHIFT-F1 context sensitive help is
   **    entered. the cursor should then change to a question mark
   **    cursor and the app should enter a modal state where the next
   **    mouse click does not perform its normal action but rather
   **    gives help corresponding to the location clicked. if the app
   **    does not implement a help system, it should at least eat the
   **    click and do nothing.
   **
   **    NOTE: the implementation here is specific to an SDI simple
   **    container. an MDI container or container/server application
   **    would have additional work to do (see the technote).
   **
   **    NOTE: (INSIDE-OUT CONTAINER) if there are currently
   **    any in-place objects active with their windows visible
   **    (ie. fIpVisible), we must forward the
   **    ContextSensitiveHelp mode on to these objects.
   */
   ContainerDoc_ContextSensitiveHelp(
            lpContainerLine->m_lpDoc,fEnterMode,TRUE /*InitiatedByObj*/);

   // release artificial AddRef
   CntrLine_IPSite_Release(lpThis);
   return NOERROR;
}


STDMETHODIMP CntrLine_IPSite_CanInPlaceActivate(LPOLEINPLACESITE lpThis)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleInPlaceSiteImpl FAR*)lpThis)->lpContainerLine;
   OleDbgOut2("CntrLine_IPSite_CanInPlaceActivate\r\n");

   /* OLE2NOTE: the container can NOT allow in-place activation if it
   **    is currently displaying the object as an ICON
   **    (DVASPECT_ICON). it can ONLY do in-place activation if it is
   **    displaying the DVASPECT_CONTENT of the OLE object.
   */
   if (lpContainerLine->m_dwDrawAspect == DVASPECT_CONTENT)
      return NOERROR;
   else
      return ResultFromScode(S_FALSE);
}

STDMETHODIMP CntrLine_IPSite_OnInPlaceActivate(LPOLEINPLACESITE lpThis)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleInPlaceSiteImpl FAR*)lpThis)->lpContainerLine;
   LPCONTAINERDOC lpContainerDoc = lpContainerLine->m_lpDoc;
```

```
    SCODE sc = S_OK;

    // artificial AddRef in case object is destroyed during call
    CntrLine_IPSite_AddRef(lpThis);
    OLEDBG_BEGIN2("CntrLine_IPSite_OnInPlaceActivate\r\n");

    /* OLE2NOTE: (OUTSIDE-IN CONTAINER) as a policy for managing
    **    running in-place servers, we will keep only 1 inplace server
    **    active at any given time. so when we start to inp-place activate
    **    another line, then we want to shut down the previously
    **    activated server. in this way we keep at most one inplace
    **    server active at a time. this is NOT a required policy. apps
    **    may choose to have a more sophisticated strategy. inside-out
    **    containers will HAVE to have a more sophisticated strategy,
    **    because they need (at a minimum) to keep all visible object
    **    servers running.
    **
    **    if the in-place activation is the result of activating a
    **    linked object in another container, then we may arrive at
    **    this method while another object is currently active.
    **    normally, if the object is in-place activated by
    **    double-clicking or Edit.<verb> from our own container, then
    **    the previous in-place object would have been de-activated in
    **    ContainerLine_DoVerb method.
    */
    if (!g_fInsideOutContainer) {
       if (lpContainerDoc->m_lpLastIpActiveLine) {
          ContainerDoc_ShutDownLastInPlaceServerIfNotNeeded(
                lpContainerDoc, lpContainerLine);
       }
       lpContainerDoc->m_lpLastIpActiveLine = lpContainerLine;
    }

    /* OLE2NOTE: to avoid LRPC problems it is important to cache the
    **    IOleInPlaceObject* pointer and NOT to call QueryInterface
    **    each time it is needed.
    */
    lpContainerLine->m_lpOleIPObj = (LPOLEINPLACEOBJECT)
          ContainerLine_GetOleObject(lpContainerLine,&IID_IOleInPlaceObject);

    if (! lpContainerLine->m_lpOleIPObj) {
#if defined( _DEBUG )
       OleDbgAssertSz(
             lpContainerLine->m_lpOleIPObj!=NULL,
             "OLE object must support IOleInPlaceObject"
       );
#endif
       // release artificial AddRef
       CntrLine_IPSite_Release(lpThis);
       return ResultFromScode(E_FAIL); // ERROR: refuse to in-place activate
    }

    lpContainerLine->m_fIpActive        = TRUE;
    lpContainerLine->m_fIpVisible       = TRUE;
```

```
   OLEDBG_BEGIN2("IOleInPlaceObject::GetWindow called\r\n")
   lpContainerLine->m_lpOleIPObj->lpVtbl->GetWindow(
         lpContainerLine->m_lpOleIPObj,
         (HWND FAR*)&lpContainerLine->m_hWndIpObject
   );
   OLEDBG_END2

   if (! lpContainerLine->m_fIpServerRunning) {
      /* OLE2NOTE: it is VERY important that an in-place container
      **    that also support linking to embeddings properly manage
      **    the running of its in-place objects. in an outside-in
      **    style in-place container, when the user clicks
      **    outside of the in-place active object, the object gets
      **    UIDeactivated and the object hides its window. in order
      **    to make the object fast to reactivate, the container
      **    deliberately does not call IOleObject::Close. the object
      **    stays running in the invisible unlocked state. the idea
      **    here is if the user simply clicks outside of the object
      **    and then wants to double click again to re-activate the
      **    object, we do not want this to be slow. if we want to
      **    keep the object running, however, we MUST Lock it
      **    running. otherwise the object will be in an unstable
      **    state where if a linking client does a "silent-update"
      **    (eg. UpdateNow from the Links dialog), then the in-place
      **    server will shut down even before the object has a chance
      **    to be saved back in its container. this saving normally
      **    occurs when the in-place container closes the object. also
      **    keeping the object in the unstable, hidden, running,
      **    not-locked state can cause problems in some scenarios.
      **    ICntrOtl keeps only one object running. if the user
      **    intiates a DoVerb on another object, then that last
      **    running in-place active object is closed. a more
      **    sophistocated in-place container may keep more object running.
      **    this lock gets unlocked in ContainerLine_CloseOleObject.
      */
      lpContainerLine->m_fIpServerRunning = TRUE;

      OLEDBG_BEGIN2("OleLockRunning(TRUE, 0) called\r\n")
      OleLockRunning((LPUNKNOWN)lpContainerLine->m_lpOleIPObj, TRUE, 0);
      OLEDBG_END2
   }

   lpContainerLine->m_lpDoc->m_cIPActiveObjects++;

   OLEDBG_END2
   // release artificial AddRef
   CntrLine_IPSite_Release(lpThis);
   return NOERROR;
}


STDMETHODIMP CntrLine_IPSite_OnUIActivate (LPOLEINPLACESITE lpThis)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleInPlaceSiteImpl FAR*)lpThis)->lpContainerLine;
   LPCONTAINERDOC lpContainerDoc = lpContainerLine->m_lpDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpContainerLine->m_lpDoc;
   LPCONTAINERLINE lpLastUIActiveLine = lpContainerDoc-
>m_lpLastUIActiveLine;

   // artificial AddRef in case object is destroyed during call
   CntrLine_IPSite_AddRef(lpThis);

   OLEDBG_BEGIN2("CntrLine_IPSite_OnUIActivate\r\n");

   lpContainerLine->m_fUIActive        = TRUE;
   lpContainerDoc->m_fAddMyUI          = FALSE;
   lpContainerDoc->m_lpLastUIActiveLine = lpContainerLine;

   if (g_fInsideOutContainer) {
      /* OLE2NOTE: (INSIDE-OUT CONTAINER) an inside-out style
      **    container must UIDeactivate the previous UIActive object
      **    when a new object is going UIActive. since the inside-out
      **    objects have their own windows visible, it is possible
      **    that a click directly in an another server window will
      **    cause it to UIActivate. OnUIActivate is the containers
      **    notification that such has occured. it must then
      **    UIDeactivate the other because at most one object can be
      **    UIActive at a time.
      */
      if (lpLastUIActiveLine && (lpLastUIActiveLine!=lpContainerLine)) {
         ContainerLine_UIDeactivate(lpLastUIActiveLine);

         // Make sure new UIActive window is on top of all others
         SetWindowPos(
               lpContainerLine->m_hWndIpObject,
               HWND_TOPMOST,
               0,0,0,0,
               SWP_NOMOVE | SWP_NOSIZE
         );

         OLEDBG_END2
         goto cleanup;
      }
   }

   lpContainerDoc->m_hWndUIActiveObj = lpContainerLine->m_hWndIpObject;

#if defined( USE_FRAMETOOLS )
   ContainerDoc_RemoveFrameLevelTools(lpContainerDoc);
#endif

#if defined( USE_DOCTOOLS )
   ContainerDoc_RemoveDocLevelTools(lpContainerDoc);
#endif

   OLEDBG_END2
cleanup:
   // release artificial AddRef
   CntrLine_IPSite_Release(lpThis);
   return NOERROR;
}


STDMETHODIMP CntrLine_IPSite_GetWindowContext(
   LPOLEINPLACESITE            lpThis,
   LPOLEINPLACEFRAME FAR*      lplpFrame,
   LPOLEINPLACEUIWINDOW FAR*   lplpDoc,
   LPRECT                      lprcPosRect,
   LPRECT                      lprcClipRect,
   LPOLEINPLACEFRAMEINFO       lpFrameInfo
)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)g_lpApp;
   LPCONTAINERLINE lpContainerLine =
         ((struct COleInPlaceSiteImpl FAR*)lpThis)->lpContainerLine;

   // artificial AddRef in case object is destroyed during call
   CntrLine_IPSite_AddRef(lpThis);

   OLEDBG_BEGIN2("CntrLine_IPSite_GetWindowContext\r\n")

   /* OLE2NOTE: The server should fill in the "cb" field so that the
   **    container can tell what size structure the server is
   **    expecting. this enables this structure to be easily extended
   **    in future releases of OLE. the container should check this
   **    field so that it doesn't try to use fields that do not exist
   **    since the server may be using an old structure definition.
   **    Since this is the first release of OLE2.0, the structure is
   **    guaranteed to be at least large enough for the current
   **    definition of OLEINPLACEFRAMEINFO struct. thus we do NOT need
   **    to consider this an error if the server did not fill in this
   **    field correctly. this server may have trouble in the future,
   **    however, when the structure is extended.
   */
   *lplpFrame = (LPOLEINPLACEFRAME)&lpContainerApp->m_OleInPlaceFrame;
   (*lplpFrame)->lpVtbl->AddRef(*lplpFrame);   // must return AddRef'ed ptr

   // OLE2NOTE: an MDI app would have to provide *lplpDoc
   *lplpDoc  = NULL;

   ContainerLine_GetPosRect(lpContainerLine, lprcPosRect);
   ContainerDoc_GetClipRect(lpContainerLine->m_lpDoc, lprcClipRect);

   OleDbgOutRect3("CntrLine_IPSite_GetWindowContext (PosRect)",
lprcPosRect);
   OleDbgOutRect3("CntrLine_IPSite_GetWindowContext
(ClipRect)",lprcClipRect);
   lpFrameInfo->hwndFrame      = lpOutlineApp->m_hWndApp;

#if defined( MDI_VERSION )
   lpFrameInfo->fMDIApp        = TRUE;
#else
```

```
   lpFrameInfo->fMDIApp        = FALSE;
#endif

   lpFrameInfo->haccel         = lpContainerApp->m_hAccelIPCntr;
   lpFrameInfo->cAccelEntries  =
      GetAccelItemCount(lpContainerApp->m_hAccelIPCntr);

   OLEDBG_END2
   // release artificial AddRef
   CntrLine_IPSite_Release(lpThis);
   return NOERROR;
}


STDMETHODIMP CntrLine_IPSite_Scroll(
   LPOLEINPLACESITE    lpThis,
   SIZE                scrollExtent
)
{
   OleDbgOut2("CntrLine_IPSite_Scroll\r\n");
   return ResultFromScode(E_FAIL);
}


STDMETHODIMP CntrLine_IPSite_OnUIDeactivate(
   LPOLEINPLACESITE    lpThis,
   BOOL                fUndoable
)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleInPlaceSiteImpl FAR*)lpThis)->lpContainerLine;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpContainerLine->m_lpDoc;
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP) g_lpApp;
   LPLINELIST lpLL;
   int nIndex;
   MSG msg;
   HRESULT hrErr;

   // artificial AddRef in case object is destroyed during call
   CntrLine_IPSite_AddRef(lpThis);

   OLEDBG_BEGIN2("CntrLine_IPSite_OnUIDeactivate\r\n")

   lpContainerLine->m_fUIActive = FALSE;
   lpContainerLine->m_fIpChangesUndoable = fUndoable;
   lpContainerLine->m_lpDoc->m_hWndUIActiveObj = NULL;

   if (lpContainerLine == lpContainerLine->m_lpDoc->m_lpLastUIActiveLine) {

      lpContainerLine->m_lpDoc->m_lpLastUIActiveLine = NULL;

      /* OLE2NOTE: here we look ahead if there is a DBLCLK sitting in our
      **    message queue. if so, it could result in in-place activating
      **    another object. we want to avoid placing our tools and
      **    repainting if immediately another object is going to do the
      **    same. SO, if there is a DBLCLK in the queue for this document
      **    we will only set the fAddMyUI flag to indicate that this work
      **    is still to be done. if another object actually in-place
      **    activates then this flag will be cleared in
      **    IOleInPlaceSite::OnUIActivate. if it does NOT get cleared,
      **    then at the end of processing the DBLCLK message in our
      **    OutlineDocWndProc we will put our tools back.
      */
      if (! PeekMessage(&msg, lpOutlineDoc->m_hWndDoc,
            WM_LBUTTONDBLCLK, WM_LBUTTONDBLCLK,
            PM_NOREMOVE | PM_NOYIELD)) {

         /* OLE2NOTE: You need to generate QueryNewPalette only if
         **    you own the top level frame (ie. you are a top-level
         **    inplace container).
         */

         OleApp_QueryNewPalette((LPOLEAPP)g_lpApp);

#if defined( USE_DOCTOOLS )
         ContainerDoc_AddDocLevelTools(lpContainerLine->m_lpDoc);
#endif

#if defined( USE_FRAMETOOLS )
         ContainerDoc_AddFrameLevelUI(lpContainerLine->m_lpDoc);
#endif
      } else {
         lpContainerLine->m_lpDoc->m_fAddMyUI = TRUE;
      }

      /* OLE2NOTE: we should re-take focus. the in-place server window
      **    previously had the focus; this window has just been removed.
      */
      SetFocus(OutlineDoc_GetWindow((LPOUTLINEDOC)lpContainerLine-
>m_lpDoc));

      // force the line to redraw to remove in-place active hatch
      lpLL = OutlineDoc_GetLineList(lpOutlineDoc);
      nIndex = LineList_GetLineIndex(lpLL, (LPLINE)lpContainerLine);
      LineList_ForceLineRedraw(lpLL, nIndex, TRUE);
   }

#if defined( UNDOSUPPORTED )

   /* OLE2NOTE: an outside-in style container that supports UNDO would
   **    call IOleObject::DoVerb(OLEIVERB_HIDE) to make the in-place
   **    object go invisible. when it wants the in-place active object
   **    to discard its undo state, it would call
   **    IOleInPlaceObject::InPlaceDeactivate when it wants the object
   **    to discard its undo state. there is no need for an outside-in
   **    style container to call
   **    IOleObject::DoVerb(OLEIVERB_DISCARDUNDOSTATE). if either the
   **    container or the object do not support UNDO, then the
   **    container might as well immediately call InPlaceDeactivate
   **    instead of calling DoVerb(HIDE).
   **
   **    an inside-out style container that supports UNDO would simply
   **    UIDeactivate the object. it would call
   **    IOleObject::DoVerb(OLEIVERB_DISCARDUNDOSTATE) when it wants
   **    the object discard its undo state. it would call
   **    IOleInPlaceObject::InPlaceDeactivate if it wants the object
   **    to take down its window.
   */
   if (! g_fInsideOutContainer || !lpContainerLine->m_fInsideOutObj) {

      if (lpContainerLine->m_fIpChangesUndoable) {
         ContainerLine_DoVerb(
               lpContainerLine,OLEIVERB_HIDE,NULL,FALSE,FALSE);
      } else {
         lpContainerLine->m_lpOleIPObj->lpVtbl->InPlaceDeactivate(
               lpContainerLine->m_lpOleIPObj);
      }
      lpContainerLine->m_fIpVisible = FALSE;
      lpContainerLine->m_hWndIpObject = NULL;
   }
#else

   /* OLE2NOTE: an outside-in style container that does NOT support
   **    UNDO would immediately tell the UIDeactivated server (UI
   **    removed) to IOleInPlaceObject::InPlaceDeactivate.
   **
   **    an inside-out style container that does NOT support UNDO
   **    would call IOleObject::DoVerb(OLEIVERB_DISCARDUNDOSTATE) to
   **    tell the object to discard its undo state. it would call
   **    IOleInPlaceObject::InPlaceDeactivate if it wants the object
   **    to take down its window.
   */

   if (g_fInsideOutContainer) {

      if (lpContainerLine->m_fInsideOutObj) {

         if (lpContainerLine->m_fIpChangesUndoable) {
            OLEDBG_BEGIN3("ContainerLine_DoVerb(OLEIVERB_DISCARDUNDOSTATE)
called!\r\n")
            ContainerLine_DoVerb(lpContainerLine,
                  OLEIVERB_DISCARDUNDOSTATE,NULL,FALSE,FALSE);
            OLEDBG_END3
         }

      } else {    // !fInsideOutObj

         /* OLE2NOTE: (INSIDEOUT CONTAINER) if the object is not
         **    registered OLEMISC_ACTIVATEWHENVISIBLE, then we will
         **    make the object behave in an outside-in manner. since
         **    we do NOT deal with UNDO we can simply
         **    InPlaceDeactivate the object. it should NOT be
         **    allowed to leave its window visible when
         **    UIDeactivated.
         */
         OLEDBG_BEGIN2("IOleInPlaceObject::InPlaceDeactivate called\r\n")
         hrErr = lpContainerLine->m_lpOleIPObj->lpVtbl->InPlaceDeactivate(
                  lpContainerLine->m_lpOleIPObj);
         OLEDBG_END2
         if (hrErr != NOERROR) {
            OleDbgOutHResult("IOleInPlaceObject::InPlaceDeactivate
returned", hrErr);
         }
      }

   } else {

      /* OLE2NOTE: (OUTSIDE-IN CONTAINER) since we do NOT deal with
      **    UNDO we can simply InPlaceDeactivate the object. it
      **    should NOT be allowed to leave its window visible when
      **    UIDeactivated.
        **
        **    NOTE: In-place servers must handle InPlaceDeactivate
        **    being called before its call to
        **    IOleInPlaceSite::OnUIDeactivate returns. in case there
        **    are misbehaving servers out there, one way to work around
        **    this problem is to call
        **    IOleObject::DoVerb(OLEIVERB_HIDE...) here.
      */
      OLEDBG_BEGIN2("IOleInPlaceObject::InPlaceDeactivate called\r\n")
      hrErr = lpContainerLine->m_lpOleIPObj->lpVtbl->InPlaceDeactivate(
            lpContainerLine->m_lpOleIPObj);
      OLEDBG_END2
      if (hrErr != NOERROR) {
         OleDbgOutHResult("IOleInPlaceObject::InPlaceDeactivate returned",
hrErr);
      }
   }

#endif // ! UNDOSUPPORTED

   OLEDBG_END2
   // release artificial AddRef
   CntrLine_IPSite_Release(lpThis);
   return NOERROR;
}


STDMETHODIMP CntrLine_IPSite_OnInPlaceDeactivate(LPOLEINPLACESITE lpThis)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleInPlaceSiteImpl FAR*)lpThis)->lpContainerLine;

   // artificial AddRef in case object is destroyed during call
   CntrLine_IPSite_AddRef(lpThis);
   OLEDBG_BEGIN2("CntrLine_IPSite_OnInPlaceDeactivate\r\n");

   lpContainerLine->m_fIpActive            = FALSE;
   lpContainerLine->m_fIpVisible           = FALSE;
   lpContainerLine->m_fIpChangesUndoable   = FALSE;
   lpContainerLine->m_hWndIpObject         = NULL;

   OleStdRelease((LPUNKNOWN) lpContainerLine->m_lpOleIPObj);
   lpContainerLine->m_lpOleIPObj = NULL;
   lpContainerLine->m_lpDoc->m_cIPActiveObjects--;

   OLEDBG_END2
   // release artificial AddRef
   CntrLine_IPSite_Release(lpThis);
   return NOERROR;
}


STDMETHODIMP CntrLine_IPSite_DiscardUndoState(LPOLEINPLACESITE lpThis)
{
   OleDbgOut2("CntrLine_IPSite_DiscardUndoState\r\n");
   return NOERROR;
}


STDMETHODIMP CntrLine_IPSite_DeactivateAndUndo(LPOLEINPLACESITE lpThis)
{
   OleDbgOut2("CntrLine_IPSite_DeactivateAndUndo\r\n");
   return NOERROR;
}


STDMETHODIMP CntrLine_IPSite_OnPosRectChange(
   LPOLEINPLACESITE    lpThis,
   LPCRECT             lprcPosRect
)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleInPlaceSiteImpl FAR*)lpThis)->lpContainerLine;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpContainerLine->m_lpDoc;
   LPSCALEFACTOR lpscale = OutlineDoc_GetScaleFactor(lpOutlineDoc);
   LPLINE lpLine = (LPLINE)lpContainerLine;
   LPLINELIST lpLL;
   int nIndex;
   RECT rcClipRect;
   RECT rcNewPosRect;
   SIZEL sizelPix;
   SIZEL sizelHim;
   int nIPObjHeight = lprcPosRect->bottom - lprcPosRect->top;
   int nIPObjWidth = lprcPosRect->right - lprcPosRect->left;

   // artificial AddRef in case object is destroyed during call
   CntrLine_IPSite_AddRef(lpThis);

   OLEDBG_BEGIN2("CntrLine_IPSite_OnPosRectChange\r\n")
   OleDbgOutRect3("CntrLine_IPSite_OnPosRectChange (PosRect --IN)",
(LPRECT)lprcPosRect);

   /* OLE2NOTE: if the in-place container does NOT impose any
   **    size contraints on its in-place active objects, then it may
   **    simply grant the size requested by the object by immediately
   **    calling IOleInPlaceObject::SetObjectRects with the
   **    lprcPosRect passed by the in-place object.
   **
   **    Container Outline, however, imposes a size constraint on its
   **    embedded objects (see comment in ContainerLine_UpdateExtent),
   **    thus it is necessary to calculate the size that the in-place
   **    active object is allowed to be.
   **
   **    Here we need to know the new extents of the in-place object.
   **    We can NOT directly ask the object via IOleObject::GetExtent
   **    because this method will retreive the extents of the last
   **    cached metafile. the cache has not yet been updated by this
   **    point. We can not reliably call IOleObject::GetExtent
   **    because, despite the fact that will be delegated to the
   **    object properly, some objects may not have reformated their
   **    object and computed the new extents at the time of calling
   **    OnPosRectChange.
   **
   **    the best we can do to get the new extents of the object is
   **    to determine the scale factor that the object was operating at
   **    prior to the OnPosRect call and scale the new lprcPosRect
   **    using this scale factor back to HIMETRIC units.
   */
   if (lpContainerLine->m_sizeInHimetric.cx > 0 &&
      lpContainerLine->m_sizeInHimetric.cy > 0) {
      sizelHim.cx = lpLine->m_nWidthInHimetric;
      sizelHim.cy = lpLine->m_nHeightInHimetric;
      XformSizeInHimetricToPixels(NULL, &sizelHim, &sizelPix);
      sizelHim.cx = lpContainerLine->m_sizeInHimetric.cx *
               nIPObjWidth / sizelPix.cx;
      sizelHim.cy = lpContainerLine->m_sizeInHimetric.cy *
               nIPObjHeight / sizelPix.cy;

      // Convert size back to 100% zoom
      sizelHim.cx = sizelHim.cx * lpscale->dwSxD / lpscale->dwSxN;
      sizelHim.cy = sizelHim.cy * lpscale->dwSyD / lpscale->dwSyN;
   } else {
      sizelHim.cx = (long)DEFOBJWIDTH;
      sizelHim.cy = (long)DEFOBJHEIGHT;
      XformSizeInHimetricToPixels(NULL, &sizelHim, &sizelPix);
      sizelHim.cx = sizelHim.cx * nIPObjWidth / sizelPix.cx;
      sizelHim.cy = sizelHim.cy * nIPObjHeight / sizelPix.cy;
   }

   ContainerLine_UpdateExtent(lpContainerLine, &sizelHim);
   ContainerLine_GetPosRect(lpContainerLine, (LPRECT)&rcNewPosRect);
   ContainerDoc_GetClipRect(lpContainerLine->m_lpDoc, (LPRECT)&rcClipRect);

#if defined( _DEBUG )
   OleDbgOutRect3("CntrLine_IPSite_OnPosRectChange (PosRect --OUT)",
         (LPRECT)&rcNewPosRect);
   OleDbgOutRect3("CntrLine_IPSite_OnPosRectChange (ClipRect--OUT)",
         (LPRECT)&rcClipRect);
#endif
```

```
   OLEDBG_BEGIN2("IOleInPlaceObject::SetObjectRects called\r\n")
   lpContainerLine->m_lpOleIPObj->lpVtbl->SetObjectRects(
         lpContainerLine->m_lpOleIPObj,
         (LPRECT)&rcNewPosRect,
         (LPRECT)&rcClipRect
   );
   OLEDBG_END2

   /* OLE2NOTE: (INSIDEOUT CONTAINER) Because this object just changed
   **    size, this may cause other in-place active objects in the
   **    document to move. in ICNTROTL's case any object below this
   **    object would be affected. in this case it would be necessary
   **    to call SetObjectRects to each affected in-place active object.
   */
   if (g_fInsideOutContainer) {
      lpLL = OutlineDoc_GetLineList(lpOutlineDoc);
      nIndex = LineList_GetLineIndex(lpLL, (LPLINE)lpContainerLine);

      ContainerDoc_UpdateInPlaceObjectRects(
            lpContainerLine->m_lpDoc, nIndex);
   }
   OLEDBG_END2
   // release artificial AddRef
   CntrLine_IPSite_Release(lpThis);
   return NOERROR;
}
```

CNTRLINE.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Container Sample Code
**
**    cntrline.c
**
**    This file contains ContainerLine methods.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"

OLEDBGDATA



extern LPOUTLINEAPP         g_lpApp;
extern IUnknownVtbl         g_CntrLine_UnknownVtbl;
extern IOleClientSiteVtbl   g_CntrLine_OleClientSiteVtbl;
extern IAdviseSinkVtbl      g_CntrLine_AdviseSinkVtbl;

#if defined( INPLACE_CNTR )
extern IOleInPlaceSiteVtbl  g_CntrLine_OleInPlaceSiteVtbl;
extern BOOL g_fInsideOutContainer;
#endif  // INPLACE_CNTR

// REVIEW: should use string resource for messages
OLECHAR ErrMsgDoVerb[] = OLESTR("OLE object action failed!");


/* prototype for static functions */
static void InvertDiffRect(LPRECT lprcPix, LPRECT lprcObj, HDC hDC);


/*************************************************************************
** ContainerLine
**    This object represents the location within the container where
**    the embedded/linked object lives. It exposes interfaces to the
**    object that allow the object to get information about its
**    embedding site and to announce notifications of important events
**    (changed, closed, saved)
**
**    The ContainerLine exposes the following interfaces:
**      IUnknown
**      IOleClientSite
**      IAdviseSink
*************************************************************************/
```

```
/*************************************************************************
** ContainerLine::IUnknown interface implementation
*************************************************************************/


// IUnknown::QueryInterface
STDMETHODIMP CntrLine_Unk_QueryInterface(
      LPUNKNOWN           lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleClientSiteImpl FAR*)lpThis)->lpContainerLine;

   return ContainerLine_QueryInterface(lpContainerLine, riid, lplpvObj);
}


// IUnknown::AddRef
STDMETHODIMP_(ULONG) CntrLine_Unk_AddRef(LPUNKNOWN lpThis)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleClientSiteImpl FAR*)lpThis)->lpContainerLine;

   OleDbgAddRefMethod(lpThis, "IUnknown");

   return ContainerLine_AddRef(lpContainerLine);
}


// IUnknown::Release
STDMETHODIMP_(ULONG) CntrLine_Unk_Release(LPUNKNOWN lpThis)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleClientSiteImpl FAR*)lpThis)->lpContainerLine;

   OleDbgReleaseMethod(lpThis, "IUnknown");

   return ContainerLine_Release(lpContainerLine);
}


/*************************************************************************
** ContainerLine::IOleClientSite interface implementation
*************************************************************************/

// IOleClientSite::QueryInterface
STDMETHODIMP CntrLine_CliSite_QueryInterface(
      LPOLECLIENTSITE     lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleClientSiteImpl FAR*)lpThis)->lpContainerLine;

   return ContainerLine_QueryInterface(lpContainerLine, riid, lplpvObj);
}


// IOleClientSite::AddRef
STDMETHODIMP_(ULONG) CntrLine_CliSite_AddRef(LPOLECLIENTSITE lpThis)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleClientSiteImpl FAR*)lpThis)->lpContainerLine;

   OleDbgAddRefMethod(lpThis, "IOleClientSite");

   return ContainerLine_AddRef(lpContainerLine);
}


// IOleClientSite::Release
STDMETHODIMP_(ULONG) CntrLine_CliSite_Release(LPOLECLIENTSITE lpThis)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleClientSiteImpl FAR*)lpThis)->lpContainerLine;

   OleDbgReleaseMethod(lpThis, "IOleClientSite");

   return ContainerLine_Release(lpContainerLine);
}


// IOleClientSite::SaveObject
STDMETHODIMP CntrLine_CliSite_SaveObject(LPOLECLIENTSITE lpThis)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleClientSiteImpl FAR*)lpThis)->lpContainerLine;
   LPPERSISTSTORAGE lpPersistStg = lpContainerLine->m_lpPersistStg;
   SCODE sc = S_OK;
   HRESULT hrErr;

   OLEDBG_BEGIN2("CntrLine_CliSite_SaveObject\r\n")

   if (! lpPersistStg) {
      /* OLE2NOTE: The object is NOT loaded. a container must be
      **    prepared for the fact that an object that it thinks it
      **    has unloaded may still call IOleClientSite::SaveObject.
      **    in this case the container should fail the save call and
      **    NOT try to reload the object. this scenario arises if you
      **    have a in-process server (DLL object) which has a
      **    connected linking client. even after the embedding
      **    container unloads the DLL object, the link connection
      **    keeps the object alive. it may then as part of its
      **    shutdown try to call IOCS::SaveObject, but then it is too
      **    late.
      */
      OLEDBG_END2
      return ResultFromScode(E_FAIL);
   }

   // mark ContainerDoc as now dirty
   OutlineDoc_SetModified(
         (LPOUTLINEDOC)lpContainerLine->m_lpDoc, TRUE, TRUE, FALSE);

   /* Tell OLE object to save itself
   ** OLE2NOTE: it is NOT sufficient to ONLY call
   **    IPersistStorage::Save method. it is also necessary to call
   **    WriteClassStg. the helper API OleSave does this automatically.
   */
   OLEDBG_BEGIN2("OleSave called\r\n")
   hrErr=OleSave(lpPersistStg,lpContainerLine->m_lpStg,
TRUE/*fSameAsLoad*/);
   OLEDBG_END2

   if (hrErr != NOERROR) {
      OleDbgOutHResult("WARNING: OleSave returned", hrErr);
      sc = GetScode(hrErr);
   }

   // OLE2NOTE: even if OleSave fails, SaveCompleted must be called.
   OLEDBG_BEGIN2("IPersistStorage::SaveCompleted called\r\n")
   hrErr = lpPersistStg->lpVtbl->SaveCompleted(lpPersistStg, NULL);
   OLEDBG_END2

   if (hrErr != NOERROR) {
      OleDbgOutHResult("WARNING: SaveCompleted returned",hrErr);
      if (sc == S_OK)
         sc = GetScode(hrErr);
   }

   OLEDBG_END2
   return ResultFromScode(sc);
}


// IOleClientSite::GetMoniker
STDMETHODIMP CntrLine_CliSite_GetMoniker(
      LPOLECLIENTSITE     lpThis,
      DWORD               dwAssign,
      DWORD               dwWhichMoniker,
      LPMONIKER FAR*      lplpmk
)
{
   LPCONTAINERLINE lpContainerLine;

   lpContainerLine=((struct COleClientSiteImpl FAR*)lpThis)-
>lpContainerLine;

   OLEDBG_BEGIN2("CntrLine_CliSite_GetMoniker\r\n")

   // OLE2NOTE: we must make sure to set output pointer parameters to NULL
   *lplpmk = NULL;

   switch (dwWhichMoniker) {

      case OLEWHICHMK_CONTAINER:
         /* OLE2NOTE: create a FileMoniker which identifies the
         **    entire container document.
         */
         *lplpmk = OleDoc_GetFullMoniker(
               (LPOLEDOC)lpContainerLine->m_lpDoc,
               dwAssign
         );
         break;

      case OLEWHICHMK_OBJREL:

         /* OLE2NOTE: create an ItemMoniker which identifies the
         **    OLE object relative to the container document.
         */
         *lplpmk = ContainerLine_GetRelMoniker(lpContainerLine, dwAssign);
         break;

      case OLEWHICHMK_OBJFULL:
      {
         /* OLE2NOTE: create an absolute moniker which identifies the
         **    OLE object in the container document. this moniker is
         **    created as a composite of the absolute moniker for the
         **    entire document appended with an item moniker which
         **    identifies the OLE object relative to the document.
         */

         *lplpmk = ContainerLine_GetFullMoniker(lpContainerLine, dwAssign);
         break;
      }
   }

   OLEDBG_END2

   if (*lplpmk != NULL)
      return NOERROR;
   else
      return ResultFromScode(E_FAIL);
}


// IOleClientSite::GetContainer
STDMETHODIMP CntrLine_CliSite_GetContainer(
      LPOLECLIENTSITE     lpThis,
      LPOLECONTAINER FAR* lplpContainer
)
{
   LPCONTAINERLINE lpContainerLine;
   HRESULT hrErr;

   // artificial AddRef in case object is deleted during call
   CntrLine_CliSite_AddRef(lpThis);
   OLEDBG_BEGIN2("CntrLine_CliSite_GetContainer\r\n")

   lpContainerLine=((struct COleClientSiteImpl FAR*)lpThis)-
>lpContainerLine;

   hrErr = OleDoc_QueryInterface(
         (LPOLEDOC)lpContainerLine->m_lpDoc,
         &IID_IOleContainer,
         (LPVOID FAR*)lplpContainer
   );

   OLEDBG_END2
   // release artificial AddRef
   CntrLine_CliSite_Release(lpThis);
   return hrErr;
}


// IOleClientSite::ShowObject
STDMETHODIMP CntrLine_CliSite_ShowObject(LPOLECLIENTSITE lpThis)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleClientSiteImpl FAR*)lpThis)->lpContainerLine;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpContainerLine->m_lpDoc;
   LPLINELIST lpLL = OutlineDoc_GetLineList(lpOutlineDoc);
   int nIndex = LineList_GetLineIndex(lpLL, (LPLINE)lpContainerLine);
   HWND hWndFrame = OutlineApp_GetFrameWindow(g_lpApp);

   // artificial AddRef in case object is deleted during call
   CntrLine_CliSite_AddRef(lpThis);

   OLEDBG_BEGIN2("CntrLine_CliSite_ShowObject\r\n")

   /* make sure our doc window is visible and not minimized.
   **    the OutlineDoc_ShowWindow function will cause the app window
   **    to show itself SW_SHOWNORMAL.
   */
   if (! IsWindowVisible(hWndFrame) || IsIconic(hWndFrame))
      OutlineDoc_ShowWindow(lpOutlineDoc);

   BringWindowToTop(hWndFrame);

   /* make sure that the OLE object is currently in view. if necessary
   **    scroll the document in order to bring it into view.
   */
   LineList_ScrollLineIntoView(lpLL, nIndex);

#if defined( INPLACE_CNTR )
   /* after the in-place object is scrolled into view, we need to ask
   **    it to update its rect for the new clip rect coordinates
   */
   ContainerDoc_UpdateInPlaceObjectRects((LPCONTAINERDOC)lpOutlineDoc, 0);
#endif
```

```
   OLEDBG_END2
   // release artificial AddRef
   CntrLine_CliSite_Release(lpThis);
   return NOERROR;
}


// IOleClientSite::OnShowWindow
STDMETHODIMP CntrLine_CliSite_OnShowWindow(LPOLECLIENTSITE lpThis, BOOL
fShow)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleClientSiteImpl FAR*)lpThis)->lpContainerLine;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpContainerLine->m_lpDoc;
   LPLINELIST lpLL = OutlineDoc_GetLineList(lpOutlineDoc);
   int nIndex = LineList_GetLineIndex(lpLL, (LPLINE)lpContainerLine);

   // artificial AddRef in case object is deleted during call
   CntrLine_CliSite_AddRef(lpThis);

   if (fShow) {
      OLEDBG_BEGIN2("CntrLine_CliSite_OnShowWindow(TRUE)\r\n")

      /* OLE2NOTE: we need to hatch out the OLE object now; it has
      **    just been opened in a window elsewhere (open editing as
      **    opposed to in-place activation).
      **    force the line to re-draw with the hatch.
      */
      lpContainerLine->m_fObjWinOpen = TRUE;
      LineList_ForceLineRedraw(lpLL, nIndex, FALSE /*fErase*/);

   } else {
      OLEDBG_BEGIN2("CntrLine_CliSite_OnShowWindow(FALSE)\r\n")

      /* OLE2NOTE: the object associated with this container site has
      **    just closed its server window. we should now remove the
      **    hatching that indicates that the object is open
      **    elsewhere. also our window should now come to the top.
      **    force the line to re-draw without the hatch.
      */
      lpContainerLine->m_fObjWinOpen = FALSE;
      LineList_ForceLineRedraw(lpLL, nIndex, TRUE /*fErase*/);

      BringWindowToTop(lpOutlineDoc->m_hWndDoc);
      SetFocus(lpOutlineDoc->m_hWndDoc);
   }

   OLEDBG_END2
   // release artificial AddRef
   CntrLine_CliSite_Release(lpThis);

   return NOERROR;
}
```

```
// IOleClientSite::RequestNewObjectLayout
STDMETHODIMP CntrLine_CliSite_RequestNewObjectLayout(LPOLECLIENTSITE lpThis)
{
   OleDbgOut2("CntrLine_CliSite_RequestNewObjectLayout\r\n");

   /* OLE2NOTE: this method is NOT yet used. it is for future layout
   **    negotiation support.
   */
   return ResultFromScode(E_NOTIMPL);
}


/*************************************************************************
** ContainerLine::IAdviseSink interface implementation
*************************************************************************/

// IAdviseSink::QueryInterface
STDMETHODIMP CntrLine_AdvSink_QueryInterface(
      LPADVISESINK        lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleClientSiteImpl FAR*)lpThis)->lpContainerLine;

   return ContainerLine_QueryInterface(lpContainerLine, riid, lplpvObj);
}


// IAdviseSink::AddRef
STDMETHODIMP_(ULONG) CntrLine_AdvSink_AddRef(LPADVISESINK lpThis)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleClientSiteImpl FAR*)lpThis)->lpContainerLine;

   OleDbgAddRefMethod(lpThis, "IAdviseSink");

   return ContainerLine_AddRef(lpContainerLine);
}


// IAdviseSink::Release
STDMETHODIMP_(ULONG) CntrLine_AdvSink_Release (LPADVISESINK lpThis)
{
   LPCONTAINERLINE lpContainerLine =
         ((struct COleClientSiteImpl FAR*)lpThis)->lpContainerLine;

   OleDbgReleaseMethod(lpThis, "IAdviseSink");

   return ContainerLine_Release(lpContainerLine);
}


// IAdviseSink::OnDataChange
```

```
STDMETHODIMP_(void) CntrLine_AdvSink_OnDataChange(
      LPADVISESINK        lpThis,
      FORMATETC FAR*      lpFormatetc,
      STGMEDIUM FAR*      lpStgmed
)
{
   OleDbgOut2("CntrLine_AdvSink_OnDataChange\r\n");
   // We are not interested in data changes (only view changes)
   //      (ie. nothing to do)
}


STDMETHODIMP_(void) CntrLine_AdvSink_OnViewChange(
      LPADVISESINK        lpThis,
      DWORD               aspects,
      LONG                lindex
)
{
   LPCONTAINERLINE lpContainerLine;
   LPOUTLINEDOC lpOutlineDoc;
   HWND hWndDoc;
   LPLINELIST lpLL;
   MSG msg;
   int nIndex;

   // artificial AddRef in case object is destroyed during call
   CntrLine_AdvSink_AddRef(lpThis);

   OLEDBG_BEGIN2("CntrLine_AdvSink_OnViewChange\r\n")

   lpContainerLine = ((struct CAdviseSinkImpl FAR*)lpThis)->lpContainerLine;
   lpOutlineDoc = (LPOUTLINEDOC)lpContainerLine->m_lpDoc;

   /* OLE2NOTE: at this point we simply invalidate the rectangle of
   **    the object to force a repaint in the future, we mark
   **    that the extents of the object may have changed
   **    (m_fDoGetExtent=TRUE), and we post a message
   **    (WM_U_UPDATEOBJECTEXTENT) to our document that one or more
   **    OLE objects may need to have their extents updated. later
   **    when this message is processed, the document loops through
   **    all lines to see if any are marked as needing an extent update.
   **    if infact the extents did change. this can be done by calling
   **    IViewObject2::GetExtent to retreive the object's current
   **    extents and comparing with the last known extents for the
   **    object. if the extents changed, then relayout space for the
   **    object before drawing. we postpone the check to get
   **    the extents now because OnViewChange is an asyncronis method,
   **    and we have to careful not to call any syncronis methods back
   **    to the object. while it WOULD be OK to call the
   **    IViewObject2::GetExtent method within the asyncronis
   **    OnViewChange method (because this method is handled by the
   **    object handler and never remoted), it is good practise to not
   **    call any object methods from within an asyncronis
   **    notification method.
   **    if there is already WM_U_UPDATEOBJECTEXTENT message waiting
   **    in our message queue, there is no need to post another one.
   **    in this way, if the server is updating quicker than we can
   **    keep up, we do not make unneccsary GetExtent calls. also if
   **    drawing is disabled, we postpone updating the extents of any
   **    objects until drawing is re-enabled.
   */
   lpContainerLine->m_fDoGetExtent = TRUE;
   hWndDoc = OutlineDoc_GetWindow((LPOUTLINEDOC)lpContainerLine->m_lpDoc);

   if (lpOutlineDoc->m_nDisableDraw == 0 &&
      ! PeekMessage(&msg, hWndDoc,
         WM_U_UPDATEOBJECTEXTENT, WM_U_UPDATEOBJECTEXTENT,
         PM_NOREMOVE | PM_NOYIELD)) {
      PostMessage(hWndDoc, WM_U_UPDATEOBJECTEXTENT, 0, 0L);
   }

   // force the modified line to redraw.
   lpLL = OutlineDoc_GetLineList(lpOutlineDoc);
   nIndex = LineList_GetLineIndex(lpLL, (LPLINE)lpContainerLine);
   LineList_ForceLineRedraw(lpLL, nIndex, TRUE /*fErase*/);

   OLEDBG_END2
   // release artificial AddRef
   CntrLine_AdvSink_Release(lpThis);
}


// IAdviseSink::OnRename
STDMETHODIMP_(void) CntrLine_AdvSink_OnRename(
      LPADVISESINK        lpThis,
      LPMONIKER           lpmk
)
{
   OleDbgOut2("CntrLine_AdvSink_OnRename\r\n");
   /* OLE2NOTE: the Embedding Container has nothing to do here. this
   **    notification is important for linking situations. it tells
   **    the OleLink objects to update their moniker because the
   **    source object has been renamed (track the link source).
   */
}


// IAdviseSink::OnSave
STDMETHODIMP_(void) CntrLine_AdvSink_OnSave(LPADVISESINK lpThis)
{
   OleDbgOut2("CntrLine_AdvSink_OnSave\r\n");
   /* OLE2NOTE: the Embedding Container has nothing to do here. this
   **    notification is only useful to clients which have set up a
   **    data cache with the ADVFCACHE_ONSAVE flag.
   */
}


// IAdviseSink::OnClose
STDMETHODIMP_(void) CntrLine_AdvSink_OnClose(LPADVISESINK lpThis)
```

```
{
   OleDbgOut2("CntrLine_AdvSink_OnClose\r\n");
   /* OLE2NOTE: the Embedding Container has nothing to do here. this
   **    notification is important for the OLE's default object handler
   **    and the OleLink object. it tells them the remote object is
   **    shutting down.
   */
}


/*************************************************************************
** ContainerLine Support Functions
*************************************************************************/


/* ContainerLine_Init
** ------------------
**  Initialize fields in a newly constructed ContainerLine line object.
**  NOTE: ref cnt of ContainerLine initialized to 0
*/
void ContainerLine_Init(LPCONTAINERLINE lpContainerLine, int nTab, HDC hDC)
{
   Line_Init((LPLINE)lpContainerLine, nTab, hDC);  // init base class fields

   ((LPLINE)lpContainerLine)->m_lineType           = CONTAINERLINETYPE;
   ((LPLINE)lpContainerLine)->m_nWidthInHimetric   = DEFOBJWIDTH;
   ((LPLINE)lpContainerLine)->m_nHeightInHimetric  = DEFOBJHEIGHT;
   lpContainerLine->m_cRef                         = 0;
   lpContainerLine->m_szStgName[0]                 = '\0';
   lpContainerLine->m_fObjWinOpen                  = FALSE;
   lpContainerLine->m_fMonikerAssigned             = FALSE;
   lpContainerLine->m_dwDrawAspect                 = DVASPECT_CONTENT;

   lpContainerLine->m_fGuardObj                    = FALSE;
   lpContainerLine->m_fDoGetExtent                 = FALSE;
   lpContainerLine->m_fDoSetExtent                 = FALSE;
   lpContainerLine->m_sizeInHimetric.cx            = -1;
   lpContainerLine->m_sizeInHimetric.cy            = -1;

   lpContainerLine->m_lpStg                        = NULL;
   lpContainerLine->m_lpDoc                        = NULL;
   lpContainerLine->m_lpOleObj                     = NULL;
   lpContainerLine->m_lpViewObj2                   = NULL;
   lpContainerLine->m_lpPersistStg                 = NULL;
   lpContainerLine->m_lpOleLink                    = NULL;
   lpContainerLine->m_dwLinkType                   = 0;
   lpContainerLine->m_fLinkUnavailable             = FALSE;
   lpContainerLine->m_lpszShortType                = NULL;

#if defined( INPLACE_CNTR )
   lpContainerLine->m_fIpActive                    = FALSE;
   lpContainerLine->m_fUIActive                    = FALSE;
   lpContainerLine->m_fIpVisible                   = FALSE;
   lpContainerLine->m_lpOleIPObj                   = NULL;
   lpContainerLine->m_fInsideOutObj                = FALSE;
   lpContainerLine->m_fIpChangesUndoable           = FALSE;
   lpContainerLine->m_fIpServerRunning             = FALSE;
   lpContainerLine->m_hWndIpObject                 = NULL;
   lpContainerLine->m_nHorizScrollShift            = 0;
#endif  // INPLACE_CNTR

   INIT_INTERFACEIMPL(
         &lpContainerLine->m_Unknown,
         &g_CntrLine_UnknownVtbl,
         lpContainerLine
   );

   INIT_INTERFACEIMPL(
         &lpContainerLine->m_OleClientSite,
         &g_CntrLine_OleClientSiteVtbl,
         lpContainerLine
   );

   INIT_INTERFACEIMPL(
         &lpContainerLine->m_AdviseSink,
         &g_CntrLine_AdviseSinkVtbl,
         lpContainerLine
   );

#if defined( INPLACE_CNTR )
   INIT_INTERFACEIMPL(
         &lpContainerLine->m_OleInPlaceSite,
         &g_CntrLine_OleInPlaceSiteVtbl,
         lpContainerLine
   );
#endif  // INPLACE_CNTR
}


/* Setup the OLE object associated with the ContainerLine */
BOOL ContainerLine_SetupOleObject(
      LPCONTAINERLINE         lpContainerLine,
      BOOL                    fDisplayAsIcon,
      HGLOBAL                 hMetaPict
)
{
   DWORD dwDrawAspect = (fDisplayAsIcon ? DVASPECT_ICON : DVASPECT_CONTENT);
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpContainerLine->m_lpDoc;

   /* Cache a pointer to the IViewObject2* interface. *Required*
   **      we need this everytime we draw the object.
   **
   ** OLE2NOTE: We require the object to support IViewObject2
   **    interface. this is an extension to the IViewObject interface
   **    that was added with the OLE 2.01 release. This interface must
   **    be supported by all object handlers and DLL-based objects.
   */
   lpContainerLine->m_lpViewObj2 = (LPVIEWOBJECT2)OleStdQueryInterface(
         (LPUNKNOWN)lpContainerLine->m_lpOleObj, &IID_IViewObject2);
   if (! lpContainerLine->m_lpViewObj2) {
```

```
#if defined( _DEBUG )
      OleDbgAssertSz(
         lpContainerLine->m_lpViewObj2,"IViewObject2 NOT supported\r\n");
#endif
      return FALSE;
   }

   // Cache a pointer to the IPersistStorage* interface. *Required*
   //      we need this everytime we save the object.
   lpContainerLine->m_lpPersistStg = (LPPERSISTSTORAGE)OleStdQueryInterface(
         (LPUNKNOWN)lpContainerLine->m_lpOleObj, &IID_IPersistStorage);
   if (! lpContainerLine->m_lpPersistStg) {
      OleDbgAssert(lpContainerLine->m_lpPersistStg);
      return FALSE;
   }

   // Cache a pointer to the IOleLink* interface if supported. *Optional*
   //      if supported the object is a link. we need this to manage the
link
   lpContainerLine->m_lpOleLink = (LPOLELINK)OleStdQueryInterface(
         (LPUNKNOWN)lpContainerLine->m_lpOleObj, &IID_IOleLink);
   if (lpContainerLine->m_lpOleLink) {
      OLEDBG_BEGIN2("IOleLink::GetUpdateOptions called\r\n")
      lpContainerLine->m_lpOleLink->lpVtbl->GetUpdateOptions(
            lpContainerLine->m_lpOleLink, &lpContainerLine->m_dwLinkType);
      OLEDBG_END2
   } else
      lpContainerLine->m_dwLinkType = 0;  // NOT a link

   /* get the short user type name of the object. this
   **    is used all the time when we have to build the object
   **    verb menu. we will cache this information to make it
   **    quicker to build the verb menu.
   */
   OleDbgAssert(lpContainerLine->m_lpszShortType == NULL);
   OLEDBG_BEGIN2("IOleObject::GetUserType called\r\n")
   lpContainerLine->m_lpOleObj->lpVtbl->GetUserType(
         lpContainerLine->m_lpOleObj,
         USERCLASSTYPE_SHORT,
         (LPOLESTR FAR*)&lpContainerLine->m_lpszShortType
   );
   OLEDBG_END2

   /* Perform that standard setup for the OLE object. this includes:
   **      setup View advise
   **      Call IOleObject::SetHostNames
   **      Call OleSetContainedObject
   */
   OleStdSetupAdvises(
         lpContainerLine->m_lpOleObj,
         dwDrawAspect,
         (LPOLESTR)APPNAME,
         lpOutlineDoc->m_lpszDocTitle,
         (LPADVISESINK)&lpContainerLine->m_AdviseSink,
         TRUE    /*fCreate*/
   );

#if defined( INPLACE_CNTR )
   /* OLE2NOTE: (INSIDE-OUT CONTAINER) An inside-out container should
   **    check if the object is an inside-out and prefers to be
   **    activated when visible type of object. if not the object
   **    should not be allowed to keep its window up after it gets
   **    UIDeactivated.
   */
   if (g_fInsideOutContainer) {
      DWORD mstat;
      OLEDBG_BEGIN2("IOleObject::GetMiscStatus called\r\n")
      lpContainerLine->m_lpOleObj->lpVtbl->GetMiscStatus(
            lpContainerLine->m_lpOleObj,
            DVASPECT_CONTENT,
            (DWORD FAR*)&mstat
      );
      OLEDBG_END2

      lpContainerLine->m_fInsideOutObj = (BOOL)
            (mstat & (OLEMISC_INSIDEOUT|OLEMISC_ACTIVATEWHENVISIBLE));
   }
#endif  // INPLACE_CNTR

   if (fDisplayAsIcon) {
      /* user has requested to display icon aspect instead of content
      **    aspect.
      **    NOTE: we do not have to delete the previous aspect cache
      **    because one did not get set up.
      */
      OleStdSwitchDisplayAspect(
            lpContainerLine->m_lpOleObj,
            &lpContainerLine->m_dwDrawAspect,
            dwDrawAspect,
            hMetaPict,
            FALSE,  /* fDeleteOldAspect */
            TRUE,   /* fSetupViewAdvise */
            (LPADVISESINK)&lpContainerLine->m_AdviseSink,
            NULL /*fMustUpdate*/        // this can be ignored; update
                                 // for switch to icon not req'd
      );
   }
   return TRUE;
}


/* Create an ContainerLine object and return the pointer */
LPCONTAINERLINE ContainerLine_Create(
      DWORD                  dwOleCreateType,
      HDC                    hDC,
      UINT                   nTab,
      LPCONTAINERDOC         lpContainerDoc,
      LPCLSID                lpclsid,
      LPOLESTR               lpszFileName,
      BOOL                   fDisplayAsIcon,
      HGLOBAL                 hMetaPict,
      LPOLESTR                lpszStgName
)
{
   LPCONTAINERLINE lpContainerLine = NULL;
   LPOLEOBJECT     lpObj = NULL;
   LPSTORAGE       lpDocStg = ContainerDoc_GetStg(lpContainerDoc);
   DWORD           dwDrawAspect =
                  (fDisplayAsIcon ? DVASPECT_ICON : DVASPECT_CONTENT);
   DWORD           dwOleRenderOpt =
                  (fDisplayAsIcon ? OLERENDER_NONE : OLERENDER_DRAW);
   HRESULT         hrErr;

   OLEDBG_BEGIN3("ContainerLine_Create\r\n")

   if (lpDocStg == NULL) {
      OleDbgAssertSz(lpDocStg != NULL, "Doc storage is NULL");
      goto error;
   }

   lpContainerLine=(LPCONTAINERLINE) New((DWORD)sizeof(CONTAINERLINE));
   if (lpContainerLine == NULL) {
      OleDbgAssertSz(lpContainerLine!=NULL,"Error allocating
ContainerLine");
      goto error;
   }

   ContainerLine_Init(lpContainerLine, nTab, hDC);

   /* OLE2NOTE: in order to avoid re-entrancy we will set a flag to
   **    guard our object. if this guard is set, then the object is
   **    not ready to have any OLE interface methods called. it is
   **    necessary to guard the object this way while it is being
   **    created or loaded.
   */
   lpContainerLine->m_fGuardObj = TRUE;

   /* OLE2NOTE: In order to have a stable ContainerLine object we must
   **    AddRef the object's refcnt. this will be later released when
   **    the ContainerLine is deleted.
   */
   ContainerLine_AddRef(lpContainerLine);

   OLESTRCPY(lpContainerLine->m_szStgName, lpszStgName);
   lpContainerLine->m_lpDoc = lpContainerDoc;

   /* Create a new storage for the object inside the doc's storage */
   lpContainerLine->m_lpStg =
OleStdCreateChildStorage(lpDocStg,lpszStgName);
   if (lpContainerLine->m_lpStg == NULL) {
      OleDbgAssert(lpContainerLine->m_lpStg != NULL);
      goto error;
   }

   lpContainerLine->m_dwLinkType = 0;
   switch (dwOleCreateType) {

      case IOF_SELECTCREATENEW:

         OLEDBG_BEGIN2("OleCreate called\r\n")
         hrErr = OleCreate (
               lpclsid,
               &IID_IOleObject,
               dwOleRenderOpt,
               NULL,
               (LPOLECLIENTSITE)&lpContainerLine->m_OleClientSite,
               lpContainerLine->m_lpStg,
               (LPVOID FAR*)&lpContainerLine->m_lpOleObj
         );
         OLEDBG_END2

#if defined( _DEBUG )
         if (hrErr != NOERROR)
            OleDbgOutHResult("OleCreate returned", hrErr);
#endif

         break;

      case IOF_SELECTCREATEFROMFILE:

         OLEDBG_BEGIN2("OleCreateFromFile called\r\n")
         hrErr = OleCreateFromFile (
               &CLSID_NULL,
               lpszFileName,
               &IID_IOleObject,
               dwOleRenderOpt,
               NULL,
               (LPOLECLIENTSITE)&lpContainerLine->m_OleClientSite,
               lpContainerLine->m_lpStg,
               (LPVOID FAR*)&lpContainerLine->m_lpOleObj
         );
         OLEDBG_END2

#if defined( _DEBUG )
         if (hrErr != NOERROR)
            OleDbgOutHResult("OleCreateFromFile returned", hrErr);
#endif
         break;

      case IOF_CHECKLINK:

         OLEDBG_BEGIN2("OleCreateLinkToFile called\r\n")
         hrErr = OleCreateLinkToFile (
               lpszFileName,
               &IID_IOleObject,
               dwOleRenderOpt,
               NULL,
               (LPOLECLIENTSITE)&lpContainerLine->m_OleClientSite,
               lpContainerLine->m_lpStg,
               (LPVOID FAR*)&lpContainerLine->m_lpOleObj
         );
         OLEDBG_END2

#if defined( _DEBUG )
         if (hrErr != NOERROR)
            OleDbgOutHResult("OleCreateLinkToFile returned", hrErr);
#endif
         break;
   }
   if (hrErr != NOERROR)
      goto error;

   if (! ContainerLine_SetupOleObject(
                        lpContainerLine, fDisplayAsIcon, hMetaPict)) {
      goto error;
   }

   /* OLE2NOTE: clear our re-entrancy guard. the object is now ready
   **    to have interface methods called.
   */
   lpContainerLine->m_fGuardObj = FALSE;

   OLEDBG_END3
   return lpContainerLine;

error:
   OutlineApp_ErrorMessage(g_lpApp, OLESTR("Could not create object!"));

   // Destroy partially created OLE object
   if (lpContainerLine)
      ContainerLine_Delete(lpContainerLine);
   OLEDBG_END3
   return NULL;
}


LPCONTAINERLINE ContainerLine_CreateFromData(
      HDC                     hDC,
      UINT                    nTab,
      LPCONTAINERDOC          lpContainerDoc,
      LPDATAOBJECT            lpSrcDataObj,
      DWORD                   dwCreateType,
      CLIPFORMAT              cfFormat,
      BOOL                    fDisplayAsIcon,
      HGLOBAL                 hMetaPict,
      LPOLESTR                lpszStgName
)
{
   HGLOBAL          hData = NULL;
   LPCONTAINERLINE lpContainerLine = NULL;
   LPOLEOBJECT      lpObj = NULL;
   LPSTORAGE        lpDocStg = ContainerDoc_GetStg(lpContainerDoc);
   DWORD            dwDrawAspect =
                  (fDisplayAsIcon ? DVASPECT_ICON : DVASPECT_CONTENT);
```

```
DWORD           dwOleRenderOpt;
FORMATETC       renderFmtEtc;
LPFORMATETC     lpRenderFmtEtc = NULL;
HRESULT         hrErr;
LPUNKNOWN       lpUnk = NULL;

OLEDBG_BEGIN3("ContainerLine_CreateFromData\r\n")

if (dwCreateType == OLECREATEFROMDATA_STATIC && cfFormat != 0) {
   // a particular type of static object should be created

   dwOleRenderOpt = OLERENDER_FORMAT;
   lpRenderFmtEtc = (LPFORMATETC)&renderFmtEtc;

   if (cfFormat == CF_METAFILEPICT)
      SETDEFAULTFORMATETC(renderFmtEtc, cfFormat, TYMED_MFPICT);
   else if (cfFormat == CF_BITMAP)
      SETDEFAULTFORMATETC(renderFmtEtc, cfFormat, TYMED_GDI);
   else
      SETDEFAULTFORMATETC(renderFmtEtc, cfFormat, TYMED_HGLOBAL);

} else if (dwCreateType == OLECREATEFROMDATA_STATIC && fDisplayAsIcon) {
   // a link that currently displayed as an icon needs to be
   // converted to a STATIC picture object. this case is driven
   // from "BreakLink" in the "Links" dialog. because the current
   // data in the source object's cache is DVASPECT_ICON we need
   // to tell the OleCreateStaticFromData API to look for
   // DVASPECT_ICON data. the static object that results however,
   // is considered to be displayed in the DVASPECT_CONTENT view.

   dwOleRenderOpt = OLERENDER_DRAW;
   lpRenderFmtEtc = (LPFORMATETC)&renderFmtEtc;
   SETFORMATETC(renderFmtEtc,0,DVASPECT_ICON,NULL,TYMED_NULL,-1);
   dwDrawAspect = DVASPECT_CONTENT;   // static obj displays only CONTENT

} else if (fDisplayAsIcon && hMetaPict) {
   // a special icon should be used. first we create the object
   // OLERENDER_NONE and then we stuff the special icon into the cache.

   dwOleRenderOpt = OLERENDER_NONE;

} else if (fDisplayAsIcon && hMetaPict == NULL) {
   // the object's default icon should be used

   dwOleRenderOpt = OLERENDER_DRAW;
   lpRenderFmtEtc = (LPFORMATETC)&renderFmtEtc;
   SETFORMATETC(renderFmtEtc,0,DVASPECT_ICON,NULL,TYMED_NULL,-1);

} else {
   // create standard DVASPECT_CONTENT/OLERENDER_DRAW object
   dwOleRenderOpt = OLERENDER_DRAW;
}

if (lpDocStg == NULL) {
   OleDbgAssertSz(lpDocStg != NULL, "Doc storage is NULL");
```

```
      goto error;
   }

   lpContainerLine=(LPCONTAINERLINE) New((DWORD)sizeof(CONTAINERLINE));
   if (lpContainerLine == NULL) {
      OleDbgAssertSz(lpContainerLine!=NULL,"Error allocating
ContainerLine");
      goto error;
   }

   ContainerLine_Init(lpContainerLine, nTab, hDC);

   /* OLE2NOTE: in order to avoid re-entrancy we will set a flag to
   **    guard our object. if this guard is set, then the object is
   **    not ready to have any OLE interface methods called. it is
   **    necessary to guard the object this way while it is being
   **    created or loaded.
   */
   lpContainerLine->m_fGuardObj = TRUE;

   /* OLE2NOTE: In order to have a stable ContainerLine object we must
   **    AddRef the object's refcnt. this will be later released when
   **    the ContainerLine is deleted.
   */
   ContainerLine_AddRef(lpContainerLine);

   OLESTRCPY(lpContainerLine->m_szStgName, lpszStgName);
   lpContainerLine->m_lpDoc = lpContainerDoc;

   /* Create a new storage for the object inside the doc's storage */
   lpContainerLine->m_lpStg =
OleStdCreateChildStorage(lpDocStg,lpszStgName);
   if (lpContainerLine->m_lpStg == NULL) {
      OleDbgAssert(lpContainerLine->m_lpStg != NULL);
      goto error;
   }

   switch (dwCreateType) {

      case OLECREATEFROMDATA_LINK:

         OLEDBG_BEGIN2("OleCreateLinkFromData called\r\n")
         hrErr = OleCreateLinkFromData (
               lpSrcDataObj,
               &IID_IOleObject,
               dwOleRenderOpt,
               lpRenderFmtEtc,
               (LPOLECLIENTSITE)&lpContainerLine->m_OleClientSite,
               lpContainerLine->m_lpStg,
               (LPVOID FAR*)&lpContainerLine->m_lpOleObj
         );
         OLEDBG_END2

#if defined( _DEBUG )
         if (hrErr != NOERROR)
```

```
            OleDbgOutHResult("OleCreateLinkFromData returned", hrErr);
#endif
         break;

      case OLECREATEFROMDATA_OBJECT:

         OLEDBG_BEGIN2("OleCreateFromData called\r\n")
         hrErr = OleCreateFromData (
               lpSrcDataObj,
               &IID_IOleObject,
               dwOleRenderOpt,
               lpRenderFmtEtc,
               (LPOLECLIENTSITE)&lpContainerLine->m_OleClientSite,
               lpContainerLine->m_lpStg,
               (LPVOID FAR*)&lpContainerLine->m_lpOleObj
         );
         OLEDBG_END2

#if defined( _DEBUG )
         if (hrErr != NOERROR)
            OleDbgOutHResult("OleCreateFromData returned", hrErr);
#endif
         break;

      case OLECREATEFROMDATA_STATIC:

         OLEDBG_BEGIN2("OleCreateStaticFromData called\r\n")
         hrErr = OleCreateStaticFromData (
               lpSrcDataObj,
               &IID_IOleObject,
               dwOleRenderOpt,
               lpRenderFmtEtc,
               (LPOLECLIENTSITE)&lpContainerLine->m_OleClientSite,
               lpContainerLine->m_lpStg,
               (LPVOID FAR*)&lpContainerLine->m_lpOleObj
         );
         OLEDBG_END2

#if defined( _DEBUG )
         if (hrErr != NOERROR)
            OleDbgOutHResult("OleCreateStaticFromData returned", hrErr);
#endif
         break;
   }

   if (hrErr != NOERROR)
      goto error;

   if (! ContainerLine_SetupOleObject(
                     lpContainerLine, fDisplayAsIcon, hMetaPict)) {
      goto error;
   }

   /* OLE2NOTE: clear our re-entrancy guard. the object is now ready
   **    to have interface methods called.
   */
   lpContainerLine->m_fGuardObj = FALSE;

   OLEDBG_END3
   return lpContainerLine;

error:
   OutlineApp_ErrorMessage(g_lpApp, OLESTR("Could not create object!"));
   // Destroy partially created OLE object
   if (lpContainerLine)
      ContainerLine_Delete(lpContainerLine);
   OLEDBG_END3
   return NULL;
}


/* ContainerLine_AddRef
** --------------------
**
**  increment the ref count of the line object.
**
**    Returns the new ref count on the object
*/
ULONG ContainerLine_AddRef(LPCONTAINERLINE lpContainerLine)
{
   ++lpContainerLine->m_cRef;

#if defined( _DEBUG )
   OleDbgOutRefCnt4(
         "ContainerLine_AddRef: cRef++\r\n",
         lpContainerLine,
         lpContainerLine->m_cRef
   );
#endif
   return lpContainerLine->m_cRef;
}


/* ContainerLine_Release
** ---------------------
**
**  decrement the ref count of the line object.
**    if the ref count goes to 0, then the line is destroyed.
**
**    Returns the remaining ref count on the object
*/
ULONG ContainerLine_Release(LPCONTAINERLINE lpContainerLine)
{
   ULONG cRef;

   /*********************************************************************
   ** OLE2NOTE: when the obj refcnt == 0, then destroy the object.     **
   **     otherwise the object is still in use.                        **
   *********************************************************************/
   cRef = --lpContainerLine->m_cRef;

#if defined( _DEBUG )
   OleDbgAssertSz(
         lpContainerLine->m_cRef >= 0,"Release called with cRef == 0");

   OleDbgOutRefCnt4(
         "ContainerLine_Release: cRef--\r\n",
         lpContainerLine,
         cRef
   );
#endif
   if (cRef == 0)
      ContainerLine_Destroy(lpContainerLine);

   return cRef;
}


/* ContainerLine_QueryInterface
** ----------------------------
**
** Retrieve a pointer to an interface on the ContainerLine object.
**
**    Returns NOERROR if interface is successfully retrieved.
**            E_NOINTERFACE if the interface is not supported
*/
HRESULT ContainerLine_QueryInterface(
      LPCONTAINERLINE        lpContainerLine,
      REFIID                 riid,
      LPVOID FAR*            lplpvObj
)
{
   SCODE sc = E_NOINTERFACE;

   /* OLE2NOTE: we must make sure to set all out ptr parameters to NULL. */
   *lplpvObj = NULL;

   if (IsEqualIID(riid, &IID_IUnknown)) {
      OleDbgOut4("ContainerLine_QueryInterface: IUnknown* RETURNED\r\n");

      *lplpvObj = (LPVOID) &lpContainerLine->m_Unknown;
      ContainerLine_AddRef(lpContainerLine);
      sc = S_OK;
   }
   else if (IsEqualIID(riid, &IID_IOleClientSite)) {
      OleDbgOut4("ContainerLine_QueryInterface: IOleClientSite*
RETURNED\r\n");

      *lplpvObj = (LPVOID) &lpContainerLine->m_OleClientSite;
      ContainerLine_AddRef(lpContainerLine);
      sc = S_OK;
   }
   else if (IsEqualIID(riid, &IID_IAdviseSink)) {
      OleDbgOut4("ContainerLine_QueryInterface: IAdviseSink* RETURNED\r\n");
      *lplpvObj = (LPVOID) &lpContainerLine->m_AdviseSink;
      ContainerLine_AddRef(lpContainerLine);
      sc = S_OK;
   }
#if defined( INPLACE_CNTR )
   else if (IsEqualIID(riid, &IID_IOleWindow)
          || IsEqualIID(riid, &IID_IOleInPlaceSite)) {
      OleDbgOut4("ContainerLine_QueryInterface: IOleInPlaceSite*
RETURNED\r\n");

      *lplpvObj = (LPVOID) &lpContainerLine->m_OleInPlaceSite;
      ContainerLine_AddRef(lpContainerLine);
      sc = S_OK;
   }
#endif  // INPLACE_CNTR

   OleDbgQueryInterfaceMethod(*lplpvObj);

   return ResultFromScode(sc);
}


BOOL ContainerLine_LoadOleObject(LPCONTAINERLINE lpContainerLine)
{
   LPOUTLINEDOC    lpOutlineDoc = (LPOUTLINEDOC)lpContainerLine->m_lpDoc;
   LPSTORAGE       lpDocStg = ContainerDoc_GetStg(lpContainerLine->m_lpDoc);
   LPOLECLIENTSITE lpOleClientSite;
   LPMONIKER       lpmkObj;
   LPOLEAPP        lpOleApp = (LPOLEAPP)g_lpApp;
   BOOL            fPrevEnable1;
   BOOL            fPrevEnable2;
   HRESULT         hrErr;

   if (lpContainerLine->m_fGuardObj)
      return FALSE;                // object in process of creation

   if (lpContainerLine->m_lpOleObj)
      return TRUE;                // object already loaded

   OLEDBG_BEGIN3("ContainerLine_LoadOleObject\r\n")

   /* OLE2NOTE: in order to avoid re-entrancy we will set a flag to
   **    guard our object. if this guard is set, then the object is
   **    not ready to have any OLE interface methods called. it is
   **    necessary to guard the object this way while it is being
   **    created or loaded.
   */
   lpContainerLine->m_fGuardObj = TRUE;

   /* if object storage is not already open, then open it */
   if (! lpContainerLine->m_lpStg) {
      lpContainerLine->m_lpStg = OleStdOpenChildStorage(
            lpDocStg,
            lpContainerLine->m_szStgName,
            STGM_READWRITE
      );
      if (lpContainerLine->m_lpStg == NULL) {
         OleDbgAssert(lpContainerLine->m_lpStg != NULL);
         goto error;
      }
   }

   /* OLE2NOTE: if the OLE object being loaded is in a data transfer
   **    document, then we should NOT pass a IOleClientSite* pointer
   **    to the OleLoad call. This particularly critical if the OLE
   **    object is an OleLink object. If a non-NULL client site is
   **    passed to the OleLoad function, then the link will bind to
   **    the source if its is running. in the situation that we are
   **    loading the object as part of a data transfer document we do
   **    not want this connection to be established. even worse, if
   **    the link source is currently blocked or busy, then this could
   **    hang the system. it is simplest to never pass a
   **    IOleClientSite* when loading an object in a data transfer
   **    document.
   */
   lpOleClientSite = (lpOutlineDoc->m_fDataTransferDoc ?
         NULL : (LPOLECLIENTSITE)&lpContainerLine->m_OleClientSite);

   /* OLE2NOTE: we do not want to ever give the Busy/NotResponding
   **    dialogs when we are loading an object. if the object is a
   **    link, it will attempt to BindIfRunning to the link source. if
   **    the link source is currently busy, this could cause the Busy
   **    dialog to come up. even if the link source is busy,
   **    we do not want put up the busy dialog. thus we will disable
   **    the dialog and later re-enable them
   */
   OleApp_DisableBusyDialogs(lpOleApp, &fPrevEnable1, &fPrevEnable2);

   OLEDBG_BEGIN2("OleLoad called\r\n")
   hrErr = OleLoad (
         lpContainerLine->m_lpStg,
         &IID_IOleObject,
         lpOleClientSite,
         (LPVOID FAR*)&lpContainerLine->m_lpOleObj
   );
   OLEDBG_END2

   // re-enable the Busy/NotResponding dialogs
   OleApp_EnableBusyDialogs(lpOleApp, fPrevEnable1, fPrevEnable2);

   if (hrErr != NOERROR) {
      OleDbgAssertSz(hrErr == NOERROR, "Could not load object!");
      OleDbgOutHResult("OleLoad returned", hrErr);
      goto error;
   }

   /* Cache a pointer to the IViewObject2* interface. *Required*
   **      we need this everytime we draw the object.
   **
```

```
   ** OLE2NOTE: We require the object to support IViewObject2
   **    interface. this is an extension to the IViewObject interface
   **    that was added with the OLE 2.01 release. This interface must
   **    be supported by all object handlers and DLL-based objects.
   */
   lpContainerLine->m_lpViewObj2 = (LPVIEWOBJECT2)OleStdQueryInterface(
         (LPUNKNOWN)lpContainerLine->m_lpOleObj, &IID_IViewObject2);
   if (! lpContainerLine->m_lpViewObj2) {
#if defined( _DEBUG )
      OleDbgAssertSz(
         lpContainerLine->m_lpViewObj2,"IViewObject2 NOT supported\r\n");
#endif
      goto error;
   }

   // Cache a pointer to the IPersistStorage* interface. *Required*
   //      we need this everytime we save the object.
   lpContainerLine->m_lpPersistStg = (LPPERSISTSTORAGE)OleStdQueryInterface(
         (LPUNKNOWN)lpContainerLine->m_lpOleObj, &IID_IPersistStorage);
   if (! lpContainerLine->m_lpPersistStg) {
      OleDbgAssert(lpContainerLine->m_lpPersistStg);
      goto error;
   }

   // Cache a pointer to the IOleLink* interface if supported. *Optional*
   //      if supported the object is a link. we need this to manage the
link
   if (lpContainerLine->m_dwLinkType != 0) {
      lpContainerLine->m_lpOleLink = (LPOLELINK)OleStdQueryInterface(
            (LPUNKNOWN)lpContainerLine->m_lpOleObj, &IID_IOleLink);
      if (! lpContainerLine->m_lpOleLink) {
         OleDbgAssert(lpContainerLine->m_lpOleLink);
         goto error;
      }
   }

   /* OLE2NOTE: clear our re-entrancy guard. the object is now ready
   **    to have interface methods called.
   */
   lpContainerLine->m_fGuardObj = FALSE;

   /* OLE2NOTE: similarly, if the OLE object being loaded is in a data
   **    transfer document, then we do NOT need to setup any advises,
   **    call SetHostNames, SetMoniker, etc.
   */
   if (lpOleClientSite) {
      /* Setup the Advises (OLE notifications) that we are interested
      ** in receiving.
      */
      OleStdSetupAdvises(
            lpContainerLine->m_lpOleObj,
            lpContainerLine->m_dwDrawAspect,
            (LPOLESTR)APPNAME,
            lpOutlineDoc->m_lpszDocTitle,
            (LPADVISESINK)&lpContainerLine->m_AdviseSink,
            FALSE   /*fCreate*/
      );

      /* OLE2NOTE: if the OLE object has a moniker assigned, we need to
      **    inform the object by calling IOleObject::SetMoniker. this
      **    will force the OLE object to register in the
      **    RunningObjectTable when it enters the running state.
      */
      if (lpContainerLine->m_fMonikerAssigned) {
         lpmkObj = ContainerLine_GetRelMoniker(
               lpContainerLine,
               GETMONIKER_ONLYIFTHERE
         );

         if (lpmkObj) {
            OLEDBG_BEGIN2("IOleObject::SetMoniker called\r\n")
            lpContainerLine->m_lpOleObj->lpVtbl->SetMoniker(
                  lpContainerLine->m_lpOleObj,
                  OLEWHICHMK_OBJREL,
                  lpmkObj
            );
            OLEDBG_END2
            OleStdRelease((LPUNKNOWN)lpmkObj);
         }
      }

      /* get the Short form of the user type name of the object. this
      **    is used all the time when we have to build the object
      **    verb menu. we will cache this information to make it
      **    quicker to build the verb menu.
      */
      OLEDBG_BEGIN2("IOleObject::GetUserType called\r\n")
      lpContainerLine->m_lpOleObj->lpVtbl->GetUserType(
            lpContainerLine->m_lpOleObj,
            USERCLASSTYPE_SHORT,
            (LPOLESTR FAR*)&lpContainerLine->m_lpszShortType
      );
      OLEDBG_END2

#if defined( INPLACE_CNTR )
      /* OLE2NOTE: an inside-out container should check if the object
      **    is an inside-out and prefers to be activated when visible
      **    type of object. if so, the object should be immediately
      **    activated in-place, BUT NOT UIActived.
      */
      if (g_fInsideOutContainer &&
            lpContainerLine->m_dwDrawAspect == DVASPECT_CONTENT) {
         DWORD mstat;
         OLEDBG_BEGIN2("IOleObject::GetMiscStatus called\r\n")
         lpContainerLine->m_lpOleObj->lpVtbl->GetMiscStatus(
               lpContainerLine->m_lpOleObj,
               DVASPECT_CONTENT,
               (DWORD FAR*)&mstat
         );
         OLEDBG_END2
         lpContainerLine->m_fInsideOutObj = (BOOL)
               (mstat & (OLEMISC_INSIDEOUT|OLEMISC_ACTIVATEWHENVISIBLE));

         if ( lpContainerLine->m_fInsideOutObj ) {
            HWND hWndDoc = OutlineDoc_GetWindow(lpOutlineDoc);

            ContainerLine_DoVerb(
                  lpContainerLine,
                  OLEIVERB_INPLACEACTIVATE,
                  NULL,
                  FALSE,
                  FALSE
            );

            /* OLE2NOTE: following this DoVerb(INPLACEACTIVATE) the
            **    object may have taken focus. but because the
            **    object is NOT UIActive it should NOT have focus.
            **    we will make sure our document has focus.
            */
            SetFocus(hWndDoc);
         }
      }
#endif  // INPLACE_CNTR
      OLEDBG_END2

   }

   OLEDBG_END2
   return TRUE;

error:
   OLEDBG_END2
   return FALSE;
}


/* ContainerLine_CloseOleObject
** ----------------------------
**    Close the OLE object associated with the ContainerLine.
**
**    Closing the object forces the object to transition from the
**    running state to the loaded state. if the object was not running,
**    then there is no effect. it is necessary to close the OLE object
**    before releasing the pointers to the OLE object.
**
**    Returns TRUE if successfully closed,
**            FALSE if closing was aborted.
*/
BOOL ContainerLine_CloseOleObject(
      LPCONTAINERLINE         lpContainerLine,
      DWORD                   dwSaveOption
)
{
   HRESULT hrErr;
   SCODE   sc;

   if (lpContainerLine->m_fGuardObj)
      return FALSE;                // object in process of creation

   if (! lpContainerLine->m_lpOleObj)
      return TRUE;    // object is NOT loaded

   OLEDBG_BEGIN2("IOleObject::Close called\r\n")
   hrErr = lpContainerLine->m_lpOleObj->lpVtbl->Close(
         lpContainerLine->m_lpOleObj,
         (dwSaveOption == OLECLOSE_NOSAVE ?
               OLECLOSE_NOSAVE : OLECLOSE_SAVEIFDIRTY)
   );
   OLEDBG_END2

#if defined( INPLACE_CNTR )
   if (lpContainerLine->m_fIpServerRunning) {
      /* OLE2NOTE: unlock the lock held on the in-place object.
      **    it is VERY important that an in-place container
      **    that also support linking to embeddings properly manage
      **    the running of its in-place objects. in an outside-in
      **    style in-place container, when the user clicks
      **    outside of the in-place active object, the object gets
      **    UIDeactivated and the object hides its window. in order
      **    to make the object fast to reactivate, the container
      **    deliberately does not call IOleObject::Close. the object
      **    stays running in the invisible unlocked state. the idea
      **    here is if the user simply clicks outside of the object
      **    and then wants to double click again to re-activate the
      **    object, we do not want this to be slow. if we want to
      **    keep the object running, however, we MUST Lock it
      **    running. otherwise the object will be in an unstable
      **    state where if a linking client does a "silent-update"
      **    (eg. UpdateNow from the Links dialog), then the in-place
      **    server will shut down even before the object has a chance
      **    to be saved back in its container. this saving normally
      **    occurs when the in-place container closes the object. also
      **    keeping the object in the unstable, hidden, running,
      **    not-locked state can cause problems in some scenarios.
      **    ICntrOtl keeps only one object running. if the user
      **    intiates a DoVerb on another object, then that last
      **    running in-place active object is closed. a more
      **    sophistocated in-place container may keep more object running.
      **    (see CntrLine_IPSite_OnInPlaceActivate)
      */
      lpContainerLine->m_fIpServerRunning = FALSE;

      OLEDBG_BEGIN2("OleLockRunning(FALSE,TRUE) called\r\n")
      OleLockRunning((LPUNKNOWN)lpContainerLine->m_lpOleObj, FALSE, TRUE);
      OLEDBG_END2
   }
#endif

   if (hrErr != NOERROR) {
      OleDbgOutHResult("IOleObject::Close returned", hrErr);
      sc = GetScode(hrErr);
      if (sc == RPC_E_CALL_REJECTED || sc==OLE_E_PROMPTSAVECANCELLED)
         return FALSE;   // object aborted shutdown
   }
   return TRUE;
}


/* ContainerLine_UnloadOleObject
** -----------------------------
**    Close the OLE object associated with the ContainerLine and
**    release all pointer held to the object.
**
**    Closing the object forces the object to transition from the
**    running state to the loaded state. if the object was not running,
**    then there is no effect. it is necessary to close the OLE object
**    before releasing the pointers to the OLE object. releasing all
**    pointers to the object allows the object to transition from
**    loaded to unloaded (or passive).
*/
void ContainerLine_UnloadOleObject(
      LPCONTAINERLINE         lpContainerLine,
      DWORD                   dwSaveOption
)
{
   if (lpContainerLine->m_lpOleObj) {

      OLEDBG_BEGIN2("IOleObject::Close called\r\n")
      lpContainerLine->m_lpOleObj->lpVtbl->Close(
            lpContainerLine->m_lpOleObj, dwSaveOption);
      OLEDBG_END2

      /* OLE2NOTE: we will take our IOleClientSite* pointer away from
      **    the object before we release all pointers to the object.
      **    in the scenario where the object is implemented as an
      **    in-proc server (DLL object), then, if there are link
      **    connections to the DLL object, it is possible that the
      **    object will not be destroyed when we release our pointers
      **    to the object. the existance of the remote link
      **    connections will hold the object alive. later when these
      **    strong connections are released, then the object may
      **    attempt to call IOleClientSite::Save if we had not taken
      **    away the client site pointer.
      */
      OLEDBG_BEGIN2("IOleObject::SetClientSite(NULL) called\r\n")
      lpContainerLine->m_lpOleObj->lpVtbl->SetClientSite(
            lpContainerLine->m_lpOleObj, NULL);
      OLEDBG_END2

      OleStdRelease((LPUNKNOWN)lpContainerLine->m_lpOleObj);
      lpContainerLine->m_lpOleObj = NULL;

      if (lpContainerLine->m_lpViewObj2) {
         OleStdRelease((LPUNKNOWN)lpContainerLine->m_lpViewObj2);
```

```
            lpContainerLine->m_lpViewObj2 = NULL;
         }
         if (lpContainerLine->m_lpPersistStg) {
            OleStdRelease((LPUNKNOWN)lpContainerLine->m_lpPersistStg);
            lpContainerLine->m_lpPersistStg = NULL;
         }

         if (lpContainerLine->m_lpOleLink) {
            OleStdRelease((LPUNKNOWN)lpContainerLine->m_lpOleLink);
            lpContainerLine->m_lpOleLink = NULL;
         }
      }

      if (lpContainerLine->m_lpszShortType) {
         OleStdFreeString(lpContainerLine->m_lpszShortType, NULL);
         lpContainerLine->m_lpszShortType = NULL;
      }
   }


/* ContainerLine_Delete
** --------------------
**    Delete the ContainerLine.
**
**    NOTE: we can NOT directly destroy the memory for the
**    ContainerLine; the ContainerLine maintains a reference count. a
**    non-zero reference count indicates that the object is still
**    in-use. The OleObject keeps a reference-counted pointer to the
**    ClientLine object. we must take the actions necessary so that the
**    ContainerLine object receives Releases for outstanding
**    references. when the reference count of the ContainerLine reaches
**    zero, then the memory for the object will actually be destroyed
**    (ContainerLine_Destroy called).
**
*/
void ContainerLine_Delete(LPCONTAINERLINE lpContainerLine)
{
   OLEDBG_BEGIN2("ContainerLine_Delete\r\n")

#if defined( INPLACE_CNTR )
   if (lpContainerLine == lpContainerLine->m_lpDoc->m_lpLastIpActiveLine)
      lpContainerLine->m_lpDoc->m_lpLastIpActiveLine = NULL;
   if (lpContainerLine == lpContainerLine->m_lpDoc->m_lpLastUIActiveLine)
      lpContainerLine->m_lpDoc->m_lpLastUIActiveLine = NULL;
#endif

   /* OLE2NOTE: in order to have a stable line object during the
   **    process of deleting, we intially AddRef the line ref cnt and
   **    later Release it. This initial AddRef is artificial; it is
   **    simply done to guarantee that our object does not destroy
   **    itself until the END of this routine.
   */
   ContainerLine_AddRef(lpContainerLine);

   // Unload the loaded OLE object
   if (lpContainerLine->m_lpOleObj)
      ContainerLine_UnloadOleObject(lpContainerLine, OLECLOSE_NOSAVE);

   /* OLE2NOTE: we can NOT directly free the memory for the ContainerLine
   **    data structure until everyone holding on to a pointer to our
   **    ClientSite interface and IAdviseSink interface has released
   **    their pointers. There is one refcnt on the ContainerLine object
   **    which is held by the container itself. we will release this
   **    refcnt here.
   */
   ContainerLine_Release(lpContainerLine);

   /* OLE2NOTE: this call forces all external connections to our
   **    ContainerLine to close down and therefore guarantees that
   **    we receive all releases associated with those external
   **    connections. Strictly this call should NOT be necessary, but
   **    it is defensive coding to make this call.
   */
   OLEDBG_BEGIN2("CoDisconnectObject(lpContainerLine) called\r\n")
   CoDisconnectObject((LPUNKNOWN)&lpContainerLine->m_Unknown, 0);
   OLEDBG_END2

#if defined( _DEBUG )
   /* at this point the object all references from the OLE object to
   **    our ContainerLine object should have been released. there
   **    should only be 1 remaining reference that will be released below.
   */
   if (lpContainerLine->m_cRef != 1) {
      OleDbgOutRefCnt(
         "WARNING: ContainerLine_Delete: cRef != 1\r\n",
         lpContainerLine,
         lpContainerLine->m_cRef
      );
   }
#endif

   ContainerLine_Release(lpContainerLine); // release artificial AddRef
above
   OLEDBG_END2
}


/* ContainerLine_Destroy
** ---------------------
**    Destroy (Free) the memory used by a ContainerLine structure.
**    This function is called when the ref count of the ContainerLine goes
**    to zero. the ref cnt goes to zero after ContainerLine_Delete forces
**    the OleObject to unload and release its pointers to the
**    ContainerLine IOleClientSite and IAdviseSink interfaces.
*/

void ContainerLine_Destroy(LPCONTAINERLINE lpContainerLine)
{
   LPUNKNOWN lpTmpObj;
   OLEDBG_BEGIN2("ContainerLine_Destroy\r\n")

   // Release the storage opened for the OLE object
   if (lpContainerLine->m_lpStg) {
      lpTmpObj = (LPUNKNOWN)lpContainerLine->m_lpStg;
      lpContainerLine->m_lpStg = NULL;

      OleStdRelease(lpTmpObj);
   }

   if (lpContainerLine->m_lpszShortType) {
      OleStdFreeString(lpContainerLine->m_lpszShortType, NULL);
      lpContainerLine->m_lpszShortType = NULL;
   }

   Delete(lpContainerLine);        // Free the memory for the structure
itself
   OLEDBG_END2
}


/* ContainerLine_CopyToDoc
 * -----------------------
 *
 *      Copy a ContainerLine to another Document (usually ClipboardDoc)
 */
BOOL ContainerLine_CopyToDoc(
      LPCONTAINERLINE         lpSrcLine,
      LPOUTLINEDOC            lpDestDoc,
      int                     nIndex
)
{
   LPCONTAINERLINE lpDestLine = NULL;
   LPLINELIST  lpDestLL = &lpDestDoc->m_LineList;
   HDC         hDC;
   HRESULT     hrErr;
   BOOL        fStatus;
   LPSTORAGE   lpDestDocStg = ((LPOLEDOC)lpDestDoc)->m_lpStg;
   LPSTORAGE   lpDestObjStg = NULL;

   lpDestLine = (LPCONTAINERLINE) New((DWORD)sizeof(CONTAINERLINE));
   if (lpDestLine == NULL) {
      OleDbgAssertSz(lpDestLine!=NULL, "Error allocating ContainerLine");
      return FALSE;
   }

   hDC = LineList_GetDC(lpDestLL);
   ContainerLine_Init(lpDestLine, ((LPLINE)lpSrcLine)->m_nTabLevel, hDC);
   LineList_ReleaseDC(lpDestLL, hDC);

   /* OLE2NOTE: In order to have a stable ContainerLine object we must
   **    AddRef the object's refcnt. this will be later released when
   **    the ContainerLine is deleted.
   */
   ContainerLine_AddRef(lpDestLine);
```

```
lpDestLine->m_lpDoc = (LPCONTAINERDOC)lpDestDoc;

// Copy data of the original source ContainerLine.
((LPLINE)lpDestLine)->m_nWidthInHimetric =
      ((LPLINE)lpSrcLine)->m_nWidthInHimetric;
((LPLINE)lpDestLine)->m_nHeightInHimetric =
      ((LPLINE)lpSrcLine)->m_nHeightInHimetric;
lpDestLine->m_fMonikerAssigned = lpSrcLine->m_fMonikerAssigned;
lpDestLine->m_dwDrawAspect = lpSrcLine->m_dwDrawAspect;
lpDestLine->m_sizeInHimetric = lpSrcLine->m_sizeInHimetric;
lpDestLine->m_dwLinkType = lpSrcLine->m_dwLinkType;


/* We must create a new sub-storage for the embedded object within
**    the destination document's storage. We will first attempt to
**    use the same storage name as the source line. if this name is
**    in use, then we will allocate a new name. in this way we try
**    to keep the name associated with the OLE object unchanged
**    through a Cut/Paste operation.
*/
lpDestObjStg = OleStdCreateChildStorage(
      lpDestDocStg,
      lpSrcLine->m_szStgName
);
if (lpDestObjStg) {
   OLESTRCPY(lpDestLine->m_szStgName, lpSrcLine->m_szStgName);
} else {
   /* the original name was in use, make up a new name. */
   ContainerDoc_GetNextStgName(
         (LPCONTAINERDOC)lpDestDoc,
         lpDestLine->m_szStgName,
         OLESTRLEN(lpDestLine->m_szStgName)
   );
   lpDestObjStg = OleStdCreateChildStorage(
         lpDestDocStg,
         lpDestLine->m_szStgName
   );
}
if (lpDestObjStg == NULL) {
   OleDbgAssertSz(lpDestObjStg != NULL, "Error creating child stg");
   goto error;
}

// Copy over storage of the embedded object itself

if (! lpSrcLine->m_lpOleObj) {

   /*****************************************************************
   ** CASE 1: object is NOT loaded.
   **    because the object is not loaded, we can simply copy the
   **    object's current storage to the new storage.
   *****************************************************************/

   /* if current object storage is not already open, then open it */
```

```
      if (! lpSrcLine->m_lpStg) {
         LPSTORAGE lpSrcDocStg = ((LPOLEDOC)lpSrcLine->m_lpDoc)->m_lpStg;

         if (! lpSrcDocStg) goto error;

         // open object storage.
         lpSrcLine->m_lpStg = OleStdOpenChildStorage(
               lpSrcDocStg,
               lpSrcLine->m_szStgName,
               STGM_READWRITE
         );
         if (lpSrcLine->m_lpStg == NULL) {
#if defined( _DEBUG )
            OleDbgAssertSz(
                  lpSrcLine->m_lpStg != NULL,
                  "Error opening child stg"
            );
#endif
            goto error;
         }
      }

      hrErr = lpSrcLine->m_lpStg->lpVtbl->CopyTo(
            lpSrcLine->m_lpStg,
            0,
            NULL,
            NULL,
            lpDestObjStg
      );
      if (hrErr != NOERROR) {
         OleDbgOutHResult("WARNING: lpSrcObjStg->CopyTo returned", hrErr);
         goto error;
      }

      fStatus = OleStdCommitStorage(lpDestObjStg);

   } else {

      /*****************************************************************
      ** CASE 2: object IS loaded.
      **    we must tell the object to save into the new storage.
      *****************************************************************/

      SCODE sc = S_OK;
      LPPERSISTSTORAGE lpPersistStg = lpSrcLine->m_lpPersistStg;
      OleDbgAssert(lpPersistStg);

      OLEDBG_BEGIN2("OleSave called\r\n")
      hrErr = OleSave(lpPersistStg, lpDestObjStg, FALSE /*fSameAsLoad*/);
      OLEDBG_END2

      if (hrErr != NOERROR) {
         OleDbgOutHResult("WARNING: OleSave returned", hrErr);
         sc = GetScode(hrErr);
      }
      // OLE2NOTE: even if OleSave fails, SaveCompleted must be called.
      OLEDBG_BEGIN2("IPersistStorage::SaveCompleted called\r\n")
      hrErr=lpPersistStg->lpVtbl->SaveCompleted(lpPersistStg,NULL);
      OLEDBG_END2

      if (hrErr != NOERROR) {
         OleDbgOutHResult("WARNING: SaveCompleted returned",hrErr);
         if (sc == S_OK)
            sc = GetScode(hrErr);
      }

      if (sc != S_OK)
         goto error;

   }

   OutlineDoc_AddLine(lpDestDoc, (LPLINE)lpDestLine, nIndex);
   OleStdVerifyRelease(
         (LPUNKNOWN)lpDestObjStg,
         OLESTR("Copied object stg not released")
   );

   return TRUE;

error:

   // Delete any partially created storage.
   if (lpDestObjStg) {
      OleStdVerifyRelease(
            (LPUNKNOWN)lpDestObjStg,
            OLESTR("Copied object stg not released")
      );

      lpDestDocStg->lpVtbl->DestroyElement(
            lpDestDocStg,
            lpDestLine->m_szStgName
      );
      lpDestLine->m_szStgName[0] = '\0';
   }

   // destroy partially created ContainerLine
   if (lpDestLine)
      ContainerLine_Delete(lpDestLine);
   return FALSE;
}


/* ContainerLine_UpdateExtent
** --------------------------
**   Update the size of the ContainerLine because the extents of the
**    object may have changed.
**
**    NOTE: because we are using a Windows OwnerDraw ListBox, we must
**    constrain the maximum possible height of a line. the ListBox has
**    a limitation (unfortunately) that no line can become larger than
**    255 pixels. thus we force the object to scale maintaining its
**    aspect ratio if this maximum line height limit is reached. the
**    actual maximum size for an object at 100% Zoom is
**    255
**
**    RETURNS TRUE -- if the extents of the object changed
**            FALSE -- if the extents did NOT change
*/
BOOL ContainerLine_UpdateExtent(
      LPCONTAINERLINE     lpContainerLine,
      LPSIZEL             lpsizelHim
)
{
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpContainerLine->m_lpDoc;
   LPLINELIST lpLL = OutlineDoc_GetLineList(lpOutlineDoc);
   LPLINE lpLine = (LPLINE)lpContainerLine;
   int nIndex = LineList_GetLineIndex(lpLL, lpLine);
   UINT nOrgWidthInHimetric = lpLine->m_nWidthInHimetric;
   UINT nOrgHeightInHimetric = lpLine->m_nHeightInHimetric;
   BOOL fWidthChanged = FALSE;
   BOOL fHeightChanged = FALSE;
   SIZEL sizelHim;
   HRESULT hrErr;

   if (!lpContainerLine || !lpContainerLine->m_lpOleObj)
      return FALSE;

   if (lpContainerLine->m_fGuardObj)
      return FALSE;                // object in process of creation

   OLEDBG_BEGIN3("ContainerLine_UpdateExtent\r\n");

   lpContainerLine->m_fDoGetExtent = FALSE;

   if (! lpsizelHim) {
      /* OLE2NOTE: We want to call IViewObject2::GetExtent instead of
      **    IOleObject::GetExtent. IViewObject2::GetExtent method was
      **    added in OLE 2.01 release. It always retrieves the
      **    extents of the object corresponding to that which will be
      **    drawn by calling IViewObject::Draw. Normally, this is
      **    determined by the data stored in the data cache. This
      **    call will never result in a remoted (LRPC) call.
      */
      OLEDBG_BEGIN2("IViewObject2::GetExtent called\r\n")
      hrErr = lpContainerLine->m_lpViewObj2->lpVtbl->GetExtent(
            lpContainerLine->m_lpViewObj2,
            lpContainerLine->m_dwDrawAspect,
            -1,     /*lindex*/
            NULL,   /*ptd*/
            (LPSIZEL)&sizelHim
      );
      OLEDBG_END2
      if (hrErr != NOERROR)
         sizelHim.cx = sizelHim.cy = 0;
      lpsizelHim = (LPSIZEL)&sizelHim;
   }

   if (lpsizelHim->cx == lpContainerLine->m_sizeInHimetric.cx &&
      lpsizelHim->cy == lpContainerLine->m_sizeInHimetric.cy) {
      goto noupdate;
   }

   if (lpsizelHim->cx > 0 || lpsizelHim->cy > 0) {
      lpContainerLine->m_sizeInHimetric = *lpsizelHim;
   } else {
      /* object does not have any extents; let's use our container
      **    chosen arbitrary size for OLE objects.
      */
      lpContainerLine->m_sizeInHimetric.cx = (long)DEFOBJWIDTH;
      lpContainerLine->m_sizeInHimetric.cy = (long)DEFOBJHEIGHT;
   }

   ContainerLine_SetLineHeightFromObjectExtent(
         lpContainerLine,
         (LPSIZEL)&lpContainerLine->m_sizeInHimetric);

   // if height of object changed, then reset the height of line in LineList
   if (nOrgHeightInHimetric != lpLine->m_nHeightInHimetric) {
      LineList_SetLineHeight(lpLL, nIndex, lpLine->m_nHeightInHimetric);
      fHeightChanged = TRUE;
   }

   fWidthChanged = LineList_RecalcMaxLineWidthInHimetric(
         lpLL,
         nOrgWidthInHimetric
   );
   fWidthChanged |= (nOrgWidthInHimetric != lpLine->m_nWidthInHimetric);

   if (fHeightChanged || fWidthChanged) {
      OutlineDoc_ForceRedraw(lpOutlineDoc, TRUE);

      // mark ContainerDoc as now dirty
      OutlineDoc_SetModified(lpOutlineDoc, TRUE, TRUE, TRUE);
   }

   OLEDBG_END3
   return TRUE;

noupdate:
   OLEDBG_END3
   return FALSE;   // No UPDATE necessary
}


/* ContainerLine_DoVerb
** --------------------
**    Activate the OLE object and perform a specific verb.
*/
```

```
BOOL ContainerLine_DoVerb(
      LPCONTAINERLINE lpContainerLine,
      LONG            iVerb,
      LPMSG           lpMsg,
      BOOL            fMessage,
      BOOL            fAction
)
{
   HRESULT hrErr;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpContainerLine->m_lpDoc;
   RECT rcPosRect;
   OLEDBG_BEGIN3("ContainerLine_DoVerb\r\n")

   if (lpContainerLine->m_fGuardObj) {
      // object in process of creation--Fail the DoVerb call
      hrErr = ResultFromScode(E_FAIL);
      goto error;
   }

   /* if object is not already loaded, then load it now. objects are
   **    loaded lazily in this manner.
   */
   if (! lpContainerLine->m_lpOleObj)
      ContainerLine_LoadOleObject(lpContainerLine);

   if (! lpContainerLine->m_lpOleObj) {
#if defined( _DEBUG )
      OleDbgAssertSz(
            lpContainerLine->m_lpOleObj != NULL,
            "OLE object not loaded"
      );
#endif
      goto error;
   }

ExecuteDoVerb:

   ContainerLine_GetPosRect(lpContainerLine, (LPRECT)&rcPosRect);

   // run the object
   hrErr = ContainerLine_RunOleObject(lpContainerLine);
   if (hrErr != NOERROR)
      goto error;

   /* Tell object server to perform a "verb". */
   OLEDBG_BEGIN2("IOleObject::DoVerb called\r\n")
   hrErr = lpContainerLine->m_lpOleObj->lpVtbl->DoVerb (
         lpContainerLine->m_lpOleObj,
         iVerb,
         lpMsg,
         (LPOLECLIENTSITE)&lpContainerLine->m_OleClientSite,
         -1,
         OutlineDoc_GetWindow(lpOutlineDoc),
         (LPCRECT)&rcPosRect
   );
   OLEDBG_END2

   /* OLE2NOTE: IOleObject::DoVerb may return a success code
   **    OLE_S_INVALIDVERB. this SCODE should NOT be considered an
   **    error; thus it is important to use the "FAILED" macro to
   **    check for an error SCODE.
   */
   if (FAILED(GetScode(hrErr))) {
      OleDbgOutHResult("WARNING: lpOleObj->DoVerb returned", hrErr);
      goto error;
   }

#if defined( INPLACE_CNTR )
   /* OLE2NOTE: we want to keep only 1 inplace server active at any
   **    given time. so when we start to do a DoVerb on another line,
   **    then we want to shut down the previously activated server. in
   **    this way we keep at most one inplace server active at a time.
   **    because it is possible that the object we want to do DoVerb
   **    on is handled by the same EXE as that of the previously
   **    activated server, then we do not want the EXE to be shut down
   **    only to be launched again. in order to avoid this we will do
   **    the DoVerb BEFORE trying to shutdown the previous object.
   */
   if (!g_fInsideOutContainer) {
      ContainerDoc_ShutDownLastInPlaceServerIfNotNeeded(
            lpContainerLine->m_lpDoc, lpContainerLine);
   }
#endif  // INPLACE_CNTR

   OLEDBG_END3
   return TRUE;

error:

   if (lpContainerLine->m_dwLinkType != 0)
      lpContainerLine->m_fLinkUnavailable = TRUE;

#if defined( INPLACE_CNTR )
   /* OLE2NOTE: we want to keep only 1 inplace server active at any
   **    given time. so when we start to do a DoVerb on another line,
   **    then we want to shut down the previously activated server. in
   **    this way we keep at most one inplace server active at a time.
   **    even though the DoVerb failed, we will still shutdown the
   **    previous server. it is possible that we ran out of memory and
   **    that the DoVerb will succeed next time after shutting down
   **    the pervious server.
   */
   if (!g_fInsideOutContainer) {
      ContainerDoc_ShutDownLastInPlaceServerIfNotNeeded(
            lpContainerLine->m_lpDoc, lpContainerLine);
   }
#endif  // INPLACE_CNTR

   /* OLE2NOTE: if an error occurs we must give the appropriate error
   **    message box. there are many potential errors that can occur.
   **    the OLE2.0 user model has specific guidelines as to the
   **    dialogs that should be displayed given the various potential
   **    errors (eg. server not registered, unavailable link source.
   **    the OLE2UI library includes support for most of the
   **    recommended message dialogs. (see OleUIPrompUser function)
   */
   if (fMessage) {
      BOOL fReDoVerb = ContainerLine_ProcessOleRunError(
            lpContainerLine,
            hrErr,
            fAction,
            (lpMsg==NULL && iVerb>=0)   /* fMenuInvoked */
      );
      if (fReDoVerb) {
         goto ExecuteDoVerb;
      }
   }

   OLEDBG_END3
   return FALSE;
}



/* ContainerLine_ProcessOleRunError
 * --------------------------------
 *
 *  Handle the various errors possible when attempting OleRun of an object.
 *  Popup up appropriate message according to the error and/or take action
 *  specified button pressed by the user.
 *
 *  OLE2NOTE: The OLE 2.0 User Interface Guidelines specify the messages
 *            that should be given for the following situations:
 *                  1. Link Source Unavailable...goto Links dialog
 *                  2. Server Not Registered...goto Convert dialog
 *                  3. Link Type Changed
 *                  4. Server Not Found
 *
 *  Returns:    TRUE -- repeat IOleObject::DoVerb call.
 *              FALSE -- do NOT repeat IOleObject::DoVerb call.
 *
 *  Comments:
 *      (see LinkTypeChanged case)
 */
BOOL ContainerLine_ProcessOleRunError(
      LPCONTAINERLINE         lpContainerLine,
      HRESULT                 hrErr,
      BOOL                    fAction,
      BOOL                    fMenuInvoked
)
{
   LPOUTLINEDOC    lpOutlineDoc = (LPOUTLINEDOC)lpContainerLine->m_lpDoc;
   HWND            hwndParent = OutlineDoc_GetWindow(lpOutlineDoc);
   SCODE           sc = GetScode(hrErr);
   BOOL            fReDoVerb = FALSE;
   char szTemp[256];
   char szTemp2[256];

   OleDbgOutHResult("ProcessError", hrErr);
   if ((sc >= MK_E_FIRST) && (sc <= MK_E_LAST))
      goto LinkSourceUnavailable;
   if (sc == OLE_E_CANT_BINDTOSOURCE)
      goto LinkSourceUnavailable;
   if (sc == STG_E_PATHNOTFOUND)
      goto LinkSourceUnavailable;
   if (sc == REGDB_E_CLASSNOTREG)
      goto ServerNotReg;
   if (sc == OLE_E_STATIC)
      goto ServerNotReg;  // user dblclk'ed a static object w/ no svr reg'd
   if (sc == OLE_E_CLASSDIFF)
      goto LinkTypeChanged;
   if (sc == CO_E_APPDIDNTREG)
      goto ServerNotFound;
   if (sc == CO_E_APPNOTFOUND)
      goto ServerNotFound;
   if (sc == E_OUTOFMEMORY)
      goto OutOfMemory;

   if (ContainerLine_IsOleLink(lpContainerLine))
      goto LinkSourceUnavailable;
   else
      goto ServerNotFound;


/*************************************************************************
** Error handling routines                                              **
*************************************************************************/
LinkSourceUnavailable:
   wcstombs(szTemp, APPNAME, 255);
   if (ID_PU_LINKS == OleUIPromptUser(
            (WORD)IDD_LINKSOURCEUNAVAILABLE,
            hwndParent,
            szTemp)) {
      if (fAction) {
         ContainerDoc_EditLinksCommand(lpContainerLine->m_lpDoc);
      }
   }
   return fReDoVerb;

ServerNotReg:
   {
   LPOLESTR lpszUserType = NULL;
   CLIPFORMAT  cfFormat;       // not used

   hrErr = ReadFmtUserTypeStg(
         lpContainerLine->m_lpStg, &cfFormat, &lpszUserType);

   wcstombs(szTemp, APPNAME, 255);
   wcstombs(szTemp2, lpszUserType, 255);
   if (ID_PU_CONVERT == OleUIPromptUser(
         (WORD)IDD_SERVERNOTREG,
         hwndParent,
         szTemp,
         (hrErr == NOERROR) ? szTemp2 : "Unknown Object")) {
      if (fAction) {
         ContainerDoc_ConvertCommand(
               lpContainerLine->m_lpDoc,
               TRUE        // fMustActivate
         );
      }
   }

   if (lpszUserType)
      OleStdFreeString(lpszUserType, NULL);

   return fReDoVerb;
   }


LinkTypeChanged:
   {
   /* OLE2NOTE: If IOleObject::DoVerb is executed on a Link object and it
   **    returns OLE_E_CLASSDIFF because the link source is no longer
   **    the expected class, then if the verb is a semantically
   **    defined verb (eg. OLEIVERB_PRIMARY, OLEIVERB_SHOW,
   **    OLEIVERB_OPEN, etc.), then the link should be re-created with
   **    the new link source and the same verb executed on the new
   **    link. there is no need to give a message to the user. if the
   **    user had selected a verb from the object's verb menu
   **    (fMenuInvoked==TRUE), then we can not be certain of the
   **    semantics of the verb and whether the new link can still
   **    support the verb. in this case the user is given a prompt
   **    telling him to "choose a different command offered by the new
   **    type".
   */

   LPOLESTR       lpszUserType = NULL;

   if (fMenuInvoked) {
      hrErr = lpContainerLine->m_lpOleObj->lpVtbl->GetUserType(
            lpContainerLine->m_lpOleObj,USERCLASSTYPE_FULL,&lpszUserType);
      wcstombs(szTemp, APPNAME, 255);
      wcstombs(szTemp2, lpszUserType, 255);
      OleUIPromptUser(
            (WORD)IDD_LINKTYPECHANGED,
            hwndParent,
            szTemp,
            (hrErr == NOERROR) ? szTemp2 : "Unknown Object"
      );
   } else {
      fReDoVerb = TRUE;
   }
   ContainerLine_ReCreateLinkBecauseClassDiff(lpContainerLine);

   if (lpszUserType)
```

```
      OleStdFreeString(lpszUserType, NULL);

   return fReDoVerb;
   }

ServerNotFound:
   wcstombs(szTemp, APPNAME, 255);
   OleUIPromptUser(
         (WORD)IDD_SERVERNOTFOUND,
         hwndParent,
         (LPOLESTR)szTemp);
   return fReDoVerb;

OutOfMemory:
   wcstombs(szTemp, APPNAME, 255);
   OleUIPromptUser(
         (WORD)IDD_OUTOFMEMORY,
         hwndParent,
         szTemp);
   return fReDoVerb;
}


/* ContainerLine_ReCreateLinkBecauseClassDiff
** -------------------------------------------
**    Re-create the link. The existing link was created when
**    the moniker binds to a link source bound of a different class
**    than the same moniker currently binds to. the link may be a
**    special link object specifically used with the old link
**    source class. thus the link object needs to be re-created to
**    give the new link source the opportunity to create its own
**    special link object. (see description "Custom Link Source")
*/
HRESULT ContainerLine_ReCreateLinkBecauseClassDiff(
      LPCONTAINERLINE lpContainerLine
)
{
   LPOLELINK   lpOleLink = lpContainerLine->m_lpOleLink;
   HGLOBAL     hMetaPict = NULL;
   LPMONIKER   lpmkLinkSrc = NULL;
   SCODE       sc = E_FAIL;
   HRESULT     hrErr;

   if (lpOleLink &&
      lpOleLink->lpVtbl->GetSourceMoniker(
            lpOleLink, (LPMONIKER FAR*)&lpmkLinkSrc) == NOERROR) {

      BOOL            fDisplayAsIcon =
                     (lpContainerLine->m_dwDrawAspect==DVASPECT_ICON);
      STGMEDIUM       medium;
      LPDATAOBJECT    lpDataObj = NULL;
      DWORD           dwOleRenderOpt;
      FORMATETC       renderFmtEtc;
      LPFORMATETC     lpRenderFmtEtc = NULL;
```

```
// get the current icon if object is displayed as icon
if (fDisplayAsIcon &&
   (lpDataObj = (LPDATAOBJECT)OleStdQueryInterface( (LPUNKNOWN)
         lpContainerLine->m_lpOleObj,&IID_IDataObject)) != NULL ) {
   hMetaPict = OleStdGetData(
         lpDataObj, CF_METAFILEPICT, NULL, DVASPECT_ICON, &medium);
   OleStdRelease((LPUNKNOWN)lpDataObj);
}

if (fDisplayAsIcon && hMetaPict) {
   // a special icon should be used. first we create the object
   // OLERENDER_NONE. then we stuff the special icon into the cache.

   dwOleRenderOpt = OLERENDER_NONE;

} else if (fDisplayAsIcon && hMetaPict == NULL) {
   // the object's default icon should be used

   dwOleRenderOpt = OLERENDER_DRAW;
   lpRenderFmtEtc = (LPFORMATETC)&renderFmtEtc;
   SETFORMATETC(renderFmtEtc,0,DVASPECT_ICON,NULL,TYMED_NULL,-1);

} else {
   // create standard DVASPECT_CONTENT/OLERENDER_DRAW object
   dwOleRenderOpt = OLERENDER_DRAW;
}

// unload original link object
ContainerLine_UnloadOleObject(lpContainerLine, OLECLOSE_SAVEIFDIRTY);

// delete entire contents of the current object's storage
OleStdDestroyAllElements(lpContainerLine->m_lpStg);

OLEDBG_BEGIN2("OleCreateLink called\r\n")
hrErr = OleCreateLink (
      lpmkLinkSrc,
      &IID_IOleObject,
      dwOleRenderOpt,
      lpRenderFmtEtc,
      (LPOLECLIENTSITE)&lpContainerLine->m_OleClientSite,
      lpContainerLine->m_lpStg,
      (LPVOID FAR*)&lpContainerLine->m_lpOleObj
);
OLEDBG_END2

if (hrErr == NOERROR) {
   if (! ContainerLine_SetupOleObject(
         lpContainerLine, fDisplayAsIcon, hMetaPict) ) {

      // ERROR: setup of the new link failed.
      // revert the storage to restore the original link.
      ContainerLine_UnloadOleObject(
            lpContainerLine, OLECLOSE_NOSAVE);
      lpContainerLine->m_lpStg->lpVtbl->Revert(
            lpContainerLine->m_lpStg);
            sc = E_FAIL;
         } else {
            sc = S_OK;  // IT WORKED!

         }
      }
      else {
         sc = GetScode(hrErr);
         OleDbgOutHResult("OleCreateLink returned", hrErr);
         // ERROR: Re-creating the link failed.
         // revert the storage to restore the original link.
         lpContainerLine->m_lpStg->lpVtbl->Revert(
               lpContainerLine->m_lpStg);
      }
   }

   if (hMetaPict)
      OleUIMetafilePictIconFree(hMetaPict); // clean up metafile
   return ResultFromScode(sc);
}

/* ContainerLine_GetOleObject
** --------------------------
**    return pointer to desired interface of embedded/linked object.
**
**    NOTE: this function causes an AddRef to the object. when the caller is
**          finished with the object, it must call Release.
**          this function does not AddRef the ContainerLine object.
*/
LPUNKNOWN ContainerLine_GetOleObject(
      LPCONTAINERLINE         lpContainerLine,
      REFIID                  riid
)
{
   /* if object is not already loaded, then load it now. objects are
   **    loaded lazily in this manner.
   */
   if (! lpContainerLine->m_lpOleObj)
      ContainerLine_LoadOleObject(lpContainerLine);

   if (lpContainerLine->m_lpOleObj)
      return OleStdQueryInterface(
            (LPUNKNOWN)lpContainerLine->m_lpOleObj,
            riid
      );
   else
      return NULL;
}



/* ContainerLine_RunOleObject
** --------------------------
**    Load and run the object. Upon running and if size of object has
changed,
**    use SetExtent to change to new size.
**
*/
HRESULT ContainerLine_RunOleObject(LPCONTAINERLINE lpContainerLine)
{
   LPLINE lpLine = (LPLINE)lpContainerLine;
   SIZEL   sizelNew;
   HRESULT hrErr;
   HCURSOR  hPrevCursor;

   if (! lpContainerLine)
      return NOERROR;

   if (lpContainerLine->m_fGuardObj) {
      // object in process of creation--Fail to Run the object
      return ResultFromScode(E_FAIL);
   }

   if (lpContainerLine->m_lpOleObj &&
      OleIsRunning(lpContainerLine->m_lpOleObj))
      return NOERROR;     // object already running

   // this may take a while, put up hourglass cursor
   hPrevCursor = SetCursor(LoadCursor(NULL, IDC_WAIT));
   OLEDBG_BEGIN3("ContainerLine_RunOleObject\r\n")

   if (! lpContainerLine->m_lpOleObj) {
      if (! ContainerLine_LoadOleObject(lpContainerLine))
      {
         SetCursor(hPrevCursor);
         return ResultFromScode(E_OUTOFMEMORY); // Error: couldn't load obj
      }
   }

   OLEDBG_BEGIN2("OleRun called\r\n")
   hrErr = OleRun((LPUNKNOWN)lpContainerLine->m_lpOleObj);
   OLEDBG_END2

   if (hrErr != NOERROR) {
      SetCursor(hPrevCursor);     // restore original cursor

      OleDbgOutHResult("OleRun returned", hrErr);
      OLEDBG_END3
      return hrErr;
   }

   if (lpContainerLine->m_fDoSetExtent) {
      /* OLE2NOTE: the OLE object was resized when it was not running
      **    and the object did not have the OLEMISC_RECOMPOSEONRESIZE
      **    bit set. if it had, the object would have been run
      **    immediately when it was resized. this flag indicates that
      **    the object does something other than simple scaling when
      **    it is resized. because the object is being run now, we
      **    will call IOleObject::SetExtent.
      */
      lpContainerLine->m_fDoSetExtent = FALSE;

      // the size stored in our Line includes the border around the object.
      // we must subtract the border to get the size of the object itself.
      sizelNew.cx = lpLine->m_nWidthInHimetric;
      sizelNew.cy = lpLine->m_nHeightInHimetric;

      if ((sizelNew.cx != lpContainerLine->m_sizeInHimetric.cx) ||
         (sizelNew.cy != lpContainerLine->m_sizeInHimetric.cy)) {

         OLEDBG_BEGIN2("IOleObject::SetExtent called\r\n")
         lpContainerLine->m_lpOleObj->lpVtbl->SetExtent(
               lpContainerLine->m_lpOleObj,
               lpContainerLine->m_dwDrawAspect,
               (LPSIZEL)&sizelNew
         );
         OLEDBG_END2
      }
   }

   SetCursor(hPrevCursor);     // restore original cursor

   OLEDBG_END3
   return NOERROR;

}


/* ContainerLine_IsOleLink
** -----------------------
**
**    return TRUE if the ContainerLine has an OleLink.
**           FALSE if the ContainerLine has an embedding
*/
BOOL ContainerLine_IsOleLink(LPCONTAINERLINE lpContainerLine)
{
   if (!lpContainerLine)
      return FALSE;

   return (lpContainerLine->m_dwLinkType != 0);
}


/*  ContainerLine_Draw
**  ------------------
**
**      Draw a ContainerLine object on a DC.
**
**  Parameters:
**      hDC     - DC to which the line will be drawn
**      lpRect  - the object rect in logical coordinates
**      lpRectWBounds - bounding rect of the metafile underneath hDC
**                      (NULL if hDC is not a metafile DC)
**      fHighlight    - TRUE if line has selection highlight
*/
```

```
void ContainerLine_Draw(
      LPCONTAINERLINE         lpContainerLine,
      HDC                     hDC,
      LPRECT                  lpRect,
      LPRECT                  lpRectWBounds,
      BOOL                    fHighlight
)
{
   LPLINE  lpLine = (LPLINE) lpContainerLine;
   HRESULT hrErr = NOERROR;
   RECTL   rclHim;
   RECTL   rclHimWBounds;
   RECT    rcHim;

   if (lpContainerLine->m_fGuardObj) {
      // object in process of creation--do NOT try to draw
      return;
   }

   /* if object is not already loaded, then load it now. objects are
   **    loaded lazily in this manner.
   */
   if (! lpContainerLine->m_lpViewObj2) {
      if (! ContainerLine_LoadOleObject(lpContainerLine))
      {
         return;     // Error: could not load object
      }
   }

   if (lpRectWBounds) {
      rclHimWBounds.left      = (long) lpRectWBounds->left;
      rclHimWBounds.bottom    = (long) lpRectWBounds->bottom;
      rclHimWBounds.top       = (long) lpRectWBounds->top;
      rclHimWBounds.right     = (long) lpRectWBounds->right;
   }

   /* construct bounds rectangle for the object.
   **  offset origin for object to correct tab indentation
   */
   rclHim.left     = (long) lpRect->left;
   rclHim.bottom   = (long) lpRect->bottom;
   rclHim.top      = (long) lpRect->top;
   rclHim.right    = (long) lpRect->right;

   rclHim.left += (long) ((LPLINE)lpContainerLine)->m_nTabWidthInHimetric;
   rclHim.right += (long) ((LPLINE)lpContainerLine)->m_nTabWidthInHimetric;

#if defined( INPLACE_CNTR )
   /* OLE2NOTE: if the OLE object currently has a visible in-place
   **    window, then we do NOT want to draw on top of its window.
   **    this could interfere with the object's display.
   */
   if ( !lpContainerLine->m_fIpVisible )
#endif
   {
   hrErr = lpContainerLine->m_lpViewObj2->lpVtbl->Draw(
         lpContainerLine->m_lpViewObj2,
         lpContainerLine->m_dwDrawAspect,
         -1,
         NULL,
         NULL,
         NULL,
         hDC,
         (LPRECTL)&rclHim,
         (lpRectWBounds ? (LPRECTL)&rclHimWBounds : NULL),
         NULL,
         0
   );
   if (hrErr != NOERROR)
      OleDbgOutHResult("IViewObject::Draw returned", hrErr);

   if (lpContainerLine->m_fObjWinOpen)
      {
      rcHim.left      = (int) rclHim.left;
      rcHim.top       = (int) rclHim.top;
      rcHim.right     = (int) rclHim.right;
      rcHim.bottom    = (int) rclHim.bottom;

      /* OLE2NOTE: if the object servers window is Open (ie. not active
      **    in-place) then we must shade the object in our document to
      **    indicate to the user that the object is open elsewhere.
      */
      OleUIDrawShading((LPRECT)&rcHim, hDC, OLEUI_SHADE_FULLRECT, 0);
      }
   }

   /* if the object associated with the ContainerLine is an automatic
   **    link then try to connect it with its LinkSource if the
   **    LinkSource is already running. we do not want to force the
   **    LinkSource to run.
   **
   **    OLE2NOTE: a sophistocated container will want to continually
   **    attempt to connect its automatic links. OLE does NOT
   **    automatically connect links when link sources become
   **    available. some containers will want to attempt to connect
   **    its links as part of idle time processing. another strategy
   **    is to attempt to connect an automatic link every time it is
   **    drawn on the screen. (this is the strategy used by this
   **    CntrOutl sample application.)
   */
   if (lpContainerLine->m_dwLinkType == OLEUPDATE_ALWAYS)
      ContainerLine_BindLinkIfLinkSrcIsRunning(lpContainerLine);

   return;
}


void ContainerLine_DrawSelHilight(
      LPCONTAINERLINE lpContainerLine,
      HDC             hDC,            // MM_TEXT mode
      LPRECT          lprcPix,        // listbox rect
      UINT            itemAction,
      UINT            itemState
)
{
   LPLINE  lpLine = (LPLINE)lpContainerLine;
   RECT    rcObj;
   DWORD   dwFlags = OLEUI_HANDLES_INSIDE | OLEUI_HANDLES_USEINVERSE;
   int     nHandleSize;
   LPCONTAINERDOC lpContainerDoc;

   if (!lpContainerLine || !hDC || !lprcPix)
      return;

   lpContainerDoc = lpContainerLine->m_lpDoc;

   // Get size of OLE object
   ContainerLine_GetOleObjectRectInPixels(lpContainerLine, (LPRECT)&rcObj);

   nHandleSize = GetProfileInt("windows", "oleinplaceborderwidth",
         DEFAULT_HATCHBORDER_WIDTH) + 1;

   OleUIDrawHandles((LPRECT)&rcObj, hDC, dwFlags, nHandleSize, TRUE);
}

/* InvertDiffRect
** --------------
**
**    Paint the surrounding of the Obj rect black but within lprcPix
**      (similar to the lprcPix minus lprcObj)
*/
static void InvertDiffRect(LPRECT lprcPix, LPRECT lprcObj, HDC hDC)
{
   RECT rcBlack;

   // draw black in all space outside of object's rectangle
   rcBlack.top = lprcPix->top;
   rcBlack.bottom = lprcPix->bottom;

   rcBlack.left = lprcPix->left + 1;
   rcBlack.right = lprcObj->left - 1;
   InvertRect(hDC, (LPRECT)&rcBlack);

   rcBlack.left = lprcObj->right + 1;
   rcBlack.right = lprcPix->right - 1;
   InvertRect(hDC, (LPRECT)&rcBlack);

   rcBlack.top = lprcPix->top;
   rcBlack.bottom = lprcPix->top + 1;
   rcBlack.left = lprcObj->left - 1;
   rcBlack.right = lprcObj->right + 1;
   InvertRect(hDC, (LPRECT)&rcBlack);

   rcBlack.top = lprcPix->bottom;
   rcBlack.bottom = lprcPix->bottom - 1;
   rcBlack.left = lprcObj->left - 1;
   rcBlack.right = lprcObj->right + 1;
   InvertRect(hDC, (LPRECT)&rcBlack);
}


/* Edit the ContainerLine line object.
**      returns TRUE if line was changed
**              FALSE if the line was NOT changed
*/
BOOL ContainerLine_Edit(LPCONTAINERLINE lpContainerLine, HWND hWndDoc,HDC
hDC)
{
   ContainerLine_DoVerb(lpContainerLine, OLEIVERB_PRIMARY, NULL, TRUE,
TRUE);

   /* assume object was NOT changed, if it was obj will send Changed
   **    or Saved notification.
   */
   return FALSE;
}



/* ContainerLine_SetHeightInHimetric
** ---------------------------------
**
** Set the height of a ContainerLine object. The widht will be changed
** to keep the aspect ratio
*/
void ContainerLine_SetHeightInHimetric(LPCONTAINERLINE lpContainerLine, int
nHeight)
{
   LPLINE  lpLine = (LPLINE)lpContainerLine;
   SIZEL   sizelOleObject;
   HRESULT hrErr;

   if (!lpContainerLine)
      return;

   if (lpContainerLine->m_fGuardObj) {
      // object in process of creation--Fail to set the Height
      return;
   }

   if (nHeight != -1) {
      BOOL    fMustClose = FALSE;
      BOOL    fMustRun   = FALSE;

      /* if object is not already loaded, then load it now. objects are
      **    loaded lazily in this manner.
      */
      if (! lpContainerLine->m_lpOleObj)
         ContainerLine_LoadOleObject(lpContainerLine);
```

```
// the height argument specifies the desired height for the Line.
sizelOleObject.cy = nHeight;

// we will calculate the corresponding width for the object by
// maintaining the current aspect ratio of the object.
sizelOleObject.cx = (int)(sizelOleObject.cy *
      lpContainerLine->m_sizeInHimetric.cx /
      lpContainerLine->m_sizeInHimetric.cy);

/* OLE2NOTE: if the OLE object is already running then we can
**    immediately call SetExtent. But, if the object is NOT
**    currently running then we will check if the object
**    indicates that it is normally recomposes itself on
**    resizing. ie. that the object does not simply scale its
**    display when it it resized. if so then we will force the
**    object to run so that we can call IOleObject::SetExtent.
**    SetExtent does not have any effect if the object is only
**    loaded. if the object does NOT indicate that it
**    recomposes on resize (OLEMISC_RECOMPOSEONRESIZE) then we
**    will wait till the next time that the object is run to
**    call SetExtent. we will store a flag in the ContainerLine
**    to indicate that a SetExtent is necessary. It is
**    necessary to persist this flag.
*/
if (! OleIsRunning(lpContainerLine->m_lpOleObj)) {
   DWORD dwStatus;

   OLEDBG_BEGIN2("IOleObject::GetMiscStatus called\r\n")
   hrErr = lpContainerLine->m_lpOleObj->lpVtbl->GetMiscStatus(
         lpContainerLine->m_lpOleObj,
         lpContainerLine->m_dwDrawAspect,
         (LPDWORD)&dwStatus
   );
   OLEDBG_END2
   if (hrErr == NOERROR && (dwStatus & OLEMISC_RECOMPOSEONRESIZE)) {
      // force the object to run
      ContainerLine_RunOleObject(lpContainerLine);
      fMustClose = TRUE;
   } else {
      /*  the OLE object is NOT running and does NOT
      **    recompose on resize. simply scale the object now
      **    and do the SetExtent the next time the object is
      **    run. we set the Line to the new size even though
      **    the object's extents have not been changed.
      **    this has the result of scaling the object's
      **    display to the new size.
      */
      lpContainerLine->m_fDoSetExtent = TRUE;
      ContainerLine_SetLineHeightFromObjectExtent(
            lpContainerLine, (LPSIZEL)&sizelOleObject);
      return;
   }
}

OLEDBG_BEGIN2("IOleObject::SetExtent called\r\n")
```

```
      hrErr = lpContainerLine->m_lpOleObj->lpVtbl->SetExtent(
            lpContainerLine->m_lpOleObj,
            lpContainerLine->m_dwDrawAspect,
            (LPSIZEL)&sizelOleObject);
      OLEDBG_END2

      if (hrErr != NOERROR) {
         /* OLE Object refuses to take on the new extents. Set the
         **    Line to the new size even though the object refused
         **    the new extents. this has the result of scaling the
         **    object's display to the new size.
         **
         **    if the object HAD accepted the new extents, then it
         **    will send out an OnViewChange/OnDataChange
         **    notification. this results in our container receiving
         **    an OnViewChange notification; the line height will be
         **    reset when this notification is received.
         */
         ContainerLine_SetLineHeightFromObjectExtent(
               lpContainerLine, (LPSIZEL)&sizelOleObject);
      }

      if (fMustClose)
         ContainerLine_CloseOleObject(
               lpContainerLine, OLECLOSE_SAVEIFDIRTY);
   }
   else {
      /* Set the line to default height given the natural (unscaled)
      **    extents of the OLE object.
      */
      ContainerLine_SetLineHeightFromObjectExtent(
            lpContainerLine,(LPSIZEL)&lpContainerLine->m_sizeInHimetric);
   }
}


/*  ContainerLine_SetLineHeightFromObjectExtent
 *
 *  Purpose:
 *      Calculate the corresponding line height from the OleObject size
 *      Scale the line height to fit the limit if necessary
 *
 *  Parameters:
 *      lpsizelOleObject        pointer to size of OLE Object
 *
 *  Returns:
 *      nil
 */
void ContainerLine_SetLineHeightFromObjectExtent(
      LPCONTAINERLINE         lpContainerLine,
      LPSIZEL                 lpsizelOleObject
)
{
   LPLINE lpLine = (LPLINE)lpContainerLine;
   UINT uMaxObjectHeight = XformHeightInPixelsToHimetric(NULL,
         LISTBOX_HEIGHT_LIMIT);

   if (!lpContainerLine || !lpsizelOleObject)
      return;

   if (lpContainerLine->m_fGuardObj) {
      // object in process of creation--Fail to set the Height
      return;
   }

   lpLine->m_nWidthInHimetric = (int)lpsizelOleObject->cx;
   lpLine->m_nHeightInHimetric = (int)lpsizelOleObject->cy;

   // Rescale the object if height is greater than the limit
   if (lpLine->m_nHeightInHimetric > (UINT)uMaxObjectHeight) {

      lpLine->m_nWidthInHimetric = (UINT)
            ((long)lpLine->m_nWidthInHimetric *
            (long)uMaxObjectHeight /
            (long)lpLine->m_nHeightInHimetric);

      lpLine->m_nHeightInHimetric = uMaxObjectHeight;
   }

}


/* ContainerLine_SaveToStg
** ----------------------
**    Save a given ContainerLine and associated OLE object to an IStorage*.
*/
BOOL ContainerLine_SaveToStm(
      LPCONTAINERLINE         lpContainerLine,
      LPSTREAM                lpLLStm
)
{
   CONTAINERLINERECORD_ONDISK objLineRecord;
   ULONG nWritten;
   HRESULT hrErr;
   //  Compilers should handle alignment correctly

   OLESTRCPY(objLineRecord.m_szStgName, lpContainerLine->m_szStgName);
   objLineRecord.m_fMonikerAssigned = (USHORT)lpContainerLine-
>m_fMonikerAssigned;
   objLineRecord.m_dwDrawAspect = lpContainerLine->m_dwDrawAspect;
   objLineRecord.m_sizeInHimetric = lpContainerLine->m_sizeInHimetric;
   objLineRecord.m_dwLinkType = lpContainerLine->m_dwLinkType;
   objLineRecord.m_fDoSetExtent = (USHORT) lpContainerLine->m_fDoSetExtent;

   /* write line record */
   hrErr = lpLLStm->lpVtbl->Write(
         lpLLStm,
         (LPVOID)&objLineRecord,
         sizeof(objLineRecord),
         &nWritten
   );

   if (hrErr != NOERROR) {
      OleDbgAssertSz(hrErr == NOERROR,"Could not write to LineList stream");
    {
      return FALSE;
    }
   }

   return TRUE;
}


/* ContainerLine_SaveOleObjectToStg
** --------------------------------
**    Save the OLE object associated with the ContainerLine to an IStorage*.
*/
BOOL ContainerLine_SaveOleObjectToStg(
      LPCONTAINERLINE         lpContainerLine,
      LPSTORAGE               lpSrcStg,
      LPSTORAGE               lpDestStg,
      BOOL                    fRemember
)
{
   HRESULT          hrErr;
   SCODE            sc = S_OK;
   BOOL             fStatus;
   BOOL             fSameAsLoad = (lpSrcStg==lpDestStg ? TRUE : FALSE);
   LPSTORAGE        lpObjDestStg;

   if (lpContainerLine->m_fGuardObj) {
      // object in process of creation--Fail to save
      return FALSE;
   }

   if (! lpContainerLine->m_lpOleObj) {

      /*****************************************************************
      ** CASE 1: object is NOT loaded.
      *****************************************************************/

      if (fSameAsLoad) {
         /************************************************************
         ** CASE 1A: we are saving to the current storage. because
         **    the object is not loaded, it is up-to-date
         **    (ie. nothing to do).
         ************************************************************/

         ;

      } else {
         /************************************************************
         ** CASE 1B: we are saving to a new storage. because
         **    the object is not loaded, we can simply copy the
```

```
        **    object's current storage to the new storage.
        *************************************************************/

        /* if current object storage is not already open, then open it */
        if (! lpContainerLine->m_lpStg) {
           lpContainerLine->m_lpStg = OleStdOpenChildStorage(
                 lpSrcStg,
                 lpContainerLine->m_szStgName,
                 STGM_READWRITE
              );
           if (lpContainerLine->m_lpStg == NULL) {
#if defined( _DEBUG )
              OleDbgAssertSz(
                    lpContainerLine->m_lpStg != NULL,
                    "Error opening child stg"
              );
#endif
              return FALSE;
           }
        }

        /* Create a child storage inside the destination storage. */
        lpObjDestStg = OleStdCreateChildStorage(
              lpDestStg,
              lpContainerLine->m_szStgName
        );

        if (lpObjDestStg == NULL) {
#if defined( _DEBUG )
           OleDbgAssertSz(
                 lpObjDestStg != NULL,
                 "Could not create obj storage!"
           );
#endif
           return FALSE;
        }

        hrErr = lpContainerLine->m_lpStg->lpVtbl->CopyTo(
              lpContainerLine->m_lpStg,
              0,
              NULL,
              NULL,
              lpObjDestStg
        );
        // REVIEW: should we handle error here?
        fStatus = OleStdCommitStorage(lpObjDestStg);

        /* if we are supposed to remember this storage as the new
        **    storage for the object, then release the old one and
        **    save the new one. else, throw away the new one.
        */
        if (fRemember) {
           OleStdVerifyRelease(
                 (LPUNKNOWN)lpContainerLine->m_lpStg,
                 OLESTR("Original object stg not released")
```

```
         );
         lpContainerLine->m_lpStg = lpObjDestStg;
      } else {
         OleStdVerifyRelease(
               (LPUNKNOWN)lpObjDestStg,
               OLESTR("Copied object stg not released")
         );
      }
   }

} else {

   /****************************************************************
   ** CASE 2: object IS loaded.
   ****************************************************************/

   if (fSameAsLoad) {
      /*************************************************************
      ** CASE 2A: we are saving to the current storage. if the object
      **    is not dirty, then the current storage is up-to-date
      **    (ie. nothing to do).
      *************************************************************/

      LPPERSISTSTORAGE lpPersistStg = lpContainerLine->m_lpPersistStg;
      OleDbgAssert(lpPersistStg);

      hrErr = lpPersistStg->lpVtbl->IsDirty(lpPersistStg);

      /* OLE2NOTE: we will only accept an explicit "no i
      **    am NOT dirty statement" (ie. S_FALSE) as an
      **    indication that the object is clean. eg. if
      **    the object returns E_NOTIMPL we must
      **    interpret it as the object IS dirty.
      */
      if (GetScode(hrErr) != S_FALSE) {

         /* OLE object IS dirty */

         OLEDBG_BEGIN2("OleSave called\r\n")
         hrErr = OleSave(
               lpPersistStg, lpContainerLine->m_lpStg, fSameAsLoad);
         OLEDBG_END2

         if (hrErr != NOERROR) {
            OleDbgOutHResult("WARNING: OleSave returned", hrErr);
            sc = GetScode(hrErr);
         }

         // OLE2NOTE: if OleSave fails, SaveCompleted must be called.
         OLEDBG_BEGIN2("IPersistStorage::SaveCompleted called\r\n")
         hrErr=lpPersistStg->lpVtbl->SaveCompleted(lpPersistStg,NULL);
         OLEDBG_END2

         if (hrErr != NOERROR) {
            OleDbgOutHResult("WARNING: SaveCompleted returned",hrErr);
```

```
                  if (sc == S_OK)
                     sc = GetScode(hrErr);
               }

               if (sc != S_OK)
               {
                  return FALSE;
               }
            }

         } else {
            /************************************************************
            ** CASE 2B: we are saving to a new storage. we must
            **    tell the object to save into the new storage.
            ************************************************************/

            LPPERSISTSTORAGE lpPersistStg = lpContainerLine->m_lpPersistStg;

            if (! lpPersistStg) return FALSE;

            /* Create a child storage inside the destination storage. */
            lpObjDestStg = OleStdCreateChildStorage(
                  lpDestStg,
                  lpContainerLine->m_szStgName
            );

            if (lpObjDestStg == NULL) {
#if defined( _DEBUG )
               OleDbgAssertSz(
                     lpObjDestStg != NULL,
                     "Could not create object storage!"
               );
#endif
               return FALSE;
            }

            OLEDBG_BEGIN2("OleSave called\r\n")
            hrErr = OleSave(lpPersistStg, lpObjDestStg, fSameAsLoad);
            OLEDBG_END2

            // OLE2NOTE: even if OleSave fails, must still call SaveCompleted
            if (hrErr != NOERROR) {
               OleDbgOutHResult("WARNING: OleSave returned", hrErr);
               sc = GetScode(hrErr);
            }

            /* OLE2NOTE: a root level container should immediately
            **    call IPersistStorage::SaveCompleted after calling
            **    OleSave. a nested level container should not call
            **    SaveCompleted now, but must wait until SaveCompleted
            **    is call on it by its container. since our container
            **    is not a container/server, then we always call
            **    SaveComplete here.
            **
            **    if this is a SaveAs operation, then we need to pass
```

```
      **    the lpStg back in SaveCompleted to inform the object
      **    of its new storage that it may hold on to. if this is
      **    a Save or a SaveCopyAs operation, then we simply pass
      **    NULL in SaveCompleted; the object can continue to hold
      **    its current storage. if an error occurs during the
      **    OleSave call we must still call SaveCompleted but we
      **    must pass NULL.
      */
      OLEDBG_BEGIN2("IPersistStorage::SaveCompleted called\r\n")
      hrErr = lpPersistStg->lpVtbl->SaveCompleted(
            lpPersistStg,
            ((FAILED(sc) || !fRemember) ? NULL : lpObjDestStg)
      );
      OLEDBG_END2

      if (hrErr != NOERROR) {
         OleDbgOutHResult("WARNING: SaveCompleted returned",hrErr);
         if (sc == S_OK)
            sc = GetScode(hrErr);
      }

      if (sc != S_OK) {
         OleStdVerifyRelease(
               (LPUNKNOWN)lpObjDestStg,
               OLESTR("Copied object stg not released")
         );
         return FALSE;
      }

      /* if we are supposed to remember this storage as the new
      **    storage for the object, then release the old one and
      **    save the new one. else, throw away the new one.
      */
      if (fRemember) {
         OleStdVerifyRelease(
               (LPUNKNOWN)lpContainerLine->m_lpStg,
               OLESTR("Original object stg not released")
         );
         lpContainerLine->m_lpStg = lpObjDestStg;
      } else {
         OleStdVerifyRelease(
               (LPUNKNOWN)lpObjDestStg,
               OLESTR("Copied object stg not released")
         );
      }
   }
}

/* OLE2NOTE: after saving an OLE object it is possible that it sent
**    an OnViewChange notification because it had been modified. in
**    this situation it is possible that the extents of the object
**    have changed. if so we want to relayout the space for the
**    object immediately so that the extent information saved with
**    the ContainerLine match the data saved with the OLE object
**    itself.
```

```
   */
   if (lpContainerLine->m_fDoGetExtent) {
      BOOL fSizeChanged = ContainerLine_UpdateExtent(lpContainerLine, NULL);
#if defined( INPLACE_CNTR )
      /* if the extents of this ContainerLine have changed, then we
      **    need to reset the fDoGetExtent flag to TRUE so that later
      **    when ContainerDoc_UpdateExtentOfAllOleObjects is called
      **    (when the WM_U_UPDATEOBJECTEXTENT message is processed),
      **    it is recognized that the extents of this line have
      **    changed. if any line changes size, then any in-place
      **    active object below this line must be told to update the
      **    position of their windows (via SetObjectRects -- see
      **    ContainerDoc_UpdateInPlaceObjectRects function).
      */
      lpContainerLine->m_fDoGetExtent = fSizeChanged;
#endif
   }

   return TRUE;
}


/* ContainerLine_LoadFromStg
** -------------------------
**    Create a ContainerLine object and initialize it with data that
**    was previously writen to an IStorage*. this function does not
**    immediately OleLoad the associated OLE object, only the data of
**    the ContainerLine object itself is loaded from the IStorage*.
*/
LPLINE ContainerLine_LoadFromStg(
      LPSTORAGE               lpSrcStg,
      LPSTREAM                lpLLStm,
      LPOUTLINEDOC            lpDestDoc
)
{
   HDC         hDC;
   LPLINELIST  lpDestLL = &lpDestDoc->m_LineList;
   ULONG nRead;
   HRESULT hrErr;
   LPCONTAINERLINE lpContainerLine;
   CONTAINERLINERECORD_ONDISK objLineRecord;

   lpContainerLine=(LPCONTAINERLINE) New((DWORD)sizeof(objLineRecord));
   if (lpContainerLine == NULL) {
      OleDbgAssertSz(lpContainerLine!=NULL,"Error allocating
ContainerLine");
      return NULL;
   }

   hDC = LineList_GetDC(lpDestLL);
   ContainerLine_Init(lpContainerLine, 0, hDC);
   LineList_ReleaseDC(lpDestLL, hDC);

   /* OLE2NOTE: In order to have a stable ContainerLine object we must
   **    AddRef the object's refcnt. this will be later released when
   **    the ContainerLine is deleted.
   */
   ContainerLine_AddRef(lpContainerLine);

   lpContainerLine->m_lpDoc = (LPCONTAINERDOC) lpDestDoc;

   /* read line record */
   hrErr = lpLLStm->lpVtbl->Read(
         lpLLStm,
         (LPVOID)&objLineRecord,
         sizeof(CONTAINERLINERECORD),
         &nRead
   );

   if (hrErr != NOERROR) {
      OleDbgAssertSz(hrErr==NOERROR, "Could not read from LineList stream");
      goto error;
   }

   //  Compilers should handle alignment correctly
   OLESTRCPY(lpContainerLine->m_szStgName, objLineRecord.m_szStgName);
   lpContainerLine->m_fMonikerAssigned = (BOOL)
objLineRecord.m_fMonikerAssigned;
   lpContainerLine->m_dwDrawAspect = objLineRecord.m_dwDrawAspect;
   lpContainerLine->m_sizeInHimetric = objLineRecord.m_sizeInHimetric;
   lpContainerLine->m_dwLinkType = objLineRecord.m_dwLinkType;
   lpContainerLine->m_fDoSetExtent = (BOOL) objLineRecord.m_fDoSetExtent;

   return (LPLINE)lpContainerLine;

error:
   // destroy partially created ContainerLine
   if (lpContainerLine)
      ContainerLine_Delete(lpContainerLine);
   return NULL;
}


/* ContainerLine_GetTextLen
 * ------------------------
 *
 * Return length of the string representation of the ContainerLine
 *  (not considering the tab level). we will use the following as the
 *  string representation of a ContainerLine:
 *      "<" + user type name of OLE object + ">"
 *  eg:
 *      <Microsoft Excel Worksheet>
 */
int ContainerLine_GetTextLen(LPCONTAINERLINE lpContainerLine)
{
   LPOLESTR   lpszUserType = NULL;
   HRESULT hrErr;
   int     nLen;
   BOOL    fIsLink = ContainerLine_IsOleLink(lpContainerLine);
   /* if object is not already loaded, then load it now. objects are
   **    loaded lazily in this manner.
   */
   if (! lpContainerLine->m_lpOleObj)
      ContainerLine_LoadOleObject(lpContainerLine);

   OLEDBG_BEGIN2("IOleObject::GetUserType called\r\n")
   hrErr = lpContainerLine->m_lpOleObj->lpVtbl->GetUserType(
         lpContainerLine->m_lpOleObj,
         USERCLASSTYPE_FULL,
         &lpszUserType
   );
   OLEDBG_END2

   if (hrErr != NOERROR)   {
      // user type is NOT available
      nLen = sizeof(UNKNOWN_OLEOBJ_TYPE) + 2; // allow space for '<' + '>'
      nLen += lstrlen((LPSTR)(fIsLink ? szOLELINK : szOLEOBJECT)) + 1;
   } else {
      nLen = OLESTRLEN(lpszUserType) + 2;   // allow space for '<' + '>'
      nLen += OLESTRLEN((LPOLESTR)(fIsLink ? szOLELINK : szOLEOBJECT)) + 1;

      /* OLE2NOTE: we must free the string that was allocated by the
      **    IOleObject::GetUserType method.
      */
      OleStdFreeString(lpszUserType, NULL);
   }

   return nLen;
}


/* ContainerLine_GetTextData
 * -------------------------
 *
 * Return the string representation of the ContainerLine
 *  (not considering the tab level). we will use the following as the
 *  string representation of a ContainerLine:
 *      "<" + user type name of OLE object + ">"
 *  eg:
 *      <Microsoft Excel Worksheet>
 */
void ContainerLine_GetTextData(LPCONTAINERLINE lpContainerLine, LPOLESTR
lpszBuf)
{
   LPOLESTR   lpszUserType = NULL;
   BOOL    fIsLink = ContainerLine_IsOleLink(lpContainerLine);
   HRESULT hrErr;
   char    szAnsiBuf[256], szAnsiUserType[256], szAnsiString[256];
   LPOLESTR lpszUniStr;

   /* if object is not already loaded, then load it now. objects are
   **    loaded lazily in this manner.
   */
   if (! lpContainerLine->m_lpOleObj)
```

```
      ContainerLine_LoadOleObject(lpContainerLine);

   hrErr = lpContainerLine->m_lpOleObj->lpVtbl->GetUserType(
         lpContainerLine->m_lpOleObj,
         USERCLASSTYPE_FULL,
         &lpszUserType
   );

   W2A (lpszBuf, szAnsiBuf, 256);
   W2A (lpszUserType, szAnsiUserType, 256);
   lpszUniStr = (fIsLink ? szOLELINK : szOLEOBJECT);
   W2A (lpszUniStr, szAnsiString, 256);
   if (hrErr != NOERROR)   {
      // user type is NOT available
      wsprintf(
            szAnsiBuf,
            "<%s %s>",
            UNKNOWN_OLEOBJ_TYPE,
            (LPSTR)szAnsiString
      );
   } else {
      wsprintf(
            szAnsiBuf,
            "<%s %s>",
            szAnsiUserType,
            (LPSTR)szAnsiString
      );

      /* OLE2NOTE: we must free the string that was allocated by the
      **    IOleObject::GetUserType method.
      */
      OleStdFreeString(lpszUserType, NULL);
   }
}


/* ContainerLine_GetOutlineData
 * ----------------------------
 *
 * Return the CF_OUTLINE format data for the ContainerLine.
 */
BOOL ContainerLine_GetOutlineData(
      LPCONTAINERLINE         lpContainerLine,
      LPTEXTLINE              lpBuf
)
{
   LPLINE      lpLine = (LPLINE)lpContainerLine;
   LPLINELIST  lpLL = &((LPOUTLINEDOC)lpContainerLine->m_lpDoc)->m_LineList;
   HDC         hDC;
   OLECHAR     szTmpBuf[MAXSTRLEN+1];
   char        szAnsiStr[256];
   LPTEXTLINE  lpTmpTextLine;

   // Create a TextLine with the Text representation of the ContainerLine.
   ContainerLine_GetTextData(lpContainerLine, (LPOLESTR)szTmpBuf);
   hDC = LineList_GetDC(lpLL);
   W2A (szTmpBuf, szAnsiStr, 256);
   lpTmpTextLine = TextLine_Create(hDC, lpLine->m_nTabLevel, szAnsiStr);
   LineList_ReleaseDC(lpLL, hDC);

   TextLine_Copy(lpTmpTextLine, lpBuf);

   // Delete the temporary TextLine
   TextLine_Delete(lpTmpTextLine);
   return TRUE;
}


/* ContainerLine_GetPosRect
** -----------------------
**    Get the PosRect in client coordinates for the OLE object's window.
**
** OLE2NOTE: the PosRect must take into account the scroll postion of
**    the document window.
*/
void ContainerLine_GetPosRect(
      LPCONTAINERLINE     lpContainerLine,
      LPRECT              lprcPosRect
)
{
   ContainerLine_GetOleObjectRectInPixels(lpContainerLine,lprcPosRect);

   // shift rect for left margin
   lprcPosRect->left += lpContainerLine->m_nHorizScrollShift;
   lprcPosRect->right += lpContainerLine->m_nHorizScrollShift;
}


/* ContainerLine_GetOleObjectRectInPixels
** --------------------------------------
**    Get the extent of the OLE Object contained in the given Line in
**    client coordinates after scaling.
*/
void ContainerLine_GetOleObjectRectInPixels(LPCONTAINERLINE lpContainerLine,
LPRECT lprc)
{
   LPOUTLINEDOC lpOutlineDoc;
   LPSCALEFACTOR lpscale;
   LPLINELIST lpLL;
   LPLINE lpLine;
   int nIndex;
   HDC hdcLL;

   if (!lpContainerLine || !lprc)
      return;

   lpOutlineDoc = (LPOUTLINEDOC)lpContainerLine->m_lpDoc;
   lpscale = OutlineDoc_GetScaleFactor(lpOutlineDoc);
   lpLL = OutlineDoc_GetLineList(lpOutlineDoc);
   lpLine = (LPLINE)lpContainerLine;
   nIndex = LineList_GetLineIndex(lpLL, lpLine);

   LineList_GetLineRect(lpLL, nIndex, lprc);

   hdcLL = GetDC(lpLL->m_hWndListBox);

   /* lprc is set to be size of Line Object (including the boundary) */
   lprc->left += (int)(
         (long)XformWidthInHimetricToPixels(hdcLL,
               lpLine->m_nTabWidthInHimetric +
               LOWORD(OutlineDoc_GetMargin(lpOutlineDoc))) *
         lpscale->dwSxN / lpscale->dwSxD);
   lprc->right = (int)(
         lprc->left + (long)
         XformWidthInHimetricToPixels(hdcLL, lpLine->m_nWidthInHimetric) *
         lpscale->dwSxN / lpscale->dwSxD);

   ReleaseDC(lpLL->m_hWndListBox, hdcLL);
}


/* ContainerLine_GetOleObjectSizeInHimetric
** ----------------------------------------
**    Get the size of the OLE Object contained in the given Line
*/
void ContainerLine_GetOleObjectSizeInHimetric(LPCONTAINERLINE
lpContainerLine, LPSIZEL lpsizel)
{
   if (!lpContainerLine || !lpsizel)
      return;

   *lpsizel = lpContainerLine->m_sizeInHimetric;
}


/* ContainerLine_BindLinkIfLinkSrcIsRunning
** ----------------------------------------
**    Try to connect the OLE link object associated with the
**    ContainerLine with its LinkSource if the LinkSource is already
**    running and the link is an automatic link. we do not want to
**    force the LinkSource to run.
**
**    OLE2NOTE: a sophistocated container will want to continually
**    attempt to connect its automatic links. OLE does NOT
**    automatically connect links when link source become available. some
**    containers will want to attempt to connect its links as part of
**    idle time processing. another strategy is to attempt to connect
**    an automatic link every time it is drawn on the screen. (this is
**    the strategy used by this CntrOutl sample application.)
*/
void ContainerLine_BindLinkIfLinkSrcIsRunning(LPCONTAINERLINE
lpContainerLine)
{
   LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;
```

```
    HRESULT hrErr;
    BOOL fPrevEnable1;
    BOOL fPrevEnable2;

    // if the link source is known to be un-bindable, then don't even try
    if (lpContainerLine->m_fLinkUnavailable)
       return;

    /* OLE2NOTE: we do not want to ever give the Busy/NotResponding
    **    dialogs when we are attempting to BindIfRunning to the link
    **    source. if the link source is currently busy, this could
    **    cause the Busy dialog to come up. even if the link source is
    **    busy, we do not want put up the busy dialog. thus we will
    **    disable the dialog and later re-enable them
    */
    OleApp_DisableBusyDialogs(lpOleApp, &fPrevEnable1, &fPrevEnable2);

    OLEDBG_BEGIN2("IOleLink::BindIfRunning called\r\n")
    hrErr = lpContainerLine->m_lpOleLink->lpVtbl->BindIfRunning(
          lpContainerLine->m_lpOleLink);
    OLEDBG_END2

    // re-enable the Busy/NotResponding dialogs
    OleApp_EnableBusyDialogs(lpOleApp, fPrevEnable1, fPrevEnable2);
}
```

CNTROUTL.DEF   (OUTLINE Sample)

```
;;
;;
;;    OLE 2.0 Container Sample Code
;;
;;    cntroutl.def
;;
;;    Definition file for cntroutl.exe
;;
;;    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
;;
;;

NAME            CntrOutl
DESCRIPTION     'Microsoft OLE 2.0 Container Sample code'
EXETYPE         WINDOWS
STUB            'WINSTUB.EXE'
CODE            PRELOAD MOVEABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        4096

;; NOTE: Do not add exports to this file.  Use __export
;; in your function prototype and definition instead.
```

CNTROUTL.H (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Container Sample Code
**
**    cntroutl.h
**
**    This file contains file contains data structure defintions,
**    function prototypes, constants, etc. used by the OLE 2.0 container
**    app version of the Outline series of sample applications:
**          Outline -- base version of the app (without OLE functionality)
**          SvrOutl -- OLE 2.0 Server sample app
**          CntrOutl -- OLE 2.0 Containter (Container) sample app
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#if !defined( _CNTROUTL_H_ )
#define _CNTROUTL_H_

#ifndef RC_INVOKED
#pragma message ("INCLUDING CNTROUTL.H from " __FILE__)
#endif  /* RC_INVOKED */

#include "oleoutl.h"
#include "cntrrc.h"
#include "ansiapi.h"

// REVIEW: should load from string resource
#define DEFOBJNAMEPREFIX    "Obj"   // Prefix for auto-generated stg names
#define DEFOBJWIDTH         5000    // default size for embedded obj.
#define DEFOBJHEIGHT        5000    // default size for embedded obj.
#define UNKNOWN_OLEOBJ_TYPE "Unknown OLE Object Type"
#define szOLEOBJECT OLESTR("Object")
#define szOLELINK   OLESTR("Link")

#ifdef TEST32
#define CONTAINERDOCFORMAT  OLESTR("CntrOu32")      // CF_CntrOutl format
name
#else
#define CONTAINERDOCFORMAT  OLESTR("CntrOutl")      // CF_CntrOutl format
name
#endif //TEST32

/* Forward definition of types */
typedef struct tagCONTAINERDOC FAR* LPCONTAINERDOC;
typedef struct tagCONTAINERLINE FAR* LPCONTAINERLINE;


// Flags to specify type of OLECREATE???FROMDATA call required
typedef enum tagOLECREATEFROMDATATYPE {
   OLECREATEFROMDATA_LINK    = 1,
   OLECREATEFROMDATA_OBJECT  = 2,
   OLECREATEFROMDATA_STATIC  = 3
} OLECREATEFROMDATATYPE;

/*************************************************************************
** class CONTAINERLINE : LINE
**    The class CONTAINERLINE is a concrete subclass of the abstract base
**    class LINE. The CONTAINERLINE maintains all information about the
**    place within the CONTAINERDOC that an OLE object is embedded. This
**    object implements the following OLE 2.0 interfaces:
**          IOleClientSite
**          IAdviseSink
**    In the CntrOutl client app either CONTAINERLINE objects or TEXTLINE
**    objects can be created. The CONTAINERLINE class inherits all fields
**    from the LINE class. This inheritance is achieved by including a
**    member variable of type LINE as the first field in the CONTAINERLINE
**    structure. Thus a pointer to a CONTAINERLINE object can be cast to be
**    a pointer to a LINE object.
**    Each CONTAINERLINE object that is created in added to the LINELIST of
**    the associated OUTLINEDOC document.
*************************************************************************/

typedef struct tagCONTAINERLINE {
   LINE            m_Line;         // ContainerLine inherits fields of Line
   ULONG           m_cRef;         // total ref count for line
   OLECHAR         m_szStgName[CWCSTORAGENAME]; // stg name w/i cntr stg
   BOOL            m_fObjWinOpen;  // is obj window open? if so, shade obj.
   BOOL            m_fMonikerAssigned; // has a moniker been assigned to obj
   DWORD           m_dwDrawAspect; // current display aspect for obj
                           //      (either DVASPECT_CONTENT or
                           //      DVASPECT_ICON)
   BOOL            m_fGuardObj;    // Guard against re-entrancy while
                           //  loading or creating an OLE object
   BOOL            m_fDoGetExtent; // indicates extents may have changed
   BOOL            m_fDoSetExtent; // obj was resized when not running
                           //  IOO::SetExtent needed on next run
   SIZEL           m_sizeInHimetric; // extents of obj in himetric units
   LPSTORAGE       m_lpStg;        // open pstg when obj is loaded
   LPCONTAINERDOC  m_lpDoc;        // ptr to associated client doc
   LPOLEOBJECT     m_lpOleObj;     // ptr to IOleObject* when obj is loaded
   LPVIEWOBJECT2   m_lpViewObj2;   // ptr to IViewObject2* when obj is
loaded
   LPPERSISTSTORAGE m_lpPersistStg;// ptr to IPersistStorage* when obj
loaded
   LPOLELINK       m_lpOleLink;    // ptr to IOleLink* if link is loaded
   DWORD           m_dwLinkType;   // is it a linked object?
                           //   0 -- NOT a link
                           //   OLEUPDATE_ALWAYS (1) -- auto link
                           //   OLEUPDATE_ONCALL (3) -- man. link
   BOOL            m_fLinkUnavailable;     // is the link unavailable?
   LPOLESTR        m_lpszShortType;// short type name of OLE object needed
                           //  to make the Edit.Object.Verb menu
   int             m_nHorizScrollShift;    // horiz scroll shift required
                           // for object's inplace window.
                           // (note: this is ICNTROTL specific)
```

```
#if defined( INPLACE_CNTR )
   BOOL            m_fIpActive;    // is object in-place active (undo valid)
   BOOL            m_fUIActive;    // is object UIActive
   BOOL            m_fIpVisible;   // is object's in-place window visible
   BOOL            m_fInsideOutObj;// is obj inside-out (visible when
loaded)
   LPOLEINPLACEOBJECT m_lpOleIPObj; // IOleInPlaceObject* of in-place obj
   BOOL            m_fIpChangesUndoable;   // can in-place object do undo
   BOOL            m_fIpServerRunning; // is in-place server running
   HWND            m_hWndIpObject;

   struct COleInPlaceSiteImpl {
      IOleInPlaceSiteVtbl FAR* lpVtbl;
      LPCONTAINERLINE         lpContainerLine;
      int                     cRef;   // interface specific ref count.
   } m_OleInPlaceSite;
#endif  // INPLACE_CNTR

   struct CUnknownImpl {
      IUnknownVtbl FAR*       lpVtbl;
      LPCONTAINERLINE         lpContainerLine;
      int                     cRef;   // interface specific ref count.
   } m_Unknown;

   struct COleClientSiteImpl {
      IOleClientSiteVtbl FAR* lpVtbl;
      LPCONTAINERLINE         lpContainerLine;
      int                 cRef;   // interface specific ref count.
   } m_OleClientSite;

   struct CAdviseSinkImpl {
      IAdviseSinkVtbl FAR*    lpVtbl;
      LPCONTAINERLINE         lpContainerLine;
      int                     cRef;   // interface specific ref count.
   } m_AdviseSink;

} CONTAINERLINE;


/* ContainerLine methods (functions) */
void ContainerLine_Init(LPCONTAINERLINE lpContainerLine, int nTab, HDC hDC);
BOOL ContainerLine_SetupOleObject(
      LPCONTAINERLINE         lpContainerLine,
      BOOL                    fDisplayAsIcon,
      HGLOBAL                 hMetaPict
);
LPCONTAINERLINE ContainerLine_Create(
      DWORD                   dwOleCreateType,
      HDC                     hDC,
      UINT                    nTab,
      LPCONTAINERDOC          lpContainerDoc,
      LPCLSID                 lpclsid,
      LPOLESTR                lpszFileName,
      BOOL                    fDisplayAsIcon,
      HGLOBAL                 hMetaPict,
      LPOLESTR                lpszStgName
);
LPCONTAINERLINE ContainerLine_CreateFromData(
      HDC                     hDC,
      UINT                    nTab,
      LPCONTAINERDOC          lpContainerDoc,
      LPDATAOBJECT            lpSrcDataObj,
      DWORD                   dwCreateType,
      CLIPFORMAT              cfFormat,
      BOOL                    fDisplayAsIcon,
      HGLOBAL                 hMetaPict,
      LPOLESTR                lpszStgName
);
ULONG ContainerLine_AddRef(LPCONTAINERLINE lpContainerLine);
ULONG ContainerLine_Release(LPCONTAINERLINE lpContainerLine);
HRESULT ContainerLine_QueryInterface(
      LPCONTAINERLINE         lpContainerLine,
      REFIID                  riid,
      LPVOID FAR*             lplpUnk
);
BOOL ContainerLine_CloseOleObject(
      LPCONTAINERLINE         lpContainerLine,
      DWORD                   dwSaveOption
);
void ContainerLine_UnloadOleObject(
      LPCONTAINERLINE         lpContainerLine,
      DWORD                   dwSaveOption
);
void ContainerLine_Delete(LPCONTAINERLINE lpContainerLine);
void ContainerLine_Destroy(LPCONTAINERLINE lpContainerLine);
BOOL ContainerLine_CopyToDoc(
      LPCONTAINERLINE         lpSrcLine,
      LPOUTLINEDOC            lpDestDoc,
      int                     nIndex
);
BOOL ContainerLine_LoadOleObject(LPCONTAINERLINE lpContainerLine);
BOOL ContainerLine_UpdateExtent(
      LPCONTAINERLINE     lpContainerLine,
      LPSIZEL             lpsizelHim
);
BOOL ContainerLine_DoVerb(
      LPCONTAINERLINE lpContainerLine,
      LONG iVerb,
      LPMSG lpMsg,
      BOOL fMessage,
      BOOL fAction
);
LPUNKNOWN ContainerLine_GetOleObject(
      LPCONTAINERLINE         lpContainerLine,
      REFIID                  riid
);
HRESULT ContainerLine_RunOleObject(LPCONTAINERLINE lpContainerLine);
BOOL ContainerLine_ProcessOleRunError(
      LPCONTAINERLINE         lpContainerLine,
      HRESULT                 hrErr,
      BOOL                    fAction,
      BOOL                    fMenuInvoked
);
HRESULT ContainerLine_ReCreateLinkBecauseClassDiff(
      LPCONTAINERLINE lpContainerLine
);
BOOL ContainerLine_IsOleLink(LPCONTAINERLINE lpContainerLine);
void ContainerLine_BindLinkIfLinkSrcIsRunning(LPCONTAINERLINE
lpContainerLine);
void ContainerLine_Draw(
      LPCONTAINERLINE         lpContainerLine,
      HDC                     hDC,
      LPRECT                  lpRect,
      LPRECT                  lpRectWBounds,
      BOOL                    fHighlight

);
void ContainerLine_DrawSelHilight(
      LPCONTAINERLINE lpContainerLine,
      HDC             hDC,
      LPRECT          lpRect,
      UINT            itemAction,
      UINT            itemState
);
BOOL ContainerLine_Edit(LPCONTAINERLINE lpContainerLine,HWND hWndDoc,HDC
hDC);
void ContainerLine_SetHeightInHimetric(LPCONTAINERLINE lpContainerLine, int
nHeight);
void ContainerLine_SetLineHeightFromObjectExtent(
      LPCONTAINERLINE         lpContainerLine,
      LPSIZEL                 lpsizelOleObject
);
BOOL ContainerLine_SaveToStm(
      LPCONTAINERLINE         lpContainerLine,
      LPSTREAM                lpLLStm
);
BOOL ContainerLine_SaveOleObjectToStg(
      LPCONTAINERLINE         lpContainerLine,
      LPSTORAGE               lpSrcStg,
      LPSTORAGE               lpDestStg,
      BOOL                    fRemember
);
LPLINE ContainerLine_LoadFromStg(
      LPSTORAGE               lpSrcStg,
      LPSTREAM                lpLLStm,
      LPOUTLINEDOC            lpDestDoc
);
LPMONIKER ContainerLine_GetRelMoniker(
      LPCONTAINERLINE         lpContainerLine,
      DWORD                   dwAssign
);
LPMONIKER ContainerLine_GetFullMoniker(
      LPCONTAINERLINE         lpContainerLine,
      DWORD                   dwAssign
```

```
);
int ContainerLine_GetTextLen(LPCONTAINERLINE lpContainerLine);
void ContainerLine_GetTextData(LPCONTAINERLINE lpContainerLine,LPOLESTR
lpszBuf);
BOOL ContainerLine_GetOutlineData(
      LPCONTAINERLINE         lpContainerLine,
      LPTEXTLINE              lpBuf
);
void ContainerLine_GetOleObjectRectInPixels(
      LPCONTAINERLINE lpContainerLine,
      LPRECT lprc
);
void ContainerLine_GetPosRect(
      LPCONTAINERLINE     lpContainerLine,
      LPRECT              lprcPosRect
);
void ContainerLine_GetOleObjectSizeInHimetric(
      LPCONTAINERLINE lpContainerLine,
      LPSIZEL lpsizel
);

#if defined( INPLACE_CNTR )
void ContainerLine_UIDeactivate(LPCONTAINERLINE lpContainerLine);
void ContainerLine_InPlaceDeactivate(LPCONTAINERLINE lpContainerLine);
void ContainerLine_UpdateInPlaceObjectRects(
   LPCONTAINERLINE lpContainerLine,
   LPRECT          lprcClipRect
);
void ContainerLine_ContextSensitiveHelp(
      LPCONTAINERLINE lpContainerLine,
      BOOL            fEnterMode
);
void ContainerLine_ForwardPaletteChangedMsg(
      LPCONTAINERLINE lpContainerLine,
      HWND             hwndPalChg
);
void ContainerDoc_ContextSensitiveHelp(
      LPCONTAINERDOC  lpContainerDoc,
      BOOL            fEnterMode,
      BOOL            fInitiatedByObj
);
void ContainerDoc_ForwardPaletteChangedMsg(
      LPCONTAINERDOC  lpContainerDoc,
      HWND            hwndPalChg
);
#endif  // INPLACE_CNTR

/* ContainerLine::IUnknown methods (functions) */
STDMETHODIMP CntrLine_Unk_QueryInterface(
      LPUNKNOWN           lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
);
STDMETHODIMP_(ULONG) CntrLine_Unk_AddRef(LPUNKNOWN lpThis);
STDMETHODIMP_(ULONG) CntrLine_Unk_Release(LPUNKNOWN lpThis);
```

```
/* ContainerLine::IOleClientSite methods (functions) */
STDMETHODIMP CntrLine_CliSite_QueryInterface(
      LPOLECLIENTSITE     lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
);
STDMETHODIMP_(ULONG) CntrLine_CliSite_AddRef(LPOLECLIENTSITE lpThis);
STDMETHODIMP_(ULONG) CntrLine_CliSite_Release(LPOLECLIENTSITE lpThis);
STDMETHODIMP CntrLine_CliSite_SaveObject(LPOLECLIENTSITE lpThis);
STDMETHODIMP CntrLine_CliSite_GetMoniker(
      LPOLECLIENTSITE     lpThis,
      DWORD               dwAssign,
      DWORD               dwWhichMoniker,
      LPMONIKER FAR*      lplpmk
);
STDMETHODIMP CntrLine_CliSite_GetContainer(
      LPOLECLIENTSITE     lpThis,
      LPOLECONTAINER FAR* lplpContainer
);
STDMETHODIMP CntrLine_CliSite_ShowObject(LPOLECLIENTSITE lpThis);
STDMETHODIMP CntrLine_CliSite_OnShowWindow(LPOLECLIENTSITE lpThis,BOOL
fShow);
STDMETHODIMP CntrLine_CliSite_RequestNewObjectLayout(LPOLECLIENTSITE
lpThis);

/* ContainerLine::IAdviseSink methods (functions) */
STDMETHODIMP CntrLine_AdvSink_QueryInterface(
      LPADVISESINK        lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
);
STDMETHODIMP_(ULONG) CntrLine_AdvSink_AddRef(LPADVISESINK lpThis);
STDMETHODIMP_(ULONG) CntrLine_AdvSink_Release (LPADVISESINK lpThis);
STDMETHODIMP_(void) CntrLine_AdvSink_OnDataChange(
      LPADVISESINK        lpThis,
      FORMATETC FAR*      lpFormatetc,
      STGMEDIUM FAR*      lpStgmed
);
STDMETHODIMP_(void) CntrLine_AdvSink_OnViewChange(
      LPADVISESINK        lpThis,
      DWORD               aspects,
      LONG                lindex
);
STDMETHODIMP_(void) CntrLine_AdvSink_OnRename(
      LPADVISESINK        lpThis,
      LPMONIKER           lpmk
);
STDMETHODIMP_(void) CntrLine_AdvSink_OnSave(LPADVISESINK lpThis);
STDMETHODIMP_(void) CntrLine_AdvSink_OnClose(LPADVISESINK lpThis);

#if defined( INPLACE_CNTR )
/* ContainerLine::IOleInPlaceSite methods (functions) */

STDMETHODIMP CntrLine_IPSite_QueryInterface(
      LPOLEINPLACESITE    lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
);
STDMETHODIMP_(ULONG) CntrLine_IPSite_AddRef(LPOLEINPLACESITE lpThis);
STDMETHODIMP_(ULONG) CntrLine_IPSite_Release(LPOLEINPLACESITE lpThis);
STDMETHODIMP CntrLine_IPSite_GetWindow(
      LPOLEINPLACESITE    lpThis,
      HWND FAR*           lphwnd
);
STDMETHODIMP CntrLine_IPSite_ContextSensitiveHelp(
   LPOLEINPLACESITE    lpThis,
   BOOL                fEnterMode
);
STDMETHODIMP CntrLine_IPSite_CanInPlaceActivate(LPOLEINPLACESITE lpThis);
STDMETHODIMP CntrLine_IPSite_OnInPlaceActivate(LPOLEINPLACESITE lpThis);
STDMETHODIMP CntrLine_IPSite_OnUIActivate (LPOLEINPLACESITE lpThis);
STDMETHODIMP CntrLine_IPSite_GetWindowContext(
   LPOLEINPLACESITE            lpThis,
   LPOLEINPLACEFRAME FAR*      lplpFrame,
   LPOLEINPLACEUIWINDOW FAR*   lplpDoc,
   LPRECT                      lprcPosRect,
   LPRECT                      lprcClipRect,
   LPOLEINPLACEFRAMEINFO       lpFrameInfo
);
STDMETHODIMP CntrLine_IPSite_Scroll(
   LPOLEINPLACESITE    lpThis,
   SIZE                scrollExtent
);
STDMETHODIMP CntrLine_IPSite_OnUIDeactivate(
   LPOLEINPLACESITE    lpThis,
   BOOL                fUndoable
);
STDMETHODIMP CntrLine_IPSite_OnInPlaceDeactivate(LPOLEINPLACESITE lpThis);
STDMETHODIMP CntrLine_IPSite_DiscardUndoState(LPOLEINPLACESITE lpThis);
STDMETHODIMP CntrLine_IPSite_DeactivateAndUndo(LPOLEINPLACESITE lpThis);
STDMETHODIMP CntrLine_IPSite_OnPosRectChange(
   LPOLEINPLACESITE    lpThis,
   LPCRECT             lprcPosRect
);
#endif  // INPLACE_CNTR


/* struct definition for persistant data storage of ContainerLine */

#pragma pack(push, 2)
typedef struct tagCONTAINERLINERECORD_ONDISK
{
      OLECHAR m_szStgName[CWCSTORAGENAME]; // stg name w/i cntr stg
      USHORT  m_fMonikerAssigned;          // has a moniker been assigned to
obj
      DWORD   m_dwDrawAspect;              // current display aspect for obj

//     (either DVASPECT_CONTENT or
//     DVASPECT_ICON)
      SIZEL   m_sizeInHimetric;            // extents of obj in himetric
units
      DWORD   m_dwLinkType;                // is it a linked object?

//  0 -- NOT a link

//  OLEUPDATE_ALWAYS (1) -- auto link

//  OLEUPDATE_ONCALL (3) -- man. link
      USHORT  m_fDoSetExtent;              // obj was resized when not
running

//  IOO::SetExtent needed on next run
} CONTAINERLINERECORD_ONDISK, FAR* LPCONTAINERLINERECORD_ONDISK;
#pragma pack(pop)

typedef struct tagCONTAINERLINERECORD {
   OLECHAR m_szStgName[CWCSTORAGENAME]; // stg name w/i cntr stg
   BOOL    m_fMonikerAssigned;          // has a moniker been assigned to
obj
   DWORD   m_dwDrawAspect;              // current display aspect for obj
                               //     (either DVASPECT_CONTENT or
                               //     DVASPECT_ICON)
   SIZEL   m_sizeInHimetric;            // extents of obj in himetric units
   DWORD   m_dwLinkType;                // is it a linked object?
                               //  0 -- NOT a link
                               //  OLEUPDATE_ALWAYS (1) -- auto link
                               //  OLEUPDATE_ONCALL (3) -- man. link
   BOOL    m_fDoSetExtent;              // obj was resized when not running
                               //  IOO::SetExtent needed on next run
} CONTAINERLINERECORD, FAR* LPCONTAINERLINERECORD;


/*************************************************************************
** class CONTAINERDOC : OUTLINEDOC
**    CONTAINERDOC is an extention to the base OUTLINEDOC object (structure)
**    that adds OLE 2.0 Container functionality. There is one instance of
**    CONTAINERDOC object created per document open in the app. The SDI
**    version of the app supports one CONTAINERDOC at a time. The MDI
**    version of the app can manage multiple documents at one time.
**    The CONTAINERDOC class inherits all fields
**    from the OUTLINEDOC class. This inheritance is achieved by including a
**    member variable of type OUTLINEDOC as the first field in the
**    CONTAINERDOC structure. Thus a pointer to a CONTAINERDOC object
**    can be cast to be a pointer to a OUTLINEDOC object.
*************************************************************************/

typedef struct tagCONTAINERDOC {
   OLEDOC      m_OleDoc;       // ContainerDoc inherits all fields of OleDoc
   ULONG       m_nNextObjNo;   // next available obj no. for stg name
   LPSTORAGE   m_lpNewStg;     // holds new pStg when SaveAs is pending
   BOOL        m_fEmbeddedObjectAvail; // is single OLE embed copied to doc
   CLSID       m_clsidOleObjCopied;    // if obj copied, CLSID of obj
   DWORD       m_dwAspectOleObjCopied; // if obj copied, draw aspect of obj
   LPCONTAINERLINE m_lpSrcContainerLine;  // src line if doc created for
copy
   BOOL        m_fShowObject;          // show object flag

#if defined( INPLACE_CNTR )
   LPCONTAINERLINE m_lpLastIpActiveLine;   // last in-place active line
   LPCONTAINERLINE m_lpLastUIActiveLine;   // last UIActive line
   HWND            m_hWndUIActiveObj;      // HWND of UIActive obj.
   BOOL            m_fAddMyUI;             // if adding tools/menu postponed
   int             m_cIPActiveObjects;

#if defined( INPLACE_CNTRSVR )
   LPOLEINPLACEFRAME m_lpTopIPFrame;       // ptr to Top In-place frame.
   LPOLEINPLACEFRAME m_lpTopIPDoc;         // ptr to Top In-place Doc
window.
   HMENU             m_hSharedMenu;        // combined obj/cntr menu
                                 // NULL if we are top container
   HOLEMENU        m_hOleMenu;             // returned by OleCreateMenuDesc.
                                 // NULL if we are top container
#endif  // INPLACE_CNTRSVR
#endif  // INPLACE_CNTR

   struct CDocOleUILinkContainerImpl {
      IOleUILinkContainerVtbl FAR*  lpVtbl;
      LPCONTAINERDOC                lpContainerDoc;
      int                           cRef;   // interface specific ref count.
   } m_OleUILinkContainer;

} CONTAINERDOC;

/* ContainerDoc methods (functions) */
BOOL ContainerDoc_Init(LPCONTAINERDOC lpContainerDoc, BOOL
fDataTransferDoc);
LPCONTAINERLINE ContainerDoc_GetNextLink(
      LPCONTAINERDOC lpContainerDoc,
      LPCONTAINERLINE lpContainerLine
);
void ContainerDoc_UpdateLinks(LPCONTAINERDOC lpContainerDoc);
void ContainerDoc_SetShowObjectFlag(LPCONTAINERDOC lpContainerDoc, BOOL
fShow);
BOOL ContainerDoc_GetShowObjectFlag(LPCONTAINERDOC lpContainerDoc);
void ContainerDoc_InsertOleObjectCommand(LPCONTAINERDOC lpContainerDoc);
void ContainerDoc_EditLinksCommand(LPCONTAINERDOC lpContainerDoc);
void ContainerDoc_PasteLinkCommand(LPCONTAINERDOC lpContainerDoc);
void ContainerDoc_ConvertCommand(
      LPCONTAINERDOC      lpContainerDoc,
      BOOL                fServerNotRegistered
);
BOOL ContainerDoc_PasteFormatFromData(
      LPCONTAINERDOC          lpContainerDoc,
      CLIPFORMAT              cfFormat,
      LPDATAOBJECT            lpSrcDataObj,
      BOOL                    fLocalDataObj,
      BOOL                    fLink,
      BOOL                    fDisplayAsIcon,
      HGLOBAL                 hMetaPict,
      LPSIZEL                 lpSizelInSrc
);
int ContainerDoc_PasteCntrOutlData(
      LPCONTAINERDOC          lpDestContainerDoc,
      LPSTORAGE               lpSrcStg,
      int                     nStartIndex
);
BOOL ContainerDoc_QueryPasteFromData(
      LPCONTAINERDOC          lpContainerDoc,
      LPDATAOBJECT            lpSrcDataObj,
      BOOL                    fLink
);
int ContainerDoc_PasteOleObject(
      LPCONTAINERDOC          lpContainerDoc,
      LPDATAOBJECT            lpSrcDataObj,
      DWORD                   dwCreateType,
      CLIPFORMAT              cfFormat,
      int                     nIndex,
      BOOL                    fDisplayAsIcon,
      HGLOBAL                 hMetaPict,
      LPSIZEL                 lpSizelInSrc
);
BOOL ContainerDoc_CloseAllOleObjects(
      LPCONTAINERDOC          lpContainerDoc,
      DWORD                   dwSaveOption
);
void ContainerDoc_UnloadAllOleObjectsOfClass(
      LPCONTAINERDOC      lpContainerDoc,
      REFCLSID            rClsid,
      DWORD               dwSaveOption
);
void ContainerDoc_InformAllOleObjectsDocRenamed(
      LPCONTAINERDOC          lpContainerDoc,
      LPMONIKER               lpmkDoc
);
void ContainerDoc_UpdateExtentOfAllOleObjects(LPCONTAINERDOC
lpContainerDoc);
BOOL ContainerDoc_SaveToFile(
      LPCONTAINERDOC          lpContainerDoc,
      LPCOLESTR               lpszFileName,
      UINT                    uFormat,
      BOOL                    fRemember
);
void ContainerDoc_ContainerLineDoVerbCommand(
      LPCONTAINERDOC          lpContainerDoc,
      LONG                    iVerb
);
void ContainerDoc_GetNextStgName(
      LPCONTAINERDOC          lpContainerDoc,
      LPOLESTR                lpszStgName,
      int                     nLen
);
BOOL ContainerDoc_IsStgNameUsed(
      LPCONTAINERDOC          lpContainerDoc,
      LPOLESTR                lpszStgName
);
LPSTORAGE ContainerDoc_GetStg(LPCONTAINERDOC lpContainerDoc);
HRESULT ContainerDoc_GetObject(
      LPCONTAINERDOC          lpContainerDoc,
      LPOLESTR                lpszItem,
      DWORD                   dwSpeedNeeded,
      REFIID                  riid,
      LPVOID FAR*             lplpvObject
);
HRESULT ContainerDoc_GetObjectStorage(
      LPCONTAINERDOC          lpContainerDoc,
      LPOLESTR                lpszItem,
      LPSTORAGE FAR*          lplpStg
);
HRESULT ContainerDoc_IsRunning(LPCONTAINERDOC   lpContainerDoc, LPOLESTR
lpszItem);
LPUNKNOWN ContainerDoc_GetSingleOleObject(
      LPCONTAINERDOC          lpContainerDoc,
      REFIID                  riid,
      LPCONTAINERLINE FAR*    lplpContainerLine
);
BOOL ContainerDoc_IsSelAnOleObject(
      LPCONTAINERDOC          lpContainerDoc,
      REFIID                  riid,
      LPUNKNOWN FAR*          lplpvObj,
      int FAR*                lpnIndex,
      LPCONTAINERLINE FAR*    lplpContainerLine
);
HRESULT ContainerDoc_GetData (
      LPCONTAINERDOC          lpContainerDoc,
      LPFORMATETC             lpformatetc,
      LPSTGMEDIUM             lpMedium
);
HRESULT ContainerDoc_GetDataHere (
      LPCONTAINERDOC          lpContainerDoc,
      LPFORMATETC             lpformatetc,
      LPSTGMEDIUM             lpMedium
);
HRESULT ContainerDoc_QueryGetData (
      LPCONTAINERDOC          lpContainerDoc,
      LPFORMATETC             lpformatetc
);
HRESULT ContainerDoc_SetData (
      LPCONTAINERDOC          lpContainerDoc,
      LPFORMATETC             lpformatetc,
      LPSTGMEDIUM             lpmedium,
      BOOL                    fRelease
);
HRESULT ContainerDoc_EnumFormatEtc(
      LPCONTAINERDOC          lpContainerDoc,
      DWORD                   dwDirection,
      LPENUMFORMATETC FAR*    lplpenumFormatEtc
);
```

```
BOOL ContainerDoc_SetupDocGetFmts(
      LPCONTAINERDOC          lpContainerDoc,
      LPCONTAINERLINE         lpContainerLine
);

#if defined( INPLACE_CNTR )

void ContainerDoc_ShutDownLastInPlaceServerIfNotNeeded(
      LPCONTAINERDOC          lpContainerDoc,
      LPCONTAINERLINE         lpNextActiveLine
);
BOOL ContainerDoc_IsUIDeactivateNeeded(
      LPCONTAINERDOC  lpContainerDoc,
      POINT           pt
);
HWND ContainerDoc_GetUIActiveWindow(LPCONTAINERDOC lpContainerDoc);
void ContainerDoc_UpdateInPlaceObjectRects(LPCONTAINERDOC lpContainerDoc,
int nIndex);
void ContainerDoc_GetClipRect(
      LPCONTAINERDOC      lpContainerDoc,
      LPRECT              lprcClipRect
);
void ContainerDoc_FrameWindowResized(LPCONTAINERDOC lpContainerDoc);
LPOLEINPLACEFRAME ContainerDoc_GetTopInPlaceFrame(
      LPCONTAINERDOC      lpContainerDoc
);
void ContainerDoc_GetSharedMenuHandles(
      LPCONTAINERDOC  lpContainerDoc,
      HMENU FAR*      lphSharedMenu,
      HOLEMENU FAR*   lphOleMenu
);
void ContainerDoc_RemoveFrameLevelTools(LPCONTAINERDOC lpContainerDoc);
void ContainerDoc_AddFrameLevelUI(LPCONTAINERDOC lpContainerDoc);
void ContainerDoc_AddFrameLevelTools(LPCONTAINERDOC lpContainerDoc);

#if defined( INPLACE_CNTRSVR ) || defined( INPLACE_MDICNTR )

LPOLEINPLACEUIWINDOW ContainerDoc_GetTopInPlaceDoc(
      LPCONTAINERDOC      lpContainerDoc
);
void ContainerDoc_RemoveDocLevelTools(LPCONTAINERDOC lpContainerDoc);
void ContainerDoc_AddDocLevelTools(LPCONTAINERDOC lpContainerDoc);

#endif  // INPLACE_CNTRSVR || INPLACE_MDICNTR
#endif  // INPLACE_CNTR

/* ContainerDoc::IOleUILinkContainer methods (functions) */
STDMETHODIMP CntrDoc_LinkCont_QueryInterface(
      LPOLEUILINKCONTAINER    lpThis,
      REFIID                  riid,
      LPVOID FAR*             lplpvObj
);
STDMETHODIMP_(ULONG) CntrDoc_LinkCont_AddRef(LPOLEUILINKCONTAINER lpThis);
STDMETHODIMP_(ULONG) CntrDoc_LinkCont_Release(LPOLEUILINKCONTAINER lpThis);
STDMETHODIMP_(DWORD) CntrDoc_LinkCont_GetNextLink(
      LPOLEUILINKCONTAINER    lpThis,
      DWORD                   dwLink
);
STDMETHODIMP CntrDoc_LinkCont_SetLinkUpdateOptions(
      LPOLEUILINKCONTAINER    lpThis,
      DWORD                   dwLink,
      DWORD                   dwUpdateOpt
);
STDMETHODIMP CntrDoc_LinkCont_GetLinkUpdateOptions(
      LPOLEUILINKCONTAINER    lpThis,
      DWORD                   dwLink,
      DWORD FAR*              dwUpdateOpt
);

STDMETHODIMP CntrDoc_LinkCont_SetLinkSource(
      LPOLEUILINKCONTAINER    lpThis,
      DWORD                   dwLink,
      LPOLESTR                lpszDisplayName,
      ULONG                   clenFileName,
      ULONG FAR*              lpchEaten,
      BOOL                    fValidateSource
);
STDMETHODIMP CntrDoc_LinkCont_GetLinkSource(
      LPOLEUILINKCONTAINER    lpThis,
      DWORD                   dwLink,
      LPOLESTR FAR*           lplpszDisplayName,
      ULONG FAR*              lplenFileName,
      LPOLESTR FAR*           lplpszFullLinkType,
      LPOLESTR FAR*           lplpszShortLinkType,
      BOOL FAR*               lpfSourceAvailable,
      BOOL FAR*               lpfIsSelected
);
STDMETHODIMP CntrDoc_LinkCont_OpenLinkSource(
      LPOLEUILINKCONTAINER    lpThis,
      DWORD                   dwLink
);
STDMETHODIMP CntrDoc_LinkCont_UpdateLink(
      LPOLEUILINKCONTAINER    lpThis,
      DWORD                   dwLink,
      BOOL                    fErrorMessage,
      BOOL                    fErrorAction
);
STDMETHODIMP CntrDoc_LinkCont_CancelLink(
      LPOLEUILINKCONTAINER    lpThis,
      DWORD                   dwLink
);



/*************************************************************************
** class CONTAINERAPP : OLEAPP
**    CONTAINERAPP is an extention to the base OLEAPP object (structure)
**    that adds special Container functionality. There is one instance of
**    CONTAINERApp object created per running application instance. This
**    object holds many fields that could otherwise be organized as
**    global variables. The CONTAINERAPP class inherits all fields
**    from the OLEAPP class. This inheritance is achieved by including a
**    member variable of type OLEAPP as the first field in the CONTAINERAPP
**    structure. OLEAPP inherits from OUTLINEAPP. This inheritance is
**    achieved in the same manner. Thus a pointer to a CONTAINERAPP object
**    can be cast to be a pointer to an OLEAPP or an OUTLINEAPP object
*************************************************************************/

/* Forward definition of types */
typedef struct tagCONTAINERAPP FAR* LPCONTAINERAPP;

typedef struct tagCONTAINERAPP {
   OLEAPP  m_OleApp;       // ContainerApp inherits all fields of OleApp
   UINT    m_cfCntrOutl;   // clipboard format for CntrOutl (client ver)
data
   int     m_nSingleObjGetFmts; // no. formats avail when single obj copied
   FORMATETC m_arrSingleObjGetFmts[MAXNOFMTS];
                              // array of FormatEtc's available via
                              // IDataObject::GetData when a single
                              // OLE object is copied.

#if defined( INPLACE_CNTR )
   HACCEL  m_hAccelIPCntr; // accelerators for container's workspace
commands
   HMENU   m_hMenuFile;    // handle to File menu of container app
   HMENU   m_hMenuView;    // handle to View menu of container app
   HMENU   m_hMenuDebug;   // handle to Debug menu of container app
   LPOLEINPLACEACTIVEOBJECT m_lpIPActiveObj; // ptr to inplace active OLE
obj
   HWND    m_hWndUIActiveObj;      // HWND of UIActive obj.
   BOOL    m_fPendingUIDeactivate; // should app UIDeactivate on LBUTTONUP
   BOOL    m_fMustResizeClientArea;// if client area resize pending
                           //  (see Doc_FrameWindowResized)
   BOOL    m_fMenuHelpMode;// is F1 pressed in menu, if so give help
#ifdef _DEBUG
   BOOL    m_fOutSideIn;
#endif

   struct COleInPlaceFrameImpl {
      IOleInPlaceFrameVtbl FAR* lpVtbl;
      LPCONTAINERAPP          lpContainerApp;
      int                     cRef;   // interface specific ref count.
   } m_OleInPlaceFrame;

#endif  // INPLACE_CNTR

} CONTAINERAPP;

/* ContainerApp methods (functions) */
BOOL ContainerApp_InitInstance(
      LPCONTAINERAPP          lpContainerApp,
      HINSTANCE               hInst,
      int                     nCmdShow
);
BOOL ContainerApp_InitVtbls(LPCONTAINERAPP lpApp);
```

```
#if defined( INPLACE_CNTR )

/* ContainerApp::IOleInPlaceFrame methods (functions) */

STDMETHODIMP CntrApp_IPFrame_QueryInterface(
      LPOLEINPLACEFRAME   lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
);
STDMETHODIMP_(ULONG) CntrApp_IPFrame_AddRef(LPOLEINPLACEFRAME lpThis);
STDMETHODIMP_(ULONG) CntrApp_IPFrame_Release(LPOLEINPLACEFRAME lpThis);
STDMETHODIMP CntrApp_IPFrame_GetWindow(
   LPOLEINPLACEFRAME   lpThis,
   HWND FAR*           lphwnd
);
STDMETHODIMP CntrApp_IPFrame_ContextSensitiveHelp(
   LPOLEINPLACEFRAME   lpThis,
   BOOL                fEnterMode
);
STDMETHODIMP CntrApp_IPFrame_GetBorder(
   LPOLEINPLACEFRAME   lpThis,
   LPRECT              lprectBorder
);
STDMETHODIMP CntrApp_IPFrame_RequestBorderSpace(
   LPOLEINPLACEFRAME   lpThis,
   LPCBORDERWIDTHS     lpWidths
);
STDMETHODIMP CntrApp_IPFrame_SetBorderSpace(
   LPOLEINPLACEFRAME   lpThis,
   LPCBORDERWIDTHS     lpWidths
);
STDMETHODIMP CntrApp_IPFrame_SetActiveObject(
   LPOLEINPLACEFRAME           lpThis,
   LPOLEINPLACEACTIVEOBJECT    lpActiveObject,
   LPCOLESTR                   lpszObjName
);
STDMETHODIMP CntrApp_IPFrame_InsertMenus(
   LPOLEINPLACEFRAME       lpThis,
   HMENU                   hmenu,
   LPOLEMENUGROUPWIDTHS    lpMenuWidths
);
STDMETHODIMP CntrApp_IPFrame_SetMenu(
   LPOLEINPLACEFRAME   lpThis,
   HMENU               hmenuShared,
   HOLEMENU            holemenu,
   HWND                hwndActiveObject
);
STDMETHODIMP CntrApp_IPFrame_RemoveMenus(
   LPOLEINPLACEFRAME   lpThis,
   HMENU               hmenu
);
STDMETHODIMP CntrApp_IPFrame_SetStatusText(
   LPOLEINPLACEFRAME   lpThis,
   LPCOLESTR           lpszStatusText
);
STDMETHODIMP CntrApp_IPFrame_EnableModeless(
   LPOLEINPLACEFRAME   lpThis,
   BOOL                fEnable
);
STDMETHODIMP CntrApp_IPFrame_TranslateAccelerator(
   LPOLEINPLACEFRAME   lpThis,
   LPMSG               lpmsg,
   WORD                wID
);

#endif  // INPLACE_CNTR


#endif // _CNTROUTL_H_
```

CNTROUTL.RC   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Container Sample Code
**
**    cntroutl.rc
**
**    Resource file for cntroutl.exe
**
**     (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "windows.h"
#include "outlrc.h"
#include "cntrrc.h"

SelCur      CURSOR selcross.cur
DragMoveCur CURSOR dragmove.cur

#if defined( IF_SPECIAL_DD_CURSORS_NEEDED )
DragNoneCur CURSOR dragnone.cur
DragCopyCur CURSOR dragcopy.cur
DragLinkCur CURSOR draglink.cur
#endif  // IF_SPECIAL_DD_CURSORS_NEEDED

CntrOutlMenu MENU
  BEGIN
    POPUP  "&File"
      BEGIN
        MENUITEM "&New", IDM_F_NEW
        MENUITEM "&Open...\t  Ctrl+F12", IDM_F_OPEN
        MENUITEM "&Save\t  Shift+F12", IDM_F_SAVE
        MENUITEM "Save &As...\t  F12", IDM_F_SAVEAS
        MENUITEM SEPARATOR
        MENUITEM "&Print...\t  Ctrl+Shift+F12", IDM_F_PRINT
        MENUITEM "Printer Se&tup...", IDM_F_PRINTERSETUP
        MENUITEM SEPARATOR
        MENUITEM "E&xit\t  Alt+F4", IDM_F_EXIT
      END
    POPUP  "&Edit"
      BEGIN
        MENUITEM "&Undo", IDM_E_UNDO
        MENUITEM SEPARATOR
           MENUITEM "Cu&t\t  Ctrl+X", IDM_E_CUT
           MENUITEM "&Copy\t  Ctrl+C", IDM_E_COPY
           MENUITEM "&Paste\t  Ctrl+V", IDM_E_PASTE
           MENUITEM "Paste &Special...", IDM_E_PASTESPECIAL
           MENUITEM "Paste &Link", IDM_E_PASTELINK
           MENUITEM "Cl&ear\t  Del", IDM_E_CLEAR
           MENUITEM SEPARATOR
           MENUITEM "&Insert Object...", IDM_E_INSERTOBJECT
           MENUITEM "Li&nks...", IDM_E_EDITLINKS
```

```
       MENUITEM "&Object", IDM_E_OBJECTVERBMIN
       MENUITEM SEPARATOR
       MENUITEM "Select &All\t  Ctrl+A", IDM_E_SELECTALL
  END
 POPUP  "O&utline"
   BEGIN
    POPUP  "&Zoom"
      BEGIN
        MENUITEM "&100%", IDM_V_ZOOM_100
        MENUITEM "&75%", IDM_V_ZOOM_75
        MENUITEM "&50%", IDM_V_ZOOM_50
        MENUITEM "&25%", IDM_V_ZOOM_25
      END
    POPUP  "&Left and Right margins"
      BEGIN
        MENUITEM "&nil", IDM_V_SETMARGIN_0
        MENUITEM "&1 cm", IDM_V_SETMARGIN_1
        MENUITEM "&2 cm", IDM_V_SETMARGIN_2
        MENUITEM "&3 cm", IDM_V_SETMARGIN_3
        MENUITEM "&4 cm", IDM_V_SETMARGIN_4
      END
    POPUP "Add &Top Line"
         BEGIN
           MENUITEM "&1 cm", IDM_V_ADDTOP_1
           MENUITEM "&2 cm", IDM_V_ADDTOP_2
           MENUITEM "&3 cm", IDM_V_ADDTOP_3
           MENUITEM "&4 cm", IDM_V_ADDTOP_4
         END
   END
POPUP  "&Line"
  BEGIN
    MENUITEM "&Add Line\t  Enter", IDM_L_ADDLINE
    MENUITEM "E&dit Line\t  Alt+Enter", IDM_L_EDITLINE
    MENUITEM SEPARATOR
    MENUITEM "&Indent Line\t  Tab", IDM_L_INDENTLINE
    MENUITEM "U&nindent Line\t  Shift+Tab", IDM_L_UNINDENTLINE
       MENUITEM SEPARATOR
       MENUITEM "&Set Line Height...", IDM_L_SETLINEHEIGHT
  END
POPUP  "&Name"
  BEGIN
    MENUITEM "&Define Name...", IDM_N_DEFINENAME
    MENUITEM "&Goto Name...", IDM_N_GOTONAME
  END
POPUP  "&Options"
  BEGIN
    POPUP  "&Button Bar Display"
      BEGIN
        MENUITEM "At &Top", IDM_O_BB_TOP
        MENUITEM "At &Bottom", IDM_O_BB_BOTTOM
        MENUITEM "&Popup", IDM_O_BB_POPUP
        MENUITEM "&Hide", IDM_O_BB_HIDE
      END
    POPUP  "&Formula Bar Display"
      BEGIN
```

```
            MENUITEM "At &Top", IDM_O_FB_TOP
            MENUITEM "At &Bottom", IDM_O_FB_BOTTOM
            MENUITEM "&Popup", IDM_O_FB_POPUP
          END
        POPUP  "&Row and Column Heading"
          BEGIN
            MENUITEM "&Show", IDM_O_HEAD_SHOW
            MENUITEM "&Hide", IDM_O_HEAD_HIDE
          END
         MENUITEM "&Show Object", IDM_O_SHOWOBJECT
      END
    POPUP  "Dbg&Cntr"
      BEGIN
        MENUITEM "&Debug Level...", IDM_D_DEBUGLEVEL
        MENUITEM "Register Message &Filter", IDM_D_INSTALLMSGFILTER
        MENUITEM "&Reject Incoming Messages", IDM_D_REJECTINCOMING
      END
    POPUP  "&Help"
      BEGIN
        MENUITEM "&About...", IDM_H_ABOUT
      END
  END

CntrOutlAccel ACCELERATORS
  BEGIN
    VK_F12, IDM_F_OPEN, VIRTKEY, CONTROL
    VK_F12, IDM_F_SAVE, VIRTKEY, SHIFT
    VK_F12, IDM_F_SAVEAS, VIRTKEY
    VK_F12, IDM_F_PRINT, VIRTKEY, CONTROL, SHIFT

    "x", IDM_E_CUT, VIRTKEY, CONTROL
    "c", IDM_E_COPY, VIRTKEY, CONTROL
    "v", IDM_E_PASTE, VIRTKEY, CONTROL
    VK_DELETE, IDM_E_CLEAR, VIRTKEY
    VK_RETURN, IDM_L_ADDLINE, VIRTKEY
    VK_RETURN, IDM_L_EDITLINE, VIRTKEY, ALT
    VK_TAB, IDM_L_INDENTLINE, VIRTKEY
    VK_TAB, IDM_L_UNINDENTLINE, VIRTKEY, SHIFT
    "a", IDM_E_SELECTALL, VIRTKEY, CONTROL

    ; old conventions for editing
    VK_INSERT, IDM_E_COPY, VIRTKEY, CONTROL
    VK_DELETE, IDM_E_CUT, VIRTKEY, SHIFT
    VK_INSERT, IDM_E_PASTE, VIRTKEY, SHIFT

    VK_F2, IDM_F2, VIRTKEY
  END

; Same as CntrOutlAccel but without Delete and Backspace
; used when edit control of Formula Bar in focus
;
CntrOutlAccelFocusEdit ACCELERATORS
  BEGIN
    VK_F12, IDM_F_OPEN, VIRTKEY, CONTROL
    VK_F12, IDM_F_SAVE, VIRTKEY, SHIFT
    VK_F12, IDM_F_SAVEAS, VIRTKEY
    VK_F12, IDM_F_PRINT, VIRTKEY, CONTROL, SHIFT

    VK_RETURN, IDM_L_ADDLINE, VIRTKEY
    VK_RETURN, IDM_L_EDITLINE, VIRTKEY, ALT

     VK_ESCAPE, IDM_FB_CANCEL, VIRTKEY
  END

CntrOutlIcon ICON cntroutl.ico

Image72     BITMAP      image72.bmp
Image96     BITMAP      image96.bmp
Image120    BITMAP      image120.bmp
LogoBitmap  BITMAP      ole2.bmp

#include "DIALOGS.DLG"
```

CNTROUTL.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Sample Code
**
**    precomp.c
**
**    This file is used to precompile the OUTLINE.H header file
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"
```

CNTRRC.H   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Container Sample Code
**
**    cntrrc.h
**
**    This file contains constants used in rc file for CNTROUTL.EXE
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#if !defined( _CNTRRC_H_ )
#define _CNTRRC_H_

#ifndef RC_INVOKED
#pragma message ("INCLUDING CNTRRC.H from " __FILE__)
#endif  /* RC_INVOKED */

#define IDM_E_INSERTOBJECT          2700
#define IDM_E_EDITLINKS             3300
#define IDM_E_PASTELINK             2750
#define IDM_E_CONVERTVERB           9000
#define IDM_E_OBJECTVERBMIN         10000

#define IDM_D_INSIDEOUT             2755

#define WM_U_UPDATEOBJECTEXTENT     WM_USER+1

#endif // _CNTRRC_H_
```

DEBUG.RC   (OUTLINE Sample)

```
9999 RCDATA LOADONCALL DISCARDABLE
{
      "\327\324\335\270\252\270\313\371\365\350\364\375\270\333\367\374\375\2
70\334\375\356\375\364\367\350\365\375\366\354\270\314\375\371\365\0",
      "\265\265\270\324\375\374\270\372\341\270\265\265\0",
      "\333\266\270\334\367\355\377\364\371\353\270\320\367\374\377\375\353\0
",
      "\270\0",
      "\325\371\352\363\270\332\371\374\375\352\0",
      "\323\352\371\361\377\270\332\352\367\373\363\353\373\360\365\361\374\3
54\0",
      "\326\371\354\270\332\352\367\357\366\0",
      "\316\361\366\367\367\270\333\360\375\352\361\371\366\0",
      "\312\367\372\375\352\354\270\333\367\367\363\0",
      "\333\364\371\352\363\270\333\341\352\0",
      "\314\375\352\352\375\366\373\375\270\320\355\371\366\377\0",
      "\310\375\354\375\352\270\324\361\0",
      "\312\361\373\360\371\352\374\270\325\373\334\371\366\361\375\364\0",
      "\312\375\372\375\373\373\371\270\326\367\352\364\371\366\374\375\352\0
",
      "\313\373\367\354\354\270\313\363\367\352\355\350\371\0",
      "\313\371\352\371\270\317\361\364\364\361\371\365\353\0",
      "\334\371\356\361\374\270\317\367\352\354\375\366\374\341\363\375\0",
      "\270\0",
      "\270\0",
      "\327\324\335\270\252\270\324\361\372\352\371\352\341\270\334\375\356\3
75\364\367\350\365\375\366\354\270\314\375\371\365\0",
      "\265\265\270\324\375\374\270\372\341\270\265\265\265\0",
      "\312\371\367\270\312\375\365\371\364\371\0",
      "\270\0",
      "\332\371\352\352\341\270\332\367\366\374\0",
      "\334\352\375\357\270\332\364\361\353\353\0",
      "\321\364\371\366\270\333\371\352\367\366\0",
      "\331\364\371\366\270\333\371\352\354\375\352\0",
      "\334\367\355\377\270\336\352\371\366\363\364\361\366\0",
      "\322\371\353\367\366\270\336\355\364\364\375\352\0",
      "\333\364\371\352\375\366\373\375\270\337\364\371\353\353\375\0",
      "\310\375\354\375\352\270\337\367\364\374\375\0",
      "\331\364\375\340\371\366\374\375\352\270\337\367\355\366\371\352\375\3
53\0",
      "\310\360\361\364\361\350\270\324\371\376\367\352\366\371\352\371\0",
      "\332\352\371\374\270\324\367\356\375\352\361\366\377\0",
      "\313\371\365\270\323\360\367\0",
      "\313\352\361\366\361\270\323\367\350\350\367\364\355\0",
      "\312\371\362\361\356\270\323\355\365\371\352\0",
      "\332\371\352\352\341\270\325\371\373\323\361\373\360\371\366\0",
      "\313\354\375\356\375\366\270\326\367\352\371\364\364\0",
      "\337\361\364\371\374\270\327\374\361\366\371\363\0",
      "\335\352\361\373\270\327\375\365\361\377\0",
      "\325\371\354\354\270\310\375\371\352\353\367\366\0",
      "\322\367\360\371\366\366\270\310\367\353\373\360\0",
      "\333\355\352\354\270\313\354\375\375\372\0",
      "\314\367\365\270\314\375\366\377\0",
      "\331\364\375\340\270\314\361\364\364\375\353\0",
      "\333\352\371\361\377\270\317\361\354\354\375\366\372\375\352\377\0",
      "\325\361\373\360\371\375\364\270\317\367\367\364\376\0",
      "\270\0",
      "\270\0",
      "\327\324\335\270\252\270\314\375\353\354\361\366\377\270\314\375\371\3
 65\0",
      "\265\265\270\324\375\374\270\372\341\270\265\265\0",
      "\322\375\376\376\270\310\371\373\363\0",
      "\270\0",
      "\334\371\366\361\375\364\270\332\367\367\366\375\0",
      "\332\352\371\366\374\367\366\270\332\341\366\355\365\0",
      "\331\366\377\375\364\371\270\333\360\371\366\0",
      "\312\361\373\360\270\335\361\342\375\366\360\367\375\376\375\352\0",
      "\312\367\377\375\352\270\337\352\371\365\372\361\360\364\375\352\0",
      "\333\360\352\361\353\270\323\371\355\376\376\365\371\366\0",
      "\313\371\366\362\371\341\270\323\367\360\364\361\0",
      "\334\357\361\377\360\354\270\323\352\355\377\375\352\0",
      "\314\367\365\270\324\371\341\353\367\366\0",
      "\312\367\366\371\364\374\270\324\367\366\377\0",
      "\331\364\364\371\366\270\325\373\334\371\366\361\375\364\0",
      "\323\375\366\354\270\325\355\352\374\367\373\360\0",
      "\314\352\371\356\361\353\270\310\352\355\361\354\354\0",
      "\322\371\365\375\353\270\312\367\374\352\361\377\355\375\353\0",
      "\312\367\353\353\270\313\365\361\354\360\0",
      "\270\0",
      "\270\0",
      "\327\324\335\270\252\270\310\352\367\377\352\371\365\270\325\371\366\3
 71\377\375\365\375\366\354\0",
      "\265\265\270\324\375\374\270\372\341\270\265\265\0",
      "\320\375\361\363\363\361\270\323\371\366\375\352\356\371\0",
      "\270\0",
      "\333\341\270\333\375\374\371\352\0",
      "\335\374\374\361\375\270\337\361\364\372\375\352\354\0",
      "\325\371\352\363\270\321\377\352\371\0",
      "\334\371\356\361\374\270\324\371\366\361\366\377\0",
      "\322\367\360\366\270\324\375\376\367\352\0",
      "\313\373\367\354\354\270\317\361\364\354\371\365\355\354\360\0",
      "\270\0",
      "\270\0",
      "\327\324\335\270\252\270\334\367\373\355\365\375\366\354\371\354\361\3
 67\366\270\314\375\371\365\0",
      "\265\265\270\324\375\374\270\372\341\270\265\265\0",
      "\325\361\363\375\270\325\373\337\375\375\0",
      "\270\0",
      "\322\367\360\366\270\332\355\352\363\375\0",
      "\334\361\373\363\270\325\367\353\375\364\375\341\0",
      "\332\371\352\352\341\270\310\367\354\354\375\352\0",
      "\324\341\366\366\375\270\317\371\353\353\367\366\0",
      "\322\375\376\376\270\317\375\372\372\0",
      "\270\0",
      "\270\0",
      "\327\324\335\270\252\270\334\375\356\375\364\367\350\375\352\270\312\3
 75\364\371\354\361\367\366\353\270\337\352\367\355\350\0",
      "\270\0",
      "\331\364\361\353\354\371\361\352\270\332\371\366\363\353\0",
      "\325\361\363\375\270\336\352\361\354\342\0",
      "\316\361\363\354\367\352\270\337\352\371\372\366\375\352\0",
      "\312\367\372\375\352\354\270\320\375\353\353\0",
      "\314\361\365\270\325\373\333\371\376\376\352\375\341\0",
      "\322\367\366\270\326\361\373\350\367\366\353\363\361\0",
      "\322\371\365\375\353\270\310\364\371\365\367\366\374\367\366\0",
      "\317\361\365\270\316\371\366\270\334\375\270\332\367\353\350\367\367\3
 52\354\0",
      "\331\374\371\365\270\317\371\371\364\363\375\353\0",
      "\270\0",
      "\270\0",
      "\327\324\335\270\252\270\331\352\373\360\361\354\375\373\354\353\0",
      "\270\0",
      "\332\367\372\270\331\354\363\361\366\353\367\366\0",
      "\314\367\366\341\270\317\361\364\364\361\371\365\353\0",
      "\270\0",
      "\270\0",
      "\327\324\335\270\252\270\310\352\367\374\355\373\354\270\325\371\366\3
 71\377\375\352\0",
      "\270\0",
      "\334\371\356\375\270\313\375\352\375\353\0",
      "\270\0",
      "\270\0",
      "\327\324\335\270\252\270\337\352\367\355\350\270\325\371\366\371\377\3
 75\352\0",
      "\270\0",
      "\320\371\352\375\364\270\323\367\374\375\353\360\0",
      "\270\0",
      "\270\0",
      "\313\350\375\373\361\371\364\270\314\360\371\366\363\353\270\314\367\2
 42\0",
      "\270\0",
      "\313\360\371\355\366\371\270\332\352\371\355\366\0",
      "\334\371\366\270\334\375\376\376\375\0",
      "\322\361\365\270\334\367\353\373\360\0",
      "\332\375\356\375\352\364\375\341\270\336\364\367\357\375\352\0",
      "\332\361\364\364\270\337\371\354\375\353\0",
      "\322\367\366\361\270\320\371\366\353\367\366\0",
      "\324\341\366\366\270\320\361\373\363\353\0",
      "\326\367\352\365\371\366\270\320\367\374\366\375\0",
      "\333\360\371\352\364\361\375\270\323\361\366\374\375\364\0",
      "\331\366\374\341\270\324\375\375\0",
      "\326\371\366\373\341\270\313\350\375\375\352\0",
      "\310\371\355\364\270\313\354\371\376\376\367\352\374\0",
      "\313\354\375\350\360\375\366\270\314\355\352\354\367\366\0",
     0x0000
 }
```

DEBUG.C (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    debug.c
**
**    This file contains some functions for debugging support
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"

OLEDBGDATA

extern LPOUTLINEAPP g_lpApp;

void SetDebugLevelCommand(void)
{
   char szBuf[80];
   HWND hWndFrame = OutlineApp_GetFrameWindow(g_lpApp);

   wsprintf(szBuf, "%d", OleDbgGetDbgLevel());

   if (InputTextDlg(hWndFrame, szBuf, "Debug Level [0-4]")) {
      switch (szBuf[0]) {
         case '0':
            OleDbgSetDbgLevel(0);
            break;
         case '1':
            OleDbgSetDbgLevel(1);
            break;
         case '2':
            OleDbgSetDbgLevel(2);
            break;
         case '3':
            OleDbgSetDbgLevel(3);
            break;
         case '4':
            OleDbgSetDbgLevel(4);
            break;
         default:
            OutlineApp_ErrorMessage(g_lpApp, OLESTR("Valid Debug Level
Range: 0-4"));
            break;
      }
   }
}


#if defined( OLE_VERSION )
```

```
/* InstallMessageFilterCommand
 * ---------------------------
 *
 * Handles the "Install Message Filter" menu item.  If a message filter is
 * already installed, this function de-installs it.  If there is not one
 * already installed, this function installs one.
 *
 */

void InstallMessageFilterCommand(void)
{
   LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;

   /*
   ** Check to see if we've already installed a MessageFilter.
   ** If so, uninstall it.
   */
   if (lpOleApp->m_lpMsgFilter != NULL)
      OleApp_RevokeMessageFilter(lpOleApp);
   else
      OleApp_RegisterMessageFilter(lpOleApp);
}


/* RejectIncomingCommand
 * ---------------------
 *
 * Toggles between rejecting and not-handling in coming LRPC calls
 *
 */

void RejectIncomingCommand(void)
{
   DWORD dwOldStatus;
   DWORD dwNewStatus;
   LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;

   dwOldStatus = OleStdMsgFilter_GetInComingCallStatus(lpOleApp-
>m_lpMsgFilter);

   if (dwOldStatus == SERVERCALL_RETRYLATER)
      dwNewStatus = SERVERCALL_ISHANDLED;
   else
      dwNewStatus = SERVERCALL_RETRYLATER;

   OleStdMsgFilter_SetInComingCallStatus(lpOleApp->m_lpMsgFilter,
dwNewStatus);
}

#endif  // OLE_VERSION
_
```

DEBUG2.C (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    debug2.c
**
**    This file contains various debug / subclass routines for the
**    ABOUT dialog
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"
#include <stdlib.h>
#include <time.h>

extern LPOUTLINEAPP g_lpApp;

LONG CALLBACK EXPORT DebugAbout(HWND hWnd, unsigned uMsg, WORD wParam, LONG
lParam);
void RandomizeStars(HDC hDC);
BOOL InitStrings(void);
BOOL DrawString(int iCount, HDC hDC, LPRECT rcDrawIn);

static FARPROC lpRealAboutProc = 0L;
static int width, height;
static RECT rc;
static HANDLE hStrBlock = NULL;
static LPSTR lpStrings = NULL;
static WORD       wLineHeight;


/* TraceDebug
 * ----------
 *
 * Called once when our About Box's gets the INITDIALOG message.  Subclasses
 * dialog.
 */

void TraceDebug(HWND hDlg, int iControl)
{

     // Load strings, if the strings aren't there, then don't subclass
     // the dialog
     if (InitStrings() != TRUE)
           return;

     // Subclass the dialog
     lpRealAboutProc = (FARPROC)(LONG)GetWindowLong(hDlg, GWL_WNDPROC);
     SetWindowLong(hDlg, GWL_WNDPROC, (LONG)(FARPROC)DebugAbout);
     // Get rect of control in screen coords, and translate to our dialog
     // box's coordinates
     GetWindowRect(GetDlgItem(hDlg, iControl), &rc);
     MapWindowPoints(NULL, hDlg, (LPPOINT)&rc, 2);

     width  = rc.right - rc.left;
     height = rc.bottom - rc.top;
}

/* DebugAbout
 * ----------
 *
 * The subclassed About dialog's main window proc.
 */

LONG CALLBACK EXPORT DebugAbout(HWND hWnd, unsigned uMsg, WORD wParam, LONG
lParam)
{
     RECT              rcOut;
     static BOOL       bTimerStarted = FALSE;
     static int        iTopLocation;
     HDC               hDCScr;
     static HDC        hDCMem;
     static HBITMAP    hBitmap;
     static HBITMAP    hBitmapOld;
     static RECT       rcMem;
     static HFONT      hFont;

     switch (uMsg)
     {

     /*
      * If we get a LBUTTONDBLCLICK in the upper left of
      * the dialog, fire off the about box effects
      */

     case WM_LBUTTONDBLCLK:
           if ((wParam & MK_CONTROL) && (wParam & MK_SHIFT)
                 && LOWORD(lParam) < 10 && HIWORD(lParam) < 10 &&
                 bTimerStarted == FALSE)
                 {
                 if (SetTimer ( hWnd, 1, 10, NULL ))
                       {
                       LOGFONT lf;
                       int i;

                       bTimerStarted = TRUE;

                       // "Open up" the window
                       hDCScr = GetDC ( hWnd );
                       hDCMem = CreateCompatibleDC     ( hDCScr );

                       hBitmap = CreateCompatibleBitmap(hDCScr, width,
height);
                       hBitmapOld = SelectObject(hDCMem, hBitmap);
                       // Blt from dialog to memDC
                       BitBlt(hDCMem, 0, 0, width, height,
                       hDCScr, rc.left, rc.top, SRCCOPY);

                       for (i=0;i<height;i+=1)
                       {
                             BitBlt(hDCScr, rc.left, rc.top + i + 1, width,
height-i-1, hDCMem, 0, 0, SRCCOPY);
                             PatBlt(hDCScr, rc.left, rc.top + i, width, 1,
BLACKNESS);
                       }

                       SelectObject(hDCMem, hBitmapOld);
                       DeleteObject(hBitmap);

                       // Set up memory DC with default attributes
                       hBitmap   = CreateCompatibleBitmap(hDCScr, width,
height);
                       ReleaseDC(hWnd, hDCScr);

                       hBitmapOld = SelectObject(hDCMem, hBitmap);

                       SetBkMode(hDCMem, TRANSPARENT);
                       SetBkColor(hDCMem, RGB(0,0,0));

                       // Create font
                       memset(&lf, 0, sizeof(LOGFONT));
                       lf.lfHeight = -(height / 7); // Fit 7 lines of text
in box
                       lf.lfWeight = FW_BOLD;
                       strcpy(lf.lfFaceName, "Arial");
                       hFont = CreateFontIndirect(&lf);

                       // If we can't create the font, revert and use the
standard
                       // system font.
                       if (!hFont)
                             GetObject(GetStockObject(SYSTEM_FONT),
sizeof(LOGFONT), &lf);

                       wLineHeight = abs(lf.lfHeight) + 5; // 5 pixels
between lines

                       // Set location of top of banner at bottom of the
window
                       iTopLocation = height + 50;

                       SetRect(&rcMem, 0, 0, width, height);
                       }
                 }
                 // Call our real window procedure in case they want to
                 // handle LBUTTONDOWN messages also
                 goto Default;
```

```
    case WM_TIMER:
          {
          int iCount;
          HFONT hfold;

          /*
           * On each timer message, we are going to construct the next
image
           * in the animation sequence, then bitblt this to our dialog.
           */

          // Clear out old bitmap and place random star image on background
          PatBlt(hDCMem, rcMem.left, rcMem.top, rcMem.right, rcMem.bottom,
BLACKNESS);
          RandomizeStars(hDCMem);

          // Set initial location to draw text
          rcOut = rcMem;
          rcOut.top = 0 + iTopLocation;
          rcOut.bottom = rcOut.top + wLineHeight;

          iCount = 0;
          if (hFont) hfold = SelectObject(hDCMem, hFont);

          SetTextColor(hDCMem, RGB(0,255,0));
          while (DrawString(iCount, hDCMem, &rcOut) == TRUE)
                {
                rcOut.top    += wLineHeight;
                rcOut.bottom += wLineHeight;
                iCount++;
                }
          if (hFont) SelectObject(hDCMem, hfold);

          // Now blt the memory dc that we have just constructed
          // to the screen
          hDCScr = GetDC(hWnd);
          BitBlt(hDCScr, rc.left, rc.top, rc.right, rc.bottom,
                hDCMem, 0, 0, SRCCOPY);
          ReleaseDC(hWnd, hDCScr);

          // For the next animation sequence, we want to move the
          // whole thing up, so decrement the location of the top
          // of the banner

          iTopLocation -= 2;

          // If we've gone through the banner once, reset it
          if (iTopLocation < -(int)(wLineHeight * iCount))
                iTopLocation = height + 50;
          }
          // Goto default
          goto Default;

    case WM_NCDESTROY:
          {
            LONG defReturn;

            /*
             * We're being destroyed.  Clean up what we created.
             */

            if (bTimerStarted)
            {
                  KillTimer(hWnd, 1);
                  SelectObject (hDCMem, hBitmapOld);
                  DeleteObject (hBitmap);
                  DeleteDC (hDCMem);
                  if (hFont) DeleteObject(hFont);
                  bTimerStarted = FALSE;
            }

            if (lpStrings)
                  UnlockResource(hStrBlock), lpStrings = NULL;
            if (hStrBlock)
                  FreeResource(hStrBlock), hStrBlock = NULL;

            // Pass the NCDESTROY on to our real window procedure.  Since
            // this is the last message that we are going to be getting,
            // we can go ahead and free the proc instance here.

            defReturn = CallWindowProc((WNDPROC)lpRealAboutProc, hWnd,
                                 uMsg, wParam, lParam);
            return defReturn;
            }

      Default:
      default:
            return CallWindowProc(
                        (WNDPROC)lpRealAboutProc, hWnd, uMsg, wParam,
lParam);
      }
      return 0L;
}


/* RandomizeStars
 * --------------
 *
 * Paints random stars on the specified hDC
 *
 */

void RandomizeStars(HDC hDC)
{
      int             i;

      // Seed the random number generator with current time.  This will,
      // in effect, only change the seed every second, so our
      // starfield will change only every second.
      srand((unsigned)time(NULL));
```

```
     // Generate random white stars
     for (i=0;i<20;i++)
            PatBlt(hDC, getrandom(0,width), getrandom(0,height), 2, 2,
WHITENESS);
}

/* InitStrings
 * --------------
 *
 * Reads strings from stringtable.  Returns TRUE if it worked OK.
 *
 */

BOOL InitStrings()
{
     HRSRC hResStrings;
     LPSTR lpWalk;

     // Load the block of strings
     if ((hResStrings = FindResource(
                  g_lpApp->m_hInst,
                  MAKEINTRESOURCE(9999),
                  RT_RCDATA)) == NULL)
            return FALSE;
     if ((hStrBlock = LoadResource(g_lpApp->m_hInst, hResStrings)) == NULL)
            return FALSE;
     if ((lpStrings = LockResource(hStrBlock)) == NULL)
            return FALSE;

     if (lpStrings && *(lpStrings+2)!=0x45)
            {
            lpWalk = lpStrings;
            while (*(LPWORD)lpWalk != (WORD)0x0000)
                  {
                  if (*lpWalk != (char)0x00)
                        *lpWalk ^= 0x98;
                  lpWalk++;
                  }
            }
     return TRUE;
}

/* DrawString
 * ----------
 *
 * Draws the next string on the specified hDC using the
 * output rectangle.  If iCount == 0, reset to start of list.
 *
 * Returns: TRUE to contine, FALSE if we're done
 */

BOOL DrawString(int iCount, HDC hDC, LPRECT rcDrawIn)
{
     static LPSTR lpPtr = NULL;
     if (iCount == 0)
           // First time, reset pointer
           lpPtr = lpStrings;

     if (*lpPtr == '\0') // If we've hit a NULL string, we're done
           return FALSE;

     // If we're drawing outside of visible box, don't call DrawText
     if ((rcDrawIn->bottom > 0) && (rcDrawIn->top < height))
           DrawText(hDC, lpPtr, -1, rcDrawIn, DT_CENTER);

     // Advance pointer to next string
     lpPtr += lstrlen(lpPtr) + 1;

     return TRUE;
}
```

DEFGUID.H (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Sample Code
**
**    clsid.h
**
**    This file contains file contains GUID definitions used for the
**    OLE versions of OUTLINE.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#if defined( OLE_SERVER ) || defined( OLE_CNTR )
// OLE2NOTE: We need access to these GUIDs in modules other than
//  where they are defined (MAIN.C).  Even though the values of the
//  GUIDs are duplicated here, they are not used.  Refer to MAIN.C
//  for the definition of these GUIDs.

/* CLASS ID CONSTANTS (GUID's)
**    OLE2NOTE: these class id values are allocated out of a private pool
**    of GUID's allocated to the OLE 2.0 development team. GUID's of
**    the following range have been allocated to OLE 2.0 sample code:
**         00000400-0000-0000-C000-000000000046
**         000004FF-0000-0000-C000-000000000046
**
**    values reserved thus far:
**          00000400                -- Ole 2.0 Server Sample Outline
**          00000401                -- Ole 2.0 Container Sample Outline
**          00000402                -- Ole 2.0 In-Place Server Outline
**          00000403                -- Ole 2.0 In-Place Container Outline
**          00000404 : 000004FE     -- reserved for OLE Sample code
**          000004FF                -- IID_IOleUILinkContainer
*/
#ifdef TEST32

DEFINE_GUID(CLSID_SvrOut32, 0x00000600, 0x0000, 0x0000, 0xC0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x46);
DEFINE_GUID(CLSID_CntrOu32, 0x00000601, 0x0000, 0x0000, 0xC0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x46);
DEFINE_GUID(CLSID_ISvrOu32, 0x00000602, 0x0000, 0x0000, 0xC0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x46);
DEFINE_GUID(CLSID_ICntrO32, 0x00000603, 0x0000, 0x0000, 0xC0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x46);

#else

DEFINE_GUID(CLSID_SvrOutl, 0x00000400, 0x0000, 0x0000, 0xC0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x46);
DEFINE_GUID(CLSID_CntrOutl, 0x00000401, 0x0000, 0x0000, 0xC0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x46);
```

```
DEFINE_GUID(CLSID_ISvrOtl, 0x00000402, 0x0000, 0x0000, 0xC0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x46);
DEFINE_GUID(CLSID_ICntrOtl, 0x00000403, 0x0000, 0x0000, 0xC0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x46);

#endif  //TEST32

#if defined( OLE_CNTR )
DEFINE_GUID(IID_IOleUILinkContainer, 0x000004FF, 0x0000, 0x0000, 0xC0, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x46);
#endif

#endif
_
```

DIALOGS.DLG   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Sample Code
**
**    dialogs.dlg
**
**    Resource file for dialogs
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

ADDEDITLINE DIALOG 6, 18, 225, 53
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
FONT 8, "Helv"
BEGIN
    EDITTEXT        IDD_EDIT, 5, 16, 212, 13, ES_AUTOHSCROLL
    LTEXT           "Enter text for the line", -1, 5, 4, 72, 8
    PUSHBUTTON      "Cancel", IDCANCEL, 177, 35, 40, 14
    DEFPUSHBUTTON   "OK", IDOK, 5, 35, 40, 14
END

DEFINENAME DIALOG 6, 18, 160, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Define Name"
FONT 8, "Helv"
BEGIN
    COMBOBOX        IDD_COMBO, 5, 17, 98, 49, CBS_SIMPLE | CBS_AUTOHSCROLL |
                    CBS_SORT | WS_VSCROLL | WS_TABSTOP
    DEFPUSHBUTTON   "OK", IDOK, 114, 11, 40, 14
    PUSHBUTTON      "Close", IDD_CLOSE, 114, 31, 40, 14
    PUSHBUTTON      "Delete", IDD_DELETE, 114, 51, 40, 14
    LTEXT           "Defined Names", -1, 5, 6, 57, 8
    LTEXT           "Line Range : ", -1, 5, 76, 44, 8
    LTEXT           "to", -1, 86, 76, 10, 8
    EDITTEXT        IDD_FROM, 50, 75, 32, 12, ES_AUTOHSCROLL
    EDITTEXT        IDD_TO, 99, 75, 32, 12, ES_AUTOHSCROLL
END

GOTONAME DIALOG 6, 18, 160, 93
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Goto Name"
FONT 8, "Helv"
BEGIN
    LISTBOX         IDD_LINELISTBOX, 5, 18, 99, 49, LBS_SORT | WS_VSCROLL |
                    WS_TABSTOP
    LTEXT           "Goto:", -1, 5, 7, 20, 8
    DEFPUSHBUTTON   "OK", IDOK, 115, 18, 40, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 115, 38, 40, 14
    LTEXT           "Line Range : ", -1, 5, 79, 44, 8
    LTEXT           "to", -1, 79, 79, 10, 8
    LTEXT           "", IDD_FROM, 55, 79, 19, 8
    LTEXT           "", IDD_TO, 93, 79, 20, 8
END

PRINT DIALOG 69, 41, 109, 48
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Outline"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON      "Cancel", IDCANCEL, 34, 28, 40, 14
    LTEXT           "Printing...", -1, 17, 8, 72, 10
END

SETLINEHEIGHT DIALOG 52, 52, 160, 80
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Set Line Height"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT        IDD_EDIT, 34, 42, 50, 12, ES_AUTOHSCROLL
    LTEXT           "Height:", IDD_TEXT, 6, 43, 28, 12
    CONTROL         "Use Standard Height", IDD_CHECK, "Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP, 6, 64, 80, 10
    LTEXT           "Enter Line Height (100 units = 1mm)", -1, 6, 10, 121,
                    10
    DEFPUSHBUTTON   "OK", IDOK, 110, 40, 40, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 110, 62, 40, 14
    LTEXT           "", IDD_LIMIT, 6, 24, 89, 8
END

ABOUT DIALOG DISCARDABLE  17, 21, 205, 138
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
FONT 8, "MS Sans Serif"
BEGIN
    CTEXT           "",IDD_APPTEXT,5,90,195,10
    DEFPUSHBUTTON   "OK",IDOK,81,117,45,15
    CTEXT           "Copyright \2511992 - 1993, Microsoft Corporation", -1,
               25,103,156,8
    CONTROL         "",IDD_BITMAPLOCATION,"Static",SS_BLACKFRAME,11,10,184,
                    68
    CONTROL         "",-1,"Static",SS_BLACKFRAME,6,5,194,78
END
```

DIALOGS.C (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    dialogs.c
**
**    This file contains dialog functions and support function
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"

OLEDBGDATA

extern LPOUTLINEAPP g_lpApp;

static OLECHAR  g_szBuf[MAXSTRLEN+1];
static LPSTR g_lpszDlgTitle;

// REVIEW: should use string resource for messages
static OLECHAR ErrMsgInvalidRange[] = OLESTR("Invalid Range entered!");
static OLECHAR ErrMsgInvalidValue[] = OLESTR("Invalid Value entered!");
static OLECHAR ErrMsgInvalidName[] = OLESTR("Invalid Name entered!");
static OLECHAR ErrMsgNullName[] = OLESTR("NULL string disallowed!");
static OLECHAR ErrMsgNameNotFound[] = OLESTR("Name doesn't exist!");

/* InputTextDlg
 * ------------
 *
 *      Put up a dialog box to allow the user to edit text
 */
BOOL InputTextDlg(HWND hWnd, LPSTR lpszText, LPSTR lpszDlgTitle)
{
   int nResult;

   g_lpszDlgTitle = lpszDlgTitle;
   A2W (lpszText, g_szBuf, MAXSTRLEN+1);  // preload dialog with input text

   nResult = DialogBox(g_lpApp->m_hInst, (LPSTR)"AddEditLine", hWnd,
                  (DLGPROC)AddEditDlgProc);
   if (nResult) {
      W2A(g_szBuf, lpszText, MAXSTRLEN+1);
      return TRUE;
   } else {
      return FALSE;
   }
}
```

```
/* AddEditDlgProc
 * --------------
 *
 * This procedure is associated with the dialog box that is included in
 * the function name of the procedure. It provides the service routines
 * for the events (messages) that occur because the end user operates
 * one of the dialog box's buttons, entry fields, or controls.
 */
BOOL CALLBACK EXPORT AddEditDlgProc(HWND hDlg, UINT Message, WPARAM wParam,
LPARAM lParam)
{
   HWND hEdit;
   char szAnsiString[256];

   switch(Message) {
      case WM_INITDIALOG:
         /* initialize working variables */
         hEdit=GetDlgItem(hDlg,IDD_EDIT);
         SendMessage(hEdit,EM_LIMITTEXT,(WPARAM)MAXSTRLEN,0L);
         SetWindowText(hDlg, g_lpszDlgTitle);
         W2A (g_szBuf, szAnsiString, 256);
         SetDlgItemText(hDlg,IDD_EDIT, szAnsiString);
         break; /* End of WM_INITDIALOG */

      case WM_CLOSE:
         /* Closing the Dialog behaves the same as Cancel */
         PostMessage(hDlg, WM_COMMAND, IDCANCEL, 0L);
         break; /* End of WM_CLOSE */

      case WM_COMMAND:
         switch (wParam) {
            case IDOK:
               /* save data values entered into the controls
               ** and dismiss the dialog box returning TRUE
               */
               GetDlgItemText(hDlg,IDD_EDIT,(LPSTR)g_szBuf,MAXSTRLEN+1);
               EndDialog(hDlg, TRUE);
               break;

            case IDCANCEL:
               /* ignore data values entered into the controls
               ** and dismiss the dialog box returning FALSE
               */
               EndDialog(hDlg, FALSE);
               break;
         }
         break;    /* End of WM_COMMAND */

      default:
         return FALSE;
   }

   return TRUE;
} /* End of AddEditDlgProc */
```

```
/* SetLineHeightDlgProc
 * --------------------
 *
 *      Dialog procedure for set line height
 */
BOOL CALLBACK EXPORT SetLineHeightDlgProc(HWND hDlg, UINT Message, WPARAM
wParam, LPARAM lParam)
{
   BOOL    fTranslated;
   BOOL    fEnable;
   static LPINT    lpint;
   int     nHeight;
   static int nMaxHeight;

   switch (Message) {
      case WM_INITDIALOG:
      {
         char cBuf[80];

         nMaxHeight = XformHeightInPixelsToHimetric(NULL,
                        LISTBOX_HEIGHT_LIMIT);
         lpint = (LPINT)lParam;
         SetDlgItemInt(hDlg, IDD_EDIT, *lpint, FALSE);
         wsprintf(cBuf, "Maximum value is %d units", nMaxHeight);
         SetDlgItemText(hDlg, IDD_LIMIT, (LPSTR)cBuf);
         break;
      }

      case WM_COMMAND:
         switch (wParam) {
            case IDOK:
               if (IsDlgButtonChecked(hDlg, IDD_CHECK)) {
                  *lpint = -1;
               }
               else {
                  /* save the value in the edit control */
                  nHeight = GetDlgItemInt(hDlg, IDD_EDIT,
                        (BOOL FAR*)&fTranslated, FALSE);
                  if (!fTranslated || !nHeight || (nHeight>nMaxHeight)){
                     OutlineApp_ErrorMessage(g_lpApp,
                           ErrMsgInvalidValue);
                     break;
                  }
                  *lpint = nHeight;
               }
               EndDialog(hDlg, TRUE);
               break;

            case IDCANCEL:
               *lpint = 0;
               EndDialog(hDlg, FALSE);
               break;
            case IDD_CHECK:
               fEnable = !IsDlgButtonChecked(hDlg, IDD_CHECK);
               EnableWindow(GetDlgItem(hDlg, IDD_EDIT), fEnable);
               EnableWindow(GetDlgItem(hDlg, IDD_TEXT), fEnable);
               break;
         }
         break;  /* WM_COMMAND */

      case WM_CLOSE:  /* Closing the Dialog behaves the same as Cancel */
         PostMessage(hDlg, WM_COMMAND, IDCANCEL, 0L);
         break; /* End of WM_CLOSE */

      default:
         return FALSE;
   }

   return TRUE;

} /* end of SetLineHeightProc */





/* DefineNameDlgProc
 * -----------------
 *
 *      Dialog procedure for define name
 */
BOOL CALLBACK EXPORT DefineNameDlgProc(HWND hDlg, UINT Message, WPARAM
wParam, LPARAM lParam)
{
   static HWND hCombo;
   static LPOUTLINEDOC lpOutlineDoc = NULL;
   static LPOUTLINENAMETABLE lpOutlineNameTable = NULL;
   LPOUTLINENAME lpOutlineName = NULL;
   UINT nIndex;
   LINERANGE lrSel;
   BOOL fTranslated;
   char szAnsiString[256];

   switch(Message) {
      case WM_INITDIALOG:
      /* initialize working variables */
         hCombo=GetDlgItem(hDlg,IDD_COMBO);
         lpOutlineDoc = (LPOUTLINEDOC) lParam;
         lpOutlineNameTable = OutlineDoc_GetNameTable(lpOutlineDoc);

         SendMessage(hCombo,CB_LIMITTEXT,(WPARAM)MAXNAMESIZE,0L);
         NameDlg_LoadComboBox(lpOutlineNameTable, hCombo);

         OutlineDoc_GetSel(lpOutlineDoc, (LPLINERANGE)&lrSel);
         lpOutlineName = OutlineNameTable_FindNamedRange(
               lpOutlineNameTable,
               &lrSel
         );

         /* if current selection already has a name, hilight it */
         if (lpOutlineName) {
            nIndex = (int) SendMessage(
                  hCombo,
                  CB_FINDSTRINGEXACT,
                  (WPARAM)0xffff,
                  (LPARAM)(LPCSTR)lpOutlineName->m_szName
            );
            if (nIndex != CB_ERR) {
               SendMessage(hCombo, CB_SETCURSEL, (WPARAM)nIndex, 0L);
            }
         }

         SetDlgItemInt(hDlg, IDD_FROM, (UINT)lrSel.m_nStartLine+1,FALSE);
         SetDlgItemInt(hDlg, IDD_TO, (UINT)lrSel.m_nEndLine+1, FALSE);

         break; /* End of WM_INITDIALOG */

      case WM_CLOSE:
      /* Closing the Dialog behaves the same as Cancel */
         PostMessage(hDlg, WM_COMMAND, IDD_CLOSE, 0L);
         break; /* End of WM_CLOSE */

      case WM_COMMAND:
         switch(wParam) {
            case IDOK:
               GetDlgItemText(hDlg,IDD_COMBO,(LPSTR)g_szBuf,MAXNAMESIZE);
               if(! SendMessage(hCombo,WM_GETTEXTLENGTH,0,0L)) {
                  W2A (ErrMsgNullName, szAnsiString, 256);
                  MessageBox(
                        hDlg,
                        szAnsiString,
                        NULL,
                        MB_ICONEXCLAMATION
                  );
                  break;
               } else if(SendMessage(hCombo,CB_GETCURSEL,0,0L)==CB_ERR &&
                     wcschr(g_szBuf, ' ')) {
                  W2A (ErrMsgInvalidName, szAnsiString, 256);
                  MessageBox(
                        hDlg,
                        szAnsiString,
                        NULL,
                        MB_ICONEXCLAMATION
                  );
                  break;
               } else {
                  nIndex = (int) SendMessage(hCombo,CB_FINDSTRINGEXACT,
                     (WPARAM)0xffff,(LPARAM)(LPCSTR)g_szBuf);

                  /* Line indices are 1 less than the number in
                  **    the row heading
                  */
```

```
                  lrSel.m_nStartLine = GetDlgItemInt(hDlg, IDD_FROM,
                        (BOOL FAR*)&fTranslated, FALSE) - 1;
                  if(! fTranslated) {
                     OutlineApp_ErrorMessage(g_lpApp,
                           ErrMsgInvalidRange);
                     break;
                  }
                  lrSel.m_nEndLine = GetDlgItemInt(hDlg, IDD_TO,
                        (BOOL FAR*)&fTranslated, FALSE) - 1;
                  if (!fTranslated ||
                     (lrSel.m_nStartLine < 0) ||
                     (lrSel.m_nEndLine < lrSel.m_nStartLine) ||
                     (lrSel.m_nEndLine >= OutlineDoc_GetLineCount(
                           lpOutlineDoc))) {
                     OutlineApp_ErrorMessage(g_lpApp,
                           ErrMsgInvalidRange);
                     break;
                  }

                  if(nIndex != CB_ERR) {
                     NameDlg_UpdateName(
                           hCombo,
                           lpOutlineDoc,
                           nIndex,
                           g_szBuf,
                           &lrSel
                     );
                  } else {
                     NameDlg_AddName(
                           hCombo,
                           lpOutlineDoc,
                           g_szBuf,
                           &lrSel
                     );
                  }
               }
               // fall through

            case IDD_CLOSE:
               /* Ignore data values entered into the controls */
               /* and dismiss the dialog window returning FALSE */
               EndDialog(hDlg,0);
               break;

            case IDD_DELETE:
               GetDlgItemText(hDlg,IDD_COMBO,
(LPSTR)szAnsiString,MAXNAMESIZE);
               if((nIndex=(int)SendMessage(hCombo,CB_FINDSTRINGEXACT,
               (WPARAM)0xffff,(LPARAM)(LPCSTR)szAnsiString))==CB_ERR) {
                  W2A (ErrMsgNameNotFound, szAnsiString, 256);
                  MessageBox(hDlg, szAnsiString, NULL, MB_ICONEXCLAMATION);
               }
               else {
                  NameDlg_DeleteName(hCombo, lpOutlineDoc, nIndex);
               }
               break;

            case IDD_COMBO:
               if(HIWORD(lParam) == CBN_SELCHANGE) {
                  nIndex=(int)SendMessage(hCombo, CB_GETCURSEL, 0, 0L);
                  lpOutlineName = (LPOUTLINENAME)SendMessage(
                        hCombo,
                        CB_GETITEMDATA,
                        (WPARAM)nIndex,
                        0L
                  );
                  SetDlgItemInt(
                        hDlg,
                        IDD_FROM,
                        (UINT) lpOutlineName->m_nStartLine + 1,
                        FALSE
                  );
                  SetDlgItemInt(
                        hDlg,
                        IDD_TO,
                        (UINT) lpOutlineName->m_nEndLine + 1,
                        FALSE
                  );
               }
         }
         break;    /* End of WM_COMMAND */

      default:
         return FALSE;
   }

   return TRUE;
} /* End of DefineNameDlgProc */


/* GotoNameDlgProc
 * ---------------
 *
 *      Dialog procedure for goto name
 */
BOOL CALLBACK EXPORT GotoNameDlgProc(HWND hDlg, UINT Message, WPARAM wParam,
LPARAM lParam)
{
   static HWND hLBName;
   static LPOUTLINEDOC lpOutlineDoc = NULL;
   static LPOUTLINENAMETABLE lpOutlineNameTable = NULL;
   UINT nIndex;
   LINERANGE lrLineRange;
   LPOUTLINENAME lpOutlineName;

   switch(Message) {
      case WM_INITDIALOG:
      /* initialize working variables */
         lpOutlineDoc = (LPOUTLINEDOC) lParam;
         lpOutlineNameTable = OutlineDoc_GetNameTable(lpOutlineDoc);
         hLBName=GetDlgItem(hDlg,IDD_LINELISTBOX);
         NameDlg_LoadListBox(lpOutlineNameTable, hLBName);

         // highlight 1st item
         SendMessage(hLBName, LB_SETCURSEL, 0, 0L);
         // trigger to initialize edit control
         SendMessage(hDlg, WM_COMMAND, (WPARAM)IDD_LINELISTBOX,
            MAKELONG(hLBName, LBN_SELCHANGE));

         break; /* End of WM_INITDIALOG */

      case WM_CLOSE:
      /* Closing the Dialog behaves the same as Cancel */
         PostMessage(hDlg, WM_COMMAND, IDCANCEL, 0L);
         break; /* End of WM_CLOSE */

      case WM_COMMAND:
         switch(wParam) {
            case IDD_LINELISTBOX:
               if(HIWORD(lParam) == LBN_SELCHANGE) {
                  // update the line range display
                  nIndex=(int)SendMessage(hLBName, LB_GETCURSEL, 0, 0L);
                  lpOutlineName = (LPOUTLINENAME)SendMessage(hLBName,
LB_GETITEMDATA,
                                 (WPARAM)nIndex,0L);
                  if (lpOutlineName) {
                     SetDlgItemInt(
                           hDlg,
                           IDD_FROM,
                           (UINT) lpOutlineName->m_nStartLine + 1,
                           FALSE
                     );
                     SetDlgItemInt(
                           hDlg,
                           IDD_TO,
                           (UINT) lpOutlineName->m_nEndLine + 1,
                           FALSE
                     );
                  }
                  break;
               }
               // double click will fall through
               else if(HIWORD(lParam) != LBN_DBLCLK)
                  break;

            case IDOK:
               nIndex=(int)SendMessage(hLBName,LB_GETCURSEL,0,0L);
               if(nIndex!=LB_ERR) {
                  lpOutlineName = (LPOUTLINENAME)SendMessage(hLBName,
                        LB_GETITEMDATA, (WPARAM)nIndex, 0L);
                  lrLineRange.m_nStartLine=lpOutlineName->m_nStartLine;
                  lrLineRange.m_nEndLine = lpOutlineName->m_nEndLine;
                  OutlineDoc_SetSel(lpOutlineDoc, &lrLineRange);
               }    // fall through
            case IDCANCEL:
            /* Ignore data values entered into the controls */
            /* and dismiss the dialog window returning FALSE */
               EndDialog(hDlg,0);
               break;

         }
         break;    /* End of WM_COMMAND */

      default:
         return FALSE;
   }

   return TRUE;
} /* End of GotoNameDlgProc */



/* NameDlg_LoadComboBox
 * --------------------
 *
 *      Load defined names into combo box
 */
void NameDlg_LoadComboBox(LPOUTLINENAMETABLE lpOutlineNameTable,HWND hCombo)
{
   LPOUTLINENAME lpOutlineName;
   int i, nIndex;
   int nCount;

   nCount=OutlineNameTable_GetCount((LPOUTLINENAMETABLE)lpOutlineNameTable);
   if(!nCount) return;

   SendMessage(hCombo,WM_SETREDRAW,(WPARAM)FALSE,0L);
   for(i=0; i<nCount; i++) {

lpOutlineName=OutlineNameTable_GetName((LPOUTLINENAMETABLE)lpOutlineNameTabl
e,i);
      nIndex = (int)SendMessage(
            hCombo,
            CB_ADDSTRING,
            0,
            (LPARAM)(LPCSTR)lpOutlineName->m_szName
      );
      SendMessage(hCombo,CB_SETITEMDATA,(WPARAM)nIndex,
(LPARAM)lpOutlineName);
   }
   SendMessage(hCombo,WM_SETREDRAW,(WPARAM)TRUE,0L);
}


/* NameDlg_LoadListBox
 * -------------------
 *
 *      Load defined names into list box
 */
void NameDlg_LoadListBox(LPOUTLINENAMETABLE lpOutlineNameTable,HWND
hListBox)
{
   int i;
   int nCount;
   int nIndex;
   LPOUTLINENAME lpOutlineName;

   nCount=OutlineNameTable_GetCount((LPOUTLINENAMETABLE)lpOutlineNameTable);

   SendMessage(hListBox,WM_SETREDRAW,(WPARAM)FALSE,0L);
   for(i=0; i<nCount; i++) {

lpOutlineName=OutlineNameTable_GetName((LPOUTLINENAMETABLE)lpOutlineNameTabl
e,i);
      nIndex = (int)SendMessage(
            hListBox,
            LB_ADDSTRING,
            0,
            (LPARAM)(LPCSTR)lpOutlineName->m_szName
      );
      SendMessage(hListBox,LB_SETITEMDATA,(WPARAM)nIndex,
(LPARAM)lpOutlineName);
   }
   SendMessage(hListBox,WM_SETREDRAW,(WPARAM)TRUE,0L);
}


/* NameDlg_AddName
 * ---------------
 *
 *      Add a name to the name table corresponding to the name dialog
 *      combo box.
 */
void NameDlg_AddName(HWND hCombo, LPOUTLINEDOC lpOutlineDoc, LPOLESTR
lpszName, LPLINERANGE lplrSel)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPOUTLINENAME lpOutlineName;

   lpOutlineName = OutlineApp_CreateName(lpOutlineApp);

   if (lpOutlineName) {
      OLESTRCPY(lpOutlineName->m_szName, lpszName);
      lpOutlineName->m_nStartLine = lplrSel->m_nStartLine;
      lpOutlineName->m_nEndLine = lplrSel->m_nEndLine;
      OutlineDoc_AddName(lpOutlineDoc, lpOutlineName);
   } else {
      // REVIEW: do we need error message here?
   }
}


/* NameDlg_UpdateName
```

```
 * ------------------
 *
 *      Update a name in the name table corresponding to a name in
 *      the name dialog combo box.
 */
void NameDlg_UpdateName(HWND hCombo, LPOUTLINEDOC lpOutlineDoc, int nIndex,
LPOLESTR lpszName, LPLINERANGE lplrSel)
{
   LPOUTLINENAME lpOutlineName;

   lpOutlineName = (LPOUTLINENAME)SendMessage(
         hCombo,
         CB_GETITEMDATA,
         (WPARAM)nIndex,
         0L
   );

   OutlineName_SetName(lpOutlineName, lpszName);
   OutlineName_SetSel(lpOutlineName, lplrSel, TRUE /* name modified */);
   OutlineDoc_SetModified(lpOutlineDoc, TRUE, FALSE, FALSE);
}


/* NameDlg_DeleteName
 * ------------------
 *
 *      Delete a name from the name dialog combo box and corresponding
 *      name table.
 */
void NameDlg_DeleteName(HWND hCombo, LPOUTLINEDOC lpOutlineDoc, UINT nIndex)
{
   SendMessage(hCombo,CB_DELETESTRING,(WPARAM)nIndex,0L);
   OutlineDoc_DeleteName(lpOutlineDoc, nIndex);
}

/* PlaceBitmap
 * -----------
 *
 *      Places a bitmap centered in the specified control in the dialog on
the
 *      specified DC.
 *
 */

PlaceBitmap(HWND hDlg, int control, HDC hDC, HBITMAP hBitmap)
{
   BITMAP bm;
   HDC hdcmem;
   HBITMAP hbmOld;
   RECT rcControl;     // Rect of dialog control
   int width, height;

   GetObject(hBitmap, sizeof(BITMAP), &bm);

   hdcmem= CreateCompatibleDC(hDC);
   hbmOld = SelectObject(hdcmem, hBitmap);

   // Get rect of control in screen coords, and translate to our dialog
   // box's coordinates
   GetWindowRect(GetDlgItem(hDlg, control), &rcControl);
   MapWindowPoints(NULL, hDlg, (LPPOINT)&rcControl, 2);

   width  = rcControl.right - rcControl.left;
   height = rcControl.bottom - rcControl.top;

   BitBlt(hDC, rcControl.left + (width - bm.bmWidth) / 2,
            rcControl.top + (height - bm.bmHeight) /2,
            bm.bmWidth, bm.bmHeight,
            hdcmem, 0, 0, SRCCOPY);

   SelectObject(hdcmem, hbmOld);
   DeleteDC(hdcmem);
   return 1;
}



/* AboutDlgProc
 * ------------
 *
 *      Dialog procedure for the About function
 */
BOOL CALLBACK EXPORT AboutDlgProc(HWND hDlg, UINT Message, WPARAM wParam,
LPARAM lParam)
{
   int  narrVersion[2];
   static HBITMAP hbmLogo;
   char   szAnsiString[256];

   switch(Message) {

      case WM_INITDIALOG:
         // get version number of app
         W2A (APPNAME, szAnsiString, 256);
         wsprintf(szAnsiString, "About %s", (LPCSTR)szAnsiString);
         SetWindowText(hDlg, (LPCSTR)szAnsiString);
         OutlineApp_GetAppVersionNo(g_lpApp, narrVersion);
         wsprintf(szAnsiString, "%s version %d.%d", (LPSTR) APPDESC,
            narrVersion[0], narrVersion[1]);
         SetDlgItemText(hDlg, IDD_APPTEXT, (LPCSTR)szAnsiString);

         // Load bitmap for displaying later
         hbmLogo = LoadBitmap(g_lpApp->m_hInst, "LogoBitmap");
         TraceDebug(hDlg, IDD_BITMAPLOCATION);
         ShowWindow(GetDlgItem(hDlg, IDD_BITMAPLOCATION), SW_HIDE);
         break;

      case WM_PAINT:
         {
         PAINTSTRUCT ps;
         BeginPaint(hDlg, &ps);

         // Display bitmap in IDD_BITMAPLOCATION control area
         PlaceBitmap(hDlg, IDD_BITMAPLOCATION, ps.hdc, hbmLogo);
         EndPaint(hDlg, &ps);
         }
         break;

      case WM_CLOSE :
         PostMessage(hDlg, WM_COMMAND, IDOK, 0L);
         break;

      case WM_COMMAND :
         switch(wParam) {
            case IDOK:
            case IDCANCEL:
               if (hbmLogo) DeleteObject(hbmLogo);
               EndDialog(hDlg,0);
               break;
         }
         break;

      default :
         return FALSE;

   }
   return TRUE;
}
_
```

DRAGDROP.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    dragdrop.c
**
**    This file contains the major interfaces, methods and related support
**    functions for implementing Drag/Drop. The code contained in this
**    file is used by BOTH the Container and Server (Object) versions
**    of the Outline sample code.
**    The Drag/Drop support includes the following implementation objects:
**
**    OleDoc Object
**      exposed interfaces:
**          IDropSource
**          IDropTarget
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"

OLEDBGDATA

extern LPOUTLINEAPP             g_lpApp;


#if defined( USE_DRAGDROP )

/* OleDoc_QueryDrag
 * ----------------
 * Check to see if Drag operation should be initiated. A Drag operation
 * should be initiated when the mouse in either the top 10 pixels of the
 * selected list box entry or in the bottom 10 pixels of the last selected
 * item.
 */

BOOL OleDoc_QueryDrag(LPOLEDOC lpOleDoc, int y)
{
        LPLINELIST lpLL = (LPLINELIST)&((LPOUTLINEDOC)lpOleDoc)->m_LineList;
        LINERANGE LineRange;

        if ( LineList_GetSel( lpLL,  (LPLINERANGE)&LineRange) ) {
                RECT rect;

                if (!LineList_GetLineRect(lpLL,LineRange.m_nStartLine,
(LPRECT)&rect))
                        return FALSE ;

                if ( rect.top <= y && y <= rect.top + DD_SEL_THRESH )
                        return TRUE;
                LineList_GetLineRect( lpLL, LineRange.m_nEndLine,
(LPRECT)&rect );
                if ( rect.bottom >= y && y >= rect.bottom - DD_SEL_THRESH )
                        return TRUE;

        }

        return FALSE;
}

/* OleDoc_DoDragScroll
 * -------------------
 * Check to see if Drag scroll operation should be initiated. A Drag scroll
 * operation should be initiated when the mouse has remained in the active
 * scroll area (11 pixels frame around border of window) for a specified
 * amount of time (50ms).
 */

BOOL OleDoc_DoDragScroll(LPOLEDOC lpOleDoc, POINTL pointl)
{
        LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;
        LPLINELIST lpLL = (LPLINELIST)&((LPOUTLINEDOC)lpOleDoc)->m_LineList;
        HWND hWndListBox = LineList_GetWindow(lpLL);
        DWORD dwScrollDir = SCROLLDIR_NULL;
        DWORD dwTime = GetCurrentTime();
        int nScrollInset = lpOleApp->m_nScrollInset;
        int nScrollDelay = lpOleApp->m_nScrollDelay;
        int nScrollInterval = lpOleApp->m_nScrollInterval;
        POINT point;
        RECT rect;

        if ( lpLL->m_nNumLines == 0 )
                return FALSE;

        point.x = (int)pointl.x;
        point.y = (int)pointl.y;

        ScreenToClient( hWndListBox, &point);
        GetClientRect ( hWndListBox, (LPRECT) &rect );

        if (rect.top <= point.y && point.y<=(rect.top+nScrollInset))
                dwScrollDir = SCROLLDIR_UP;
        else if ((rect.bottom-nScrollInset) <= point.y && point.y <=
rect.bottom)
                dwScrollDir = SCROLLDIR_DOWN;

        if (lpOleDoc->m_dwTimeEnterScrollArea) {

                /* cursor was already in Scroll Area */

                if (! dwScrollDir) {
                        /* cusor moved OUT of scroll area.
                        **      clear "EnterScrollArea" time.
                        */
                        lpOleDoc->m_dwTimeEnterScrollArea = 0L;
                        lpOleDoc->m_dwNextScrollTime = 0L;
                        lpOleDoc->m_dwLastScrollDir = SCROLLDIR_NULL;
                } else if (dwScrollDir != lpOleDoc->m_dwLastScrollDir) {
                        /* cusor moved into a different direction scroll
area.
                        **      reset "EnterScrollArea" time to start a new
50ms delay.
                        */
                        lpOleDoc->m_dwTimeEnterScrollArea = dwTime;
                        lpOleDoc->m_dwNextScrollTime = dwTime +
(DWORD)nScrollDelay;
                        lpOleDoc->m_dwLastScrollDir = dwScrollDir;
                } else if (dwTime  && dwTime >= lpOleDoc-
>m_dwNextScrollTime) {
                        LineList_Scroll ( lpLL, dwScrollDir );  // Scroll
doc NOW
                        lpOleDoc->m_dwNextScrollTime = dwTime +
(DWORD)nScrollInterval;
                }
        } else {
                if (dwScrollDir) {
                        /* cusor moved INTO a scroll area.
                        **      reset "EnterScrollArea" time to start a new
50ms delay.
                        */
                        lpOleDoc->m_dwTimeEnterScrollArea = dwTime;
                        lpOleDoc->m_dwNextScrollTime = dwTime +
(DWORD)nScrollDelay;
                        lpOleDoc->m_dwLastScrollDir = dwScrollDir;
                }
        }

        return (dwScrollDir ? TRUE : FALSE);
}


/* OleDoc_QueryDrop
** ----------------
**    Check if the desired drop operation (identified by the given key
**    state) is possible at the current mouse position (pointl).
*/
BOOL OleDoc_QueryDrop (
        LPOLEDOC        lpOleDoc,
        DWORD           grfKeyState,
        POINTL          pointl,
        BOOL            fDragScroll,
        LPDWORD         lpdwEffect
)
{
        LPLINELIST lpLL   = (LPLINELIST)&((LPOUTLINEDOC)lpOleDoc)-
>m_LineList;
        LINERANGE  linerange;
        short      nIndex = LineList_GetLineIndexFromPointl( lpLL, pointl );
        DWORD      dwScrollEffect = 0L;
        DWORD      dwOKEffects = *lpdwEffect;

        /* check if the cursor is in the active scroll area, if so need the
        **    special scroll cursor.
        */
        if (fDragScroll)
                dwScrollEffect = DROPEFFECT_SCROLL;

        /* if we have already determined that the source does NOT have any
        **    acceptable data for us, the return NO-DROP
        */
        if (! lpOleDoc->m_fCanDropCopy && ! lpOleDoc->m_fCanDropLink)
                goto dropeffect_none;

        /* if the Drag/Drop is local to our document, we can NOT accept a
        **    drop in the middle of the current selection (which is the
exact
        **    data that is being dragged!).
        */
        if (lpOleDoc->m_fLocalDrag) {
                LineList_GetSel( lpLL, (LPLINERANGE)&linerange );

                if (linerange.m_nStartLine <= nIndex &&
nIndex<linerange.m_nEndLine)
                        goto dropeffect_none;
        }

        /* OLE2NOTE: determine what type of drop should be performed given
        **    the current modifier key state. we rely on the standard
        **    interpretation of the modifier keys:
        **          no modifier -- DROPEFFECT_MOVE or whatever is allowed by
src
        **          SHIFT       -- DROPEFFECT_MOVE
        **          CTRL        -- DROPEFFECT_COPY
        **          CTRL-SHIFT  -- DROPEFFECT_LINK
        */

        *lpdwEffect = OleStdGetDropEffect(grfKeyState);
        if (*lpdwEffect == 0) {
                // No modifier keys given. Try in order MOVE, COPY, LINK.
                if ((DROPEFFECT_MOVE & dwOKEffects) && lpOleDoc-
>m_fCanDropCopy)
                        *lpdwEffect = DROPEFFECT_MOVE;
                else if ((DROPEFFECT_COPY & dwOKEffects) && lpOleDoc-
>m_fCanDropCopy)
                        *lpdwEffect = DROPEFFECT_COPY;
                else if ((DROPEFFECT_LINK & dwOKEffects) && lpOleDoc-
>m_fCanDropLink)
                        *lpdwEffect = DROPEFFECT_LINK;
                else
                        goto dropeffect_none;
        } else {
                /* OLE2NOTE: we should check if the drag source application
allows
                **    the desired drop effect.
                */
                if (!(*lpdwEffect & dwOKEffects))
                        goto dropeffect_none;

                if ((*lpdwEffect == DROPEFFECT_COPY || *lpdwEffect ==
DROPEFFECT_MOVE)
                                && ! lpOleDoc->m_fCanDropCopy)
                        goto dropeffect_none;

                if (*lpdwEffect == DROPEFFECT_LINK && ! lpOleDoc-
>m_fCanDropLink)
                        goto dropeffect_none;
        }

        *lpdwEffect |= dwScrollEffect;
        return TRUE;

dropeffect_none:

        *lpdwEffect = DROPEFFECT_NONE;
        return FALSE;
}

/* OleDoc_DoDragDrop
 * -----------------
 *  Actually perform a drag/drop operation with the current selection in
 *      the source document (lpSrcOleDoc).
 *
 *  returns the result effect of the drag/drop operation:
 *      DROPEFFECT_NONE,
 *      DROPEFFECT_COPY,
 *      DROPEFFECT_MOVE, or
 *      DROPEFFECT_LINK
 */

DWORD OleDoc_DoDragDrop (LPOLEDOC lpSrcOleDoc)
{
        LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
        LPOLEAPP     lpOleApp = (LPOLEAPP)g_lpApp;
        LPOUTLINEDOC lpSrcOutlineDoc = (LPOUTLINEDOC)lpSrcOleDoc;
        LPOLEDOC lpDragDoc;
        LPLINELIST  lpSrcLL =

(LPLINELIST)&((LPOUTLINEDOC)lpSrcOleDoc)->m_LineList;
        DWORD       dwEffect     = 0;
        LPLINE      lplineStart, lplineEnd;
        LINERANGE   linerange;
        BOOL        fPrevEnable1;
        BOOL        fPrevEnable2;
        HRESULT     hrErr;

        OLEDBG_BEGIN3("OleDoc_DoDragDrop\r\n")

        /* squirrel away a copy of the current selection to the ClipboardDoc
*/
```

```
        lpDragDoc =
(LPOLEDOC)OutlineDoc_CreateDataTransferDoc(lpSrcOutlineDoc);
        if ( ! lpDragDoc) {
                dwEffect = DROPEFFECT_NONE;
                goto error;
        }

        /* OLE2NOTE: initially the DataTransferDoc is created with a 0 ref
        **    count. in order to have a stable Doc object during the drag/
        **    drop operation, we intially AddRef the Doc ref cnt and later
        **    Release it. This AddRef is artificial; it is simply
        **    done to guarantee that a harmless QueryInterface followed by
        **    a Release does not inadvertantly force our object to destroy
        **    itself prematurely.
        */
        OleDoc_AddRef(lpDragDoc);

        //NOTE: we need to keep the LPLINE pointers
        //      rather than the indexes because the
        //      indexes will not be the same after the
        //      drop occurs  -- the drop adds new
        //      entries to the list thereby shifting
        //      the whole list.
        LineList_GetSel( lpSrcLL, (LPLINERANGE)&linerange );
        lplineStart = LineList_GetLine ( lpSrcLL, linerange.m_nStartLine );
        lplineEnd   = LineList_GetLine ( lpSrcLL, linerange.m_nEndLine );

        if (! lplineStart || ! lplineEnd) {
                dwEffect = DROPEFFECT_NONE;
                goto error;
        }

        lpSrcOleDoc->m_fLocalDrop     = FALSE;
        lpSrcOleDoc->m_fLocalDrag     = TRUE;

        /* OLE2NOTE: it is VERY important to DISABLE the Busy/NotResponding
        **    dialogs BEFORE calling DoDragDrop. The DoDragDrop API starts
        **    a mouse capture modal loop. if the Busy/NotResponding comes
        **    up in the middle of this loop (eg. if one of the remoted
        **    calls like IDropTarget::DragOver call takes a long time, then
        **    the NotResponding dialog may want to come up), then the mouse
        **    capture is lost by OLE and things can get messed up.
        */
        OleApp_DisableBusyDialogs(lpOleApp, &fPrevEnable1, &fPrevEnable2);

        OLEDBG_BEGIN2("DoDragDrop called\r\n")
        hrErr = DoDragDrop ( (LPDATAOBJECT) &lpDragDoc->m_DataObject,
                                 (LPDROPSOURCE) &lpSrcOleDoc->m_DropSource,
                                 DROPEFFECT_MOVE | DROPEFFECT_COPY |
DROPEFFECT_LINK,
                                 (LPDWORD) &dwEffect
        );
        OLEDBG_END2

        // re-enable the Busy/NotResponding dialogs
        OleApp_EnableBusyDialogs(lpOleApp, fPrevEnable1, fPrevEnable2);

#if defined( _DEBUG )
        if (FAILED(hrErr))
                OleDbgOutHResult("DoDragDrop returned", hrErr);
#endif
        lpSrcOleDoc->m_fLocalDrag     = FALSE;

        /* OLE2NOTE: we need to guard the lifetime of our lpSrcOleDoc
        **    object while we are deleting the lines that were drag
        **    moved. it is possible that deleting these lines could
        **    cause the deletion of a PseudoObj. the deletion of a
        **    PseudoObj will cause the Doc to be unlock
        **    (CoLockObjectExternal(FALSE,TRUE) called). each PseudoObj
        **    holds a strong lock on the Doc. It is always best to have
        **    a memory guard around such critical sections of code. in
        **    this case, it is particularly important if we were an
        **    in-place active server and this drag ended up in a drop
        **    in our outer container. this scenario will lead to a
        **    crash if we do not hold this memory guard.
        */
        OleDoc_Lock(lpSrcOleDoc, TRUE, 0);

        /* if after the Drag/Drop modal (mouse capture) loop is finished
        **    and a drag MOVE operation was performed, then we must delete
        **    the lines that were dragged.
        */
        if ( GetScode(hrErr) == DRAGDROP_S_DROP
                        && (dwEffect & DROPEFFECT_MOVE) != 0 ) {

                int i,j,iEnd;
                LPLINE lplineFocusLine;

                /* disable repainting and sending data changed notifications
                **    until after all lines have been deleted.
                */
                OutlineDoc_SetRedraw ( (LPOUTLINEDOC)lpSrcOleDoc, FALSE );

                /* if the drop was local to our document, then we must take
                **    into account that the line indices of the original
source
                **    of the drag could have shifted because the dropped
lines
                **    have been inserted into our document. thus we will
                **    re-determine the source line indices.
                */
                if (lpSrcOleDoc->m_fLocalDrop) {
                        i = LineList_GetFocusLineIndex ( lpSrcLL );
                        lplineFocusLine = LineList_GetLine ( lpSrcLL, i );
                }

                for ( i = j = LineList_GetLineIndex(lpSrcLL,lplineStart ) ,
                         iEnd  =
LineList_GetLineIndex(lpSrcLL,lplineEnd ) ;
                         i <= iEnd ;
```

```
                          i++
                ) OutlineDoc_DeleteLine ((LPOUTLINEDOC)lpSrcOleDoc, j );

                LineList_RecalcMaxLineWidthInHimetric(lpSrcLL, 0);

                if (lpSrcOleDoc->m_fLocalDrop) {
                        i = LineList_GetLineIndex ( lpSrcLL, lplineFocusLine
);
                        LineList_SetFocusLine ( lpSrcLL, (WORD)i );
                }

                OutlineDoc_SetRedraw ( (LPOUTLINEDOC)lpSrcOleDoc, TRUE );

                /* if it is a local Drag/Drop move, we need to balance the
                **    SetRedraw(FALSE) call that was made in the
implementation
                **    of IDropTarget::Drop.
                */
                if (lpSrcOleDoc->m_fLocalDrop)
                        OutlineDoc_SetRedraw ( (LPOUTLINEDOC)lpSrcOleDoc,
TRUE );

                LineList_ForceRedraw ( lpSrcLL, FALSE );
        }

        OleDoc_Release(lpDragDoc);  // rel artificial AddRef above
        OleDoc_Lock(lpSrcOleDoc, FALSE, FALSE);  // unlock artificial lock
guard

        OLEDBG_END3
        return dwEffect;

error:
        OLEDBG_END3
        return dwEffect;
}



/*************************************************************************
** OleDoc::IDropSource interface implementation
*************************************************************************/

STDMETHODIMP OleDoc_DropSource_QueryInterface(
        LPDROPSOURCE            lpThis,
        REFIID                  riid,
        LPVOID FAR*             lplpvObj
)
{
        LPOLEDOC lpOleDoc = ((struct CDocDropSourceImpl FAR*)lpThis)-
>lpOleDoc;

        return OleDoc_QueryInterface(lpOleDoc, riid, lplpvObj);
}
```

```
STDMETHODIMP_(ULONG) OleDoc_DropSource_AddRef( LPDROPSOURCE lpThis )
{
        LPOLEDOC lpOleDoc = ((struct CDocDropSourceImpl FAR*)lpThis)-
>lpOleDoc;

        OleDbgAddRefMethod(lpThis, "IDropSource");

        return OleDoc_AddRef(lpOleDoc);
}


STDMETHODIMP_(ULONG) OleDoc_DropSource_Release ( LPDROPSOURCE lpThis)
{
        LPOLEDOC lpOleDoc = ((struct CDocDropSourceImpl FAR*)lpThis)-
>lpOleDoc;

        OleDbgReleaseMethod(lpThis, "IDropSource");

        return OleDoc_Release(lpOleDoc);
}


STDMETHODIMP    OleDoc_DropSource_QueryContinueDrag (
        LPDROPSOURCE            lpThis,
        BOOL                    fEscapePressed,
        DWORD                   grfKeyState
){
        if (fEscapePressed)
                return ResultFromScode(DRAGDROP_S_CANCEL);
        else if (!(grfKeyState & MK_LBUTTON))
                return ResultFromScode(DRAGDROP_S_DROP);
        else
                return NOERROR;
}


STDMETHODIMP    OleDoc_DropSource_GiveFeedback (
        LPDROPSOURCE            lpThis,
        DWORD                   dwEffect
)
{
        // Tell OLE to use the standard drag/drop feedback cursors
        return ResultFromScode(DRAGDROP_S_USEDEFAULTCURSORS);

#if defined( IF_SPECIAL_DD_CURSORS_NEEDED )
        switch (dwEffect) {

                case DROPEFFECT_NONE:
                        SetCursor ( ((LPOLEAPP)g_lpApp)-
>m_hcursorDragNone );
                        break;

                case DROPEFFECT_COPY:
                        SetCursor ( ((LPOLEAPP)g_lpApp)-
>m_hcursorDragCopy );
                        break;

                case DROPEFFECT_MOVE:
                        SetCursor ( ((LPOLEAPP)g_lpApp)-
>m_hcursorDragMove );
                        break;

                case DROPEFFECT_LINK:
                        SetCursor ( ((LPOLEAPP)g_lpApp)-
>m_hcursorDragLink );
                        break;

        }

        return NOERROR;
#endif

}

/*************************************************************************
** OleDoc::IDropTarget interface implementation
*************************************************************************/

STDMETHODIMP OleDoc_DropTarget_QueryInterface(
                LPDROPTARGET        lpThis,
                REFIID              riid,
                LPVOID FAR*         lplpvObj
)
{
        LPOLEDOC lpOleDoc = ((struct CDocDropTargetImpl FAR*)lpThis)-
>lpOleDoc;

        return OleDoc_QueryInterface(lpOleDoc, riid, lplpvObj);
}


STDMETHODIMP_(ULONG) OleDoc_DropTarget_AddRef(LPDROPTARGET lpThis)
{
        LPOLEDOC lpOleDoc = ((struct CDocDropTargetImpl FAR*)lpThis)-
>lpOleDoc;

        OleDbgAddRefMethod(lpThis, "IDropTarget");

        return OleDoc_AddRef(lpOleDoc);
}


STDMETHODIMP_(ULONG) OleDoc_DropTarget_Release ( LPDROPTARGET lpThis)
{
        LPOLEDOC lpOleDoc = ((struct CDocDropTargetImpl FAR*)lpThis)-
>lpOleDoc;

        OleDbgReleaseMethod(lpThis, "IDropTarget");
        return OleDoc_Release(lpOleDoc);
}


STDMETHODIMP    OleDoc_DropTarget_DragEnter (
        LPDROPTARGET            lpThis,
        LPDATAOBJECT            lpDataObj,
        DWORD                   grfKeyState,
        POINTL                  pointl,
        LPDWORD                 lpdwEffect
)
{
        LPOLEAPP   lpOleApp = (LPOLEAPP)g_lpApp;
        LPOLEDOC   lpOleDoc = ((struct CDocDropTargetImpl FAR*)lpThis)-
>lpOleDoc;
        LPLINELIST lpLL     = (LPLINELIST)&((LPOUTLINEDOC)lpOleDoc)-
>m_LineList;
        BOOL       fDragScroll;

        // artificial AddRef in case object is destroyed during call
        OleDoc_DropTarget_AddRef(lpThis);

        OLEDBG_BEGIN2("OleDoc_DropTarget_DragEnter\r\n")

        lpOleDoc->m_fDragLeave              = FALSE;
        lpOleDoc->m_dwTimeEnterScrollArea   = 0;
        lpOleDoc->m_dwLastScrollDir         = SCROLLDIR_NULL;


        /* Determine if the drag source data object offers a data format
        **    that we can copy and/or link to.
        */

        lpOleDoc->m_fCanDropCopy = OleDoc_QueryPasteFromData(
                        lpOleDoc,
                        lpDataObj,
                        FALSE   /* fLink */
        );

        lpOleDoc->m_fCanDropLink = OleDoc_QueryPasteFromData(
                        lpOleDoc,
                        lpDataObj,
                        TRUE   /* fLink */
        );

        fDragScroll = OleDoc_DoDragScroll ( lpOleDoc, pointl );

        if
(OleDoc_QueryDrop(lpOleDoc,grfKeyState,pointl,fDragScroll,lpdwEffect))
                LineList_SetDragOverLineFromPointl( lpLL, pointl );

        OLEDBG_END2

        // release artificial AddRef
        OleDoc_DropTarget_Release(lpThis);

        return NOERROR;
}

STDMETHODIMP OleDoc_DropTarget_DragOver (
        LPDROPTARGET            lpThis,
        DWORD                   grfKeyState,
        POINTL                  pointl,
        LPDWORD                 lpdwEffect
)
{
        LPOLEAPP   lpOleApp = (LPOLEAPP)g_lpApp;
        LPOLEDOC   lpOleDoc = ((struct CDocDropTargetImpl FAR*)lpThis)-
>lpOleDoc;
        LPLINELIST lpLL   = (LPLINELIST)&((LPOUTLINEDOC)lpOleDoc)-
>m_LineList;
        BOOL       fDragScroll;

        // artificial AddRef in case object is destroyed during call
        OleDoc_DropTarget_AddRef(lpThis);

        fDragScroll = OleDoc_DoDragScroll ( lpOleDoc, pointl );

        if (OleDoc_QueryDrop(lpOleDoc,grfKeyState,
pointl,fDragScroll,lpdwEffect))
                LineList_SetDragOverLineFromPointl( lpLL, pointl );
        else
                LineList_RestoreDragFeedback( lpLL );

        // release artificial AddRef
        OleDoc_DropTarget_Release(lpThis);
        return NOERROR;
}


STDMETHODIMP    OleDoc_DropTarget_DragLeave ( LPDROPTARGET lpThis)
{
        LPOLEDOC   lpOleDoc = ((struct CDocDropTargetImpl FAR*)lpThis)-
>lpOleDoc;
        LPLINELIST lpLL     = (LPLINELIST)&((LPOUTLINEDOC)lpOleDoc)-
>m_LineList;

        // artificial AddRef in case object is destroyed during call
        OleDoc_DropTarget_AddRef(lpThis);

        OLEDBG_BEGIN2("OleDoc_DropTarget_DragLeave\r\n")

        lpOleDoc->m_fDragLeave = TRUE;

        LineList_RestoreDragFeedback( lpLL );

        OLEDBG_END2

        // release artificial AddRef
        OleDoc_DropTarget_Release(lpThis);
        return NOERROR;

}

STDMETHODIMP    OleDoc_DropTarget_Drop (
        LPDROPTARGET            lpThis,
        LPDATAOBJECT            lpDataObj,
        DWORD                   grfKeyState,
        POINTL                  pointl,
        LPDWORD                 lpdwEffect
)
{
        LPOLEDOC   lpOleDoc = ((struct CDocDropTargetImpl FAR*)lpThis)-
>lpOleDoc;
        LPLINELIST lpLL     = (LPLINELIST)&((LPOUTLINEDOC)lpOleDoc)-
>m_LineList;

        // artificial AddRef in case object is destroyed during call
        OleDoc_DropTarget_AddRef(lpThis);

        OLEDBG_BEGIN2("OleDoc_DropTarget_Drop\r\n")

        lpOleDoc->m_fDragLeave = TRUE;
        lpOleDoc->m_fLocalDrop = TRUE;

        LineList_RestoreDragFeedback( lpLL );
        SetFocus( LineList_GetWindow( lpLL) );

        if (OleDoc_QueryDrop(lpOleDoc, grfKeyState, pointl, FALSE,
lpdwEffect)) {
                BOOL fLink     = (*lpdwEffect == DROPEFFECT_LINK);
                int iFocusLine = LineList_GetFocusLineIndex( lpLL );
                BOOL fStatus;

                OutlineDoc_SetRedraw ( (LPOUTLINEDOC)lpOleDoc, FALSE );
                LineList_SetFocusLineFromPointl ( lpLL, pointl );

                fStatus = OleDoc_PasteFromData(
                                lpOleDoc,
                                lpDataObj,
                                lpOleDoc->m_fLocalDrag, /* data source is
local to app */
                                fLink
                );

                // if drop was unsuccessfull, restore the original focus
line
                if (! fStatus)
                        LineList_SetFocusLine( lpLL, (WORD)iFocusLine );

#if defined( INPLACE_CNTR )
                {
                        LPCONTAINERDOC lpContainerDoc =
(LPCONTAINERDOC)lpOleDoc;
```

```
                        /* OLE2NOTE: if there is currently a UIActive OLE
object,
                        **    then we must tell it to UIDeactivate after
                        **    the drop has completed.
                        */
                        if (lpContainerDoc->m_lpLastUIActiveLine) {
                                ContainerLine_UIDeactivate(
                                                lpContainerDoc-
>m_lpLastUIActiveLine);
                        }
                }
#endif

#if defined( INPLACE_SVR )
                {
                        /* OLE2NOTE: if the drop was into a in-place visible
                        **    (in-place active but NOT UIActive object),
then we
                        **    want to UIActivate the object after the drop
is
                        **    complete.
                        */
                        ServerDoc_UIActivate((LPSERVERDOC) lpOleDoc);
                }
#endif


                /* if it is a local Drag/Drop move, don't enable redraw.
                **    after the source is done deleting the moved lines, it
                **    will re-enable redraw
                */
                if (! (lpOleDoc->m_fLocalDrag
                        && (*lpdwEffect & DROPEFFECT_MOVE) != 0 ))
                        OutlineDoc_SetRedraw ( (LPOUTLINEDOC)lpOleDoc,
TRUE );
        }

        OLEDBG_END2
        // release artificial AddRef
        OleDoc_DropTarget_Release(lpThis);
        return NOERROR;
}


#endif  // USE_DRAGDROP
```

FRAMETLS.H   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    frametls.h
**
**    This file contains file contains data structure defintions,
**    function prototypes, constants, etc. used by the frame level
**    tools used by the Outline series of sample applications. The
**    frame level tools include a formula bar and a button bar (toolbar)
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#if !defined( _FRAMETLS_H_ )
#define _FRAMETLS_H_

#ifndef RC_INVOKED
#pragma message ("INCLUDING FRAMETLS.H from " __FILE__)
#endif  /* RC_INVOKED */

#include "bttncur.h"
#include "gizmobar.h"

#define SPACE   5
#define POPUPSTUB_HEIGHT    5


/* forward type references */
typedef struct tagOUTLINEDOC FAR* LPOUTLINEDOC;

#define IDC_GIZMOBAR    1000
#define IDC_FORMULABAR  1001

#define IDB_CANCEL          0
#define IDB_EDITLINE        1
#define IDB_ADDLINE         2
#define IDB_UNINDENTLINE    3
#define IDB_INDENTLINE      4

#define BARSTATE_TOP        1
#define BARSTATE_BOTTOM     2
#define BARSTATE_POPUP      3
#define BARSTATE_HIDE       4

#define CLASS_PALETTE   "Tool Palette"

typedef struct tagBAR{
   UINT        m_uHeight;
   HWND        m_hWnd;
   int         m_nState;
} BAR, FAR* LPBAR;

typedef struct tagFRAMETOOLS {
   HWND        m_hWndPopupPalette;     // Popup Tool Palette window
   HWND        m_hWndApp;              // App Frame window
   UINT        m_uPopupWidth;          // Width of the popup palette
   HBITMAP     m_hBmp;                 // Image bitmaps
   BOOL        m_fInFormulaBar;        // does formula bar have edit focus
   BOOL        m_fToolsDisabled;       // when TRUE all tools are hidden

   BAR         m_ButtonBar;            // Button Bar
   BAR         m_FormulaBar;           // Formula Bar

   TOOLDISPLAYDATA m_tdd;      // from UIToolConfigureForDisplay
} FRAMETOOLS, FAR* LPFRAMETOOLS;


BOOL FrameToolsRegisterClass(HINSTANCE hInst);
BOOL FrameTools_Init(LPFRAMETOOLS lpft, HWND hWndParent, HINSTANCE hInst);
void FrameTools_AttachToFrame(LPFRAMETOOLS lpft, HWND hWndFrame);
void FrameTools_AssociateDoc(LPFRAMETOOLS lpft, LPOUTLINEDOC lpOutlineDoc);
void FrameTools_Destroy(LPFRAMETOOLS lpft);
void FrameTools_Move(LPFRAMETOOLS lpft, LPRECT lprcClient);
void FrameTools_PopupTools(LPFRAMETOOLS lpft);
void FrameTools_Enable(LPFRAMETOOLS lpft, BOOL fEnable);
void FrameTools_EnableWindow(LPFRAMETOOLS lpft, BOOL fEnable);

#if defined( INPLACE_CNTR ) || defined( INPLACE_SVR )
void FrameTools_NegotiateForSpaceAndShow(
      LPFRAMETOOLS            lpft,
      LPRECT                  lprcFrameRect,
      LPOLEINPLACEFRAME       lpTopIPFrame
);
#endif  // INPLACE_CNTR || INPLACE_SVR

void FrameTools_GetRequiredBorderSpace(LPFRAMETOOLS lpft, LPBORDERWIDTHS
lpBorderWidths);

void FrameTools_UpdateButtons(LPFRAMETOOLS lpft, LPOUTLINEDOC lpOutlineDoc);
void FrameTools_FB_SetEditText(LPFRAMETOOLS lpft, LPSTR lpsz);
void FrameTools_FB_GetEditText(LPFRAMETOOLS lpft, LPSTR lpsz, UINT cch);
void FrameTools_FB_FocusEdit(LPFRAMETOOLS lpft);
void FrameTools_FB_SendMessage(LPFRAMETOOLS lpft, UINT uID, UINT msg, WPARAM
wParam, LPARAM lParam);
void FrameTools_ForceRedraw(LPFRAMETOOLS lpft);
void FrameTools_BB_SetState(LPFRAMETOOLS lpft, int nState);
void FrameTools_FB_SetState(LPFRAMETOOLS lpft, int nState);
int FrameTools_BB_GetState(LPFRAMETOOLS lpft);
int FrameTools_FB_GetState(LPFRAMETOOLS lpft);
LRESULT FAR PASCAL FrameToolsWndProc(HWND hWnd, UINT Message, WPARAM wParam,
LPARAM lParam);

#endif // _FRAMETLS_H_
_
```

FRAMETLS.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Server Sample Code
**
**    frametls.c
**
**    This file contains all FrameTools methods and related support
**    functions. The FrameTools object is an encapsulation of the apps
**    formula bar and button bar.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"

OLEDBGDATA

/* private function prototype */
static void Bar_Move(LPBAR lpbar, LPRECT lprcClient, LPRECT lprcPopup);
static void FB_ResizeEdit(LPBAR lpbar);

extern LPOUTLINEAPP g_lpApp;
extern RECT g_rectNull;

/*
 * FrameToolsRegisterClass
 *
 * Purpose:
 *  Register the popup toolbar window class
 *
 * Parameters:
 *  hInst           Process instance
 *
 * Return Value:
 *  TRUE            if successful
 *  FALSE           if failed
 *
 */
BOOL FrameToolsRegisterClass(HINSTANCE hInst)
{
     WNDCLASS wc;

     // Register Tool Palette Class
     wc.style = CS_BYTEALIGNWINDOW;
     wc.lpfnWndProc = FrameToolsWndProc;
     wc.cbClsExtra = 0;
     wc.cbWndExtra = 4;
     wc.hInstance = hInst;
     wc.hIcon = NULL;
     wc.hCursor = LoadCursor(NULL, IDC_ARROW);
     wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
     wc.lpszMenuName = NULL;
     wc.lpszClassName = CLASS_PALETTE;

     if (!RegisterClass(&wc))
           return FALSE;
     else
           return TRUE;
}


static BOOL FrameTools_CreatePopupPalette(LPFRAMETOOLS lpft, HWND hWndFrame)
{
     if (lpft->m_hWndPopupPalette)
           DestroyWindow(lpft->m_hWndPopupPalette);

     lpft->m_hWndPopupPalette = CreateWindow(
           CLASS_PALETTE,
           "Tool Palette",
           WS_POPUP | WS_CAPTION | WS_CLIPCHILDREN,
           CW_USEDEFAULT, 0, 0, 0,
           hWndFrame,
           (HMENU)NULL,
           g_lpApp->m_hInst,
           0L
     );

     if (!lpft->m_hWndPopupPalette)
           return FALSE;

     SetWindowLong(lpft->m_hWndPopupPalette, 0, (LONG)lpft);
     return TRUE;
}


/*
 * FrameTools_Init
 *
 * Purpose:
 *  Init and create the toolbar
 *
 * Parameters:
 *  lpft            FrameTools object
 *  hWndParent      The window which owns the toolbar
 *  hInst           Process instance
 *
 * Return Value:
 *  TRUE            if successful
 *  FALSE           if failed
 *
 */
BOOL FrameTools_Init(LPFRAMETOOLS lpft, HWND hWndParent, HINSTANCE hInst)
{
     RECT        rc;
     UINT        uPos;
     UINT        dx;
     UINT        dy;

     if (!lpft || !hWndParent || !hInst)
           return FALSE;

     //Get BTTNCUR's display information
     UIToolConfigureForDisplay(&lpft->m_tdd);

     dx=lpft->m_tdd.cxButton;
     dy=lpft->m_tdd.cyButton;

     // 15 is calculated from the total number of buttons and separators
     lpft->m_uPopupWidth = dx * 15;

     lpft->m_hWndApp = hWndParent;
     lpft->m_ButtonBar.m_nState = BARSTATE_TOP;
     lpft->m_FormulaBar.m_nState = BARSTATE_TOP;
     lpft->m_fInFormulaBar = FALSE;

     lpft->m_fToolsDisabled = FALSE;

     lpft->m_ButtonBar.m_uHeight = lpft->m_tdd.cyBar;
     lpft->m_FormulaBar.m_uHeight = lpft->m_tdd.cyBar;


     //Get our image bitmaps for the display type we're on
     if (72 == lpft->m_tdd.uDPI)
           lpft->m_hBmp = LoadBitmap(hInst, (LPCSTR)"Image72");
     if (96 == lpft->m_tdd.uDPI)
           lpft->m_hBmp = LoadBitmap(hInst, (LPCSTR)"Image96");
     if (120 == lpft->m_tdd.uDPI)
           lpft->m_hBmp = LoadBitmap(hInst, (LPCSTR)"Image120");

     if (!lpft->m_hBmp)
           return FALSE;

     /* Create Popup Tool Palette window */
     lpft->m_hWndPopupPalette = NULL;
     if (! FrameTools_CreatePopupPalette(lpft, hWndParent))
           return FALSE;

     uPos = 0;
     //Create the GizmoBar and the client area window
     GetClientRect(hWndParent, &rc);
     lpft->m_ButtonBar.m_hWnd = CreateWindow(
           CLASS_GIZMOBAR,
           "ButtonBar",
           WS_CHILD | WS_VISIBLE,
           0, 0, rc.right-rc.left, lpft->m_tdd.cyBar,
           hWndParent,
           (HMENU)IDC_GIZMOBAR,
           hInst,
           0L
     );
```

```
     if (!lpft->m_ButtonBar.m_hWnd)
           return FALSE;


     SendMessage(lpft->m_ButtonBar.m_hWnd, WM_SETREDRAW, FALSE, 0L);

     //File new, open, save, print
     GBGizmoAdd(lpft->m_ButtonBar.m_hWnd, GIZMOTYPE_BUTTONCOMMAND, uPos++,
IDM_F_NEW, dx, dy, NULL, NULL, TOOLIMAGE_FILENEW, GIZMO_NORMAL);
     GBGizmoAdd(lpft->m_ButtonBar.m_hWnd, GIZMOTYPE_BUTTONCOMMAND, uPos++,
IDM_F_OPEN, dx, dy, NULL, NULL, TOOLIMAGE_FILEOPEN, GIZMO_NORMAL);
     GBGizmoAdd(lpft->m_ButtonBar.m_hWnd, GIZMOTYPE_BUTTONCOMMAND, uPos++,
IDM_F_SAVE, dx, dy, NULL, NULL, TOOLIMAGE_FILESAVE, GIZMO_NORMAL);
     GBGizmoAdd(lpft->m_ButtonBar.m_hWnd, GIZMOTYPE_BUTTONCOMMAND, uPos++,
IDM_F_PRINT, dx, dy, NULL, NULL, TOOLIMAGE_FILEPRINT, GIZMO_NORMAL);

     // separator
     GBGizmoAdd(lpft->m_ButtonBar.m_hWnd, GIZMOTYPE_SEPARATOR, uPos++, 0,
dx/2, dy, NULL, NULL, 0, GIZMO_NORMAL);

     // Edit cut, copy, paste
     GBGizmoAdd(lpft->m_ButtonBar.m_hWnd, GIZMOTYPE_BUTTONCOMMAND, uPos++,
IDM_E_CUT, dx, dy, NULL, NULL, TOOLIMAGE_EDITCUT, GIZMO_NORMAL);
     GBGizmoAdd(lpft->m_ButtonBar.m_hWnd, GIZMOTYPE_BUTTONCOMMAND, uPos++,
IDM_E_COPY, dx, dy, NULL, NULL, TOOLIMAGE_EDITCOPY, GIZMO_NORMAL);
     GBGizmoAdd(lpft->m_ButtonBar.m_hWnd, GIZMOTYPE_BUTTONCOMMAND, uPos++,
IDM_E_PASTE, dx, dy, NULL, NULL, TOOLIMAGE_EDITPASTE, GIZMO_NORMAL);

     // separator
     GBGizmoAdd(lpft->m_ButtonBar.m_hWnd, GIZMOTYPE_SEPARATOR, uPos++, 0,
dx/2, dy, NULL, NULL, 0, GIZMO_NORMAL);

     // Line indent, unindent
     GBGizmoAdd(lpft->m_ButtonBar.m_hWnd, GIZMOTYPE_BUTTONCOMMAND, uPos++,
IDM_L_UNINDENTLINE, dx, dy, NULL, lpft->m_hBmp, IDB_UNINDENTLINE,
GIZMO_NORMAL);
     GBGizmoAdd(lpft->m_ButtonBar.m_hWnd, GIZMOTYPE_BUTTONCOMMAND, uPos++,
IDM_L_INDENTLINE, dx, dy, NULL, lpft->m_hBmp, IDB_INDENTLINE, GIZMO_NORMAL);

     // separator
     GBGizmoAdd(lpft->m_ButtonBar.m_hWnd, GIZMOTYPE_SEPARATOR, uPos++, 0,
dx/2, dy, NULL, NULL, 0, GIZMO_NORMAL);

     // Help
     GBGizmoAdd(lpft->m_ButtonBar.m_hWnd, GIZMOTYPE_BUTTONCOMMAND, uPos++,
IDM_H_ABOUT, dx, dy, NULL, NULL, TOOLIMAGE_HELP, GIZMO_NORMAL);

     SendMessage(lpft->m_ButtonBar.m_hWnd, WM_SETREDRAW, TRUE, 0L);


     uPos = 0;
     lpft->m_FormulaBar.m_hWnd = CreateWindow(
           CLASS_GIZMOBAR,
           "FormulaBar",
           WS_CHILD | WS_VISIBLE | WS_CLIPSIBLINGS,
           0, lpft->m_tdd.cyBar, rc.right-rc.left, lpft->m_tdd.cyBar,
           hWndParent,
           (HMENU)IDC_FORMULABAR,
           hInst,
           0L
     );

     if (!lpft->m_FormulaBar.m_hWnd)
           return FALSE;

     SendMessage(lpft->m_FormulaBar.m_hWnd, WM_SETREDRAW, FALSE, 0L);

     // Line add line
     GBGizmoAdd(lpft->m_FormulaBar.m_hWnd, GIZMOTYPE_BUTTONCOMMAND, uPos++,
IDM_L_ADDLINE, dx, dy, NULL, lpft->m_hBmp, IDB_ADDLINE, GIZMO_NORMAL);

     // separator
     GBGizmoAdd(lpft->m_FormulaBar.m_hWnd, GIZMOTYPE_SEPARATOR, uPos++, 0,
dx/2, dy, NULL, NULL, 0, GIZMO_NORMAL);

     // Line edit line, Cancel
     GBGizmoAdd(lpft->m_FormulaBar.m_hWnd, GIZMOTYPE_BUTTONCOMMAND, uPos++,
IDM_L_EDITLINE, dx, dy, NULL, lpft->m_hBmp, IDB_EDITLINE, GIZMO_NORMAL);
     GBGizmoAdd(lpft->m_FormulaBar.m_hWnd, GIZMOTYPE_BUTTONCOMMAND, uPos++,
IDM_FB_CANCEL, dx, dy, NULL, lpft->m_hBmp, IDB_CANCEL, GIZMO_NORMAL);

     // separator
     GBGizmoAdd(lpft->m_FormulaBar.m_hWnd, GIZMOTYPE_SEPARATOR, uPos++, 0,
dx/2, dy, NULL, NULL, 0, GIZMO_NORMAL);

     // Edit control for line input
     GBGizmoAdd(lpft->m_FormulaBar.m_hWnd, GIZMOTYPE_EDIT, uPos++,
IDM_FB_EDIT, dx*10, lpft->m_tdd.cyBar-5, NULL, NULL, 0, GIZMO_NORMAL);


     SendMessage(lpft->m_FormulaBar.m_hWnd, WM_SETREDRAW, TRUE, 0L);

     // Limit the text lenght of edit control
     GBGizmoSendMessage(lpft->m_FormulaBar.m_hWnd, IDM_FB_EDIT,
EM_LIMITTEXT,
           (WPARAM)MAXSTRLEN, 0L);

     //Set the GizmoBar's associate to be this client window
     GBHwndAssociateSet(lpft->m_ButtonBar.m_hWnd, hWndParent);

     //Set the FormulaBar's associate to be this client window
     GBHwndAssociateSet(lpft->m_FormulaBar.m_hWnd, hWndParent);

     return TRUE;
}


void FrameTools_AttachToFrame(LPFRAMETOOLS lpft, HWND hWndFrame)
{
     if (! lpft)
```

```
            return;

      if (hWndFrame == NULL)
            hWndFrame = OutlineApp_GetFrameWindow((LPOUTLINEAPP)g_lpApp);

      if (lpft->m_hWndApp == hWndFrame)
            return;     // already have this parent frame

      lpft->m_hWndApp = hWndFrame;

      /* parent the tool bars to the frame so we can safely
      **    destroy/recreate the palette window.
      */
      SetParent(lpft->m_ButtonBar.m_hWnd, hWndFrame);
      SetParent(lpft->m_FormulaBar.m_hWnd, hWndFrame);

      // recreate popup palette so that it is owned by the hWndFrame
      FrameTools_CreatePopupPalette(lpft, hWndFrame);

      // restore the correct parent for the tool bars
      FrameTools_BB_SetState(lpft, lpft->m_ButtonBar.m_nState);
      FrameTools_FB_SetState(lpft, lpft->m_FormulaBar.m_nState);
}


void FrameTools_AssociateDoc(LPFRAMETOOLS lpft, LPOUTLINEDOC lpOutlineDoc)
{
      HWND hWnd = OutlineDoc_GetWindow(lpOutlineDoc);

      if (! lpft)
            return;

      // if no Doc is given, then associate with the App's frame window.
      if (lpOutlineDoc)
            hWnd = OutlineDoc_GetWindow(lpOutlineDoc);
      else
            hWnd = OutlineApp_GetWindow((LPOUTLINEAPP)g_lpApp);

      //Set the GizmoBar's associate to be this client window
      GBHwndAssociateSet(lpft->m_ButtonBar.m_hWnd, hWnd);

      //Set the FormulaBar's associate to be this client window
      GBHwndAssociateSet(lpft->m_FormulaBar.m_hWnd, hWnd);
}


/*
 * FrameTools_Destroy
 *
 * Purpose:
 *  Destroy the toolbar
 *
 * Parameters:
 *  lpft            FrameTools object
 *
 * Return Value:
 *  nil
 */
void FrameTools_Destroy(LPFRAMETOOLS lpft)
{
      if (!lpft)
            return;

      if (IsWindow(lpft->m_ButtonBar.m_hWnd))
            DestroyWindow(lpft->m_ButtonBar.m_hWnd);
      if (IsWindow(lpft->m_FormulaBar.m_hWnd))
            DestroyWindow(lpft->m_FormulaBar.m_hWnd);
      if (IsWindow(lpft->m_hWndPopupPalette))
            DestroyWindow(lpft->m_hWndPopupPalette);

      if (lpft->m_hBmp)
            DeleteObject(lpft->m_hBmp);
}


/*
 * FrameTools_Move
 *
 * Purpose:
 *  Move and resize the toolbar
 *
 * Parameters:
 *  lpft            FrameTools object
 *  lprc            Pointer to client rectangle
 *
 * Return Value:
 *  nil
 */
void FrameTools_Move(LPFRAMETOOLS lpft, LPRECT lprcClient)
{
      RECT rcPopup;
      LPRECT lprcPopup = (LPRECT)&rcPopup;
      int nCmdShow = SW_HIDE;

      if (!lpft || lpft->m_fToolsDisabled)
            return;

      lprcPopup->left = 0;
      lprcPopup->top = 0;
      lprcPopup->right = lpft->m_uPopupWidth;
      lprcPopup->bottom = lpft->m_ButtonBar.m_uHeight +
                  lpft->m_FormulaBar.m_uHeight;

      switch (lpft->m_ButtonBar.m_nState) {
            case BARSTATE_HIDE:
            case BARSTATE_POPUP:
            case BARSTATE_TOP:
                  Bar_Move(&lpft->m_ButtonBar, lprcClient, lprcPopup);
                  Bar_Move(&lpft->m_FormulaBar, lprcClient, lprcPopup);
                  break;
            case BARSTATE_BOTTOM:
                  Bar_Move(&lpft->m_FormulaBar, lprcClient, lprcPopup);
                  Bar_Move(&lpft->m_ButtonBar, lprcClient, lprcPopup);
                  break;
      }

      if (lprcPopup->top) {
            SetWindowPos(lpft->m_hWndPopupPalette, NULL, 0, 0, lprcPopup-
 >right,
                        lprcPopup->top + GetSystemMetrics(SM_CYCAPTION),
                        SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE |
 SWP_SHOWWINDOW);
      }
      else
            ShowWindow(lpft->m_hWndPopupPalette, SW_HIDE);

      FB_ResizeEdit(&lpft->m_FormulaBar);

      InvalidateRect(lpft->m_ButtonBar.m_hWnd, NULL, TRUE);
      InvalidateRect(lpft->m_FormulaBar.m_hWnd, NULL, TRUE);
 }


 /*
  * FrameTools_PopupTools
  *
  * Purpose:
  *  Put both formula bar and button bar in Popup Window.
  *
  * Parameters:
  *  lpft            FrameTools object
  *
  * Return Value:
  *  nil
  */
 void FrameTools_PopupTools(LPFRAMETOOLS lpft)
 {
      if (! lpft)
            return;

      FrameTools_BB_SetState(lpft, BARSTATE_POPUP);
      FrameTools_FB_SetState(lpft, BARSTATE_POPUP);
      FrameTools_Move(lpft, NULL);
 }


 /*
  * FrameTools_Enable
  *
  * Purpose:
  *  Enable/Disable(hide) all the tools of the toolbar.
  *  this will hide both the buttonbar and the
  *  formulabar independent of whether they are floating or anchored.
  *
  * Parameters:
  *  lpft            FrameTools object
  *  fEnable
  *
  * Return Value:
  *  nil
  */
 void FrameTools_Enable(LPFRAMETOOLS lpft, BOOL fEnable)
 {
      lpft->m_fToolsDisabled = !fEnable;
      if (lpft->m_fToolsDisabled) {
            ShowWindow(lpft->m_hWndPopupPalette, SW_HIDE);
            ShowWindow(lpft->m_ButtonBar.m_hWnd, SW_HIDE);
            ShowWindow(lpft->m_FormulaBar.m_hWnd, SW_HIDE);
      }
 }


 /*
  * FrameTools_EnableWindow
  *
  * Purpose:
  *  EnableWindow for all the tools of the toolbar.
  *  this enables/disables mouse and keyboard input to the tools.
  *  while a modal dialog is up, it is inportant to disable the
  *  floating tool windows.
  *  this will NOT hide any windows; it will only call EnableWindow.
  *
  * Parameters:
  *  lpft            FrameTools object
  *  fEnable
  *
  * Return Value:
  *  nil
  */
 void FrameTools_EnableWindow(LPFRAMETOOLS lpft, BOOL fEnable)
 {
      EnableWindow(lpft->m_hWndPopupPalette, fEnable);
      EnableWindow(lpft->m_ButtonBar.m_hWnd, fEnable);
      EnableWindow(lpft->m_FormulaBar.m_hWnd, fEnable);
 }


 #if defined( INPLACE_CNTR ) || defined( INPLACE_SVR )

 /*
  * FrameTools_NegotiateForSpaceAndShow
  *
  * Purpose:
  *  Negotiate for space for the toolbar tools with the given frame window.
  *  and make them visible.
  *  Negotiation steps:
  *     1. try to get enough space at top/bottom of window
  *     2. float the tools as a palette if space not available
  *
  * Parameters:
  *  lpft                FrameTools object
  *
  * Return Value:
  *  none
  */
 void FrameTools_NegotiateForSpaceAndShow(
            LPFRAMETOOLS            lpft,
            LPRECT                  lprcFrameRect,
            LPOLEINPLACEFRAME       lpTopIPFrame
 )
 {
      BORDERWIDTHS    borderwidths;
      RECT            rectBorder;
      HRESULT         hrErr;

      if (lprcFrameRect)
            rectBorder = *lprcFrameRect;
      else {
            /* OLE2NOTE: by calling GetBorder, the server can find out the
            **    size of the frame window. it can use this information to
            **    make decisions about how to orient/organize it tools (eg.
            **    if window is taller than wide put tools vertically at
            **    left edge).
            */
            OLEDBG_BEGIN2("IOleInPlaceFrame::GetBorder called\r\n")
            hrErr = lpTopIPFrame->lpVtbl->GetBorder(
                        lpTopIPFrame,
                        (LPRECT)&rectBorder
            );
            OLEDBG_END2
      }

      /* Try SetBorderSpace() with the space that you need. If it fails then
      ** you can negotiate for space and then do the SetBorderSpace().
      */
      FrameTools_GetRequiredBorderSpace(lpft,(LPBORDERWIDTHS)&borderwidths);
      OLEDBG_BEGIN2("IOleInPlaceFrame::SetBorderSpace called\r\n")
      hrErr = lpTopIPFrame->lpVtbl->SetBorderSpace(
                  lpTopIPFrame,
                  (LPCBORDERWIDTHS)&borderwidths
      );
      OLEDBG_END2

 #if defined( LATER )
      if (hrErr != NOERROR) {
            /* Frame did not give the toolsspace that we want. So negotiate
 */

            // REVIEW: try a different placement of the tools here

            OLEDBG_BEGIN2("IOleInPlaceFrame::RequestBorderSpace called\r\n")
            hrErr = lpTopIPFrame->lpVtbl->RequestBorderSpace(
                        lpTopIPFrame,
                        (LPCBORDERWIDTHS)&borderwidths
```

```
            );
            OLEDBG_END2

            if (hrErr == NOERROR) {
                  OLEDBG_BEGIN2("IOleInPlaceFrame::SetBorderSpace
called\r\n")
                  hrErr = lpTopIPFrame->lpVtbl->SetBorderSpace(
                              lpTopIPFrame,
                              (LPCBORDERWIDTHS)&borderwidths
                  );
                  OLEDBG_END2
            }
      }
#endif

      if (hrErr == NOERROR) {
            FrameTools_Move(lpft, (LPRECT)&rectBorder);   // we got what we
wanted
      } else {
            /* We did not get tool space, so POP them up.
            /* OLE2NOTE: since we are poping up our tools, we MUST inform
            **    the top in-place frame window that we need NO tool space
            **    BUT that it should NOT put its own tools up. if we were
            **    to pass NULL instead of (0,0,0,0), then the container
            **    would have the option to leave its own tools up.
            */
            OLEDBG_BEGIN2("IOleInPlaceFrame::SetBorderSpace(NULL)
called\r\n")
            hrErr = lpTopIPFrame->lpVtbl->SetBorderSpace(
                        lpTopIPFrame,
                        (LPCBORDERWIDTHS)&g_rectNull
            );
            OLEDBG_END2
            FrameTools_PopupTools(lpft);
      }
}

#endif  // INPLACE_CNTR || INPLACE_SVR


/*
 * FrameTools_GetRequiredBorderSpace
 *
 * Purpose:
 *  Calculate the desired space for the toolbar tools.
 *
 * Parameters:
 *  lpft            FrameTools object
 *  lpBorderWidths  Widths required at top,bottom,left,right
 *
 * Return Value:
 *  nil
 */
void FrameTools_GetRequiredBorderSpace(LPFRAMETOOLS lpft, LPBORDERWIDTHS
lpBorderWidths)
```

```
{
     *lpBorderWidths = g_rectNull;

     switch (lpft->m_ButtonBar.m_nState) {
           case BARSTATE_TOP:
                 lpBorderWidths->top += lpft->m_ButtonBar.m_uHeight;
                 break;

           case BARSTATE_BOTTOM:
                 lpBorderWidths->bottom += lpft->m_ButtonBar.m_uHeight;
                 break;
     }

     switch (lpft->m_FormulaBar.m_nState) {
           case BARSTATE_TOP:
                 lpBorderWidths->top += lpft->m_FormulaBar.m_uHeight;
                 break;

           case BARSTATE_BOTTOM:
                 lpBorderWidths->bottom += lpft->m_FormulaBar.m_uHeight;
                 break;
     }
}



/*
 * FrameTools_UpdateButtons
 *
 * Purpose:
 *  Enable/disable individual buttons of the toolbar according to the
 *  state of the app
 *
 * Parameters:
 *  lpft            FrameTools object
 *
 * Return Value:
 *  nil
 */
void FrameTools_UpdateButtons(LPFRAMETOOLS lpft, LPOUTLINEDOC lpOutlineDoc)
{
     BOOL            fEnable;

#if defined( OLE_VERSION )
     LPDATAOBJECT    lpClipboardDataObj;
     HRESULT         hrErr;
     LPOLEAPP        lpOleApp = (LPOLEAPP)g_lpApp;
     BOOL            fPrevEnable1;
     BOOL            fPrevEnable2;
#endif

     if (!lpft)
           return;

#if defined( INPLACE_CNTR )
```

```
     {
           LPCONTAINERDOC lpContainerDoc = (LPCONTAINERDOC)lpOutlineDoc;
           if (lpContainerDoc->m_lpLastUIActiveLine &&
                 lpContainerDoc->m_lpLastUIActiveLine->m_fUIActive) {

                 /* if there is a UIActive object, then we should disable
                 **    all of our "active editor" commands. we should enable
                 **    only those commands that are "workspace" commands.
                 */
                 if (lpft->m_FormulaBar.m_nState != BARSTATE_HIDE) {

                       GBGizmoEnable(lpft-
>m_FormulaBar.m_hWnd,IDM_L_EDITLINE,FALSE);
                       GBGizmoEnable(lpft-
>m_FormulaBar.m_hWnd,IDM_L_ADDLINE,FALSE);
                       GBGizmoEnable(lpft-
>m_FormulaBar.m_hWnd,IDM_FB_CANCEL,FALSE);
                       GBGizmoEnable(lpft-
>m_FormulaBar.m_hWnd,IDM_L_EDITLINE,FALSE);
                 }

                 if (lpft->m_ButtonBar.m_nState != BARSTATE_HIDE)
                 {
                       GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_E_CUT,
FALSE);
                       GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_E_COPY,
FALSE);
                       GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_E_PASTE,
FALSE);
                       GBGizmoEnable(lpft-
>m_ButtonBar.m_hWnd,IDM_L_INDENTLINE,FALSE);
                       GBGizmoEnable(lpft->m_ButtonBar.m_hWnd,
IDM_L_UNINDENTLINE, FALSE);
                       GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_H_ABOUT,
FALSE);
                       GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_FB_EDIT,
FALSE);
                       GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_F_NEW,
TRUE);
                       GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_F_OPEN,
TRUE);
                       GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_F_SAVE,
TRUE);
                 }
                 return;
           }
     }
#endif    // INPLACE_CNTR

     fEnable = (BOOL)OutlineDoc_GetLineCount(lpOutlineDoc);

     if (lpft->m_FormulaBar.m_nState != BARSTATE_HIDE) {

           GBGizmoEnable(lpft->m_FormulaBar.m_hWnd, IDM_L_EDITLINE,
fEnable);
```

```
            if (! lpft->m_fInFormulaBar) {
                  GBGizmoEnable(lpft->m_FormulaBar.m_hWnd, IDM_L_ADDLINE,
FALSE);
                  GBGizmoEnable(lpft->m_FormulaBar.m_hWnd, IDM_FB_CANCEL,
FALSE);
                  GBGizmoEnable(lpft->m_FormulaBar.m_hWnd, IDM_L_EDITLINE,
FALSE);
                  if (!fEnable) {
                        GBGizmoTextSet(lpft->m_FormulaBar.m_hWnd,
IDM_FB_EDIT, "");
                  }
            } else {
                  GBGizmoEnable(lpft->m_FormulaBar.m_hWnd, IDM_L_ADDLINE,
TRUE);
                  GBGizmoEnable(lpft->m_FormulaBar.m_hWnd, IDM_FB_CANCEL,
TRUE);
            }
      }

      if (lpft->m_ButtonBar.m_nState != BARSTATE_HIDE)
      {
            GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_E_CUT, fEnable);
            GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_E_COPY, fEnable);
            GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_L_INDENTLINE,
fEnable);
            GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_L_UNINDENTLINE,
fEnable);

#if defined( OLE_SERVER )

            {
                  LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOutlineDoc;

#if defined( INPLACE_SVR )
                  fEnable = ((lpServerDoc->m_fUIActive) ? FALSE : TRUE);
#else
                  fEnable = (lpOutlineDoc->m_docInitType !=
DOCTYPE_EMBEDDED);
#endif  // INPLACE_SVR

                  GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_F_NEW,
fEnable);
                  GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_F_OPEN,
fEnable);
                  GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_F_SAVE,
fEnable);
            }

#endif  // OLE_SERVER

#if defined( OLE_VERSION )

            /* OLE2NOTE: we do not want to ever give the busy dialog when we
            **    are trying to enable or disable our tool bar buttons eg.
```

```
           **    even if the source of data on the clipboard is busy, we do
           **    not want put up the busy dialog. thus we will disable the
           **    dialog and at the end re-enable it.
           */
           OleApp_DisableBusyDialogs(lpOleApp, &fPrevEnable1, 
&fPrevEnable2);

           /* OLE2NOTE: perform OLE specific menu initialization.
           **    the OLE versions use the OleGetClipboard mechanism for
           **    clipboard handling. thus, they determine if the Paste
           **    command should be enabled in an OLE specific manner.
           */
           fEnable = FALSE;
           hrErr = OleGetClipboard((LPDATAOBJECT FAR*)&lpClipboardDataObj);

           if (hrErr == NOERROR) {
                 int nFmtEtc;

                 nFmtEtc = OleStdGetPriorityClipboardFormat(
                             lpClipboardDataObj,
                             lpOleApp->m_arrPasteEntries,
                             lpOleApp->m_nPasteEntries
                 );

                 fEnable = (nFmtEtc >= 0);  // there IS a format we like

                 OleStdRelease((LPUNKNOWN)lpClipboardDataObj);
           }

           // re-enable the busy dialog
           OleApp_EnableBusyDialogs(lpOleApp, fPrevEnable1, fPrevEnable2);

           GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_E_PASTE, fEnable);

#else

           // Base Outline version uses standard Windows clipboard handling
           if(IsClipboardFormatAvailable(g_lpApp->m_cfOutline) ||
                       IsClipboardFormatAvailable(CF_TEXT))
                 GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_E_PASTE, TRUE);
           else
                 GBGizmoEnable(lpft->m_ButtonBar.m_hWnd, IDM_E_PASTE, 
FALSE);

#endif  // OLE_VERSION

     }
}

/*
 * FrameTools_FB_SetEditText
 *
 * Purpose:
 *  Set text in the edit control in FormulaBar
 *
 * Parameters:
 *  lpft            FrameTools object
 *  lpsz            pointer to string to be set
 *
 * Return Value:
 *  nil
 */
void FrameTools_FB_SetEditText(LPFRAMETOOLS lpft, LPSTR lpsz)
{
     GBGizmoTextSet(lpft->m_FormulaBar.m_hWnd, IDM_FB_EDIT, lpsz);
}


/*
 * FrameTools_FB_GetEditText
 *
 * Purpose:
 *  Get text from the edit control in FormulaBar
 *
 * Parameters:
 *  lpft            FrameTools object
 *  lpsz            pointer to buffer to receive the text
 *  cch             buffer size
 *
 * Return Value:
 *  nil
 */
void FrameTools_FB_GetEditText(LPFRAMETOOLS lpft, LPSTR lpsz, UINT cch)
{
     GBGizmoTextGet(lpft->m_FormulaBar.m_hWnd, IDM_FB_EDIT, lpsz, cch);
}


/*
 * FrameTools_FB_FocusEdit
 *
 * Purpose:
 *  Set the focus in the edit control of FormulaBar
 *
 * Parameters:
 *  lpft            FrameTools object
 *
 * Return Value:
 *  nil
 */
void FrameTools_FB_FocusEdit(LPFRAMETOOLS lpft)
{
     GBGizmoFocusSet(lpft->m_FormulaBar.m_hWnd, IDM_FB_EDIT);

     // select the whole text in the edit control
     GBGizmoSendMessage(lpft->m_FormulaBar.m_hWnd, IDM_FB_EDIT, EM_SETSEL,
                 (WPARAM)TRUE, MAKELPARAM(0, -1));
}
```

```
/*
 * FrameTools_FB_SendMessage
 *
 * Purpose:
 *  Send a message to the FormulaBar window gizmo
 *
 * Parameters:
 *  lpft            FrameTools object
 *  uID             gizmo ID
 *  msg
 *  wParam
 *  lParam
 *
 * Return Value:
 *  nil
 */
void FrameTools_FB_SendMessage(LPFRAMETOOLS lpft, UINT uID, UINT msg, WPARAM
wParam, LPARAM lParam)
{
     GBGizmoSendMessage(lpft->m_FormulaBar.m_hWnd, uID, msg, wParam,
lParam);
}


/*
 * FrameTools_FB_ForceRedraw
 *
 * Purpose:
 *  Force the toolbar to draw itself
 *
 * Parameters:
 *  lpft            FrameTools object
 *
 * Return Value:
 *  nil
 */
void FrameTools_ForceRedraw(LPFRAMETOOLS lpft)
{
     InvalidateRect(lpft->m_ButtonBar.m_hWnd, NULL, TRUE);
     InvalidateRect(lpft->m_FormulaBar.m_hWnd, NULL, TRUE);
     InvalidateRect(lpft->m_hWndPopupPalette, NULL, TRUE);
}


/*
 * FrameTools_BB_SetState
 *
 * Purpose:
 *  Set display state of ButtonBar
 *
 * Parameters:
 *  lpft            FrameTools object
 *  nState          new display state
 *
 * Return Value:
```

```
 *  nil
 */
void FrameTools_BB_SetState(LPFRAMETOOLS lpft, int nState)
{
     if (!lpft) {
           return;
     }

     lpft->m_ButtonBar.m_nState = nState;

     if (nState == BARSTATE_POPUP)
           SetParent(lpft->m_ButtonBar.m_hWnd, lpft->m_hWndPopupPalette);
     else
           SetParent(lpft->m_ButtonBar.m_hWnd, lpft->m_hWndApp);
}


/*
 * FrameTools_BB_GetState
 *
 * Purpose:
 *  Get display state of ButtonBar
 *
 * Parameters:
 *  lpft            FrameTools object
 *
 * Return Value:
 *  nState          current display state
 */
int FrameTools_BB_GetState(LPFRAMETOOLS lpft)
{
     return lpft->m_ButtonBar.m_nState;
}


/*
 * FrameTools_FB_SetState
 *
 * Purpose:
 *  Set display state of FormulaBar
 *
 * Parameters:
 *  lpft            FrameTools object
 *  nState          new display state
 *
 * Return Value:
4 *  nil
 */
void FrameTools_FB_SetState(LPFRAMETOOLS lpft, int nState)
{
     if (!lpft) {
           return;
     }

     lpft->m_FormulaBar.m_nState = nState;
     if (nState == BARSTATE_POPUP)
          SetParent(lpft->m_FormulaBar.m_hWnd, lpft->m_hWndPopupPalette);

#if defined( INPLACE_SVR )
     /* OLE2NOTE: it is dangerous for an in-place server to hide its
     **    toolbar window and leave it parented to the hWndFrame of the
     **    in-place container. if the in-place container call
     **    ShowOwnedPopups, then it could inadvertantly be made visible.
     **    to avoid this we will parent the toolbar window back to our
     **    own application main window. if we are not in-place active
     **    then this is the same as lpft->m_hWndApp.
     */
     else if (nState == BARSTATE_HIDE)
          SetParent(lpft->m_FormulaBar.m_hWnd, g_lpApp->m_hWndApp);
#endif

     else
          SetParent(lpft->m_FormulaBar.m_hWnd, lpft->m_hWndApp);
}


/*
 * FrameTools_FB_GetState
 *
 * Purpose:
 *  Get display state of FormulaBar
 *
 * Parameters:
 *  lpft            FrameTools object
 *
 * Return Value:
 *  nState          current display state
 */
int FrameTools_FB_GetState(LPFRAMETOOLS lpft)
{
     return lpft->m_FormulaBar.m_nState;
}


/*
 * FrameToolsWndProc
 *
 * Purpose:
 *  WndProc for toolbar window
 *
 * Parameters:
 *  hWnd
 *  Message
 *  wParam
 *  lParam
 *
 * Return Value:
 *  message dependent
 */
```

```
LRESULT FAR PASCAL FrameToolsWndProc(HWND hWnd, UINT Message, WPARAM wParam,
LPARAM lParam)
{
     LPFRAMETOOLS lpft = (LPFRAMETOOLS)GetWindowLong(hWnd, 0);

     switch (Message) {

           case WM_MOUSEACTIVATE:
                 return MA_NOACTIVATE;

           default:
                 return DefWindowProc(hWnd, Message, wParam, lParam);
     }

     return 0L;
}


/*
 * Bar_Move
 *
 * Purpose:
 *  Resize and reposition a bar
 *
 * Parameters:
 *  lpbar           Bar object
 *  lprcClient      pointer to Client rect
 *  lprcPopup       pointer to Popup rect
 *
 * Return Value:
 *  nil
 */
static void Bar_Move(LPBAR lpbar, LPRECT lprcClient, LPRECT lprcPopup)
{
     if (lpbar->m_nState == BARSTATE_HIDE) {
           ShowWindow(lpbar->m_hWnd, SW_HIDE);
     }
     else {
           ShowWindow(lpbar->m_hWnd, SW_SHOW);
           switch (lpbar->m_nState) {
                 case BARSTATE_POPUP:
                       MoveWindow(lpbar->m_hWnd, lprcPopup->left, lprcPopup-
>top,
                                   lprcPopup->right - lprcPopup->left,
lpbar->m_uHeight,
                                   TRUE);
                       lprcPopup->top += lpbar->m_uHeight;
                       break;

                 case BARSTATE_TOP:
                       MoveWindow(lpbar->m_hWnd, lprcClient->left,
lprcClient->top,
                                   lprcClient->right - lprcClient->left,
                                   lpbar->m_uHeight, TRUE);
                       lprcClient->top += lpbar->m_uHeight;
                       break;

                 case BARSTATE_BOTTOM:
                       MoveWindow(lpbar->m_hWnd, lprcClient->left,
                                   lprcClient->bottom - lpbar->m_uHeight,
                                   lprcClient->right - lprcClient->left,
                                   lpbar->m_uHeight, TRUE);
                       lprcClient->bottom -= lpbar->m_uHeight;
                       break;
           }
     }
}


/*
 * FB_ResizeEdit
 *
 * Purpose:
 *  Resize the edit control in FormulaBar
 *
 * Parameters:
 *  lpft            Bar object
 *
 * Return Value:
 *  nil
 */
static void FB_ResizeEdit(LPBAR lpbar)
{
     RECT rcClient;
     RECT rcEdit;
     HWND hwndEdit;

     GetClientRect(lpbar->m_hWnd, (LPRECT)&rcClient);
     hwndEdit = GetDlgItem(lpbar->m_hWnd, IDM_FB_EDIT);
     GetWindowRect(hwndEdit, (LPRECT)&rcEdit);
     ScreenToClient(lpbar->m_hWnd, (LPPOINT)&rcEdit.left);
     ScreenToClient(lpbar->m_hWnd, (LPPOINT)&rcEdit.right);

     SetWindowPos(hwndEdit, NULL, 0, 0, rcClient.right - rcEdit.left -
SPACE,
                 rcEdit.bottom - rcEdit.top, SWP_NOMOVE | SWP_NOZORDER);
}
```

HEADING.H (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    heading.c
**
**    This file contains definitions used by OutlineDoc's row and
**    column headings.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#define COLUMN  10

#define IDC_ROWHEADING  2000
#define IDC_COLHEADING  2001
#define IDC_BUTTON      2002

#define HEADING_FONT    "Arial"

#define COLUMN_LETTER   'A'


typedef struct tagCOLHEADING {
     HWND m_hWnd;
     UINT m_uHeight;
} COLHEADING, FAR* LPCOLHEADING;

typedef struct tagROWHEADING {
     HWND m_hWnd;
     UINT m_uWidth;
     FARPROC    m_WndProc;
} ROWHEADING, FAR* LPROWHEADING;

typedef struct tagHEADING {
     COLHEADING m_colhead;
     ROWHEADING m_rowhead;
     HWND       m_hwndButton;
     BOOL       m_fShow;
     HFONT      m_hfont;
} HEADING, FAR* LPHEADING;

BOOL Heading_Create(LPHEADING lphead, HWND hWndParent, HINSTANCE hInst);
void Heading_Destroy(LPHEADING lphead);
void Heading_Move(LPHEADING lphead, HWND hwndListBox, LPSCALEFACTOR
lpscale);
void Heading_Show(LPHEADING lphead, BOOL fShow);
void Heading_ReScale(LPHEADING lphead, LPSCALEFACTOR lpscale);
void Heading_CH_Draw(LPHEADING lphead, LPDRAWITEMSTRUCT lpdis, LPRECT
lprcScreen, LPRECT lprcObject);
void Heading_CH_SetHorizontalExtent(LPHEADING lphead, HWND hwndListBox);
```

```
UINT Heading_CH_GetHeight(LPHEADING lphead, LPSCALEFACTOR lpscale);
LRESULT Heading_CH_SendMessage(LPHEADING lphead, UINT msg, WPARAM wParam,
LPARAM lParam);
void Heading_CH_ForceRedraw(LPHEADING lphead, BOOL fErase);
void Heading_RH_ForceRedraw(LPHEADING lphead, BOOL fErase);
void Heading_RH_Draw(LPHEADING lphead, LPDRAWITEMSTRUCT lpdis);
LRESULT Heading_RH_SendMessage(LPHEADING lphead, UINT msg, WPARAM wParam,
LPARAM lParam);
UINT Heading_RH_GetWidth(LPHEADING lphead, LPSCALEFACTOR lpscale);
void Heading_RH_Scroll(LPHEADING lphead, HWND hwndListBox);
LRESULT FAR PASCAL RowHeadWndProc(HWND hWnd, UINT Message, WPARAM wParam,
LPARAM lParam);
```

HEADING.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    heading.c
**
**    This file contains functions and support for OutlineDoc's row and
**    column headings.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"

extern LPOUTLINEAPP g_lpApp;


BOOL Heading_Create(LPHEADING lphead, HWND hWndParent, HINSTANCE hInst)
{
   HDC         hDC;
   TEXTMETRIC  tm;

   if (!lphead || !hWndParent || !hInst)
      return FALSE;

   hDC = GetDC(hWndParent);
   if (!hDC)
      return FALSE;

   if (!GetTextMetrics(hDC, (TEXTMETRIC FAR*)&tm))
      return FALSE;
   lphead->m_colhead.m_uHeight = tm.tmHeight;
   lphead->m_rowhead.m_uWidth = 4 * tm.tmAveCharWidth;
   lphead->m_fShow = TRUE;

   ReleaseDC(hWndParent, hDC);

   lphead->m_hfont = CreateFont(
         tm.tmHeight,
         0,0,0,0,0,0,0,0,
         OUT_TT_PRECIS,      // use TrueType
         CLIP_DEFAULT_PRECIS,
         DEFAULT_QUALITY,
         DEFAULT_PITCH | FF_DONTCARE,
         HEADING_FONT
   );

   if (!lphead->m_hfont)
      return FALSE;

   lphead->m_colhead.m_hWnd = CreateWindow(
         "listbox",
         "Column Heading",
         WS_VISIBLE | WS_CHILD | WS_DISABLED | LBS_OWNERDRAWVARIABLE |
               LBS_NOINTEGRALHEIGHT,
         0,0,0,0,        // any values
         hWndParent,
         (HMENU)IDC_COLHEADING,
         hInst,
         NULL);

   if (!lphead->m_colhead.m_hWnd)
      return FALSE;

   // add a dummy line to get WM_DRAWITEM message
   SendMessage(lphead->m_colhead.m_hWnd, LB_ADDSTRING, 0,
         MAKELPARAM(lphead->m_colhead.m_uHeight,0));

   lphead->m_rowhead.m_hWnd = CreateWindow(
         "listbox",
         "Row Heading",
         WS_VISIBLE | WS_CHILD | WS_DISABLED | LBS_OWNERDRAWVARIABLE,
         0,0,0,0,        // any values
         hWndParent,
         (HMENU)IDC_ROWHEADING,
         hInst,
         NULL);

   if (!lphead->m_rowhead.m_hWnd)
      return FALSE;

   SendMessage(lphead->m_rowhead.m_hWnd, LB_ADDSTRING, 0,
         MAKELPARAM(lphead->m_colhead.m_uHeight,0));

   lphead->m_rowhead.m_WndProc =
         (FARPROC) GetWindowLong(lphead->m_rowhead.m_hWnd, GWL_WNDPROC );
   SetWindowLong(lphead->m_rowhead.m_hWnd, GWL_WNDPROC,
         (LONG) RowHeadWndProc);

   lphead->m_hwndButton = CreateWindow(
         "button",
         NULL,
         WS_VISIBLE | WS_CHILD,
         0,0,0,0,        // any values
         hWndParent,
         (HMENU)IDC_BUTTON,
         hInst,
         NULL);

   if (!lphead->m_hwndButton)
      return FALSE;

   return TRUE;
}
```

```
void Heading_Destroy(LPHEADING lphead)
{
   if (!lphead)
      return;

   if (IsWindow(lphead->m_colhead.m_hWnd)) {
      DestroyWindow(lphead->m_colhead.m_hWnd);
      lphead->m_colhead.m_hWnd = NULL;
   }
   if (IsWindow(lphead->m_rowhead.m_hWnd)) {
      DestroyWindow(lphead->m_rowhead.m_hWnd);
      lphead->m_rowhead.m_hWnd = NULL;
   }
   if (IsWindow(lphead->m_hwndButton)) {
      DestroyWindow(lphead->m_hwndButton);
      lphead->m_hwndButton = NULL;
   }
#ifdef WIN32
   if (GetObjectType(lphead->m_hfont)) {
#else
   if (IsGDIObject(lphead->m_hfont)) {
#endif
      DeleteObject(lphead->m_hfont);
      lphead->m_hfont = NULL;
   }

}


void Heading_Move(LPHEADING lphead, HWND hwndDoc, LPSCALEFACTOR lpscale)
{
   int nOffsetX;
   int nOffsetY;
   RECT rcDoc;

   if (!lphead || !hwndDoc || !lpscale)
      return;

   if (!lphead->m_fShow)
      return;

   nOffsetX = (int) Heading_RH_GetWidth(lphead, lpscale);
   nOffsetY = (int) Heading_CH_GetHeight(lphead, lpscale);
   GetClientRect(hwndDoc, (LPRECT)&rcDoc);

   MoveWindow(lphead->m_hwndButton, 0, 0, nOffsetX, nOffsetY, TRUE);

   MoveWindow(
         lphead->m_colhead.m_hWnd,
         nOffsetX, 0,
         rcDoc.right-rcDoc.left-nOffsetX, nOffsetY,
         TRUE
   );

   MoveWindow(lphead->m_rowhead.m_hWnd, 0, nOffsetY, nOffsetX,
          rcDoc.bottom-rcDoc.top-nOffsetY, TRUE);
}


void Heading_Show(LPHEADING lphead, BOOL fShow)
{
   int nCmdShow;

   if (!lphead)
      return;

   lphead->m_fShow = fShow;
   nCmdShow = fShow ? SW_SHOW : SW_HIDE;

   ShowWindow(lphead->m_hwndButton, nCmdShow);
   ShowWindow(lphead->m_colhead.m_hWnd, nCmdShow);
   ShowWindow(lphead->m_rowhead.m_hWnd, nCmdShow);
}


void Heading_ReScale(LPHEADING lphead, LPSCALEFACTOR lpscale)
{
   UINT uHeight;

   if (!lphead || !lpscale)
      return;

   // Row heading is scaled with the LineList_Rescale. So, only
   // Column heading needed to be scaled here.
   uHeight = (UINT)(lphead->m_colhead.m_uHeight * lpscale->dwSyN /
         lpscale->dwSyD);
   SendMessage(lphead->m_colhead.m_hWnd, LB_SETITEMHEIGHT, 0,
         MAKELPARAM(uHeight, 0));
}


void Heading_CH_Draw(LPHEADING lphead, LPDRAWITEMSTRUCT lpdis, LPRECT
lprcScreen, LPRECT lprcObject)
{
   HPEN        hpenOld;
   HPEN        hpen;
   HBRUSH      hbr;
   HFONT       hfOld;
   int         nTabInPix;
   char        letter;
   int         i;
   int         nOldMapMode;
   RECT        rcWindowOld;
   RECT        rcViewportOld;
   POINT    point;

   if (!lpdis || !lphead)
      return;

   hbr = GetStockObject(LTGRAY_BRUSH);
   FillRect(lpdis->hDC, (LPRECT)&lpdis->rcItem, hbr);

   nOldMapMode = SetDCToAnisotropic(lpdis->hDC, lprcScreen, lprcObject,
         (LPRECT)&rcWindowOld, (LPRECT)&rcViewportOld);

   hfOld = SelectObject(lpdis->hDC, lphead->m_hfont);
   hpen = GetStockObject(BLACK_PEN);
   hpenOld = SelectObject(lpdis->hDC, hpen);

   nTabInPix = XformWidthInHimetricToPixels(lpdis->hDC, TABWIDTH);
   SetBkMode(lpdis->hDC, TRANSPARENT);

   letter = COLUMN_LETTER;
   MoveToEx(lpdis->hDC, lprcObject->left, lprcObject->bottom,&point);
   LineTo(lpdis->hDC, lprcObject->left, lprcObject->top);

   for (i = 0; i < COLUMN; i++) {
      lprcObject->right = lprcObject->left + nTabInPix;
      DrawText(lpdis->hDC, (LPCSTR)&letter, 1, lprcObject,
            DT_SINGLELINE | DT_CENTER | DT_VCENTER);
      MoveToEx(lpdis->hDC, lprcObject->right, lprcObject->bottom, &point);
      LineTo(lpdis->hDC, lprcObject->right, lprcObject->top);

      letter++;
      lprcObject->left += nTabInPix;
   }

   SelectObject(lpdis->hDC, hpenOld);
   SelectObject(lpdis->hDC, hfOld);

   ResetOrigDC(lpdis->hDC, nOldMapMode, (LPRECT)&rcWindowOld,
         (LPRECT)&rcViewportOld);
}


void Heading_CH_SetHorizontalExtent(LPHEADING lphead, HWND hwndListBox)
{
   RECT rcLL;
   RECT rcCH;
   int  nLLWidth;
   int  nCHWidth;
   int  nHorizExtent;

   if (!lphead || !hwndListBox)
      return;

   nHorizExtent=(int)SendMessage(hwndListBox, LB_GETHORIZONTALEXTENT, 0,
0L);
   GetClientRect(hwndListBox, (LPRECT)&rcLL);
   GetClientRect(lphead->m_colhead.m_hWnd, (LPRECT)&rcCH);

   nLLWidth = rcLL.right - rcLL.left;
   nCHWidth = rcCH.right - rcCH.left;
   nHorizExtent += nCHWidth - nLLWidth;
   SendMessage(lphead->m_colhead.m_hWnd, LB_SETHORIZONTALEXTENT,
         nHorizExtent, 0L);
}


UINT Heading_CH_GetHeight(LPHEADING lphead, LPSCALEFACTOR lpscale)
{
   if (!lphead || !lpscale)
      return 0;

   if (lphead->m_fShow)
      return (UINT)(lphead->m_colhead.m_uHeight * lpscale->dwSyN /
            lpscale->dwSyD);
   else
      return 0;
}


LRESULT Heading_CH_SendMessage(LPHEADING lphead, UINT msg, WPARAM wParam,
LPARAM lParam)
{
   if (!lphead)
      return 0;

   if (lphead->m_colhead.m_hWnd)
      return SendMessage(lphead->m_colhead.m_hWnd, msg, wParam, lParam);
}


void Heading_CH_ForceRedraw(LPHEADING lphead, BOOL fErase)
{
   if (!lphead)
      return;

   InvalidateRect(lphead->m_colhead.m_hWnd, NULL, fErase);
}

void Heading_RH_ForceRedraw(LPHEADING lphead, BOOL fErase)
{
   if (!lphead)
      return;

   InvalidateRect(lphead->m_rowhead.m_hWnd, NULL, fErase);
}

void Heading_RH_Draw(LPHEADING lphead, LPDRAWITEMSTRUCT lpdis)
{
   char        cBuf[5];
   HPEN        hpenOld;
   HPEN        hpen;
   HBRUSH      hbrOld;
   HBRUSH      hbr;
   HFONT       hfOld;
   RECT        rc;
   RECT        rcWindowOld;
   RECT        rcViewportOld;
   int         nMapModeOld;

   if (!lpdis || !lphead)
      return;

   lpdis->rcItem;

   rc.left = 0;
   rc.bottom = 0;
   rc.top = (int)lpdis->itemData;
   rc.right = lphead->m_rowhead.m_uWidth;

   nMapModeOld = SetDCToAnisotropic(lpdis->hDC, &lpdis->rcItem, &rc,
         (LPRECT)&rcWindowOld, (LPRECT)&rcViewportOld);

   hpen = GetStockObject(BLACK_PEN);
   hpenOld = SelectObject(lpdis->hDC, hpen);
   hbr = GetStockObject(LTGRAY_BRUSH);
   hbrOld = SelectObject(lpdis->hDC, hbr);

   Rectangle(lpdis->hDC, rc.left, rc.top, rc.right,
         rc.bottom);

   hfOld = SelectObject(lpdis->hDC, lphead->m_hfont);

   SetBkMode(lpdis->hDC, TRANSPARENT);

   wsprintf(cBuf, "%d", lpdis->itemID + 1);

   DrawText(lpdis->hDC, (LPSTR)cBuf, lstrlen(cBuf), (LPRECT)&rc,
         DT_SINGLELINE | DT_CENTER | DT_VCENTER);

   SelectObject(lpdis->hDC, hfOld);

   SelectObject(lpdis->hDC, hpenOld);
   SelectObject(lpdis->hDC, hbrOld);

   ResetOrigDC(lpdis->hDC, nMapModeOld, (LPRECT)&rcWindowOld,
         (LPRECT)&rcViewportOld);
}

LRESULT Heading_RH_SendMessage(LPHEADING lphead, UINT msg, WPARAM wParam,
LPARAM lParam)
{
   if (!lphead)
      return 0;

   if (lphead->m_rowhead.m_hWnd)
      return SendMessage(lphead->m_rowhead.m_hWnd, msg, wParam, lParam);
}


UINT Heading_RH_GetWidth(LPHEADING lphead, LPSCALEFACTOR lpscale)
{
   if (!lphead || !lpscale)
      return 0;

   if (lphead->m_fShow)
      return (UINT)(lphead->m_rowhead.m_uWidth * lpscale->dwSxN /
            lpscale->dwSxD);
   else
      return 0;
}


void Heading_RH_Scroll(LPHEADING lphead, HWND hwndListBox)
{
   int nTopLL;
   int nTopRH;

   if (!lphead || !hwndListBox)
      return;

   nTopLL = (int)SendMessage(hwndListBox, LB_GETTOPINDEX, 0, 0L);
   nTopRH = (int)SendMessage(
         lphead->m_rowhead.m_hWnd, LB_GETTOPINDEX, 0, 0L);

   if (nTopLL != nTopRH)
      SendMessage(
            lphead->m_rowhead.m_hWnd,LB_SETTOPINDEX,(WPARAM)nTopLL,0L);
}


LRESULT FAR PASCAL RowHeadWndProc(HWND hWnd, UINT Message, WPARAM wParam,
LPARAM lParam)
{
   HWND     hwndParent = GetParent (hWnd);
   LPOUTLINEDOC lpDoc = (LPOUTLINEDOC)GetWindowLong(hwndParent, 0);
   LPHEADING lphead = OutlineDoc_GetHeading(lpDoc);

   switch (Message) {
      case WM_PAINT:
      {
         LPLINELIST lpLL = OutlineDoc_GetLineList(lpDoc);
         PAINTSTRUCT ps;

         // If there is no line in listbox, trap the message and draw the
         // background gray. Without this, the background will be painted
         // as default color.
         if (!LineList_GetCount(lpLL)) {
            BeginPaint(hWnd, &ps);
            EndPaint(hWnd, &ps);
            return 0;
         }

         break;
      }

      case WM_ERASEBKGND:
      {
         HDC hDC = (HDC)wParam;
         RECT rc;

         GetClientRect(hWnd, (LPRECT)&rc);
         FillRect(hDC, (LPRECT)&rc, GetStockObject(GRAY_BRUSH));

         return 1;
      }
   }

   return CallWindowProc(
         (WNDPROC)lphead->m_rowhead.m_WndProc,
         hWnd,
         Message,
         wParam,
         lParam
   );
}
_
```

ICNTROTL.DEF   (OUTLINE Sample)

```
;;
;;
;;    OLE 2.0 Container Sample Code
;;
;;    icntrotl.def
;;
;;    Definition file for icntrotl.exe
;;
;;    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
;;
;;

NAME            ICntrOtl
DESCRIPTION             'Microsoft OLE 2.0 In-Place Container Sample code'
EXETYPE         WINDOWS
STUB            'WINSTUB.EXE'
CODE            PRELOAD MOVEABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        4096

;; NOTE: Do not add exports to this file.  Use __export
;; in your function prototype and definition instead.
```

ICNTROTL.RC   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Container Sample Code
**
**    icntrotl.rc
**
**    Resource file for icntrotl.exe
**
**     (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "windows.h"
#include "outlrc.h"
#include "cntrrc.h"

SelCur      CURSOR selcross.cur
DragMoveCur CURSOR dragmove.cur

#if defined( IF_SPECIAL_DD_CURSORS_NEEDED )
DragNoneCur CURSOR dragnone.cur
DragCopyCur CURSOR dragcopy.cur
DragLinkCur CURSOR draglink.cur
#endif  // IF_SPECIAL_DD_CURSORS_NEEDED

CntrOutlMenu MENU
  BEGIN
    POPUP  "&File"
      BEGIN
        MENUITEM "&New", IDM_F_NEW
        MENUITEM "&Open...\t  Ctrl+F12", IDM_F_OPEN
        MENUITEM "&Save\t  Shift+F12", IDM_F_SAVE
        MENUITEM "Save &As...\t  F12", IDM_F_SAVEAS
        MENUITEM SEPARATOR
        MENUITEM "&Print...\t  Ctrl+Shift+F12", IDM_F_PRINT
        MENUITEM "Printer Se&tup...", IDM_F_PRINTERSETUP
        MENUITEM SEPARATOR
        MENUITEM "E&xit\t  Alt+F4", IDM_F_EXIT
      END
    POPUP  "&Edit"
      BEGIN
        MENUITEM "&Undo", IDM_E_UNDO
        MENUITEM SEPARATOR
        MENUITEM "Cu&t\t  Ctrl+X", IDM_E_CUT
        MENUITEM "&Copy\t  Ctrl+C", IDM_E_COPY
        MENUITEM "&Paste\t  Ctrl+V", IDM_E_PASTE
        MENUITEM "Paste &Special...", IDM_E_PASTESPECIAL
        MENUITEM "Paste &Link", IDM_E_PASTELINK
        MENUITEM "Cl&ear\t  Del", IDM_E_CLEAR
        MENUITEM SEPARATOR
        MENUITEM "&Insert Object...", IDM_E_INSERTOBJECT
        MENUITEM "Li&nks...", IDM_E_EDITLINKS
```

```
       MENUITEM "&Object", IDM_E_OBJECTVERBMIN
    MENUITEM SEPARATOR
    MENUITEM "Select &All\t  Ctrl+A", IDM_E_SELECTALL
  END
 POPUP  "O&utline"
   BEGIN
    POPUP  "&Zoom"
      BEGIN
        MENUITEM "&100%\t  Ctrl+1", IDM_V_ZOOM_100
        MENUITEM "&75%\t  Ctrl+2", IDM_V_ZOOM_75
        MENUITEM "&50%\t  Ctrl+3", IDM_V_ZOOM_50
        MENUITEM "&25%\t  Ctrl+4", IDM_V_ZOOM_25
      END
    POPUP  "&Left and Right margins"
      BEGIN
        MENUITEM "&nil", IDM_V_SETMARGIN_0
        MENUITEM "&1 cm", IDM_V_SETMARGIN_1
        MENUITEM "&2 cm", IDM_V_SETMARGIN_2
        MENUITEM "&3 cm", IDM_V_SETMARGIN_3
        MENUITEM "&4 cm", IDM_V_SETMARGIN_4
      END
    POPUP "Add &Top Line"
         BEGIN
           MENUITEM "&1 cm", IDM_V_ADDTOP_1
           MENUITEM "&2 cm", IDM_V_ADDTOP_2
           MENUITEM "&3 cm", IDM_V_ADDTOP_3
           MENUITEM "&4 cm", IDM_V_ADDTOP_4
         END
   END
POPUP  "&Line"
  BEGIN
    MENUITEM "&Add Line\t  Enter", IDM_L_ADDLINE
    MENUITEM "E&dit Line\t  Alt+Enter", IDM_L_EDITLINE
    MENUITEM SEPARATOR
    MENUITEM "&Indent Line\t  Tab", IDM_L_INDENTLINE
    MENUITEM "U&nindent Line\t  Shift+Tab", IDM_L_UNINDENTLINE
       MENUITEM SEPARATOR
       MENUITEM "&Set Line Height...", IDM_L_SETLINEHEIGHT
  END
POPUP  "&Name"
  BEGIN
    MENUITEM "&Define Name...", IDM_N_DEFINENAME
    MENUITEM "&Goto Name...", IDM_N_GOTONAME
  END
POPUP  "&Options"
  BEGIN
    POPUP  "&Button Bar Display"
      BEGIN
        MENUITEM "At &Top", IDM_O_BB_TOP
        MENUITEM "At &Bottom", IDM_O_BB_BOTTOM
        MENUITEM "&Popup", IDM_O_BB_POPUP
        MENUITEM "&Hide", IDM_O_BB_HIDE
      END
    POPUP  "&Formula Bar Display"
      BEGIN
```

```
            MENUITEM "At &Top", IDM_O_FB_TOP
            MENUITEM "At &Bottom", IDM_O_FB_BOTTOM
            MENUITEM "&Popup", IDM_O_FB_POPUP
          END
        POPUP  "&Row and Column Heading"
          BEGIN
            MENUITEM "&Show", IDM_O_HEAD_SHOW
            MENUITEM "&Hide", IDM_O_HEAD_HIDE
          END
         MENUITEM "&Show Object", IDM_O_SHOWOBJECT
      END
    POPUP  "DbgI&Cntr"
      BEGIN
        MENUITEM "&Debug Level...", IDM_D_DEBUGLEVEL
        MENUITEM "Register Message &Filter", IDM_D_INSTALLMSGFILTER
        MENUITEM "&Reject Incoming Messages", IDM_D_REJECTINCOMING
        MENUITEM "&Inside-out Activation", IDM_D_INSIDEOUT
      END
    POPUP  "&Help"
      BEGIN
        MENUITEM "&About...", IDM_H_ABOUT
      END
  END

CntrOutlAccel ACCELERATORS
  BEGIN
    VK_F12, IDM_F_OPEN, VIRTKEY, CONTROL
    VK_F12, IDM_F_SAVE, VIRTKEY, SHIFT
    VK_F12, IDM_F_SAVEAS, VIRTKEY
    VK_F12, IDM_F_PRINT, VIRTKEY, CONTROL, SHIFT

    "x", IDM_E_CUT, VIRTKEY, CONTROL
    "c", IDM_E_COPY, VIRTKEY, CONTROL
    "v", IDM_E_PASTE, VIRTKEY, CONTROL
    VK_DELETE, IDM_E_CLEAR, VIRTKEY
    VK_RETURN, IDM_L_ADDLINE, VIRTKEY
    VK_RETURN, IDM_L_EDITLINE, VIRTKEY, ALT
    VK_TAB, IDM_L_INDENTLINE, VIRTKEY
    VK_TAB, IDM_L_UNINDENTLINE, VIRTKEY, SHIFT
    "a", IDM_E_SELECTALL, VIRTKEY, CONTROL

    ; old conventions for editing
    VK_INSERT, IDM_E_COPY, VIRTKEY, CONTROL
    VK_DELETE, IDM_E_CUT, VIRTKEY, SHIFT
    VK_INSERT, IDM_E_PASTE, VIRTKEY, SHIFT

    VK_F2, IDM_F2, VIRTKEY
     VK_ESCAPE, IDM_ESCAPE, VIRTKEY

    "1", IDM_V_ZOOM_100, VIRTKEY, CONTROL
    "2", IDM_V_ZOOM_75, VIRTKEY, CONTROL
    "3", IDM_V_ZOOM_50, VIRTKEY, CONTROL
    "4", IDM_V_ZOOM_25, VIRTKEY, CONTROL
  END
```

```
; Same as CntrOutlAccel but without Delete and Backspace
; used when edit control of Formula Bar in focus
;
CntrOutlAccelFocusEdit ACCELERATORS
  BEGIN
    VK_F12, IDM_F_OPEN, VIRTKEY, CONTROL
    VK_F12, IDM_F_SAVE, VIRTKEY, SHIFT
    VK_F12, IDM_F_SAVEAS, VIRTKEY
    VK_F12, IDM_F_PRINT, VIRTKEY, CONTROL, SHIFT

    "x", IDM_E_CUT, VIRTKEY, CONTROL
    "c", IDM_E_COPY, VIRTKEY, CONTROL
    "v", IDM_E_PASTE, VIRTKEY, CONTROL
    VK_RETURN, IDM_L_ADDLINE, VIRTKEY
    VK_RETURN, IDM_L_EDITLINE, VIRTKEY, ALT
    VK_TAB, IDM_L_INDENTLINE, VIRTKEY
    VK_TAB, IDM_L_UNINDENTLINE, VIRTKEY, SHIFT
    "a", IDM_E_SELECTALL, VIRTKEY, CONTROL

     VK_ESCAPE, IDM_ESCAPE, VIRTKEY

    ; old conventions for editing
    VK_INSERT, IDM_E_COPY, VIRTKEY, CONTROL
    VK_DELETE, IDM_E_CUT, VIRTKEY, SHIFT
    VK_INSERT, IDM_E_PASTE, VIRTKEY, SHIFT
  END

InPlaceCntrOutlAccel ACCELERATORS
  BEGIN
    VK_F12, IDM_F_OPEN, VIRTKEY, CONTROL
    VK_F12, IDM_F_SAVE, VIRTKEY, SHIFT
    VK_F12, IDM_F_SAVEAS, VIRTKEY
    VK_F12, IDM_F_PRINT, VIRTKEY, CONTROL, SHIFT
     VK_ESCAPE, IDM_ESCAPE, VIRTKEY
    "1", IDM_V_ZOOM_100, VIRTKEY, CONTROL
    "2", IDM_V_ZOOM_75, VIRTKEY, CONTROL
    "3", IDM_V_ZOOM_50, VIRTKEY, CONTROL
    "4", IDM_V_ZOOM_25, VIRTKEY, CONTROL
  END

CntrOutlIcon ICON icntrotl.ico

Image72     BITMAP      image72.bmp
Image96     BITMAP      image96.bmp
Image120    BITMAP      image120.bmp
LogoBitmap  BITMAP      ole2.bmp

#include "DIALOGS.DLG"
```

ICNTROTL.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Sample Code
**
**    precomp.c
**
**    This file is used to precompile the OUTLINE.H header file
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"
```

ISVROTL.RC   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Server Sample Code
**
**    isvrotl.rc
**
**    Resource file for isvrotl.exe
**
**     (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "windows.h"
#include "outlrc.h"

SelCur      CURSOR selcross.cur
DragMoveCur CURSOR dragmove.cur

#if defined( IF_SPECIAL_DD_CURSORS_NEEDED )
DragNoneCur CURSOR dragnone.cur
DragCopyCur CURSOR dragcopy.cur
DragLinkCur CURSOR draglink.cur
#endif  // IF_SPECIAL_DD_CURSORS_NEEDED

SvrOutlMenu MENU
  BEGIN
    POPUP  "&File"
      BEGIN
        MENUITEM "&New", IDM_F_NEW
        MENUITEM "&Open...\t  Ctrl+F12", IDM_F_OPEN
        MENUITEM "&Save\t  Shift+F12", IDM_F_SAVE
        MENUITEM "Save &As...\t  F12", IDM_F_SAVEAS
        MENUITEM SEPARATOR
        MENUITEM "&Print...\t  Ctrl+Shift+F12", IDM_F_PRINT
        MENUITEM "Printer Se&tup...", IDM_F_PRINTERSETUP
        MENUITEM SEPARATOR
        MENUITEM "E&xit\t  Alt+F4", IDM_F_EXIT
      END
    POPUP  "&Edit"
      BEGIN
        MENUITEM "&Undo", IDM_E_UNDO
        MENUITEM SEPARATOR
        MENUITEM "Cu&t\t  Ctrl+X", IDM_E_CUT
        MENUITEM "&Copy\t  Ctrl+C", IDM_E_COPY
        MENUITEM "&Paste\t  Ctrl+V", IDM_E_PASTE
        MENUITEM "Paste &Special...", IDM_E_PASTESPECIAL
        MENUITEM "Cl&ear\t  Del", IDM_E_CLEAR
        MENUITEM SEPARATOR
        MENUITEM "Select A&ll\t  Ctrl+A", IDM_E_SELECTALL
      END
     POPUP  "O&utline"
       BEGIN
        POPUP  "&Zoom"
          BEGIN
            MENUITEM "&400%", IDM_V_ZOOM_400
            MENUITEM "&300%", IDM_V_ZOOM_300
            MENUITEM "&200%", IDM_V_ZOOM_200
            MENUITEM "&100%", IDM_V_ZOOM_100
            MENUITEM "&75%", IDM_V_ZOOM_75
            MENUITEM "&50%", IDM_V_ZOOM_50
            MENUITEM "&25%", IDM_V_ZOOM_25
          END
        POPUP  "&Left and Right margins"
          BEGIN
            MENUITEM "&nil", IDM_V_SETMARGIN_0
            MENUITEM "&1 cm", IDM_V_SETMARGIN_1
            MENUITEM "&2 cm", IDM_V_SETMARGIN_2
            MENUITEM "&3 cm", IDM_V_SETMARGIN_3
            MENUITEM "&4 cm", IDM_V_SETMARGIN_4
          END
        POPUP "Add &Top Line"
             BEGIN
               MENUITEM "&1 cm", IDM_V_ADDTOP_1
               MENUITEM "&2 cm", IDM_V_ADDTOP_2
               MENUITEM "&3 cm", IDM_V_ADDTOP_3
               MENUITEM "&4 cm", IDM_V_ADDTOP_4
             END
       END
    POPUP  "&Line"
      BEGIN
        MENUITEM "&Add Line\t  Enter", IDM_L_ADDLINE
        MENUITEM "E&dit Line\t  Alt+Enter", IDM_L_EDITLINE
        MENUITEM SEPARATOR
        MENUITEM "&Indent Line\t  Tab", IDM_L_INDENTLINE
        MENUITEM "U&nindent Line\t  Shift+Tab", IDM_L_UNINDENTLINE
           MENUITEM SEPARATOR
           MENUITEM "&Set Line Height...", IDM_L_SETLINEHEIGHT
      END
    POPUP  "&Name"
      BEGIN
        MENUITEM "&Define Name...", IDM_N_DEFINENAME
        MENUITEM "&Goto Name...", IDM_N_GOTONAME
      END
    POPUP  "&Options"
      BEGIN
        POPUP  "&Button Bar Display"
          BEGIN
            MENUITEM "At &Top", IDM_O_BB_TOP
            MENUITEM "At &Bottom", IDM_O_BB_BOTTOM
            MENUITEM "&Popup", IDM_O_BB_POPUP
            MENUITEM "&Hide", IDM_O_BB_HIDE
          END
        POPUP  "&Formula Bar Display"
          BEGIN
            MENUITEM "At &Top", IDM_O_FB_TOP
            MENUITEM "At &Bottom", IDM_O_FB_BOTTOM
            MENUITEM "&Popup", IDM_O_FB_POPUP
```

```
          END
        POPUP  "&Row and Column Heading"
          BEGIN
            MENUITEM "&Show", IDM_O_HEAD_SHOW
            MENUITEM "&Hide", IDM_O_HEAD_HIDE
          END
      END
    POPUP  "DbgI&Svr"
      BEGIN
        MENUITEM "&Debug Level...", IDM_D_DEBUGLEVEL
        MENUITEM "Register Message &Filter", IDM_D_INSTALLMSGFILTER
        MENUITEM "&Reject Incoming Messages", IDM_D_REJECTINCOMING
      END
    POPUP  "&Help"
      BEGIN
        MENUITEM "&About...", IDM_H_ABOUT
      END
  END

SvrOutlAccel ACCELERATORS
  BEGIN
    VK_F12, IDM_F_OPEN, VIRTKEY, CONTROL
    VK_F12, IDM_F_SAVE, VIRTKEY, SHIFT
    VK_F12, IDM_F_SAVEAS, VIRTKEY
    VK_F12, IDM_F_PRINT, VIRTKEY, CONTROL, SHIFT

    "x", IDM_E_CUT, VIRTKEY, CONTROL
    "c", IDM_E_COPY, VIRTKEY, CONTROL
    "v", IDM_E_PASTE, VIRTKEY, CONTROL
    VK_DELETE, IDM_E_CLEAR, VIRTKEY
    VK_RETURN, IDM_L_ADDLINE, VIRTKEY
    VK_RETURN, IDM_L_EDITLINE, VIRTKEY, ALT
    VK_TAB, IDM_L_INDENTLINE, VIRTKEY
    VK_TAB, IDM_L_UNINDENTLINE, VIRTKEY, SHIFT
    "a", IDM_E_SELECTALL, VIRTKEY, CONTROL

    ; old conventions for editing
    VK_INSERT, IDM_E_COPY, VIRTKEY, CONTROL
    VK_DELETE, IDM_E_CUT, VIRTKEY, SHIFT
    VK_INSERT, IDM_E_PASTE, VIRTKEY, SHIFT

    VK_F2, IDM_F2, VIRTKEY
  END

; used when edit control of Formula Bar in focus
;
SvrOutlAccelFocusEdit ACCELERATORS
  BEGIN
    VK_F12, IDM_F_OPEN, VIRTKEY, CONTROL
    VK_F12, IDM_F_SAVE, VIRTKEY, SHIFT
    VK_F12, IDM_F_SAVEAS, VIRTKEY
    VK_F12, IDM_F_PRINT, VIRTKEY, CONTROL, SHIFT

    VK_RETURN, IDM_L_ADDLINE, VIRTKEY
    VK_RETURN, IDM_L_EDITLINE, VIRTKEY, ALT
```

```
     VK_ESCAPE, IDM_ESCAPE, VIRTKEY
  END


InPlaceSvrOutlAccel ACCELERATORS
  BEGIN
    "x", IDM_E_CUT, VIRTKEY, CONTROL
    "c", IDM_E_COPY, VIRTKEY, CONTROL
    "v", IDM_E_PASTE, VIRTKEY, CONTROL
    VK_DELETE, IDM_E_CLEAR, VIRTKEY
    VK_RETURN, IDM_L_ADDLINE, VIRTKEY
    VK_RETURN, IDM_L_EDITLINE, VIRTKEY, ALT
    VK_TAB, IDM_L_INDENTLINE, VIRTKEY
    VK_TAB, IDM_L_UNINDENTLINE, VIRTKEY, SHIFT
    "a", IDM_E_SELECTALL, VIRTKEY, CONTROL

    ; old conventions for editing
    VK_INSERT, IDM_E_COPY, VIRTKEY, CONTROL
    VK_DELETE, IDM_E_CUT, VIRTKEY, SHIFT
    VK_INSERT, IDM_E_PASTE, VIRTKEY, SHIFT

    VK_F2, IDM_F2, VIRTKEY
     VK_ESCAPE, IDM_ESCAPE, VIRTKEY
  END

; used when edit control of Formula Bar in focus
; REVIEW: currently not properly used
InPlaceSvrOutlAccelFocusEdit ACCELERATORS
  BEGIN
    VK_F12, IDM_F_OPEN, VIRTKEY, CONTROL
    VK_F12, IDM_F_SAVE, VIRTKEY, SHIFT
    VK_F12, IDM_F_SAVEAS, VIRTKEY
    VK_F12, IDM_F_PRINT, VIRTKEY, CONTROL, SHIFT

    VK_RETURN, IDM_L_ADDLINE, VIRTKEY
    VK_RETURN, IDM_L_EDITLINE, VIRTKEY, ALT

     VK_ESCAPE, IDM_FB_CANCEL, VIRTKEY
  END

SvrOutlIcon ICON isvrotl.ico

Image72     BITMAP      image72.bmp
Image96     BITMAP      image96.bmp
Image120    BITMAP      image120.bmp
LogoBitmap  BITMAP      ole2.bmp

#include "DIALOGS.DLG"
```

ISVROTL.C (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Sample Code
**
**    precomp.c
**
**    This file is used to precompile the OUTLINE.H header file
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"
```

LINKING.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    linking.c
**
**    This file contains the major interfaces, methods and related support
**    functions for implementing linking to items. The code
**    contained in this file is used by BOTH the Container and Server
**    (Object) versions of the Outline sample code.
**
**    As a server SVROUTL supports linking to the whole document object
**    (either a file-based document or as an embedded object). It also
**    supports linking to ranges (or PseudoObjects).
**
**    As a container CNTROUTL supports linking to embedded objects.
**    (see file svrpsobj.c for Pseudo Object implementation)
**
**    OleDoc Object
**      exposed interfaces:
**          IPersistFile
**          IOleItemContainer
**          IExternalConnection
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"

OLEDBGDATA

extern LPOUTLINEAPP             g_lpApp;


/*************************************************************************
** OleDoc::IPersistFile interface implementation
*************************************************************************/

// IPersistFile::QueryInterface
STDMETHODIMP OleDoc_PFile_QueryInterface(
      LPPERSISTFILE       lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
)
{
   LPOLEDOC lpOleDoc = ((struct CDocPersistFileImpl FAR*)lpThis)->lpOleDoc;

   return OleDoc_QueryInterface(lpOleDoc, riid, lplpvObj);
}
```

```
// IPersistFile::AddRef
STDMETHODIMP_(ULONG) OleDoc_PFile_AddRef(LPPERSISTFILE lpThis)
{
   LPOLEDOC lpOleDoc = ((struct CDocPersistFileImpl FAR*)lpThis)->lpOleDoc;

   OleDbgAddRefMethod(lpThis, "IPersistFile");

   return OleDoc_AddRef(lpOleDoc);
}


// IPersistFile::Release
STDMETHODIMP_(ULONG) OleDoc_PFile_Release (LPPERSISTFILE lpThis)
{
   LPOLEDOC lpOleDoc = ((struct CDocPersistFileImpl FAR*)lpThis)->lpOleDoc;

   OleDbgReleaseMethod(lpThis, "IPersistFile");

   return OleDoc_Release(lpOleDoc);
}


// IPersistFile::GetClassID
STDMETHODIMP OleDoc_PFile_GetClassID (
      LPPERSISTFILE       lpThis,
      CLSID FAR*          lpclsid
)
{
   LPOLEDOC lpOleDoc = ((struct CDocPersistFileImpl FAR*)lpThis)->lpOleDoc;
   LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;
   OleDbgOut2("OleDoc_PFile_GetClassID\r\n");

#if defined( OLE_SERVER ) && defined( SVR_TREATAS )

   /* OLE2NOTE: we must be carefull to return the correct CLSID here.
   **    if we are currently preforming a "TreatAs (aka. ActivateAs)"
   **    operation then we need to return the class of the object
   **    written in the storage of the object. otherwise we would
   **    return our own class id.
   */
   return ServerDoc_GetClassID((LPSERVERDOC)lpOleDoc, lpclsid);
#else
   *lpclsid = CLSID_APP;
#endif
   return NOERROR;
}


// IPersistFile::IsDirty
STDMETHODIMP  OleDoc_PFile_IsDirty(LPPERSISTFILE lpThis)
{
   LPOLEDOC lpOleDoc = ((struct CDocPersistFileImpl FAR*)lpThis)->lpOleDoc;
   OleDbgOut2("OleDoc_PFile_IsDirty\r\n");

   if (OutlineDoc_IsModified((LPOUTLINEDOC)lpOleDoc))
```

```
      return NOERROR;
   else
      return ResultFromScode(S_FALSE);
}


// IPersistFile::Load
STDMETHODIMP OleDoc_PFile_Load (
      LPPERSISTFILE       lpThis,
      LPCOLESTR           lpszFileName,
      DWORD               grfMode
)
{
   LPOLEDOC lpOleDoc = ((struct CDocPersistFileImpl FAR*)lpThis)->lpOleDoc;
   SCODE sc;

   OLEDBG_BEGIN2("OleDoc_PFile_Load\r\n")

   /* OLE2NOTE: grfMode passed from the caller indicates if the caller
   **    needs Read or ReadWrite permissions. if appropriate the
   **    callee should open the file with the requested permissions.
   **    the caller will normally not impose sharing permissions.
   **
   **    the sample code currently always opens its file ReadWrite.
   */

   if (OutlineDoc_LoadFromFile((LPOUTLINEDOC)lpOleDoc,
(LPOLESTR)lpszFileName))
      sc = S_OK;
   else
      sc = E_FAIL;

   OLEDBG_END2
   return ResultFromScode(sc);
}


// IPersistFile::Save
STDMETHODIMP OleDoc_PFile_Save (
      LPPERSISTFILE       lpThis,
      LPCOLESTR           lpszFileName,
      BOOL                fRemember
)
{
   LPOLEDOC lpOleDoc = ((struct CDocPersistFileImpl FAR*)lpThis)->lpOleDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpOleDoc;
   SCODE sc;

   OLEDBG_BEGIN2("OleDoc_PFile_Save\r\n")

   /* OLE2NOTE: it is only legal to perform a Save or SaveAs operation
   **    on a file-based document. if the document is an embedded
   **    object then we can not be changed to a file-base object.
   **
   **      fRemember   lpszFileName     Type of Save
   **    ----------------------------------------------
   **        TRUE        NULL           SAVE
   **        TRUE        ! NULL         SAVE AS
   **        FALSE       ! NULL         SAVE COPY AS
   **        FALSE       NULL           ***error***
   */
   if ( (lpszFileName==NULL || (lpszFileName != NULL && fRemember))
         && ((lpOutlineDoc->m_docInitType != DOCTYPE_FROMFILE
            && lpOutlineDoc->m_docInitType != DOCTYPE_NEW)) ) {
      OLEDBG_END2
      return ResultFromScode(E_INVALIDARG);
   }

   if (OutlineDoc_SaveToFile(
         (LPOUTLINEDOC)lpOleDoc,
         lpszFileName,
         lpOutlineDoc->m_cfSaveFormat,
         fRemember)) {
      sc = S_OK;
   } else
      sc = E_FAIL;

   OLEDBG_END2
   return ResultFromScode(sc);
}


// IPersistFile::SaveCompleted
STDMETHODIMP OleDoc_PFile_SaveCompleted (
      LPPERSISTFILE       lpThis,
      LPCOLESTR           lpszFileName
)
{
   LPOLEDOC lpOleDoc = ((struct CDocPersistFileImpl FAR*)lpThis)->lpOleDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpOleDoc;

   OleDbgOut2("OleDoc_PFile_SaveCompleted\r\n");

   /* This method is called after IPersistFile::Save is called. during
   **    the period between Save and SaveCompleted the object must
   **    consider itself in NOSCRIBBLE mode (ie. it is NOT allowed to
   **    write to its file. here the object can clear its NOSCRIBBLE
   **    mode flag. the outline app never scribbles to its storage, so
   **    we have nothing to do.
   */
   return NOERROR;
}


// IPersistFile::GetCurFile
STDMETHODIMP OleDoc_PFile_GetCurFile (
      LPPERSISTFILE   lpThis,
      LPOLESTR FAR*   lplpszFileName
)
{
   LPOLEDOC lpOleDoc = ((struct CDocPersistFileImpl FAR*)lpThis)->lpOleDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpOleDoc;
   LPMALLOC lpMalloc;
   LPOLESTR lpsz;
   SCODE sc;
   char  szAnsiStr[256];

   OleDbgOut2("OleDoc_PFile_GetCurFile\r\n");

   /* OLE2NOTE: we must make sure to set all out ptr parameters to NULL. */
   *lplpszFileName = NULL;

   /*********************************************************************
   ** OLE2NOTE: memory returned for the lplpszFileName must be
   **    allocated appropriately using the current registered IMalloc
   **    interface. the allows the ownership of the memory to be
   **    passed to the caller (even if in another process).
   *********************************************************************/

   CoGetMalloc(MEMCTX_TASK, &lpMalloc);
   if (! lpMalloc) {

      return ResultFromScode(E_FAIL);
   }

   if (lpOutlineDoc->m_docInitType == DOCTYPE_FROMFILE) {
      /* valid filename associated; return file name */
      lpsz = (LPOLESTR)lpMalloc->lpVtbl->Alloc(
            lpMalloc,
            (OLESTRLEN(lpOutlineDoc->m_szFileName)+1)*sizeof(OLECHAR)
      );
      if (! lpsz) {
         sc = E_OUTOFMEMORY;
         goto error;
      }

      OLESTRCPY(lpsz, (LPOLESTR)lpOutlineDoc->m_szFileName);
      sc = S_OK;
   } else {
      /* no file associated; return default file name prompt */
      lpsz=(LPOLESTR)lpMalloc->lpVtbl->Alloc(lpMalloc,
(lstrlen(DEFEXTENSION)+3)*sizeof(OLECHAR));
      wsprintf(szAnsiStr, "*.%s", DEFEXTENSION);
      A2W (szAnsiStr, lpsz, OLEUI_CCHPATHMAX);
      sc = S_FALSE;
   }

error:
   OleStdRelease((LPUNKNOWN)lpMalloc);
   *lplpszFileName = lpsz;


   return ResultFromScode(sc);
}
```

```
/*************************************************************************
** OleDoc::IOleItemContainer interface implementation
*************************************************************************/

// IOleItemContainer::QueryInterface
STDMETHODIMP OleDoc_ItemCont_QueryInterface(
      LPOLEITEMCONTAINER  lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
)
{
   LPOLEDOC lpOleDoc =
         ((struct CDocOleItemContainerImpl FAR*)lpThis)->lpOleDoc;

   return OleDoc_QueryInterface(lpOleDoc, riid, lplpvObj);
}


// IOleItemContainer::AddRef
STDMETHODIMP_(ULONG) OleDoc_ItemCont_AddRef(LPOLEITEMCONTAINER lpThis)
{
   LPOLEDOC lpOleDoc =
         ((struct CDocOleItemContainerImpl FAR*)lpThis)->lpOleDoc;

   OleDbgAddRefMethod(lpThis, "IOleItemContainer");

   return OleDoc_AddRef((LPOLEDOC)lpOleDoc);
}


// IOleItemContainer::Release
STDMETHODIMP_(ULONG) OleDoc_ItemCont_Release(LPOLEITEMCONTAINER lpThis)
{
   LPOLEDOC lpOleDoc =
         ((struct CDocOleItemContainerImpl FAR*)lpThis)->lpOleDoc;

   OleDbgReleaseMethod(lpThis, "IOleItemContainer");

   return OleDoc_Release((LPOLEDOC)lpOleDoc);
}


// IOleItemContainer::ParseDisplayName
STDMETHODIMP OleDoc_ItemCont_ParseDisplayName(
      LPOLEITEMCONTAINER  lpThis,
      LPBC                lpbc,
      LPOLESTR            lpszDisplayName,
      ULONG FAR*          lpchEaten,
      LPMONIKER FAR*      lplpmkOut
)
{
   LPOLEDOC lpOleDoc =
         ((struct CDocOleItemContainerImpl FAR*)lpThis)->lpOleDoc;
   OLECHAR szItemName[MAXNAMESIZE];
   LPUNKNOWN lpUnk;
   HRESULT hrErr;

   OleDbgOut2("OleDoc_ItemCont_ParseDisplayName\r\n");

   /* OLE2NOTE: we must make sure to set all out ptr parameters to NULL. */
   *lplpmkOut = NULL;

   *lpchEaten = OleStdGetItemToken(
         lpszDisplayName,
         szItemName,
         MAXNAMESIZE*sizeof(OLECHAR)
   );

   /* OLE2NOTE: get a pointer to a running instance of the object. we
   **    should force the object to go running if necessary (even if
   **    this means launching its server EXE). this is the meaining of
   **    BINDSPEED_INDEFINITE. Parsing a Moniker is known to be an
   **    "EXPENSIVE" operation.
   */
   hrErr = OleDoc_ItemCont_GetObject(
         lpThis,
         szItemName,
         BINDSPEED_INDEFINITE,
         lpbc,
         &IID_IUnknown,
         (LPVOID FAR*)&lpUnk
   );

   if (hrErr == NOERROR) {
      OleStdRelease(lpUnk);   // item name FOUND; don't need obj ptr.
      CreateItemMoniker(OLESTDDELIM, szItemName, lplpmkOut);
   } else
      *lpchEaten = 0;     // item name is NOT valid

   return hrErr;
}


// IOleItemContainer::EnumObjects
STDMETHODIMP OleDoc_ItemCont_EnumObjects(
      LPOLEITEMCONTAINER  lpThis,
      DWORD               grfFlags,
      LPENUMUNKNOWN FAR*  lplpenumUnknown
)
{
   LPOLEDOC lpOleDoc =
         ((struct CDocOleItemContainerImpl FAR*)lpThis)->lpOleDoc;

   OLEDBG_BEGIN2("OleDoc_ItemCont_EnumObjects\r\n")

   /* OLE2NOTE: we must make sure to set all out ptr parameters to NULL. */
   *lplpenumUnknown = NULL;

   /* OLE2NOTE: this method should be implemented to allow programatic
   **    clients the ability to what elements the container holds.
   **    this method is NOT called in the standard linking scenarios.
   **
   **    grfFlags can be one of the following:
   **        OLECONTF_EMBEDDINGS   -- enumerate embedded objects
   **        OLECONTF_LINKS        -- enumerate linked objects
   **        OLECONTF_OTHERS       -- enumerate non-OLE compound doc objs
   **        OLECONTF_ONLYUSER     -- enumerate only objs named by user
   **        OLECONTF_ONLYIFRUNNING-- enumerate only objs in running state
   */

   OleDbgAssertSz(0, "NOT YET IMPLEMENTED!");

   OLEDBG_END2

   return ResultFromScode(E_NOTIMPL);
}


// IOleItemContainer::LockContainer
STDMETHODIMP OleDoc_ItemCont_LockContainer(
      LPOLEITEMCONTAINER  lpThis,
      BOOL                fLock
)
{
   LPOLEDOC lpOleDoc =
         ((struct CDocOleItemContainerImpl FAR*)lpThis)->lpOleDoc;
   HRESULT hrErr;
   OLEDBG_BEGIN2("OleDoc_ItemCont_LockContainer\r\n")

#if defined( _DEBUG )
   if (fLock) {
      ++lpOleDoc->m_cCntrLock;
      OleDbgOutRefCnt3(
            "OleDoc_ItemCont_LockContainer: cLock++\r\n",
            lpOleDoc,
            lpOleDoc->m_cCntrLock
      );
   } else {
      /* OLE2NOTE: when there are no open documents and the app is not
      **    under the control of the user and there are no outstanding
      **    locks on the app, then revoke our ClassFactory to enable the
      **    app to shut down.
      */
      --lpOleDoc->m_cCntrLock;
      OleDbgAssertSz (
            lpOleDoc->m_cCntrLock >= 0,
            "OleDoc_ItemCont_LockContainer(FALSE) called with cLock == 0"
      );

      if (lpOleDoc->m_cCntrLock == 0) {
         OleDbgOutRefCnt2(
               "OleDoc_ItemCont_LockContainer: UNLOCKED\r\n",
               lpOleDoc, lpOleDoc->m_cCntrLock);
      } else {
         OleDbgOutRefCnt3(
               "OleDoc_ItemCont_LockContainer: cLock--\r\n",
               lpOleDoc, lpOleDoc->m_cCntrLock);
      }
   }
#endif  // _DEBUG

   /* OLE2NOTE: in order to hold the document alive we call
   **    CoLockObjectExternal to add a strong reference to our Doc
   **    object. this will keep the Doc alive when all other external
   **    references release us. whenever an embedded object goes
   **    running a LockContainer(TRUE) is called. when the embedded
   **    object shuts down (ie. transitions from running to loaded)
   **    LockContainer(FALSE) is called. if the user issues File.Close
   **    the document will shut down in any case ignoring any
   **    outstanding LockContainer locks because CoDisconnectObject is
   **    called in OleDoc_Close. this will forceably break any
   **    existing strong reference counts including counts that we add
   **    ourselves by calling CoLockObjectExternal and guarantee that
   **    the Doc object gets its final release (ie. cRefs goes to 0).
   */
   hrErr = OleDoc_Lock(lpOleDoc, fLock, TRUE /* fLastUnlockReleases */);

   OLEDBG_END2
   return hrErr;
}


// IOleItemContainer::GetObject
STDMETHODIMP OleDoc_ItemCont_GetObject(
      LPOLEITEMCONTAINER  lpThis,
      LPOLESTR            lpszItem,
      DWORD               dwSpeedNeeded,
      LPBINDCTX           lpbc,
      REFIID              riid,
      LPVOID FAR*         lplpvObject
)
{
   LPOLEDOC lpOleDoc =
         ((struct CDocOleItemContainerImpl FAR*)lpThis)->lpOleDoc;
   HRESULT hrErr;

   OLEDBG_BEGIN2("OleDoc_ItemCont_GetObject\r\n")

   /* OLE2NOTE: we must make sure to set all out ptr parameters to NULL. */
   *lplpvObject = NULL;

#if defined( OLE_SERVER )

   /* OLE2NOTE: SERVER ONLY version should return PseudoObjects with
   **    BINDSPEED_IMMEDIATE, thus the dwSpeedNeeded is not important
   **    in the case of a pure server.
   */
   hrErr = ServerDoc_GetObject(
         (LPSERVERDOC)lpOleDoc, lpszItem,riid,lplpvObject);
```

```
#endif
#if defined( OLE_CNTR )

   /* OLE2NOTE: dwSpeedNeeded indicates how long the caller is willing
   **    to wait for us to get the object:
   **      BINDSPEED_IMMEDIATE -- only if obj already loaded && IsRunning
   **      BINDSPEED_MODERATE  -- load obj if necessary && if IsRunning
   **      BINDSPEED_INDEFINITE-- force obj to load and run if necessary
   */
   hrErr = ContainerDoc_GetObject(
         (LPCONTAINERDOC)lpOleDoc,lpszItem,dwSpeedNeeded,riid,lplpvObject);
#endif

   OLEDBG_END2

   return hrErr;
}


// IOleItemContainer::GetObjectStorage
STDMETHODIMP OleDoc_ItemCont_GetObjectStorage(
      LPOLEITEMCONTAINER  lpThis,
      LPOLESTR            lpszItem,
      LPBINDCTX           lpbc,
      REFIID              riid,
      LPVOID FAR*         lplpvStorage
)
{
   LPOLEDOC lpOleDoc =
         ((struct CDocOleItemContainerImpl FAR*)lpThis)->lpOleDoc;
   OleDbgOut2("OleDoc_ItemCont_GetObjectStorage\r\n");

   /* OLE2NOTE: we must make sure to set all out ptr parameters to NULL. */
   *lplpvStorage = NULL;

#if defined( OLE_SERVER )
   /* OLE2NOTE: in the SERVER ONLY version, item names identify pseudo
   **    objects. pseudo objects, do NOT have identifiable storage.
   */
   return ResultFromScode(E_FAIL);
#endif
#if defined( OLE_CNTR )
   // We can only return an IStorage* type pointer
   if (! IsEqualIID(riid, &IID_IStorage))
   {
      return ResultFromScode(E_FAIL);
   }

   return ContainerDoc_GetObjectStorage(
         (LPCONTAINERDOC)lpOleDoc,
         lpszItem,
         (LPSTORAGE FAR*)lplpvStorage
   );
#endif
}
```

```
// IOleItemContainer::IsRunning
STDMETHODIMP OleDoc_ItemCont_IsRunning(
      LPOLEITEMCONTAINER  lpThis,
      LPOLESTR            lpszItem
)
{
   LPOLEDOC lpOleDoc =
         ((struct CDocOleItemContainerImpl FAR*)lpThis)->lpOleDoc;
   HRESULT hrErr;

   OLEDBG_BEGIN2("OleDoc_ItemCont_IsRunning\r\n")

   /* OLE2NOTE: Check if item name is valid. if so then return if
   **    Object is running. PseudoObjects in the Server version are
   **    always considered running. Ole objects in the container must
   **    be checked if they are running.
   */

#if defined( OLE_SERVER )
   hrErr = ServerDoc_IsRunning((LPSERVERDOC)lpOleDoc, lpszItem);
#endif
#if defined( OLE_CNTR )
   hrErr = ContainerDoc_IsRunning((LPCONTAINERDOC)lpOleDoc, lpszItem);
#endif

   OLEDBG_END2
   return hrErr;
}


/*************************************************************************
** OleDoc::IExternalConnection interface implementation
*************************************************************************/

// IExternalConnection::QueryInterface
STDMETHODIMP OleDoc_ExtConn_QueryInterface(
      LPEXTERNALCONNECTION    lpThis,
      REFIID                  riid,
      LPVOID FAR*             lplpvObj
)
{
   LPOLEDOC lpOleDoc =
         ((struct CDocExternalConnectionImpl FAR*)lpThis)->lpOleDoc;

   return OleDoc_QueryInterface(lpOleDoc, riid, lplpvObj);
}


// IExternalConnection::AddRef
STDMETHODIMP_(ULONG) OleDoc_ExtConn_AddRef(LPEXTERNALCONNECTION lpThis)
{
   LPOLEDOC lpOleDoc =
         ((struct CDocExternalConnectionImpl FAR*)lpThis)->lpOleDoc;
   OleDbgAddRefMethod(lpThis, "IExternalConnection");

   return OleDoc_AddRef(lpOleDoc);
}


// IExternalConnection::Release
STDMETHODIMP_(ULONG) OleDoc_ExtConn_Release (LPEXTERNALCONNECTION lpThis)
{
   LPOLEDOC lpOleDoc =
         ((struct CDocExternalConnectionImpl FAR*)lpThis)->lpOleDoc;

   OleDbgReleaseMethod(lpThis, "IExternalConnection");

   return OleDoc_Release(lpOleDoc);
}


// IExternalConnection::AddConnection
STDMETHODIMP_(DWORD) OleDoc_ExtConn_AddConnection(
      LPEXTERNALCONNECTION    lpThis,
      DWORD                   extconn,
      DWORD                   reserved
)
{
   LPOLEDOC lpOleDoc =
         ((struct CDocExternalConnectionImpl FAR*)lpThis)->lpOleDoc;

   if( extconn & EXTCONN_STRONG ) {

#if defined( _DEBUG )
      OleDbgOutRefCnt3(
            "OleDoc_ExtConn_AddConnection: dwStrongExtConn++\r\n",
            lpOleDoc,
            lpOleDoc->m_dwStrongExtConn + 1
      );
#endif
      return ++(lpOleDoc->m_dwStrongExtConn);
   } else
   {
      return 0;
   }
}


// IExternalConnection::ReleaseConnection
STDMETHODIMP_(DWORD) OleDoc_ExtConn_ReleaseConnection(
      LPEXTERNALCONNECTION    lpThis,
      DWORD                   extconn,
      DWORD                   reserved,
      BOOL                    fLastReleaseCloses
)
{
   LPOLEDOC lpOleDoc =
         ((struct CDocExternalConnectionImpl FAR*)lpThis)->lpOleDoc;

   if( extconn & EXTCONN_STRONG ){
      DWORD dwSave = --(lpOleDoc->m_dwStrongExtConn);
#if defined( _DEBUG )
      OLEDBG_BEGIN2( (fLastReleaseCloses ?
                  "OleDoc_ExtConn_ReleaseConnection(TRUE)\r\n" :
                  "OleDoc_ExtConn_ReleaseConnection(FALSE)\r\n") )
      OleDbgOutRefCnt3(
            "OleDoc_ExtConn_ReleaseConnection: dwStrongExtConn--\r\n",
            lpOleDoc,
            lpOleDoc->m_dwStrongExtConn
      );
      OleDbgAssertSz (
            lpOleDoc->m_dwStrongExtConn >= 0,
            "OleDoc_ExtConn_ReleaseConnection called with dwStrong == 0"
      );
#endif  // _DEBUG

      if( lpOleDoc->m_dwStrongExtConn == 0 && fLastReleaseCloses )
         OleDoc_Close(lpOleDoc, OLECLOSE_SAVEIFDIRTY);

      OLEDBG_END2
      return dwSave;
   } else
   {
      return 0;
   }
}


/*************************************************************************
** OleDoc Common Support Functions
*************************************************************************/


/* OleDoc_GetFullMoniker
** ---------------------
**    Return the full, absolute moniker of the document.
**
**    NOTE: the caller must release the pointer returned when done.
*/
LPMONIKER OleDoc_GetFullMoniker(LPOLEDOC lpOleDoc, DWORD dwAssign)
{
   LPMONIKER lpMoniker = NULL;

   OLEDBG_BEGIN3("OleDoc_GetFullMoniker\r\n")

   if (lpOleDoc->m_lpSrcDocOfCopy) {
      /* CASE I: this document was created for a copy or drag/drop
      **    operation. generate the moniker which identifies the
      **    source document of the original copy.
      */
      if (! lpOleDoc->m_fLinkSourceAvail)
         goto done;        // we already know a moniker is not available
      lpMoniker=OleDoc_GetFullMoniker(lpOleDoc->m_lpSrcDocOfCopy, dwAssign);
   }
   else if (lpOleDoc->m_lpFileMoniker) {

      /* CASE II: this document is a top-level user document (either
      **    file-based or untitled). return the FileMoniker stored
      **    with the document; it uniquely identifies the document.
      */
      // we must AddRef the moniker to pass out a ptr
      lpOleDoc->m_lpFileMoniker->lpVtbl->AddRef(lpOleDoc->m_lpFileMoniker);

      lpMoniker = lpOleDoc->m_lpFileMoniker;
   }

#if defined( OLE_SERVER )

   else if (((LPSERVERDOC)lpOleDoc)->m_lpOleClientSite) {

      /* CASE III: this document is an embedded object, ask our
      **    container for our moniker.
      */
      OLEDBG_BEGIN2("IOleClientSite::GetMoniker called\r\n");
      ((LPSERVERDOC)lpOleDoc)->m_lpOleClientSite->lpVtbl->GetMoniker(
            ((LPSERVERDOC)lpOleDoc)->m_lpOleClientSite,
            dwAssign,
            OLEWHICHMK_OBJFULL,
            &lpMoniker
      );
      OLEDBG_END2
   }

#endif

   else {
      lpMoniker = NULL;
   }

done:
   OLEDBG_END3
   return lpMoniker;
}


/* OleDoc_DocRenamedUpdate
** -----------------------
**    Update the documents registration in the running object table (ROT).
**    Also inform all embedded OLE objects (container only) and/or psedudo
**    objects (server only) that the name of the document has changed.
*/
void OleDoc_DocRenamedUpdate(LPOLEDOC lpOleDoc, LPMONIKER lpmkDoc)
{
   OLEDBG_BEGIN3("OleDoc_DocRenamedUpdate\r\n")

   OleDoc_AddRef(lpOleDoc);
   /* OLE2NOTE: we must re-register ourselves as running when we
   **    get a new moniker assigned (ie. when we are renamed).
   */
   OLEDBG_BEGIN3("OleStdRegisterAsRunning called\r\n")
   OleStdRegisterAsRunning(
         (LPUNKNOWN)&lpOleDoc->m_Unknown,
         lpmkDoc,
         &lpOleDoc->m_dwRegROT
   );
   OLEDBG_END3

#if defined( OLE_SERVER )
   {
      LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOleDoc;
      LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpOleDoc;

      /* OLE2NOTE: inform any linking clients that the document has been
      **    renamed.
      */
      ServerDoc_SendAdvise (
            lpServerDoc,
            OLE_ONRENAME,
            lpmkDoc,
            0        /* advf -- not relevant here */
      );

      /* OLE2NOTE: inform any clients of pseudo objects
      **    within our document, that our document's
      **    Moniker has changed.
      */
      ServerNameTable_InformAllPseudoObjectsDocRenamed(
            (LPSERVERNAMETABLE)lpOutlineDoc->m_lpNameTable, lpmkDoc);
   }
#endif
#if defined( OLE_CNTR )
   {
      LPCONTAINERDOC lpContainerDoc = (LPCONTAINERDOC)lpOleDoc;

      /* OLE2NOTE: must tell all OLE objects that our container
      **    moniker changed.
      */
      ContainerDoc_InformAllOleObjectsDocRenamed(
            lpContainerDoc,
            lpmkDoc
      );
   }
#endif

   OleDoc_Release(lpOleDoc);       // release artificial AddRef above
   OLEDBG_END3
}
```

```
#if defined( OLE_SERVER )

/*************************************************************************
** ServerDoc Supprt Functions Used by Server versions
*************************************************************************/


/* ServerDoc_PseudoObjLockDoc
** --------------------------
**    Add a lock on the Doc on behalf of the PseudoObject. the Doc may not
**    close while the Doc exists.
**
**    when a pseudo object is first created, it calls this method to
**    guarantee that the document stays alive (PseudoObj_Init).
**    when a pseudo object is destroyed, it call
**    ServerDoc_PseudoObjUnlockDoc to release this hold on the document.
*/
void ServerDoc_PseudoObjLockDoc(LPSERVERDOC lpServerDoc)
{
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpServerDoc;
   ULONG cPseudoObj;

   cPseudoObj = ++lpServerDoc->m_cPseudoObj;

#if defined( _DEBUG )
   OleDbgOutRefCnt3(
         "ServerDoc_PseudoObjLockDoc: cPseudoObj++\r\n",
         lpServerDoc,
         cPseudoObj
   );
#endif
   OleDoc_Lock(lpOleDoc, TRUE /* fLock */, 0 /* not applicable */);
   return;
}


/* ServerDoc_PseudoObjUnlockDoc
** ----------------------------
**    Release the lock on the Doc on behalf of the PseudoObject. if this was
**    the last lock on the Doc, then it will shutdown.
*/
void ServerDoc_PseudoObjUnlockDoc(
      LPSERVERDOC         lpServerDoc,
      LPPSEUDOOBJ         lpPseudoObj
)
{
   ULONG cPseudoObj;
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpServerDoc;
   OLEDBG_BEGIN3("ServerDoc_PseudoObjUnlockDoc\r\n")

   /* OLE2NOTE: when there are no active pseudo objects in the Doc and
   **    the Doc is not visible, and if there are no outstanding locks
   **    on the Doc, then this is a "silent update"
   **    situation. our Doc is being used programatically by some
   **    client; it is NOT accessible to the user because it is
   **    NOT visible. thus since all Locks have been released, we
   **    will close the document. if the app is only running due
   **    to the presence of this document, then the app will now
   **    also shut down.
   */
   cPseudoObj = --lpServerDoc->m_cPseudoObj;

#if defined( _DEBUG )
   OleDbgAssertSz (
         lpServerDoc->m_cPseudoObj >= 0,
         "PseudoObjUnlockDoc called with cPseudoObj == 0"
   );

   OleDbgOutRefCnt3(
         "ServerDoc_PseudoObjUnlockDoc: cPseudoObj--\r\n",
         lpServerDoc,
         cPseudoObj
   );
#endif
   OleDoc_Lock(lpOleDoc, FALSE /* fLock */, TRUE /* fLastUnlockReleases */);

   OLEDBG_END3
   return;
}


/* ServerDoc_GetObject
** -------------------
**
**    Return a pointer to an object identified by an item string
**    (lpszItem). For a server-only app, the object returned will be a
**    pseudo object.
*/
HRESULT ServerDoc_GetObject(
      LPSERVERDOC            lpServerDoc,
      LPOLESTR               lpszItem,
      REFIID                 riid,
      LPVOID FAR*            lplpvObject
)
{
   LPPSEUDOOBJ lpPseudoObj;
   LPSERVERNAMETABLE lpServerNameTable =
         (LPSERVERNAMETABLE)((LPOUTLINEDOC)lpServerDoc)->m_lpNameTable;

   *lplpvObject = NULL;

   /* Get the PseudoObj which corresponds to an item name. if the item
   **    name does NOT exist in the name table then NO object is
   **    returned. the ServerNameTable_GetPseudoObj routine finds a
   **    name entry corresponding to the item name, it then checks if
   **    a PseudoObj has already been allocated. if so, it returns the
   **    existing object, otherwise it allocates a new PseudoObj.
   */
   lpPseudoObj = ServerNameTable_GetPseudoObj(
         lpServerNameTable,
            lpszItem,
            lpServerDoc
    );

    if (! lpPseudoObj) {
        *lplpvObject = NULL;
        return ResultFromScode(MK_E_NOOBJECT);
    }

    // return the desired interface pointer of the pseudo object.
    return PseudoObj_QueryInterface(lpPseudoObj, riid, lplpvObject);
}


/* ServerDoc_IsRunning
** -------------------
**
**    Check if the object identified by an item string (lpszItem) is in
**    the running state. For a server-only app, if the item name exists in
**    in the NameTable then the item name is considered running.
**    IOleItemContainer::GetObject would succeed.
*/

HRESULT ServerDoc_IsRunning(LPSERVERDOC lpServerDoc, LPOLESTR lpszItem)
{
    LPOUTLINENAMETABLE lpOutlineNameTable =
            ((LPOUTLINEDOC)lpServerDoc)->m_lpNameTable;
    LPSERVERNAME lpServerName;

    lpServerName = (LPSERVERNAME)OutlineNameTable_FindName(
            lpOutlineNameTable,
            lpszItem
    );

    if (lpServerName)
        return NOERROR;
    else
        return ResultFromScode(MK_E_NOOBJECT);
}


/* ServerDoc_GetSelRelMoniker
** --------------------------
**    Retrieve the relative item moniker which identifies the given
**    selection (lplrSel).
**
**    Returns NULL if a moniker can NOT be created.
*/

LPMONIKER ServerDoc_GetSelRelMoniker(
        LPSERVERDOC             lpServerDoc,
        LPLINERANGE             lplrSel,
        DWORD                   dwAssign
)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPSERVERNAMETABLE lpServerNameTable =
         (LPSERVERNAMETABLE)lpOutlineDoc->m_lpNameTable;
   LPOUTLINENAMETABLE lpOutlineNameTable =
         (LPOUTLINENAMETABLE)lpServerNameTable;
   LPOUTLINENAME lpOutlineName;
   LPMONIKER lpmk;


lpOutlineName=OutlineNameTable_FindNamedRange(lpOutlineNameTable,lplrSel);

   if (lpOutlineName) {
      /* the selection range already has a name assigned */
      CreateItemMoniker(OLESTDDELIM, lpOutlineName->m_szName, &lpmk);
   } else {
      char szbuf[MAXNAMESIZE];
      OLECHAR szUniBuf[MAXNAMESIZE];

      switch (dwAssign) {

         case GETMONIKER_FORCEASSIGN:

            /* Force the assignment of the name. This is called when a
            **    Paste Link actually occurs. At this point we want to
            **    create a Name and add it to the NameTable in order to
            **    track the source of the link. This name (as all
            **    names) will be updated upon editing of the document.
            */
            wsprintf(
                  szbuf,
                  "%s %ld",
                  (LPSTR)DEFRANGENAMEPREFIX,
                  ++(lpServerDoc->m_nNextRangeNo)
            );

            lpOutlineName = OutlineApp_CreateName(lpOutlineApp);

            if (lpOutlineName) {
               A2W (szbuf, lpOutlineName->m_szName, OLEUI_CCHPATHMAX);
               lpOutlineName->m_nStartLine = lplrSel->m_nStartLine;
               lpOutlineName->m_nEndLine = lplrSel->m_nEndLine;
               OutlineDoc_AddName(lpOutlineDoc, lpOutlineName);
            } else {
               // REVIEW: do we need "Out-of-Memory" error message here?
            }
            break;

         case GETMONIKER_TEMPFORUSER:

            /* Create a name to show to the user in the Paste
            **    Special dialog but do NOT yet incur the overhead
            **    of adding a Name to the NameTable. The Moniker
            **    generated should be useful to display to the user
            **    to indicate the source of the copy, but will NOT
            **    be used to create a link directly (the caller
            **    should ask again for a moniker specifying FORCEASSIGN).
            **    we will generate the name that would be the next
            **    auto-generated range name, BUT will NOT actually
            **    increment the range counter.
            */
            wsprintf(
                  szbuf,
                  "%s %ld",
                  (LPSTR)DEFRANGENAMEPREFIX,
                  (lpServerDoc->m_nNextRangeNo)+1
            );
            break;

         case GETMONIKER_ONLYIFTHERE:

            /* the caller only wants a name if one has already been
            **    assigned. we have already above checked if the
            **    current selection has a name, so we will simply
            **    return NULL here.
            */
            return NULL;    // no moniker is assigned

         default:
            return NULL;    // unknown flag given
      }

      A2W (szbuf, szUniBuf, MAXNAMESIZE);
      CreateItemMoniker(OLESTDDELIM, szUniBuf, &lpmk);
   }

   return lpmk;
}


/* ServerDoc_GetSelFullMoniker
** ---------------------------
**    Retrieve the full absolute moniker which identifies the given
**    selection (lplrSel).
**    this moniker is created as a composite of the absolute moniker for
**    the entire document appended with an item moniker which identifies
**    the selection relative to the document.
**    Returns NULL if a moniker can NOT be created.
*/
LPMONIKER ServerDoc_GetSelFullMoniker(
      LPSERVERDOC             lpServerDoc,
      LPLINERANGE             lplrSel,
      DWORD                   dwAssign
)
{
   LPMONIKER lpmkDoc = NULL;
   LPMONIKER lpmkItem = NULL;
   LPMONIKER lpmkFull = NULL;

   lpmkDoc = OleDoc_GetFullMoniker(
            (LPOLEDOC)lpServerDoc,
            dwAssign
    );
    if (! lpmkDoc) return NULL;

    lpmkItem = ServerDoc_GetSelRelMoniker(
            lpServerDoc,
            lplrSel,
            dwAssign
    );
    if (lpmkItem) {
        CreateGenericComposite(lpmkDoc, lpmkItem, (LPMONIKER FAR*)&lpmkFull);
        OleStdRelease((LPUNKNOWN)lpmkItem);
    }

    if (lpmkDoc)
        OleStdRelease((LPUNKNOWN)lpmkDoc);

    return lpmkFull;
}


/* ServerNameTable_EditLineUpdate
 * -------------------------------
 *
 *      Update the table when a line at nEditIndex is edited.
 */
void ServerNameTable_EditLineUpdate(
        LPSERVERNAMETABLE       lpServerNameTable,
        int                     nEditIndex
)
{
    LPOUTLINENAMETABLE lpOutlineNameTable =
                            (LPOUTLINENAMETABLE)lpServerNameTable;
    LPOUTLINENAME lpOutlineName;
    LINERANGE lrSel;
    LPPSEUDOOBJ lpPseudoObj;
    int i;

    for(i = 0; i < lpOutlineNameTable->m_nCount; i++) {
        lpOutlineName=OutlineNameTable_GetName(lpOutlineNameTable, i);

        lpPseudoObj = ((LPSERVERNAME)lpOutlineName)->m_lpPseudoObj;

        /* if there is a pseudo object associated with this name, then
        **    check if the line that was modified is included within
        **    the named range.
        */
        if (lpPseudoObj) {
            OutlineName_GetSel(lpOutlineName, &lrSel);

            if(((int)lrSel.m_nStartLine <= nEditIndex) &&
                ((int)lrSel.m_nEndLine >= nEditIndex)) {

                // inform linking clients data has changed
            PseudoObj_SendAdvise(
                  lpPseudoObj,
                  OLE_ONDATACHANGE,
                  NULL,   /* lpmkDoc -- not relevant here */
                  0       /* advf -- no flags necessary */
            );
         }

      }
   }
}


/* ServerNameTable_InformAllPseudoObjectsDocRenamed
 * ------------------------------------------------
 *
 *      Inform all pseudo object clients that the name of the pseudo
 *      object has changed.
 */
void ServerNameTable_InformAllPseudoObjectsDocRenamed(
      LPSERVERNAMETABLE       lpServerNameTable,
      LPMONIKER               lpmkDoc
)
{
   LPOUTLINENAMETABLE lpOutlineNameTable =
                           (LPOUTLINENAMETABLE)lpServerNameTable;
   LPOUTLINENAME lpOutlineName;
   LPPSEUDOOBJ lpPseudoObj;
   LPMONIKER lpmkObj;
   int i;

   OLEDBG_BEGIN2("ServerNameTable_InformAllPseudoObjectsDocRenamed\r\n");

   for(i = 0; i < lpOutlineNameTable->m_nCount; i++) {
      lpOutlineName=OutlineNameTable_GetName(lpOutlineNameTable, i);

      lpPseudoObj = ((LPSERVERNAME)lpOutlineName)->m_lpPseudoObj;

      /* if there is a pseudo object associated with this name, then
      **    send OnRename advise to its linking clients.
      */
      if (lpPseudoObj &&
         ((lpmkObj=PseudoObj_GetFullMoniker(lpPseudoObj,lpmkDoc))!=NULL)) {

         // inform the clients that the name has changed
         PseudoObj_SendAdvise (
               lpPseudoObj,
               OLE_ONRENAME,
               lpmkObj,
               0           /* advf -- not relevant here */
         );
      }
   }
   OLEDBG_END2
}
```

```
/* ServerNameTable_InformAllPseudoObjectsDocSaved
 * ------------------------------------------------
 *
 *      Inform all pseudo object clients that the name of the pseudo
 *      object has changed.
 */
void ServerNameTable_InformAllPseudoObjectsDocSaved(
      LPSERVERNAMETABLE       lpServerNameTable,
      LPMONIKER               lpmkDoc
)
{
   LPOUTLINENAMETABLE lpOutlineNameTable =
                            (LPOUTLINENAMETABLE)lpServerNameTable;
   LPOUTLINENAME lpOutlineName;
   LPPSEUDOOBJ lpPseudoObj;
   LPMONIKER lpmkObj;
   int i;

   OLEDBG_BEGIN2("ServerNameTable_InformAllPseudoObjectsDocSaved\r\n");

   for(i = 0; i < lpOutlineNameTable->m_nCount; i++) {
      lpOutlineName=OutlineNameTable_GetName(lpOutlineNameTable, i);

      lpPseudoObj = ((LPSERVERNAME)lpOutlineName)->m_lpPseudoObj;

      /* if there is a pseudo object associated with this name, then
      **    send OnSave advise to its linking clients.
      */
      if (lpPseudoObj &&
         ((lpmkObj=PseudoObj_GetFullMoniker(lpPseudoObj,lpmkDoc))!=NULL)) {

         // inform the clients that the name has been saved
         PseudoObj_SendAdvise (
               lpPseudoObj,
               OLE_ONSAVE,
               NULL,   /* lpmkDoc -- not relevant here */
               0       /* advf -- not relevant here */
         );
      }
   }
   OLEDBG_END2
}


/* ServerNameTable_SendPendingAdvises
 * ----------------------------------
 *
 *      Send any pending change notifications for pseudo objects.
 *  while ReDraw is diabled on the ServerDoc, then change advise
 *  notifications are not sent to pseudo object clients.
 */
void ServerNameTable_SendPendingAdvises(LPSERVERNAMETABLE lpServerNameTable)
{
   LPOUTLINENAMETABLE lpOutlineNameTable =
                           (LPOUTLINENAMETABLE)lpServerNameTable;
   LPSERVERNAME lpServerName;
   int i;

   for(i = 0; i < lpOutlineNameTable->m_nCount; i++) {
      lpServerName = (LPSERVERNAME)OutlineNameTable_GetName(
            lpOutlineNameTable,
            i
      );
      ServerName_SendPendingAdvises(lpServerName);
   }
}


/* ServerNameTable_GetPseudoObj
** ----------------------------
**
**    Return a pointer to a pseudo object identified by an item string
**    (lpszItem). if the pseudo object already exists, then return the
**    existing object, otherwise allocate a new pseudo object.
*/
LPPSEUDOOBJ ServerNameTable_GetPseudoObj(
      LPSERVERNAMETABLE       lpServerNameTable,
      LPOLESTR                lpszItem,
      LPSERVERDOC             lpServerDoc
)
{
   LPSERVERNAME lpServerName;

   lpServerName = (LPSERVERNAME)OutlineNameTable_FindName(
         (LPOUTLINENAMETABLE)lpServerNameTable,
         lpszItem
   );

   if (lpServerName)
      return ServerName_GetPseudoObj(lpServerName, lpServerDoc);
   else
      return NULL;
}


/* ServerNameTable_CloseAllPseudoObjs
 * ----------------------------------
 *
 *  Force all pseudo objects to close. this results in sending OnClose
 *  notification to each pseudo object's linking clients.
 */
void ServerNameTable_CloseAllPseudoObjs(LPSERVERNAMETABLE lpServerNameTable)
{
   LPOUTLINENAMETABLE lpOutlineNameTable =
                           (LPOUTLINENAMETABLE)lpServerNameTable;
   LPSERVERNAME lpServerName;
   int i;
   OLEDBG_BEGIN3("ServerNameTable_CloseAllPseudoObjs\r\n")

   for(i = 0; i < lpOutlineNameTable->m_nCount; i++) {
      lpServerName = (LPSERVERNAME)OutlineNameTable_GetName(
            lpOutlineNameTable,
            i
      );
        ServerName_ClosePseudoObj(lpServerName);
   }

   OLEDBG_END3
}



/* ServerName_SetSel
 * -----------------
 *
 *      Change the line range of a  name.
 */
void ServerName_SetSel(
      LPSERVERNAME            lpServerName,
      LPLINERANGE             lplrSel,
      BOOL                    fRangeModified
)
{
   LPOUTLINENAME lpOutlineName = (LPOUTLINENAME)lpServerName;
   BOOL fPseudoObjChanged = fRangeModified;

   if (lpOutlineName->m_nStartLine != lplrSel->m_nStartLine) {
      lpOutlineName->m_nStartLine = lplrSel->m_nStartLine;
      fPseudoObjChanged = TRUE;
   }

   if (lpOutlineName->m_nEndLine != lplrSel->m_nEndLine) {
      lpOutlineName->m_nEndLine = lplrSel->m_nEndLine;
      fPseudoObjChanged = TRUE;
   }

   /* OLE2NOTE: if the range of an active pseudo object has
   **    changed, then inform any linking clients that the object
   **    has changed.
   */
   if (lpServerName->m_lpPseudoObj && fPseudoObjChanged) {
      PseudoObj_SendAdvise(
            lpServerName->m_lpPseudoObj,
            OLE_ONDATACHANGE,
            NULL,   /* lpmkDoc -- not relevant here */
            0       /* advf -- no flags necessary */
      );
   }
}


/* ServerName_SendPendingAdvises
```

```
 * ------------------------------
 *
 *      Send any pending change notifications for the associated
 *  pseudo objects for this name (if one exists).
 *  while ReDraw is diabled on the ServerDoc, then change advise
 *  notifications are not sent to pseudo object clients.
 */
void ServerName_SendPendingAdvises(LPSERVERNAME lpServerName)
{
   if (! lpServerName->m_lpPseudoObj)
      return;     // no associated pseudo object

   if (lpServerName->m_lpPseudoObj->m_fDataChanged)
      PseudoObj_SendAdvise(
            lpServerName->m_lpPseudoObj,
            OLE_ONDATACHANGE,
            NULL,   /* lpmkDoc -- not relevant here */
            0       /* advf -- no flags necessary */
      );
}


/* ServerName_GetPseudoObj
** -----------------------
**
**    Return a pointer to a pseudo object associated to a ServerName.
**    if the pseudo object already exists, then return the
**    existing object, otherwise allocate a new pseudo object.
**
**    NOTE: the PseudoObj is returned with a 0 refcnt if first created,
**    else the existing refcnt is unchanged.
*/
LPPSEUDOOBJ ServerName_GetPseudoObj(
      LPSERVERNAME            lpServerName,
      LPSERVERDOC             lpServerDoc
)
{
   // Check if a PseudoObj already exists
   if (lpServerName->m_lpPseudoObj)
      return lpServerName->m_lpPseudoObj;

   // A PseudoObj does NOT already exist, allocate a new one.
   lpServerName->m_lpPseudoObj=(LPPSEUDOOBJ) New((DWORD)sizeof(PSEUDOOBJ));
   if (lpServerName->m_lpPseudoObj == NULL) {
      OleDbgAssertSz(lpServerName->m_lpPseudoObj != NULL,   "Error
allocating PseudoObj");
      return NULL;
   }

   PseudoObj_Init(lpServerName->m_lpPseudoObj, lpServerName, lpServerDoc);
   return lpServerName->m_lpPseudoObj;
}


/* ServerName_ClosePseudoObj
```

```
 * -------------------------
 *
 *      if there is an associated pseudo objects for this name (if one
 *  exists), then close it. this results in sending OnClose
 *  notification to the pseudo object's linking clients.
 */
void ServerName_ClosePseudoObj(LPSERVERNAME lpServerName)
{
   if (!lpServerName || !lpServerName->m_lpPseudoObj)
      return;     // no associated pseudo object

   PseudoObj_Close(lpServerName->m_lpPseudoObj);
}


#endif  // OLE_SERVER


#if defined( OLE_CNTR )


/*************************************************************************
** ContainerDoc Supprt Functions Used by Container versions
*************************************************************************/


/* ContainerLine_GetRelMoniker
** ---------------------------
**    Retrieve the relative item moniker which identifies the OLE object
**    relative to the container document.
**
**    Returns NULL if a moniker can NOT be created.
*/
LPMONIKER ContainerLine_GetRelMoniker(
      LPCONTAINERLINE         lpContainerLine,
      DWORD                   dwAssign
)
{
   LPMONIKER lpmk = NULL;

   /* OLE2NOTE: we should only give out a moniker for the OLE object
   **    if the object is allowed to be linked to from the inside. if
   **    so we are allowed to give out a moniker which binds to the
   **    running OLE object). if the object is an OLE 2.0 embedded
   **    object then it is allowed to be linked to from the inside. if
   **    the object is either an OleLink or an OLE 1.0 embedding
   **    then it can not be linked to from the inside.
   **    if we were a container/server app then we could offer linking
   **    to the outside of the object (ie. a pseudo object within our
   **    document). we are a container only app that does not support
   **    linking to ranges of its data.
   */

   switch (dwAssign) {
```

```
case GETMONIKER_FORCEASSIGN:

      /* Force the assignment of the name. This is called when a
      **    Paste Link actually occurs. From now on we want
      **    to inform the OLE object that its moniker is
      **    assigned and is thus necessary to register itself
      **    in the RunningObjectTable.
      */
      CreateItemMoniker(
            OLESTDDELIM, lpContainerLine->m_szStgName, &lpmk);

      /* OLE2NOTE: if the OLE object is already loaded and it
      **    is being assigned a moniker for the first time,
      **    then we need to inform it that it now has a moniker
      **    assigned by calling IOleObject::SetMoniker. this
      **    will force the OLE object to register in the
      **    RunningObjectTable when it enters the running
      **    state. if the object is not currently loaded,
      **    SetMoniker will be called automatically later when
      **    the object is loaded by the function
      **    ContainerLine_LoadOleObject.
      */
      if (! lpContainerLine->m_fMonikerAssigned) {

         /* we must remember forever more that this object has a
         **    moniker assigned.
         */
         lpContainerLine->m_fMonikerAssigned = TRUE;

         // we are now dirty and must be saved
         OutlineDoc_SetModified(
               (LPOUTLINEDOC)lpContainerLine->m_lpDoc,
               TRUE,   /* fModified */
               FALSE,  /* fDataChanged--N/A for container ver. */
               FALSE   /* fSizeChanged--N/A for container ver. */
         );

         if (lpContainerLine->m_lpOleObj) {
            OLEDBG_BEGIN2("IOleObject::SetMoniker called\r\n")
            lpContainerLine->m_lpOleObj->lpVtbl->SetMoniker(
                  lpContainerLine->m_lpOleObj,
                  OLEWHICHMK_OBJREL,
                  lpmk
            );
            OLEDBG_END2
         }
      }
      break;

case GETMONIKER_ONLYIFTHERE:

      /* If the OLE object currently has a moniker assigned,
      **    then return it.
      */
      if (lpContainerLine->m_fMonikerAssigned) {
```

```
                CreateItemMoniker(
                      OLESTDDELIM,
                      lpContainerLine->m_szStgName,
                      &lpmk
                );
            }
            break;

      case GETMONIKER_TEMPFORUSER:

            /* Return the moniker that would be used for the OLE
            **    object but do NOT force moniker assignment at
            **    this point. Since our strategy is to use the
            **    storage name of the object as its item name, we
            **    can simply create the corresponding ItemMoniker
            **    (indepenedent of whether the moniker is currently
            **    assigned or not).
            */
            CreateItemMoniker(
                  OLESTDDELIM,
                  lpContainerLine->m_szStgName,
                  &lpmk
            );
            break;

      case GETMONIKER_UNASSIGN:

            lpContainerLine->m_fMonikerAssigned = FALSE;
            break;

   }

   return lpmk;
}


/* ContainerLine_GetFullMoniker
** ----------------------------
**    Retrieve the full absolute moniker which identifies the OLE object
**    in the container document.
**    this moniker is created as a composite of the absolute moniker for
**    the entire document appended with an item moniker which identifies
**    the OLE object relative to the document.
**    Returns NULL if a moniker can NOT be created.
*/
LPMONIKER ContainerLine_GetFullMoniker(
      LPCONTAINERLINE         lpContainerLine,
      DWORD                   dwAssign
)
{
   LPMONIKER lpmkDoc = NULL;
   LPMONIKER lpmkItem = NULL;
   LPMONIKER lpmkFull = NULL;

   lpmkDoc = OleDoc_GetFullMoniker(
         (LPOLEDOC)lpContainerLine->m_lpDoc,
         dwAssign
   );
   if (! lpmkDoc)
   {
       return NULL;
   }
   lpmkItem = ContainerLine_GetRelMoniker(lpContainerLine, dwAssign);

   if (lpmkItem) {
      CreateGenericComposite(lpmkDoc, lpmkItem, (LPMONIKER FAR*)&lpmkFull);
      OleStdRelease((LPUNKNOWN)lpmkItem);
   }

   if (lpmkDoc)
      OleStdRelease((LPUNKNOWN)lpmkDoc);

   return lpmkFull;
}


/* ContainerDoc_InformAllOleObjectsDocRenamed
** ------------------------------------------
**    Inform all OLE objects that the name of the ContainerDoc has changed.
*/
void ContainerDoc_InformAllOleObjectsDocRenamed(
      LPCONTAINERDOC          lpContainerDoc,
      LPMONIKER               lpmkDoc
)
{
   LPLINELIST lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   int i;
   LPLINE lpLine;

   // artificial AddRef in case someone releases object during call
   OleDoc_AddRef((LPOLEDOC)lpContainerDoc);

   for (i = 0; i < lpLL->m_nNumLines; i++) {
      lpLine=LineList_GetLine(lpLL, i);

      if (lpLine && (Line_GetLineType(lpLine)==CONTAINERLINETYPE)) {
         LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)lpLine;

         /* OLE2NOTE: if the OLE object is already loaded AND the
         **    object already has a moniker assigned, then we need
         **    to inform it that the moniker of the ContainerDoc has
         **    changed. of course, this means the full moniker of
         **    the object has changed. to do this we call
         **    IOleObject::SetMoniker. this will force the OLE
         **    object to re-register in the RunningObjectTable if it
         **    is currently in the running state. it is not in the
         **    running state, the object handler can make not that
         **    the object has a new moniker. if the object is not
         **    currently loaded, SetMoniker will be called
         **    automatically later when the object is loaded by the
```

```
        **    function ContainerLine_LoadOleObject.
        **    also if the object is a linked object, we always want
        **    to call SetMoniker on the link so that in case the
        **    link source is contained within our same container,
        **    the link source will be tracked. the link rebuilds
        **    its absolute moniker if it has a relative moniker.
        */
        if (lpContainerLine->m_lpOleObj) {
           if (lpContainerLine->m_fMonikerAssigned ||
              lpContainerLine->m_dwLinkType != 0) {
              OLEDBG_BEGIN2("IOleObject::SetMoniker called\r\n")
              lpContainerLine->m_lpOleObj->lpVtbl->SetMoniker(
                    lpContainerLine->m_lpOleObj,
                    OLEWHICHMK_CONTAINER,
                    lpmkDoc
              );
              OLEDBG_END2
           }

           /* OLE2NOTE: we must call IOleObject::SetHostNames so
           **    any open objects can update their window titles.
           */
           OLEDBG_BEGIN2("IOleObject::SetHostNames called\r\n")
           lpContainerLine->m_lpOleObj->lpVtbl->SetHostNames(
                 lpContainerLine->m_lpOleObj,
                 (LPOLESTR)APPNAME,
                 ((LPOUTLINEDOC)lpContainerDoc)->m_lpszDocTitle
           );
           OLEDBG_END2
        }
     }
  }
  // release artificial AddRef
  OleDoc_Release((LPOLEDOC)lpContainerDoc);
}


/* ContainerDoc_GetObject
** ----------------------
**    Return a pointer to the desired interface of an object identified
**    by an item string (lpszItem). the object returned will be an OLE
**    object (either link or embedding).
**
**      OLE2NOTE: we must force the object to run because we are
**          REQUIRED to return a pointer the OLE object in the
**          RUNNING state.
**
**    dwSpeedNeeded indicates how long the caller is willing
**    to wait for us to get the object:
**      BINDSPEED_IMMEDIATE -- only if obj already loaded && IsRunning
**      BINDSPEED_MODERATE  -- load obj if necessary && if IsRunning
**      BINDSPEED_INDEFINITE-- force obj to load and run if necessary
*/
HRESULT ContainerDoc_GetObject(
      LPCONTAINERDOC          lpContainerDoc,
      LPOLESTR                lpszItem,
      DWORD                   dwSpeedNeeded,
      REFIID                  riid,
      LPVOID FAR*             lplpvObject
)
{
   LPLINELIST  lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   int         i;
   LPLINE      lpLine;
   BOOL        fMatchFound = FALSE;
   DWORD       dwStatus;
   HRESULT     hrErr;

   *lplpvObject = NULL;

   for (i = 0; i < lpLL->m_nNumLines; i++) {
      lpLine=LineList_GetLine(lpLL, i);

      if (lpLine && (Line_GetLineType(lpLine)==CONTAINERLINETYPE)) {
         LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)lpLine;

         if (OLESTRCMP(lpContainerLine->m_szStgName, lpszItem) == 0) {

            fMatchFound = TRUE;     // valid item name

            // check if object is loaded.
            if (lpContainerLine->m_lpOleObj == NULL) {

               // if BINDSPEED_IMMEDIATE is requested, object must
               // ALREADY be loadded.
               if (dwSpeedNeeded == BINDSPEED_IMMEDIATE)
                  return ResultFromScode(MK_E_EXCEEDEDDEADLINE);

               ContainerLine_LoadOleObject(lpContainerLine);
               if (! lpContainerLine->m_lpOleObj)
                  return ResultFromScode(E_OUTOFMEMORY);
            }

            /* OLE2NOTE: check if the object is allowed to be linked
            **    to from the inside (ie. we are allowed to
            **    give out a moniker which binds to the running
            **    OLE object). if the object is an OLE
            **    2.0 embedded object then it is allowed to be
            **    linked to from the inside. if the object is
            **    either an OleLink or an OLE 1.0 embedding
            **    then it can not be linked to from the inside.
            **    if we were a container/server app then we
            **    could offer linking to the outside of the
            **    object (ie. a pseudo object within our
            **    document). we are a container only app that
            **    does not support linking to ranges of its data.
            */
            OLEDBG_BEGIN2("IOleObject::GetMiscStatus called\r\n");
            lpContainerLine->m_lpOleObj->lpVtbl->GetMiscStatus(
                  lpContainerLine->m_lpOleObj,
                  DVASPECT_CONTENT, /* aspect is not important */
                  (LPDWORD)&dwStatus
            );
            OLEDBG_END2
            if (dwStatus & OLEMISC_CANTLINKINSIDE)
            {
               return ResultFromScode(MK_E_NOOBJECT);
            }

            // check if object is running.
            if (! OleIsRunning(lpContainerLine->m_lpOleObj)) {

               // if BINDSPEED_MODERATE is requested, object must
               // ALREADY be running.
               if (dwSpeedNeeded == BINDSPEED_MODERATE)
               {
                  return ResultFromScode(MK_E_EXCEEDEDDEADLINE);
               }

               /* OLE2NOTE: we have found a match for the item name.
               **    now we must return a pointer to the desired
               **    interface on the RUNNING object. we must
               **    carefully load the object and initially ask for
               **    an interface that we are sure the loaded form of
               **    the object supports. if we immediately ask the
               **    loaded object for the desired interface, the
               **    QueryInterface call might fail if it is an
               **    interface that is supported only when the object
               **    is running. thus we force the object to load and
               **    return its IUnknown*. then we force the object to
               **    run, and then finally, we can ask for the
               **    actually requested interface.
               */
               hrErr = ContainerLine_RunOleObject(lpContainerLine);
               if (hrErr != NOERROR) {
                  return hrErr;
               }
            }

            // Retrieve the requested interface
            *lplpvObject = OleStdQueryInterface(
                  (LPUNKNOWN)lpContainerLine->m_lpOleObj, riid);

            break;  // Match FOUND!
         }
      }
   }

   if (*lplpvObject != NULL) {
      return NOERROR;
   } else
   {
      return (fMatchFound ? ResultFromScode(E_NOINTERFACE)
                     : ResultFromScode(MK_E_NOOBJECT));
   }
```

```
}


/* ContainerDoc_GetObjectStorage
** ----------------------------
**    Return a pointer to the IStorage* used by the object identified
**    by an item string (lpszItem). the object identified could be either
**    an OLE object (either link or embedding).
*/
HRESULT ContainerDoc_GetObjectStorage(
      LPCONTAINERDOC          lpContainerDoc,
      LPOLESTR                lpszItem,
      LPSTORAGE FAR*          lplpStg
)
{
   LPLINELIST lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
   int i;
   LPLINE lpLine;

   *lplpStg = NULL;

   for (i = 0; i < lpLL->m_nNumLines; i++) {
      lpLine=LineList_GetLine(lpLL, i);

      if (lpLine && (Line_GetLineType(lpLine)==CONTAINERLINETYPE)) {
         LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)lpLine;

         if (OLESTRCMP(lpContainerLine->m_szStgName, lpszItem) == 0) {

            *lplpStg = lpContainerLine->m_lpStg;
            break;  // Match FOUND!
         }
      }
   }

   if (*lplpStg != NULL) {
      return NOERROR;
   } else
   {
      return ResultFromScode(MK_E_NOOBJECT);
   }
}


/* ContainerDoc_IsRunning
** ----------------------
**    Check if the object identified by an item string (lpszItem) is in
**    the running state.
**    For a container-only app, a check is made if the OLE object
**    associated with the item name is running.
*/
HRESULT ContainerDoc_IsRunning(LPCONTAINERDOC   lpContainerDoc, LPOLESTR
lpszItem)
{
   LPLINELIST  lpLL = &((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
```

```
int         i;
LPLINE      lpLine;
DWORD       dwStatus;

for (i = 0; i < lpLL->m_nNumLines; i++) {
   lpLine=LineList_GetLine(lpLL, i);

   if (lpLine && (Line_GetLineType(lpLine)==CONTAINERLINETYPE)) {
      LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)lpLine;

      if (OLESTRCPY(lpContainerLine->m_szStgName, lpszItem) == 0) {

         /* OLE2NOTE: we have found a match for the item name.
         **    now we must check if the OLE object is running.
         **    we will load the object if not already loaded.
         */
         if (! lpContainerLine->m_lpOleObj) {
            ContainerLine_LoadOleObject(lpContainerLine);
            if (! lpContainerLine->m_lpOleObj)
               return ResultFromScode(E_OUTOFMEMORY);
         }

         /* OLE2NOTE: check if the object is allowed to be linked
         **    to from the inside (ie. we are allowed to
         **    give out a moniker which binds to the running
         **    OLE object). if the object is an OLE
         **    2.0 embedded object then it is allowed to be
         **    linked to from the inside. if the object is
         **    either an OleLink or an OLE 1.0 embedding
         **    then it can not be linked to from the inside.
         **    if we were a container/server app then we
         **    could offer linking to the outside of the
         **    object (ie. a pseudo object within our
         **    document). we are a container only app that
         **    does not support linking to ranges of its data.
         */
         OLEDBG_BEGIN2("IOleObject::GetMiscStatus called\r\n")
         lpContainerLine->m_lpOleObj->lpVtbl->GetMiscStatus(
               lpContainerLine->m_lpOleObj,
               DVASPECT_CONTENT, /* aspect is not important */
               (LPDWORD)&dwStatus
         );
         OLEDBG_END2
         if (dwStatus & OLEMISC_CANTLINKINSIDE)
         {
            return ResultFromScode(MK_E_NOOBJECT);
         }

         if (OleIsRunning(lpContainerLine->m_lpOleObj))
         {
            return NOERROR;
         }
         else
         {
            return ResultFromScode(S_FALSE);
            }
         }
      }
   }
   // no object was found corresponding to the item name
   return ResultFromScode(MK_E_NOOBJECT);
}

#endif  // OLE_CNTR
```

MAIN.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    main.c
**
**    This file contains initialization functions which are WinMain,
**    WndProc, and OutlineApp_InitalizeMenu.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "ansiapi.h"
#include "outline.h"
#if defined( USE_STATUSBAR )
#include "status.h"
#endif

#if !defined( WIN32 )
#if defined( USE_CTL3D )
//#include "..\include\ctl3d.h"
#include "ctl3d.h"
#endif  // USE_CTL3D
#endif  // !WIN32

#include "initguid.h"            // forces our GUIDs to be initialized
#include "defguid.h"

#if defined( OLE_CNTR )
//*************************************************************************

#if defined( INPLACE_CNTR )
OLEDBGDATA_MAIN("ICNTR")
#else
OLEDBGDATA_MAIN("CNTR")
#endif

CONTAINERAPP g_OutlineApp;  // Global App object maintains app instance
state

/* Global interface Vtbl's
 * OLE2NOTE: we only need one copy of each Vtbl. When an object which
 *      exposes an interface is instantiated, its lpVtbl is intialized
 *      to point to one of these global Vtbl's.
 */
IUnknownVtbl            g_OleApp_UnknownVtbl;
IClassFactoryVtbl       g_OleApp_ClassFactoryVtbl;
IMessageFilterVtbl      g_OleApp_MessageFilterVtbl;

IUnknownVtbl            g_OleDoc_UnknownVtbl;
IPersistFileVtbl        g_OleDoc_PersistFileVtbl;
```

```
IOleItemContainerVtbl   g_OleDoc_OleItemContainerVtbl;
IExternalConnectionVtbl g_OleDoc_ExternalConnectionVtbl;
IDataObjectVtbl         g_OleDoc_DataObjectVtbl;

#if defined( USE_DRAGDROP )
IDropSourceVtbl         g_OleDoc_DropSourceVtbl;
IDropTargetVtbl         g_OleDoc_DropTargetVtbl;
#endif  // USE_DRAGDROP

IOleUILinkContainerVtbl g_CntrDoc_OleUILinkContainerVtbl;

IOleClientSiteVtbl      g_CntrLine_UnknownVtbl;
IOleClientSiteVtbl      g_CntrLine_OleClientSiteVtbl;
IAdviseSinkVtbl         g_CntrLine_AdviseSinkVtbl;

#if defined( INPLACE_CNTR )
IOleInPlaceSiteVtbl     g_CntrLine_OleInPlaceSiteVtbl;
IOleInPlaceFrameVtbl    g_CntrApp_OleInPlaceFrameVtbl;
BOOL g_fInsideOutContainer = FALSE;     // default to outside-in activation
#endif  // INPLACE_CNTR

//*********************************************************************
#endif  // OLE_CNTR

#if defined( OLE_SERVER )
//*********************************************************************

#if defined( INPLACE_SVR )
OLEDBGDATA_MAIN("ISVR")
#else
OLEDBGDATA_MAIN("SVR")
#endif

SERVERAPP g_OutlineApp; // Global App object maintains app instance state

/* Global interface Vtbl's
 * OLE2NOTE: we only need one copy of each Vtbl. When an object which
 *      exposes an interface is instantiated, its lpVtbl is intialized
 *      to point to one of these global Vtbl's.
 */
IUnknownVtbl            g_OleApp_UnknownVtbl;
IClassFactoryVtbl       g_OleApp_ClassFactoryVtbl;
IMessageFilterVtbl      g_OleApp_MessageFilterVtbl;

IUnknownVtbl            g_OleDoc_UnknownVtbl;
IPersistFileVtbl        g_OleDoc_PersistFileVtbl;
IOleItemContainerVtbl   g_OleDoc_OleItemContainerVtbl;
IExternalConnectionVtbl g_OleDoc_ExternalConnectionVtbl;
IDataObjectVtbl         g_OleDoc_DataObjectVtbl;

#if defined( USE_DRAGDROP )
IDropSourceVtbl         g_OleDoc_DropSourceVtbl;
IDropTargetVtbl         g_OleDoc_DropTargetVtbl;
#endif  // USE_DRAGDROP
```

```
IOleObjectVtbl          g_SvrDoc_OleObjectVtbl;
IPersistStorageVtbl     g_SvrDoc_PersistStorageVtbl;

#if defined( SVR_TREATAS )
IStdMarshalInfoVtbl     g_SvrDoc_StdMarshalInfoVtbl;
#endif  // SVR_TREATAS

#if defined( INPLACE_SVR )
IOleInPlaceObjectVtbl       g_SvrDoc_OleInPlaceObjectVtbl;
IOleInPlaceActiveObjectVtbl g_SvrDoc_OleInPlaceActiveObjectVtbl;
#endif // INPLACE_SVR

IUnknownVtbl            g_PseudoObj_UnknownVtbl;
IOleObjectVtbl          g_PseudoObj_OleObjectVtbl;
IDataObjectVtbl         g_PseudoObj_DataObjectVtbl;

//**************************************************************************
#endif  // OLE_SVR

#if !defined( OLE_VERSION )
OLEDBGDATA_MAIN("OUTL")
OUTLINEAPP g_OutlineApp;    // Global App object maintains app instance
state
#endif

LPOUTLINEAPP g_lpApp=(LPOUTLINEAPP)&g_OutlineApp;   // ptr to global app obj
RECT        g_rectNull = {0, 0, 0, 0};
UINT        g_uMsgHelp = 0;  // help msg from ole2ui dialogs
BOOL        g_fAppActive = FALSE;

/* WinMain
** -------
**    Main routine for the Windows application.
*/
int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                  LPSTR lpszCmdLine, int nCmdShow)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   MSG         msg;            /* MSG structure to store your messages */

#if defined( OLE_VERSION )
   /* OLE2NOTE: it is recommended that all OLE applications to set
   **    their message queue size to 96. this improves the capacity
   **    and performance of OLE's LRPC mechanism.
   */
   int cMsg = 96;   // recommend msg queue size for OLE
   while (cMsg && ! SetMessageQueue(cMsg))  // take largest size we can get.
      cMsg -= 8;
   if (! cMsg)
      return -1;  // ERROR: we got no message queue
#endif

#if defined( USE_CTL3D )
   Ctl3dRegister(hInstance);
   Ctl3dAutoSubclass(hInstance);
```

```
#endif

   if(! hPrevInstance) {
      /* register window classes if first instance of application */
      if(! OutlineApp_InitApplication(lpOutlineApp, hInstance))
         return 0;
   }

   /* Create App Frame window */
   if (! OutlineApp_InitInstance(lpOutlineApp, hInstance, nCmdShow))
      return 0;

   if (! OutlineApp_ParseCmdLine(lpOutlineApp, lpszCmdLine, nCmdShow))
      return 0;

   lpOutlineApp->m_hAccelApp = LoadAccelerators(hInstance, APPACCEL);
   lpOutlineApp->m_hAccelFocusEdit = LoadAccelerators(hInstance,
         FB_EDIT_ACCEL);
   lpOutlineApp->m_hAccel = lpOutlineApp->m_hAccelApp;
   lpOutlineApp->m_hWndAccelTarget = lpOutlineApp->m_hWndApp;


   // Main message loop
   while(GetMessage(&msg, NULL, 0, 0)) {        /* Until WM_QUIT message */
      if(!MyTranslateAccelerator(&msg)) {
         TranslateMessage(&msg);
         DispatchMessage(&msg);
      }
   }

#if defined( OLE_VERSION )
   OleApp_TerminateApplication((LPOLEAPP)lpOutlineApp);
#else
   /* OLE2NOTE: CoInitialize() is called in OutlineApp_InitInstance
   **    and therefore we need to uninitialize it when exit.
   */
   CoUninitialize();
#endif

#if defined( USE_CTL3D )
   Ctl3dUnregister(hInstance);
#endif

   return msg.wParam;

} /*  End of WinMain */

BOOL MyTranslateAccelerator(LPMSG lpmsg)
{
   // if it's not a keystroke it can not be an accelerator
   if (lpmsg->message < WM_KEYFIRST || lpmsg->message > WM_KEYLAST)
      return FALSE;

   if (g_lpApp->m_hWndAccelTarget &&
      TranslateAccelerator(g_lpApp->m_hWndAccelTarget,
                                        g_lpApp->m_hAccel,lpmsg))
      return TRUE;

#if defined( INPLACE_SVR )
   /* OLE2NOTE: if we are in-place active and we did not translate the
   **    accelerator, we need to give the top-level (frame) in-place
   **    container a chance to translate the accelerator.
   **    we ONLY need to call OleTranslateAccelerator API if the
   **    message is a keyboard message. otherwise it is harmless but
   **    unnecessary.
   **
   **    NOTE: even a in-place server that does NOT have any
   **    Accelerators must still call OleTranslateAccelerator for all
   **    keyboard messages so that the server's OWN menu mneumonics
   **    (eg. &Edit -- Alt-e) function properly.
   **
   **    NOTE: an in-place server MUST check that the accelerator is
   **    NOT one of its own accelerators BEFORE calling
   **    OleTranslateAccelerator which tries to see if it is a
   **    container accelerator. if this is a server accelerator that
   **    was not translateed because the associated menu command was
   **    disabled, we MUST NOT call OleTranslateAccelerator. The
   **    IsAccelerator helper API has been added to assist with this
   **    check.
   */
   if (g_OutlineApp.m_lpIPData &&
      !IsAccelerator(g_lpApp->m_hAccel,
         GetAccelItemCount(g_lpApp->m_hAccel), lpmsg,NULL) &&
      OleTranslateAccelerator(g_OutlineApp.m_lpIPData->lpFrame,
            (LPOLEINPLACEFRAMEINFO)&g_OutlineApp.m_lpIPData->frameInfo,
            lpmsg) == NOERROR) {
      return TRUE;
   }
#endif

   return FALSE;
}


/****************************************************************************/
/*                                                                          */
/* Main Window Procedure                                                    */
/*                                                                          */
/* This procedure provides service routines for the Windows events         */
/* (messages) that Windows sends to the window, as well as the user        */
/* initiated events (messages) that are generated when the user selects */
/* the action bar and pulldown menu controls or the corresponding          */
/* keyboard accelerators.                                                   */
/*                                                                          */
/****************************************************************************/

LRESULT FAR PASCAL AppWndProc(HWND hWnd, UINT Message, WPARAM wParam, LPARAM
lParam)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)GetWindowLong(hWnd, 0);
   LPOUTLINEDOC lpOutlineDoc = NULL;
#if defined( OLE_VERSION )
   LPOLEAPP lpOleApp = (LPOLEAPP)lpOutlineApp;
#endif
#if defined( OLE_CNTR )
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)lpOutlineApp;
   char          szAnsiString[256];
#endif
   HWND          hWndDoc = NULL;

#if defined( USE_FRAMETOOLS )
   LPFRAMETOOLS lptb = OutlineApp_GetFrameTools(lpOutlineApp);
#endif

   if (lpOutlineApp) {
      lpOutlineDoc = OutlineApp_GetActiveDoc(lpOutlineApp);

      if (lpOutlineDoc)
         hWndDoc = OutlineDoc_GetWindow(lpOutlineDoc);
   }

   switch (Message) {
      case WM_COMMAND:
      {
#ifdef WIN32
         WORD wID    = LOWORD(wParam);
#else
         WORD wID    = wParam;
#endif

#if defined( INPLACE_CNTR )
         LPCONTAINERDOC lpContainerDoc = (LPCONTAINERDOC)lpOutlineDoc;
         LPOLEDOC lpOleDoc = (LPOLEDOC)lpOutlineDoc;

         /* OLE2NOTE: see context sensitive help technote (CSHELP.DOC)
         **    m_fMenuHelpMode flag is set when F1 is pressed when a
         **    menu item is selected. this flag is set in
         **    IOleInPlaceFrame::ContextSensitveHelp method.
         **    m_fCSHelpMode flag is set when SHIFT-F1 context
         **    sensitive help is entered. this flag is set in
         **    IOleInPlaceSite::ContextSensitiveHelp method.
         **    if either of these flags are set then the WM_COMMAND
         **    message is received then, the corresponding command
         **    should NOT executed; help can be given (if desired).
         **    also the context sensitve help mode should be exited.
         **    the two different cases have their own way to exit
         **    the mode (please refer to the technote).
         */
         if (lpOleDoc &&
            (lpContainerApp->m_fMenuHelpMode||lpOleDoc->m_fCSHelpMode) &&
            (wID > IDM_FILE)   /* min wID for app command */ &&
            (wID!=IDM_FB_EDIT) /* special wID to control FormulaBar */ ) {

            if ((lpContainerApp->m_fMenuHelpMode)) {
               LPOLEINPLACEACTIVEOBJECT lpIPActiveObj =
                        lpContainerApp->m_lpIPActiveObj;

                lpContainerApp->m_fMenuHelpMode = FALSE;

                // inform the in-place active object that we handled the
                //   menu help mode (F1) selection.
                if (lpIPActiveObj) {

OLEDBG_BEGIN2("IOleInPlaceActiveObject::ContextSensitiveHelp(FALSE)
called\r\n")
                    lpIPActiveObj->lpVtbl->ContextSensitiveHelp(
                            lpIPActiveObj, FALSE);
                    OLEDBG_END2
                }
            }

            if ((lpOleDoc->m_fCSHelpMode)) {
                LPOLEINPLACEOBJECT lpIPObj;
                LPCONTAINERLINE lpLastIpActiveLine =
                        lpContainerDoc->m_lpLastIpActiveLine;

                lpOleDoc->m_fCSHelpMode = FALSE;

                /* inform immediate in-place container parent and,
                **    if we were a container/server, immediate
                **    in-place object children that we handled the
                **    context sensitive help mode.
                */
                if (lpLastIpActiveLine &&
                        (lpIPObj=lpLastIpActiveLine->m_lpOleIPObj)!=NULL){

OLEDBG_BEGIN2("IOleInPlaceObject::ContextSensitiveHelp(FALSE) called\r\n")
                    lpIPObj->lpVtbl->ContextSensitiveHelp(lpIPObj, FALSE);
                    OLEDBG_END2
                }
            }

            // if we provided help, we would do it here...

            // remove context sensitive help cursor
            SetCursor(LoadCursor(NULL,IDC_ARROW));
            return 0L;
        }
#endif  // INPLACE_CNTR

        switch (wID) {

            case IDM_F_NEW:
                OleDbgIndent(-2);   // Reset debug output indent level
                OleDbgOutNoPrefix2("\r\n");

                OLEDBG_BEGIN3("OutlineApp_NewCommand\r\n")
                OutlineApp_NewCommand(lpOutlineApp);
                OLEDBG_END3
```

```
#if defined( OLE_CNTR )
               /* OLE2NOTE: this call will attempt to recover
               **    resources by unloading DLL's that were loaded
               **    by OLE and are no longer being used. it is a
               **    good idea to call this API now and then if
               **    your app tends to run for a long time.
               **    otherwise these DLL's will be unloaded when
               **    the app exits. some apps may want to call
               **    this as part of idle-time processing. this
               **    call is optional.
               */
               OLEDBG_BEGIN2("CoFreeUnusedLibraries called\r\n")
               CoFreeUnusedLibraries();
               OLEDBG_END2
#endif

#if defined( USE_FRAMETOOLS )
               OutlineDoc_UpdateFrameToolButtons(
                     OutlineApp_GetActiveDoc(lpOutlineApp));
#endif
               break;

            case IDM_F_OPEN:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineApp_OpenCommand\r\n")
               OutlineApp_OpenCommand(lpOutlineApp);
               OLEDBG_END3

#if defined( OLE_CNTR )
               /* OLE2NOTE: this call will attempt to recover
               **    resources by unloading DLL's that were loaded
               **    by OLE and are no longer being used. it is a
               **    good idea to call this API now and then if
               **    your app tends to run for a long time.
               **    otherwise these DLL's will be unloaded when
               **    the app exits. some apps may want to call
               **    this as part of idle-time processing. this
               **    call is optional.
               */
               OLEDBG_BEGIN2("CoFreeUnusedLibraries called\r\n")
               CoFreeUnusedLibraries();
               OLEDBG_END2
#endif

#if defined( USE_FRAMETOOLS )
               OutlineDoc_UpdateFrameToolButtons(
                     OutlineApp_GetActiveDoc(lpOutlineApp));
#endif
               break;

            case IDM_F_SAVE:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineApp_SaveCommand\r\n")
```

```
                OutlineApp_SaveCommand(lpOutlineApp);
                OLEDBG_END3
                break;

            case IDM_F_SAVEAS:
                OleDbgOutNoPrefix2("\r\n");

                OLEDBG_BEGIN3("OutlineApp_SaveAsCommand\r\n")
                OutlineApp_SaveAsCommand(lpOutlineApp);
                OLEDBG_END3
                break;

            case IDM_F_PRINT:
                OleDbgOutNoPrefix2("\r\n");

                OLEDBG_BEGIN3("OutlineApp_PrintCommand\r\n")
                OutlineApp_PrintCommand(lpOutlineApp);
                OLEDBG_END3
                break;

            case IDM_F_PRINTERSETUP:
                OleDbgOutNoPrefix2("\r\n");

                OLEDBG_BEGIN3("OutlineApp_PrinterSetupCommand\r\n")
                OutlineApp_PrinterSetupCommand(lpOutlineApp);
                OLEDBG_END3
                break;

            case IDM_F_EXIT:
                SendMessage(hWnd, WM_CLOSE, 0, 0L);
                break;

            case IDM_H_ABOUT:
                OutlineApp_AboutCommand(lpOutlineApp);
                break;

#if defined( INPLACE_CNTR )
            case IDM_ESCAPE:
            {
                /* ESCAPE key pressed */
                LPCONTAINERDOC lpContainerDoc =
                        (LPCONTAINERDOC)lpOutlineDoc;

                /* OLE2NOTE: The standard OLE 2.0 UI convention
                **    is to have ESCAPE key exit in-place
                **    activation (ie. UIDeactivate). If
                **    possible it is recommended for both
                **    in-place servers AND in-place containers
                **    to take responsibility to handle the
                **    ESCAPE key accelerator. The server has
                **    the first crack at handling accelerator
                **    keys and normally the server should do
                **    the UIDeactivation. It is a good idea for
                **    in-place containers, in order to
                **    guarantee consistent behavior, to also
```

```
               **    handle the ESCAPE key and UIDeactivate
               **    the object in case the object does not do
               **    it itself. normally this should be
               **    unnecessary.
               */
               if (lpContainerDoc->m_lpLastUIActiveLine &&
                  lpContainerDoc->m_lpLastUIActiveLine->m_fUIActive)
               {
                  ContainerLine_UIDeactivate(
                        lpContainerDoc->m_lpLastUIActiveLine);
               }
               break;
            }
#endif  // INPLACE_CNTR


            default:
               // forward message to document window
               if (hWndDoc) {
                  return DocWndProc(hWndDoc, Message,wParam,lParam);
               }
         }

         break;  /* End of WM_COMMAND */
      }

      case WM_INITMENU:
         OutlineApp_InitMenu(lpOutlineApp, lpOutlineDoc, (HMENU)wParam);
         break;

#if defined( OLE_VERSION )

      /* OLE2NOTE: WM_INITMENUPOPUP is trapped primarily for the Edit
      **    menu. We didn't update the Edit menu until it is popped
      **    up to avoid the overheads of the OLE calls which are
      **    required to initialize some Edit menu items.
      */
      case WM_INITMENUPOPUP:
      {
         HMENU hMenuEdit = GetSubMenu(lpOutlineApp->m_hMenuApp, 1);
#if defined( INPLACE_CNTR )
         LPCONTAINERDOC lpContainerDoc = (LPCONTAINERDOC)lpOutlineDoc;

         /* OLE2NOTE: we must check if there is an object currently
         **    in-place UIActive. if so, then our edit menu is not
         **    on the menu; we do not want to bother updating the
         **    edit menu when it is not even there.
         */
         if (lpContainerDoc && lpContainerDoc->m_lpLastUIActiveLine &&
            lpContainerDoc->m_lpLastUIActiveLine->m_fUIActive)
            break;  // an object is in-place UI active
#endif
         if ((HMENU)wParam == hMenuEdit &&
            (LOWORD(lParam) == POS_EDITMENU) &&
            OleDoc_GetUpdateEditMenuFlag((LPOLEDOC)lpOutlineDoc)) {
```

```
            OleApp_UpdateEditMenu(lpOleApp, lpOutlineDoc, hMenuEdit);
         }
         break;
      }
#endif      // OLE_VERSION

      case WM_SIZE:
         if (wParam != SIZE_MINIMIZED)
            OutlineApp_ResizeWindows(lpOutlineApp);
         break;


      case WM_ACTIVATEAPP:
#if defined (OLE_CNTR)
         if (g_fAppActive = (BOOL) wParam)
            OleApp_QueryNewPalette(lpOleApp);
#endif

#if defined( INPLACE_CNTR )
         {
            BOOL fActivate = (BOOL)wParam;
            LPOLEINPLACEACTIVEOBJECT lpIPActiveObj =
                  lpContainerApp->m_lpIPActiveObj;

            /* OLE2NOTE: the in-place container MUST inform the
            **    inner most in-place active object (this is NOT
            **    necessarily our immediate child if there are
            **    nested levels of embedding) of the WM_ACTIVATEAPP
            **    status.
            */
            if (lpIPActiveObj) {
#if defined( _DEBUG )
               OLEDBG_BEGIN2((fActivate ?
                  "IOleInPlaceActiveObject::OnFrameWindowActivate(TRUE)
called\r\n" :
                  "IOleInPlaceActiveObject::OnFrameWindowActivate(FALSE)
called\r\n"))
#endif  // _DEUBG
               lpIPActiveObj->lpVtbl->OnFrameWindowActivate(
                  lpIPActiveObj, fActivate);
               OLEDBG_END2
            }
         }

#endif  // INPLACE_CNTR

         // OLE2NOTE: We can't call OutlineDoc_UpdateFrameToolButtons
         //           right away which
         //           would generate some OLE calls and eventually
         //           WM_ACTIVATEAPP and a loop was formed. Therefore, we
         //           should delay the frame tool initialization until
         //           WM_ACTIVATEAPP is finished by posting a message
         //           to ourselves.
         //           we want to ignore the WM_ACTIVATEAPP that comes
         //           as we bring up a modal dialog.
         /* Update enable/disable state of buttons in toolbar */
         if (wParam
#if defined( OLE_VERSION )
               && lpOleApp->m_cModalDlgActive == 0
#endif
         ) {
            PostMessage(hWnd, WM_U_INITFRAMETOOLS, 0, 0L);
         }
         return 0L;

      case WM_SETFOCUS:
         SetFocus(hWndDoc);
         break;


#if defined( OLE_CNTR )
      case WM_QUERYNEWPALETTE:
         if (!g_fAppActive)
            return 0L;

         return OleApp_QueryNewPalette(lpOleApp);

      case WM_PALETTECHANGED:
      {
         HWND hWndPalChg = (HWND) wParam;
         static BOOL fInPaletteChanged = FALSE;

         if (fInPaletteChanged)  // Guard against recursion
            return 0L;

         fInPaletteChanged = TRUE;

         if (hWnd != hWndPalChg)
            wSelectPalette(hWnd, lpOleApp->m_hStdPal,TRUE/*fBackground*/);

#if defined( INPLACE_CNTR )
         /* OLE2NOTE: always forward the WM_PALETTECHANGED message (via
         **    SendMessage) to any in-place objects that currently have
         **    their window visible. this gives these objects the chance
         **    to select their palettes. this is
         **    REQUIRED by all in-place containers independent of
         **    whether they use color palettes themselves--their objects
         **    may use color palettes.
         **    (see ContainerDoc_ForwardPaletteChangedMsg for more info)
         */
         if (lpOutlineDoc){
            ContainerDoc_ForwardPaletteChangedMsg(
                  (LPCONTAINERDOC)lpOutlineDoc, hWndPalChg);
         }
#endif  // INPLACE_CNTR

         fInPaletteChanged = FALSE;
         return 0L;
      }
```

```
#endif  // OLE_CNTR

      case WM_DESTROY:
         PostQuitMessage(0);
         break;

      case WM_CLOSE:  /* close the window */

         /* Close all active documents. if successful, then exit */
         OleDbgOutNoPrefix2("\r\n");

         OutlineApp_CloseAllDocsAndExitCommand(lpOutlineApp, FALSE);
         break;

      case WM_QUERYENDSESSION:
      {
#if defined( OLE_CNTR )
         /* OLE2NOTE: we are not able to make OLE LRPC calls when
         **    WM_QUERYENDSESSION is recieved (this is a
         **    SendMessage). this means, for example, that we are
         **    NOT able to ask objects to save. thus the most we can
         **    do is ask the user if he wants to exit with
         **    discarding changes or else abort shutting down.
         */

//???         W2A (APPNAME, szAnsiString, 256);

         int nResponse = MessageBox(
               hWnd,
               "Discard changes?",
               szAnsiString,
               MB_ICONQUESTION | MB_OKCANCEL
         );
         if(nResponse == IDOK)
            return 1L;      /* can terminate */

#endif
#if defined( OLE_SERVER )
         /* OLE2NOTE: an embedded object should never prompt whether
         **    it should be saved (according the OLE 2.0 User
         **    Model). therefore, an embedded object will never
         **    complain that it needs to be saved. it will always
         **    allow the QueryEndSession to proceed.
         */
         if (lpOutlineApp->m_lpDoc->m_docInitType == DOCTYPE_EMBEDDED)
            return 1L;      /* can terminate */
         else
#endif
         {
            /* this is not an embedded object; it is a user
            **    document. we will prompt if the user wants to
            **    save the document now in WM_QUERYENDSESSION. if
            **    the user cancels then that would abort the
            **    shutdown. if the user does not abort, then later
            **    in WM_ENDSESSION the document will be actually
            **    closed.
            **
            **    Because this is an SDI app, there is only one
            **    document. An MDI would need to loop through all
            **    open documents.
            */
            DWORD dwSaveOption = OLECLOSE_PROMPTSAVE;
            if (OutlineDoc_CheckSaveChanges(
                  lpOutlineApp->m_lpDoc, &dwSaveOption))
               return 1L;      /* can terminate */
         }

         /* else: can't terminate now */

         break;
      }

#if defined( OLE_VERSION)
      case WM_ENDSESSION:
      {
         BOOL fEndSession = (BOOL)wParam;

         if (fEndSession) {
            OutlineApp_CloseAllDocsAndExitCommand(lpOutlineApp, TRUE);
            return 0L;
         }
      }
      break;
#endif  // OLE_VERSION


#if defined( USE_STATUSBAR )
      case WM_MENUSELECT:
      {
         LPSTR lpszMessage;
         OLECHAR lpszUniMessage[256];
#ifdef WIN32
         UINT fuFlags    = (UINT)HIWORD(wParam);
         UINT uItem      = (UINT)LOWORD(wParam);
#else
         UINT fuFlags    = (UINT)LOWORD(lParam);
         UINT uItem      = (UINT)wParam;
#endif

         if (uItem == 0 && fuFlags == (UINT)-1) {
            GetControlMessage(STATUS_READY, &lpszMessage);
            A2W (lpszMessage, lpszUniMessage, 256);
            OutlineApp_SetStatusText(lpOutlineApp, lpszUniMessage);
         }
         else if (fuFlags & MF_POPUP) {
#ifdef WIN32
            HMENU hMainMenu = (HMENU)lParam;
            HMENU hPopupMenu = GetSubMenu(hMainMenu,uItem);
#else
            HMENU hPopupMenu = (HMENU)wParam;
```

```
#endif
            GetPopupMessage(hPopupMenu, &lpszMessage);
            A2W (lpszMessage, lpszUniMessage, 256);
            OutlineApp_SetStatusText(lpOutlineApp, lpszUniMessage);
         }
         else if (fuFlags & MF_SYSMENU) {
            GetSysMenuMessage(uItem, &lpszMessage);
            A2W (lpszMessage, lpszUniMessage, 256);
            OutlineApp_SetStatusText(lpOutlineApp, lpszUniMessage);
         }
         else if (uItem != 0) {  // Command Item
            GetItemMessage(uItem, &lpszMessage);
            A2W (lpszMessage, lpszUniMessage, 256);
            OutlineApp_SetStatusText(lpOutlineApp, lpszUniMessage);
         }
         else {
            GetControlMessage(STATUS_BLANK, &lpszMessage);
            A2W (lpszMessage, lpszUniMessage, 256);
            OutlineApp_SetStatusText(lpOutlineApp, lpszUniMessage);
         }
         break;
      }
#endif  // USE_STATUSBAR


#if defined( USE_FRAMETOOLS )
      case WM_U_INITFRAMETOOLS:
         OutlineDoc_UpdateFrameToolButtons(lpOutlineDoc);
         break;
#endif

      default:
         /* For any message for which you don't specifically provide a  */
         /* service routine, you should return the message to Windows   */
         /* for default message processing.                             */

         return DefWindowProc(hWnd, Message, wParam, lParam);
   }

   return (LRESULT)0;
}     /* End of AppWndProc*/


/****************************************************************************/
/*                                                                          */
/* Document Window Procedure                                                */
/*                                                                          */
/*   The Document Window is the parent of the OwnerDraw Listbox which       */
/* maintains the list of lines in the current document. This window         */
/* receives the ownerdraw callback messages from the list box.              */
/****************************************************************************/

LRESULT FAR PASCAL DocWndProc(HWND hWnd, UINT Message, WPARAM wParam, LPARAM
lParam)
{
   LPOUTLINEAPP    lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPOUTLINEDOC    lpOutlineDoc = (LPOUTLINEDOC)GetWindowLong(hWnd, 0);
   LPLINELIST      lpLL = OutlineDoc_GetLineList(lpOutlineDoc);
   LPSCALEFACTOR   lpscale = OutlineDoc_GetScaleFactor(lpOutlineDoc);

#if defined( OLE_VERSION )
   LPOLEAPP lpOleApp = (LPOLEAPP)lpOutlineApp;
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpOutlineDoc;
#if defined( OLE_CNTR )
   LPCONTAINERDOC lpContainerDoc = (LPCONTAINERDOC)lpOutlineDoc;
#endif
#if defined( OLE_SERVER )
   LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOutlineDoc;
#endif
#if defined( INPLACE_CNTR )
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)lpOutlineApp;
#endif
#endif  // OLE_VERSION

   switch(Message) {

#if defined( INPLACE_SVR )

      /* OLE2NOTE: ISVROTL doesn't use color palettes. The inplace objects
      **    that use color palettes must implement the following
      **    lines of code.
      **
         case WM_QUERYNEWPALETTE:
            return wSelectPalette(hWnd, hPal, FALSE); // foreground

         case WM_PALETTECHANGED:
            if (hWnd != (HWND) wParam)
               wSelectPalette(hWnd, hPal, TRUE);     // background
         break;
      **
      **
      **
      */
#endif

      case WM_MEASUREITEM:
      {
         LPMEASUREITEMSTRUCT lpmis = ((LPMEASUREITEMSTRUCT)lParam);

         switch (wParam) {
            case IDC_LINELIST:
            {
               HDC hDC = LineList_GetDC(lpLL);
               UINT uHeight;

               uHeight=Line_GetHeightInHimetric((LPLINE)lpmis->itemData);
               uHeight = XformHeightInHimetricToPixels(hDC, uHeight);
               uHeight = (UINT) (uHeight * lpscale->dwSyN /
                     lpscale->dwSyD);
               if (uHeight >LISTBOX_HEIGHT_LIMIT)
                  uHeight = LISTBOX_HEIGHT_LIMIT;

               lpmis->itemHeight = uHeight;
               LineList_ReleaseDC(lpLL, hDC);
               break;
            }

            case IDC_NAMETABLE:
            {
               // NOTE: NameTable is never made visible. do nothing.
               break;
            }

#if defined( USE_HEADING )
            case IDC_ROWHEADING:
            {
               UINT uHeight;

               uHeight = LOWORD(lpmis->itemData);
               uHeight = (UINT) (uHeight * lpscale->dwSyN /
                     lpscale->dwSyD);
               if (uHeight >LISTBOX_HEIGHT_LIMIT)
                  uHeight = LISTBOX_HEIGHT_LIMIT;
               lpmis->itemHeight = uHeight;
               break;
            }

            case IDC_COLHEADING:
            {
               UINT uHeight;

               uHeight = LOWORD(lpmis->itemData);
               uHeight = (UINT) (uHeight * lpscale->dwSyN /
                     lpscale->dwSyD);
               if (uHeight > LISTBOX_HEIGHT_LIMIT)
                  uHeight = LISTBOX_HEIGHT_LIMIT;
               lpmis->itemHeight = uHeight;
               break;
            }
#endif  // USE_HEADING

         }
         return (LRESULT)TRUE;
      }

      case WM_DRAWITEM:
      {
         LPDRAWITEMSTRUCT lpdis = ((LPDRAWITEMSTRUCT)lParam);

         switch (lpdis->CtlID) {

            case IDC_LINELIST:
            {
               RECT   rcClient;
               RECT   rcDevice;
               HWND   hWndLL = LineList_GetWindow(lpLL);
               LPLINE lpLine = (LPLINE)lpdis->itemData;

               // NOTE: When itemID == -1, the listbox is empty.
               //       We are supposed to draw focus rect only
               //       But it is not done in this app. If this line is
               //       removed, the app will crash in Line_DrawToScreen
               //       because of invalid lpLine.
               if (lpdis->itemID == -1)
                  break;

               GetClientRect(hWndLL, &rcClient);

               rcDevice = lpdis->rcItem;

               // shift the item rect to account for horizontal scrolling

               rcDevice.left += rcClient.right - lpdis->rcItem.right;

#if defined( OLE_CNTR )
               /* we need to remember the horizontal scroll offset
               **    needed for the in-place object's window.
               **    (this is specific to ICNTROTL)
               */
               if(lpdis->itemAction & ODA_DRAWENTIRE) {
                  if (Line_GetLineType(lpLine) == CONTAINERLINETYPE)
                     ((LPCONTAINERLINE)lpLine)->m_nHorizScrollShift =
                        rcDevice.left;
               }
#endif  // OLE_CNTR

               // shift rect for left margin
               rcDevice.left += (int)(XformWidthInHimetricToPixels(NULL,
                     LOWORD(OutlineDoc_GetMargin(lpOutlineDoc))) *
                     lpscale->dwSxN / lpscale->dwSxD);

               rcDevice.right = rcDevice.left +
                     (int)(XformWidthInHimetricToPixels(lpdis->hDC,
                           Line_GetWidthInHimetric(lpLine)) *
                     lpscale->dwSxN / lpscale->dwSxD);

               Line_DrawToScreen(
                     lpLine,
                     lpdis->hDC,
                     &lpdis->rcItem,
                     lpdis->itemAction,
                     lpdis->itemState,
                     &rcDevice
               );

#if defined( USE_FRAMETOOLS )
               if (lpdis->itemState & ODS_FOCUS)
                  OutlineDoc_SetFormulaBarEditText(lpOutlineDoc,lpLine);
#endif
```

```
               break;
            }
            case IDC_NAMETABLE:
            {
               // NOTE: NameTable is never made visible. do nothing
               break;
            }

#if defined( USE_HEADING )
            case IDC_ROWHEADING:
            {
               LPHEADING lphead;

               // Last dummy item shouldn't be drawn
               if (lpdis->itemID == (UINT)LineList_GetCount(lpLL))
                  break;

               // only DrawEntire need be trapped as window is disabled
               if (lpdis->itemAction == ODA_DRAWENTIRE) {
                  lphead = OutlineDoc_GetHeading(lpOutlineDoc);
                  Heading_RH_Draw(lphead, lpdis);
               }
               break;
            }

            case IDC_COLHEADING:
            {
               RECT   rect;
               RECT   rcDevice;
               RECT   rcLogical;
               LPHEADING lphead;

               // only DrawEntire need be trapped as window is disabled
               if (lpdis->itemAction == ODA_DRAWENTIRE) {
                  lphead = OutlineDoc_GetHeading(lpOutlineDoc);
                  GetClientRect(lpdis->hwndItem, &rect);

                  rcDevice = lpdis->rcItem;

                  // shift the item rect to account for
                  // horizontal scrolling
                  rcDevice.left = -(rcDevice.right - rect.right);

                  // shift rect for left margin
                  rcDevice.left += (int)(XformWidthInHimetricToPixels(
                        NULL,
                        LOWORD(OutlineDoc_GetMargin(lpOutlineDoc))) *
                     lpscale->dwSxN / lpscale->dwSxD);

                  rcDevice.right = rcDevice.left + (int)lpscale->dwSxN;
                  rcLogical.left = 0;
                  rcLogical.bottom = 0;
                  rcLogical.right = (int)lpscale->dwSxD;
                  rcLogical.top = LOWORD(lpdis->itemData);
                  Heading_CH_Draw(lphead, lpdis, &rcDevice, &rcLogical);
               }
               break;
            }
#endif  // USE_HEADING

         }
         return (LRESULT)TRUE;
      }

      case WM_SETFOCUS:
         if (lpLL)
            SetFocus(LineList_GetWindow(lpLL));
         break;

#if !defined( OLE_VERSION )
      case WM_RENDERFORMAT:
      {
         LPOUTLINEDOC lpClipboardDoc = lpOutlineApp->m_lpClipboardDoc;
         if (lpClipboardDoc)
            OutlineDoc_RenderFormat(lpClipboardDoc, wParam);

         break;
      }
      case WM_RENDERALLFORMATS:
      {
         LPOUTLINEDOC lpClipboardDoc = lpOutlineApp->m_lpClipboardDoc;
         if (lpClipboardDoc)
            OutlineDoc_RenderAllFormats(lpClipboardDoc);

         break;
      }
      case WM_DESTROYCLIPBOARD:
         if (g_lpApp->m_lpClipboardDoc) {
            OutlineDoc_Destroy(g_lpApp->m_lpClipboardDoc);
            g_lpApp->m_lpClipboardDoc = NULL;
         }
         break;

#endif   // OLE_VERSION

#if defined( OLE_CNTR )
      case WM_U_UPDATEOBJECTEXTENT:
      {
         /* Update the extents of any OLE object that is marked that
         **    its size may  have changed. when an
         **    IAdviseSink::OnViewChange notification is received,
         **    the corresponding ContainerLine is marked
         **    (m_fDoGetExtent==TRUE) and a message
         **    (WM_U_UPDATEOBJECTEXTENT) is posted to the document
         **    indicating that there are dirty objects.
         */
         ContainerDoc_UpdateExtentOfAllOleObjects(lpContainerDoc);
         break;
      }
```

```
#endif  // OLE_CNTR

#if defined( INPLACE_SVR ) || defined( INPLACE_CNTR )
      /* OLE2NOTE: Any window that is used during in-place activation
      **    must handle the WM_SETCURSOR message or else the cursor
      **    of the in-place parent will be used. if WM_SETCURSOR is
      **    not handled, then DefWindowProc sends the message to the
      **    window's parent.
      **
      **    see context sensitive help technote (CSHELP.DOC).
      **    m_fCSHelpMode flag is set when SHIFT-F1 context
      **    sensitive help is entered.
      **    if this flag is set then the context sensitive help
      **    cursor should be shown.
      */
      case WM_SETCURSOR:
         if (lpOleDoc->m_fCSHelpMode)
            SetCursor(UICursorLoad(IDC_CONTEXTHELP));
         else
            SetCursor(LoadCursor(NULL, IDC_ARROW) );
         return (LRESULT)TRUE;
#endif  // INPLACE_SVR || INPLACE_CNTR

#if defined( INPLACE_SVR )
      /* OLE2NOTE: when the in-place active, our in-place server
      **    document window (passed to IOleInPlaceFrame::SetMenu)
      **    will receive the WM_INITMENU and WM_INITMENUPOPUP messages.
      */
      case WM_INITMENU:
         OutlineApp_InitMenu(lpOutlineApp, lpOutlineDoc, (HMENU)wParam);
         break;

      /* OLE2NOTE: WM_INITMENUPOPUP is trapped primarily for the Edit
      **    menu. We didn't update the Edit menu until it is popped
      **    up to avoid the overheads of the OLE calls which are
      **    required to initialize some Edit menu items.
      */
      case WM_INITMENUPOPUP:
      {
         HMENU hMenuEdit = GetSubMenu(lpOutlineApp->m_hMenuApp, 1);
         if ((HMENU)wParam == hMenuEdit &&
            (LOWORD(lParam) == POS_EDITMENU) &&
            OleDoc_GetUpdateEditMenuFlag((LPOLEDOC)lpOutlineDoc)) {
            OleApp_UpdateEditMenu(
                  (LPOLEAPP)lpOutlineApp, lpOutlineDoc, hMenuEdit);
         }
         break;
      }
#endif      // INPLACE_SVR

#if defined( INPLACE_SVR ) && defined( USE_STATUSBAR )
      /* OLE2NOTE: when the server is in-place active the
      **    WM_MENUSELECT message is sent to the object's window and
      **    not the server app's frame window. processing this
      **    message allows there in-place server to give status bar
      **    help text for menu commands.
      */
      case WM_MENUSELECT:
      {
         LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOleDoc;
         LPSTR lpszMessage;
         OLECHAR szUniMessage[256];

#ifdef WIN32
         UINT fuFlags    = (UINT)HIWORD(wParam);
         UINT uItem      = (UINT)LOWORD(wParam);
#else
         UINT fuFlags    = (UINT)LOWORD(lParam);
         UINT uItem      = (UINT)wParam;
#endif

         if (uItem == 0 && fuFlags == (UINT)-1) {
            GetControlMessage(STATUS_READY, &lpszMessage);
            A2W( lpszMessage, szUniMessage, 256);
            ServerDoc_SetStatusText(lpServerDoc, szUniMessage);
         }
         else if (fuFlags & MF_POPUP) {
#ifdef WIN32
            HMENU hMainMenu = (HMENU)lParam;
            HMENU hPopupMenu = GetSubMenu(hMainMenu,uItem);
#else
            HMENU hPopupMenu = (HMENU)wParam;
#endif
            GetPopupMessage(hPopupMenu, &lpszMessage);
            A2W( lpszMessage, szUniMessage, 256);
            ServerDoc_SetStatusText(lpServerDoc, szUniMessage);
         }
         else if (uItem != 0) {  // Command Item
            GetItemMessage(uItem, &lpszMessage);
            A2W( lpszMessage, szUniMessage, 256);
            ServerDoc_SetStatusText(lpServerDoc, szUniMessage);
         }
         else {
            GetControlMessage(STATUS_BLANK, &lpszMessage);
            A2W( lpszMessage, szUniMessage, 256);
            ServerDoc_SetStatusText(lpServerDoc, szUniMessage);
         }
         break;
      }
#endif  // INPLACE_SVR && USE_STATUSBAR
#if defined( INPLACE_SVR ) && defined( USE_FRAMETOOLS )

      case WM_U_INITFRAMETOOLS:
         OutlineDoc_UpdateFrameToolButtons(lpOutlineDoc);
         break;
#endif      // INPLACE_SVR && USE_FRAMETOOLS


      case WM_COMMAND:
      {
```

```
#ifdef WIN32
         WORD wNotifyCode = HIWORD(wParam);
         WORD wID          = LOWORD(wParam);
         HWND hwndCtl     = (HWND) lParam;
#else
         WORD wNotifyCode = HIWORD(lParam);
         WORD wID             = wParam;
         HWND hwndCtl     = (HWND) LOWORD(lParam);
#endif

#if defined( INPLACE_SVR )
         /* OLE2NOTE: see context sensitive help technote (CSHELP.DOC)
         **    m_fMenuHelpMode flag is set when F1 is pressed when a
         **    menu item is selected. this flag is set in
         **    IOleInPlaceActiveObject::ContextSensitveHelp method.
         **    m_fCSHelpMode flag is set when SHIFT-F1 context
         **    sensitive help is entered. this flag is set in
         **    IOleInPlaceObject::ContextSensitiveHelp method.
         **    if either of these flags are set then the WM_COMMAND
         **    message is received then, the corresponding command
         **    should NOT executed; help can be given (if desired).
         **    also the context sensitve help mode should be exited.
         **    the two different cases have their own way to exit
         **    the mode (please refer to the technote).
         */
         if (lpOleDoc &&
            (lpServerDoc->m_fMenuHelpMode||lpOleDoc->m_fCSHelpMode) &&
            (wID > IDM_FILE)   /* min wID for app command */ &&
            (wID!=IDM_FB_EDIT) /* special wID to control FormulaBar */ ) {

            if ((lpServerDoc->m_fMenuHelpMode)) {
               LPOLEINPLACEFRAME lpFrame;

               lpServerDoc->m_fMenuHelpMode = FALSE;

               // inform top-level frame that we handled the
               //   menu help mode (F1) selection.
               if (lpServerDoc->m_lpIPData &&
                     (lpFrame=lpServerDoc->m_lpIPData->lpFrame)!=NULL){

OLEDBG_BEGIN2("IOleInPlaceFrame::ContextSensitiveHelp(FALSE) called\r\n")
                  lpFrame->lpVtbl->ContextSensitiveHelp(lpFrame, FALSE);
                  OLEDBG_END2
               }
            }

            if ((lpOleDoc->m_fCSHelpMode)) {
               LPOLEINPLACESITE lpSite;

               lpOleDoc->m_fCSHelpMode = FALSE;

               /* inform immediate in-place container parent and,
               **    if we were a container/server, immediate
               **    in-place object children that we handled the
               **    context sensitive help mode.
               */
               if (lpServerDoc->m_lpIPData &&
                     (lpSite=lpServerDoc->m_lpIPData->lpSite) !=NULL) {

OLEDBG_BEGIN2("IOleInPlaceSite::ContextSensitiveHelp(FALSE) called\r\n")
                  lpSite->lpVtbl->ContextSensitiveHelp(lpSite, FALSE);
                  OLEDBG_END2
               }
            }

            // if we provided help, we would do it here...

            // remove context sensitive help cursor
            SetCursor(LoadCursor(NULL,IDC_ARROW));
            return 0L;
         }
#endif  // INPLACE_SVR
#if defined( INPLACE_CNTR )

         /* OLE2NOTE: see context sensitive help technote (CSHELP.DOC)
         **    m_fMenuHelpMode flag is set when F1 is pressed when a
         **    menu item is selected. this flag is set in
         **    IOleInPlaceFrame::ContextSensitveHelp method.
         **    m_fCSHelpMode flag is set when SHIFT-F1 context
         **    sensitive help is entered. this flag is set in
         **    IOleInPlaceSite::ContextSensitiveHelp method.
         **    if either of these flags are set then the WM_COMMAND
         **    message is received then, the corresponding command
         **    should NOT executed; help can be given (if desired).
         **    also the context sensitve help mode should be exited.
         **    the two different cases have their own way to exit
         **    the mode (please refer to the technote).
         */
         if (lpOleDoc &&
            (lpContainerApp->m_fMenuHelpMode||lpOleDoc->m_fCSHelpMode) &&
            (wID > IDM_FILE)   /* min wID for app command */ &&
            (wID!=IDM_FB_EDIT) /* special wID to control FormulaBar */ ) {

            if ((lpContainerApp->m_fMenuHelpMode)) {
               LPOLEINPLACEACTIVEOBJECT lpIPActiveObj =
                     lpContainerApp->m_lpIPActiveObj;

               lpContainerApp->m_fMenuHelpMode = FALSE;

               // inform the in-place active object that we handled the
               //   menu help mode (F1) selection.
               if (lpIPActiveObj) {

OLEDBG_BEGIN2("IOleInPlaceActiveObject::ContextSensitiveHelp(FALSE)
called\r\n")
                  lpIPActiveObj->lpVtbl->ContextSensitiveHelp(
                        lpIPActiveObj, FALSE);
                  OLEDBG_END2
               }
            }
            if ((lpOleDoc->m_fCSHelpMode)) {
               LPOLEINPLACEOBJECT lpIPObj;
               LPCONTAINERLINE lpLastIpActiveLine =
                     lpContainerDoc->m_lpLastIpActiveLine;

               lpOleDoc->m_fCSHelpMode = FALSE;

               /* inform immediate in-place container parent and,
               **    if we were a container/server, immediate
               **    in-place object children that we handled the
               **    context sensitive help mode.
               */
               if (lpLastIpActiveLine &&
                     (lpIPObj=lpLastIpActiveLine->m_lpOleIPObj)!=NULL){

OLEDBG_BEGIN2("IOleInPlaceObject::ContextSensitiveHelp(FALSE) called\r\n")
                  lpIPObj->lpVtbl->ContextSensitiveHelp(lpIPObj, FALSE);
                  OLEDBG_END2
               }
            }

            // if we provided help, we would do it here...

            // remove context sensitive help cursor
            SetCursor(LoadCursor(NULL,IDC_ARROW));
            return 0L;
         }
#endif  // INPLACE_CNTR

         switch (wID) {

            /*********************************************************
            ** File new, open, save and print as well as Help about
            **    are duplicated in this switch statement and they are
            **    used to trap the message from the toolbar
            **
            *********************************************************/

            case IDM_F_NEW:
               OleDbgIndent(-2);   // Reset debug output indent level
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineApp_NewCommand\r\n")
               OutlineApp_NewCommand(lpOutlineApp);
               OLEDBG_END3

#if defined( OLE_CNTR )
               /* OLE2NOTE: this call will attempt to recover
               **    resources by unloading DLL's that were loaded
               **    by OLE and are no longer being used. it is a
               **    good idea to call this API now and then if
               **    your app tends to run for a long time.
               **    otherwise these DLL's will be unloaded when
               **    the app exits. some apps may want to call
               **    this as part of idle-time processing. this
               **    call is optional.
               */
               OLEDBG_BEGIN2("CoFreeUnusedLibraries called\r\n")
               CoFreeUnusedLibraries();
               OLEDBG_END2
#endif

#if defined( USE_FRAMETOOLS )
               OutlineDoc_UpdateFrameToolButtons(
                     OutlineApp_GetActiveDoc(lpOutlineApp));
#endif
               break;

            case IDM_F_OPEN:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineApp_OpenCommand\r\n")
               OutlineApp_OpenCommand(lpOutlineApp);
               OLEDBG_END3

#if defined( OLE_CNTR )
               /* OLE2NOTE: this call will attempt to recover
               **    resources by unloading DLL's that were loaded
               **    by OLE and are no longer being used. it is a
               **    good idea to call this API now and then if
               **    your app tends to run for a long time.
               **    otherwise these DLL's will be unloaded when
               **    the app exits. some apps may want to call
               **    this as part of idle-time processing. this
               **    call is optional.
               */
               OLEDBG_BEGIN2("CoFreeUnusedLibraries called\r\n")
               CoFreeUnusedLibraries();
               OLEDBG_END2
#endif

#if defined( USE_FRAMETOOLS )
               OutlineDoc_UpdateFrameToolButtons(
                     OutlineApp_GetActiveDoc(lpOutlineApp));
#endif
               break;

            case IDM_F_SAVE:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineApp_SaveCommand\r\n")
               OutlineApp_SaveCommand(lpOutlineApp);
               OLEDBG_END3
               break;

            case IDM_F_PRINT:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineApp_PrintCommand\r\n")
               OutlineApp_PrintCommand(lpOutlineApp);
               OLEDBG_END3
               break;


            case IDM_E_UNDO:
               // SORRY. NOT IMPLEMENTED
               break;

            case IDM_E_CUT:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineDoc_CutCommand\r\n")
               OutlineDoc_CutCommand(lpOutlineDoc);
               OLEDBG_END3

#if defined( USE_FRAMETOOLS )
               OutlineDoc_UpdateFrameToolButtons(lpOutlineDoc);
#endif
               break;

            case IDM_E_COPY:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineDoc_CopyCommand\r\n")
               OutlineDoc_CopyCommand(lpOutlineDoc);
               OLEDBG_END3

#if defined( USE_FRAMETOOLS )
               OutlineDoc_UpdateFrameToolButtons(lpOutlineDoc);
#endif
               break;

            case IDM_E_PASTE:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineDoc_PasteCommand\r\n")
               OutlineDoc_PasteCommand(lpOutlineDoc);
               OLEDBG_END3

#if defined( USE_FRAMETOOLS )
               OutlineDoc_UpdateFrameToolButtons(lpOutlineDoc);
#endif
               break;

#if defined( OLE_VERSION )
            case IDM_E_PASTESPECIAL:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OleDoc_PasteSpecialCommand\r\n")
               OleDoc_PasteSpecialCommand((LPOLEDOC)lpOutlineDoc);
               OLEDBG_END3

#if defined( USE_FRAMETOOLS )
               OutlineDoc_UpdateFrameToolButtons(lpOutlineDoc);
```

```
#endif
               break;

#endif  // OLE_VERSION

            case IDM_E_CLEAR:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineDoc_ClearCommand\r\n")
               OutlineDoc_ClearCommand(lpOutlineDoc);
               OLEDBG_END3

#if defined( OLE_CNTR )
               /* OLE2NOTE: this call will attempt to recover
               **    resources by unloading DLL's that were loaded
               **    by OLE and are no longer being used. it is a
               **    good idea to call this API now and then if
               **    your app tends to run for a long time.
               **    otherwise these DLL's will be unloaded when
               **    the app exits. some apps may want to call
               **    this as part of idle-time processing. this
               **    call is optional.
               */
               OLEDBG_BEGIN2("CoFreeUnusedLibraries called\r\n")
               CoFreeUnusedLibraries();
               OLEDBG_END2
#endif

#if defined( USE_FRAMETOOLS )
               OutlineDoc_UpdateFrameToolButtons(lpOutlineDoc);
#endif
               break;

            case IDM_L_ADDLINE:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineDoc_AddTextLineCommand\r\n")
               OutlineDoc_AddTextLineCommand(lpOutlineDoc);
               OLEDBG_END3

#if defined( USE_FRAMETOOLS )
               OutlineDoc_UpdateFrameToolButtons(lpOutlineDoc);
               SetFocus(LineList_GetWindow(lpLL));
#endif
               break;

            case IDM_L_EDITLINE:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineDoc_EditLineCommand\r\n")
               OutlineDoc_EditLineCommand(lpOutlineDoc);
               OLEDBG_END3
               SetFocus(LineList_GetWindow(lpLL));
               break;
            case IDM_L_INDENTLINE:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineDoc_IndentCommand\r\n")
               OutlineDoc_IndentCommand(lpOutlineDoc);
               OLEDBG_END3
               break;

            case IDM_L_UNINDENTLINE:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineDoc_UnindentCommand\r\n")
               OutlineDoc_UnindentCommand(lpOutlineDoc);
               OLEDBG_END3
               break;

            case IDM_L_SETLINEHEIGHT:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineDoc_SetLineHeight\r\n")
               OutlineDoc_SetLineHeightCommand(lpOutlineDoc);
               OLEDBG_END3
               break;

            case IDM_E_SELECTALL:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineDoc_SelectAllCommand\r\n")
               OutlineDoc_SelectAllCommand(lpOutlineDoc);
               OLEDBG_END3
               break;

#if defined( OLE_CNTR )
            case IDM_E_INSERTOBJECT:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("ContainerDoc_InsertOleObjectCommand\r\n")
               ContainerDoc_InsertOleObjectCommand(lpContainerDoc);
               OLEDBG_END3

#if defined( USE_FRAMETOOLS )
               OutlineDoc_UpdateFrameToolButtons(lpOutlineDoc);
#endif
               break;

            case IDM_E_EDITLINKS:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("ContainerDoc_EditLinksCommand\r\n")
               ContainerDoc_EditLinksCommand(lpContainerDoc);
               OLEDBG_END3
               break;

            case IDM_E_CONVERTVERB:
               OleDbgOutNoPrefix2("\r\n");
               OLEDBG_BEGIN3("ContainerDoc_ConvertCommand\r\n")
               ContainerDoc_ConvertCommand(
                     lpContainerDoc, FALSE /*fMustActivate*/);
               OLEDBG_END3
               break;


            case IDM_E_PASTELINK:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("ContainerDoc_PasteLinkCommand\r\n")
               ContainerDoc_PasteLinkCommand(lpContainerDoc);
               OLEDBG_END3

#if defined( USE_FRAMETOOLS )
               OutlineDoc_UpdateFrameToolButtons(lpOutlineDoc);
#endif
               break;

#endif  // OLE_CNTR

            case IDM_N_DEFINENAME:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineDoc_DefineNameCommand\r\n")
               OutlineDoc_DefineNameCommand(lpOutlineDoc);
               OLEDBG_END3
               break;

            case IDM_N_GOTONAME:
               OleDbgOutNoPrefix2("\r\n");

               OLEDBG_BEGIN3("OutlineDoc_GotoNameCommand\r\n")
               OutlineDoc_GotoNameCommand(lpOutlineDoc);
               OLEDBG_END3
               break;

#if defined( USE_FRAMETOOLS )
            case IDM_O_BB_TOP:
               FrameTools_BB_SetState(
                     lpOutlineDoc->m_lpFrameTools, BARSTATE_TOP);
               OutlineDoc_AddFrameLevelTools(lpOutlineDoc);
               break;

            case IDM_O_BB_BOTTOM:
               FrameTools_BB_SetState(
                     lpOutlineDoc->m_lpFrameTools, BARSTATE_BOTTOM);
               OutlineDoc_AddFrameLevelTools(lpOutlineDoc);
               break;

            case IDM_O_BB_POPUP:
               FrameTools_BB_SetState(
                     lpOutlineDoc->m_lpFrameTools, BARSTATE_POPUP);
               OutlineDoc_AddFrameLevelTools(lpOutlineDoc);
```

```
                break;

            case IDM_O_BB_HIDE:
                FrameTools_BB_SetState(
                        lpOutlineDoc->m_lpFrameTools, BARSTATE_HIDE);
                OutlineDoc_AddFrameLevelTools(lpOutlineDoc);
                break;

            case IDM_O_FB_TOP:
                FrameTools_FB_SetState(
                        lpOutlineDoc->m_lpFrameTools, BARSTATE_TOP);
                OutlineDoc_AddFrameLevelTools(lpOutlineDoc);
                break;

            case IDM_O_FB_BOTTOM:
                FrameTools_FB_SetState(
                        lpOutlineDoc->m_lpFrameTools, BARSTATE_BOTTOM);
                OutlineDoc_AddFrameLevelTools(lpOutlineDoc);
                break;

            case IDM_O_FB_POPUP:
                FrameTools_FB_SetState(
                        lpOutlineDoc->m_lpFrameTools, BARSTATE_POPUP);
                OutlineDoc_AddFrameLevelTools(lpOutlineDoc);
                break;

            case IDM_FB_EDIT:

                switch (wNotifyCode) {
                    case EN_SETFOCUS:
                        OutlineDoc_SetFormulaBarEditFocus(
                                lpOutlineDoc, TRUE);
                        OutlineDoc_UpdateFrameToolButtons(lpOutlineDoc);
                        break;

                    case EN_KILLFOCUS:
                        OutlineDoc_SetFormulaBarEditFocus(
                                lpOutlineDoc, FALSE);
                        OutlineDoc_UpdateFrameToolButtons(lpOutlineDoc);
                        break;
                }
                break;

            case IDM_FB_CANCEL:

                SetFocus(hWnd);
                break;


            case IDM_F2:
                SendMessage(hWnd, WM_COMMAND, (WPARAM)IDM_FB_EDIT,
                        MAKELONG(0, EN_SETFOCUS));
                break;
#endif  // USE_FRAMETOOLS
```

```
            case IDM_ESCAPE:
               /* ESCAPE key pressed */

#if defined( USE_FRAMETOOLS )
               if (OutlineDoc_IsEditFocusInFormulaBar(lpOutlineDoc))
                  SendMessage(
                     hWnd, WM_COMMAND,(WPARAM)IDM_FB_CANCEL,(LPARAM)0);
#endif  // USE_FRAMETOOLS

#if defined( INPLACE_SVR )
               else {
                  LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOutlineDoc;

                  /* OLE2NOTE: The standard OLE 2.0 UI convention
                  **    is to have ESCAPE key exit in-place
                  **    activation (ie. UIDeactivate). If
                  **    possible it is recommended for both
                  **    in-place servers AND in-place containers
                  **    to take responsibility to handle the
                  **    ESCAPE key accelerator. The server has
                  **    the first crack at handling accelerator
                  **    keys and normally the server should do
                  **    the UIDeactivation.
                  */
                  if (lpServerDoc->m_fUIActive) {
                     SvrDoc_IPObj_UIDeactivate( (LPOLEINPLACEOBJECT)&
                           lpServerDoc->m_OleInPlaceObject);
                  }
               }
#endif  // INPLACE_SVR

               break;


#if defined( USE_HEADING )
            case IDC_BUTTON:
               if (wNotifyCode == BN_CLICKED) {
                  SendMessage(hWnd, WM_COMMAND, IDM_E_SELECTALL, 0L);
                  SetFocus(hWnd);
               }
               break;

            case IDM_O_HEAD_SHOW:
               OutlineDoc_ShowHeading(lpOutlineDoc, TRUE);
               break;

            case IDM_O_HEAD_HIDE:
               OutlineDoc_ShowHeading(lpOutlineDoc, FALSE);
               break;
#endif  // USE_HEADING


#if defined( OLE_CNTR )
            case IDM_O_SHOWOBJECT:
            {
               LPCONTAINERDOC lpContainerDoc =
                        (LPCONTAINERDOC)lpOutlineDoc;
               BOOL        fShowObject;

               fShowObject = !ContainerDoc_GetShowObjectFlag(
                     lpContainerDoc);
               ContainerDoc_SetShowObjectFlag(
                     lpContainerDoc, fShowObject);
               LineList_ForceRedraw(lpLL, TRUE);

               break;
            }
#endif  // OLE_CNTR

#if !defined( OLE_CNTR )
            // Container does not allow zoom factors > 100%
            case IDM_V_ZOOM_400:
            case IDM_V_ZOOM_300:
            case IDM_V_ZOOM_200:
#endif      // !OLE_CNTR

            case IDM_V_ZOOM_100:
            case IDM_V_ZOOM_75:
            case IDM_V_ZOOM_50:
            case IDM_V_ZOOM_25:
               OutlineDoc_SetCurrentZoomCommand(lpOutlineDoc, wID);
               break;

            case IDM_V_SETMARGIN_0:
            case IDM_V_SETMARGIN_1:
            case IDM_V_SETMARGIN_2:
            case IDM_V_SETMARGIN_3:
            case IDM_V_SETMARGIN_4:
               OutlineDoc_SetCurrentMarginCommand(lpOutlineDoc, wID);
               break;

            case IDM_V_ADDTOP_1:
            case IDM_V_ADDTOP_2:
            case IDM_V_ADDTOP_3:
            case IDM_V_ADDTOP_4:
            {
               UINT nHeightInHimetric;

               switch (wID) {
                  case IDM_V_ADDTOP_1:
                     nHeightInHimetric = 1000;
                     break;

                  case IDM_V_ADDTOP_2:
                     nHeightInHimetric = 2000;
                     break;

                  case IDM_V_ADDTOP_3:
                     nHeightInHimetric = 3000;
                     break;
                  case IDM_V_ADDTOP_4:
                     nHeightInHimetric = 4000;
                     break;
               }

               OutlineDoc_AddTopLineCommand(
                     lpOutlineDoc, nHeightInHimetric);
               break;
            }


            case IDM_H_ABOUT:
               OutlineApp_AboutCommand(lpOutlineApp);
               break;

            case IDM_D_DEBUGLEVEL:
               SetDebugLevelCommand();
               break;

#if defined( OLE_VERSION )
            case IDM_D_INSTALLMSGFILTER:
               InstallMessageFilterCommand();
               break;

            case IDM_D_REJECTINCOMING:
               RejectIncomingCommand();
               break;
#endif  // OLE_VERSION

#if defined( INPLACE_CNTR )
            case IDM_D_INSIDEOUT:
               g_fInsideOutContainer = !g_fInsideOutContainer;

               // force all object to unload so they can start new
               // activation behavior.
               ContainerDoc_UnloadAllOleObjectsOfClass(
                     (LPCONTAINERDOC)lpOutlineDoc,
                     &CLSID_NULL,
                     OLECLOSE_SAVEIFDIRTY
               );
               OutlineDoc_ForceRedraw(lpOutlineDoc, TRUE);
               break;
#endif  // INPLACE_CNTR


#if defined( OLE_CNTR )
            case IDC_LINELIST: {

               if (wNotifyCode == LBN_DBLCLK) {

                  /* OLE2NOTE: a container should execute the
                  **    OLEIVERB_PRIMARY verb on an OLE object
                  **    when the user DBLCLK's the object.
                  */
                  int nIndex = LineList_GetFocusLineIndex(lpLL);
                  LPLINE lpLine = LineList_GetLine(lpLL, nIndex);

                  if (lpLine &&
                        Line_GetLineType(lpLine)==CONTAINERLINETYPE) {
                     MSG msg;

                     _fmemset((LPMSG)&msg,0,sizeof(msg));
                     msg.hwnd = hWnd;
                     msg.message = Message;
                     msg.wParam = wParam;
                     msg.lParam = lParam;

                     ContainerLine_DoVerb(
                           (LPCONTAINERLINE)lpLine,
                           OLEIVERB_PRIMARY,
                           (LPMSG)&msg,
                           TRUE,
                           TRUE
                     );
                  }

#if defined( INPLACE_CNTR )
                  { // BEGIN BLOCK
                     LPCONTAINERDOC lpContainerDoc =
                           (LPCONTAINERDOC) lpOutlineDoc;
                     if (lpContainerDoc->m_fAddMyUI) {
                        /* OLE2NOTE: fAddMyUI is TRUE when
                        **    there was previously an in-place
                        **    active object which got
                        **    UIDeactivated as a result of this
                        **    DBLCLK AND the DBLCLK did NOT
                        **    result in in-place activating
                        **    another object.
                        **    (see IOleInPlaceSite::OnUIActivate and
                        **    IOleInPlaceSite::OnUIDeactivate
                        **    methods).
                        */

                        /* OLE2NOTE: You need to generate
                        **    QueryNewPalette only if you own
                        **    the top level frame (ie. you are
                        **    a top-level inplace container).
                        */


                        OleApp_QueryNewPalette((LPOLEAPP)g_lpApp);

#if defined( USE_DOCTOOLS )
                        ContainerDoc_AddDocLevelTools(lpContainerDoc);
#endif

#if defined( USE_FRAMETOOLS )
                        ContainerDoc_AddFrameLevelUI(lpContainerDoc);
#endif
```

```
                        lpContainerDoc->m_fAddMyUI = FALSE;
                     }
                  } // END BLOCK
#endif // INPLACE_CNTR
               }
               break;
            }
#endif  // OLE_CNTR


            default:

#if defined( OLE_CNTR )
               if (wID >= IDM_E_OBJECTVERBMIN) {

                  OleDbgOutNoPrefix2("\r\n");

OLEDBG_BEGIN3("ContainerDoc_ContainerLineDoVerbCommand\r\n")
                  ContainerDoc_ContainerLineDoVerbCommand(
                        (LPCONTAINERDOC)lpOutlineDoc,
                        (LONG)(wID-IDM_E_OBJECTVERBMIN)
                  );
                  OLEDBG_END3
                  break;
               }
#endif  // OLE_CNTR
               return DefWindowProc(hWnd, Message, wParam, lParam);
         }

         break;  /* End of WM_COMMAND */
      }
      default:

         if (Message == g_uMsgHelp) {
            /* Handle OLE2UI dialog's help messages.
            ** We get the hDlg of the dialog that called us in the wParam
            ** and the dialog type in the LOWORD of the lParam,
            ** so we pass this along to our help function.
            */
            OutlineDoc_DialogHelp((HWND)wParam, LOWORD(lParam));
            break;
         }

         /* For any message for which you don't specifically provide a  */
         /* service routine, you should return the message to Windows   */
         /* for default message processing.                             */
         return DefWindowProc(hWnd, Message, wParam, lParam);
   }

   return (LRESULT)0;

} /* End of DocWndProc */
```

```
//***********************************************************************
//*
//* LineListWndProc()  Drag and Drop Listbox Window Proc Sub-Class
//*
//* Sub Class the Ownerdraw list box in order to activate the drag drop.
//***********************************************************************

LRESULT FAR PASCAL LineListWndProc(
   HWND   hWnd,
   UINT   Message,
   WPARAM wParam,
   LPARAM lParam
)
{
   HWND         hwndParent = GetParent ( hWnd );
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC) GetWindowLong( hwndParent,
0 );
   LPLINELIST   lpLL = OutlineDoc_GetLineList(lpOutlineDoc);

#if defined( OLE_VERSION )
   LPOLEAPP     lpOleApp = (LPOLEAPP)lpOutlineApp;
   LPOLEDOC     lpOleDoc = (LPOLEDOC)lpOutlineDoc;
#endif  // OLE_VERSION

#if defined( INPLACE_SVR )
   LPSERVERDOC  lpServerDoc = (LPSERVERDOC)lpOutlineDoc;
   static BOOL  fUIActivateClick = FALSE;
   static BOOL  fInWinPosChged = FALSE;
#endif  // INPLACE_SVR

#if defined( INPLACE_CNTR )
   LPCONTAINERAPP lpContainerApp=(LPCONTAINERAPP)lpOutlineApp;
   LPCONTAINERDOC lpContainerDoc=(LPCONTAINERDOC)lpOutlineDoc;
#endif  // INPLACE_CNTR

   switch (Message) {

      case WM_KILLFOCUS:
         /* OLE2NOTE: when our window looses focus we
         **    should not display any active selection
         */
#if defined( INPLACE_CNTR )
         if (! lpContainerApp->m_fPendingUIDeactivate)
#endif  // INPLACE_CNTR
            LineList_RemoveSel(lpLL);
         break;

      case WM_SETFOCUS:

#if defined( INPLACE_CNTR )
         {
            HWND hWndObj=ContainerDoc_GetUIActiveWindow(lpContainerDoc);

            /* OLE2NOTE: if there is a UIActive in-place object, we must
            **    forward focus to its window as long as there is
            **    not a pending UIDeactivate. if the mouse is
            **    clicked outside of the object and the object is
            **    about to be deactivated then we do NOT want to
            **    forward focus to the object. we do NOT want it to
            **    restore its selection feedback.
            */
            if (lpContainerApp->m_fPendingUIDeactivate)
               break;
            else if (hWndObj) {
               SetFocus(hWndObj);
               break;      // do not restore containers selection state
            }
         }
#endif  // INPLACE_CNTR

         /* OLE2NOTE: when our window gains focus we
         **    should restore the previous selection
         */
         LineList_RestoreSel(lpLL);

         break;

#if defined( INPLACE_SVR )
      case WM_MOUSEACTIVATE:
      {
         if (lpServerDoc->m_fInPlaceActive && !lpServerDoc->m_fUIActive) {
            fUIActivateClick = TRUE;
         };
         break;
      }

#endif  // INPLACE_SVR


#if defined( USE_FRAMETOOLS )
      case WM_CHAR:
      {
         OutlineDoc_SetFormulaBarEditFocus(lpOutlineDoc, TRUE);
         FrameTools_FB_SetEditText(lpOutlineDoc->m_lpFrameTools, NULL);
         FrameTools_FB_SendMessage(
               lpOutlineDoc->m_lpFrameTools,
               IDM_FB_EDIT,
               Message,
               wParam,
               lParam
         );

         return (LRESULT)0;   // don't do default listbox processing
      }
#endif  // USE_FRAMETOOLS

#if defined( INPLACE_CNTR )
      case WM_VSCROLL:
      {
         if (wParam == SB_ENDSCROLL) {
            /* OLE2NOTE: after scrolling finishes, update position of
            **    in-place visible windows.
            **    (ICNTROTL specific) we first let the ListBox
            **    perform it normal processing with the EndScroll
            **    message. also we let the ListBox handle all other
            **    scroll messages.
            */
            LRESULT lResult =  CallWindowProc(
                  (WNDPROC)lpOutlineApp->m_ListBoxWndProc,
                  hWnd,
                  Message,
                  wParam,
                  lParam
            );
            ContainerDoc_UpdateInPlaceObjectRects(lpContainerDoc, 0);
            return lResult;
         }

         break;
      }
#endif  // INPLACR_CNTR

#if defined( USE_HEADING )
      case WM_HSCROLL:
      {
         LPHEADING lphead = OutlineDoc_GetHeading(lpOutlineDoc);

         Heading_CH_SendMessage(lphead, Message, wParam, lParam);

         break;
      }

      /* NOTE: WM_PAINT trapped in order to track vertical scrolling
      **    that has taken place so the row headings can be
      **    coordinated with the LineList. we wanted to trap instead
      **    but it is not generated from scrolling without using
      **    scroll bar (e.g. use keyboard).
      */
      case WM_PAINT:
      {
         Heading_RH_Scroll(OutlineDoc_GetHeading(lpOutlineDoc), hWnd);
         break;
      }

#endif  // USE_HEADING

      case WM_LBUTTONUP:
      {

#if defined( USE_DRAGDROP )
         if (lpOleDoc->m_fPendingDrag) {
            /* ButtonUP came BEFORE distance/time threshholds were
            **    exceeded. clear fPendingDrag state.
            */
            ReleaseCapture();
            KillTimer(hWnd, 1);
            lpOleDoc->m_fPendingDrag = FALSE;
         }
#endif  // USE_DRAGDROP

#if defined( INPLACE_SVR )
         if (fUIActivateClick) {
            fUIActivateClick = FALSE;
            ServerDoc_UIActivate((LPSERVERDOC) lpOleDoc);
         }
#endif  // INPLACE_SVR

#if defined( INPLACE_CNTR )
         {
            /* check if a UIDeactivate is pending.
            **      (see comment in WM_LBUTTONDOWN)
            */
            if ( lpContainerApp->m_fPendingUIDeactivate ) {
               ContainerLine_UIDeactivate(
                     lpContainerDoc->m_lpLastUIActiveLine);

               lpContainerApp->m_fPendingUIDeactivate = FALSE;
            }
         }
#endif  // INPLACE_CNTR

         break;
      }

      case WM_LBUTTONDOWN:
      {
         POINT pt;

         pt.x = (int)(short)LOWORD (lParam );
         pt.y = (int)(short)HIWORD (lParam );

#if defined( INPLACE_SVR ) || defined( INPLACE_CNTR )
         /* OLE2NOTE: see context sensitive help technote (CSHELP.DOC)
         **    m_fCSHelpMode flag is set when SHIFT-F1 context
         **    sensitive help is entered.
         **    if this flag is set then the button click should not
         **    cause any action. if the application implements a
         **    help system, then context sensitive help should be
         **    given for the location clicked by the user.
         */
         if (lpOleDoc->m_fCSHelpMode) {
            return (LRESULT)0;    // eat the button click because we do
                            // not give any help.
         }
#endif  // INPLACE_SVR || INPLACE_CNTR

#if defined( INPLACE_CNTR )
         {
            /* OLE2NOTE: both inside-out and outside-in style
            **    containers must check if the mouse click is
            **    outside of the current UIActive object (if
            **    any). If so, then set the flag indicating that
            **    there is a pending UIDeactivate needed. We do NOT
            **    want to do it now,
            **    because that would result in un-wanted movement of
            **    the data on the screen as frame adornments (eg.
            **    toolbar) and/or object adornments (eg. ruler) would
            **    be removed from the screen. we want to defer the
            **    UIDeactivate till the mouse up event. The listbox's
            **    default processing captures the mouse on button down
            **    so that it is sure to get the button up message.
            **
            **    SPECIAL NOTE: there is potential interaction here
            **    with Drag/Drop. if this button down event actually
            **    starts a Drag/Drop operation, then OLE does the mouse
            **    capture. in this situation we will NOT get our button
            **    up event. we must instead perform the UIDeactivate
            **    when the drop operation finishes
            */
            lpContainerApp->m_fPendingUIDeactivate =
                  ContainerDoc_IsUIDeactivateNeeded(lpContainerDoc, pt);
         }
#endif  // INPLACE_CNTR

#if defined( USE_DRAGDROP )

         /* OLE2NOTE: check if this is a button down on the region
         **    that is a handle to start a drag operation. for us,
         **    this this the top/bottom border of the selection.
         **    do NOT want to start a drag immediately; we want to
         **    wait until the mouse moves a certain threshold. if
         **    LButtonUp comes before mouse move to start drag, then
         **    the fPendingDrag state is cleared. we must capture
         **    the mouse to ensure the modal state is handled
         **    properly.
         */
         if ( OleDoc_QueryDrag(lpOleDoc, pt.y) ) {
            lpOleDoc->m_fPendingDrag = TRUE;
            lpOleDoc->m_ptButDown = pt;
            SetTimer(hWnd, 1, lpOleApp->m_nDragDelay, NULL);
            SetCapture(hWnd);

            /* We do NOT want to do the listbox's default
            **    processing which would be to capture the mouse
            **    and enter a modal multiple selection state until
            **    a mouse up occurs. we have just finished a modal
            **    drag/drop operation where OLE has captured the
            **    mouse. thus by now the mouse up has already occured.
            */

            return (LRESULT)0;   // don't do default listbox processing
         }
#endif  // USE_DRAGDROP
```

```
         break;
      }


      case WM_MOUSEMOVE: {

#if defined( USE_DRAGDROP )

         int  x = (int)(short)LOWORD (lParam );
         int  y = (int)(short)HIWORD (lParam );
         POINT pt = lpOleDoc->m_ptButDown;
         int nDragMinDist = lpOleApp->m_nDragMinDist;

         if (lpOleDoc->m_fPendingDrag) {

            if (! ( ((pt.x - nDragMinDist) <= x)
                  && (x <= (pt.x + nDragMinDist))
                  && ((pt.y - nDragMinDist) <= y)
                  && (y <= (pt.y + nDragMinDist)) ) ) {

               DWORD dwEffect;

               // mouse moved beyond threshhold to start drag
               ReleaseCapture();
               KillTimer(hWnd, 1);
               lpOleDoc->m_fPendingDrag = FALSE;

               // perform the modal drag/drop operation.
               dwEffect = OleDoc_DoDragDrop( lpOleDoc );

#if defined( INPLACE_CNTR )
               {
                  /* if necessary UIDeactive the in-place object.
                  **    this applies to outside-in style
                  **    container only.
                  **    (see comment above)
                  */
                  if (lpContainerApp->m_fPendingUIDeactivate) {
                     lpContainerApp->m_fPendingUIDeactivate = FALSE;

                     // do not UIDeactivate if drag/drop was canceled
                     if (dwEffect != DROPEFFECT_NONE)
                        ContainerLine_UIDeactivate(
                              lpContainerDoc->m_lpLastUIActiveLine
                        );
                  }
               }
#endif  // INPLACE_CNTR

               return (LRESULT)0; // don't do default listbox process
            }
            else {
               /* cursor did not move from initial mouse down
               **    (pending drag) point.
               */
               return (LRESULT)0; // don't do default listbox process
            }
         }

#endif  // USE_DRAGDROP

#if defined( INPLACE_CNTR )
         { // BEGIN BLOCK
            if (lpContainerDoc->m_fAddMyUI) {
               /* OLE2NOTE: fAddMyUI is TRUE when
               **    there was previously an in-place
               **    active object which got
               **    UIDeactivated as a result of a
               **    DBLCLK AND the DBLCLK did NOT
               **    result in in-place activating
               **    another object.
               **    (see IOleInPlaceSite::OnUIActivate and
               **    IOleInPlaceSite::OnUIDeactivate
               **    methods).
               */
#if defined( USE_DOCTOOLS )
               ContainerDoc_AddDocLevelTools(lpContainerDoc);
#endif

#if defined( USE_FRAMETOOLS )
               ContainerDoc_AddFrameLevelUI(lpContainerDoc);
#endif
               lpContainerDoc->m_fAddMyUI = FALSE;
            }
         } // END BLOCK
#endif // INPLACE_CNTR

         break;
      }


#if defined( USE_DRAGDROP )
      case WM_TIMER:
      {
         DWORD dwEffect;

         // drag time delay threshhold exceeded -- start drag
         ReleaseCapture();
         KillTimer(hWnd, 1);
         lpOleDoc->m_fPendingDrag = FALSE;

         // perform the modal drag/drop operation.
         dwEffect = OleDoc_DoDragDrop( lpOleDoc );

#if defined( INPLACE_CNTR )
         /* if necessary UIDeactive the in-place object.
         **    this applies to outside-in style
         **    container only.
         **    (see comment above)
         */
         if (lpContainerApp->m_fPendingUIDeactivate) {
            lpContainerApp->m_fPendingUIDeactivate = FALSE;

            // do not UIDeactivate if drag/drop was canceled
            if (dwEffect != DROPEFFECT_NONE)
               ContainerLine_UIDeactivate(
                     lpContainerDoc->m_lpLastUIActiveLine);
         }
#endif  // INPLACE_CNTR
         break;
      }
#endif  // USE_DRAGDROP

      case WM_SETCURSOR:
      {
         RECT rc;
         POINT ptCursor;
#if defined( INPLACE_SVR ) || defined( INPLACE_CNTR )
         /* OLE2NOTE: see context sensitive help technote (CSHELP.DOC)
         **    m_fCSHelpMode flag is set when SHIFT-F1 context
         **    sensitive help is entered.
         **    if this flag is set then the context sensitive help
         **    cursor should be shown.
         */
         if (lpOleDoc->m_fCSHelpMode) {
            SetCursor(UICursorLoad(IDC_CONTEXTHELP));
            return (LRESULT)TRUE;
         }
#endif  // INPLACE_SVR || INPLACE_CNTR

         GetCursorPos((POINT FAR*)&ptCursor);
         ScreenToClient(hWnd, (POINT FAR*)&ptCursor);
         GetClientRect(hWnd, (LPRECT)&rc);

         // use arrow cursor if in scroll bar
         if (! PtInRect((LPRECT)&rc, ptCursor) )
            SetCursor(LoadCursor(NULL, IDC_ARROW) );

#if defined( USE_DRAGDROP )
         // use arrow cursor if on drag handle (top/bottom of selection)
         else if ( OleDoc_QueryDrag ( lpOleDoc, ptCursor.y) )
            SetCursor(LoadCursor(NULL, IDC_ARROW) );
#endif  // USE_DRAGDROP

         else
            SetCursor(lpOutlineApp->m_hcursorSelCur);

         return (LRESULT)TRUE;
      }

#if defined( INPLACE_SVR )

      /* The handling of WM_WINDOWPOSCHANGED message is ISVROTL
      **    application specific. The nature of the owner-draw list
      **    box used by the ISVROTL application causes a recursive
      **    call to this message in some situations when in-place
      **    active. in order not to crash this recursive call must be
      **    guarded.
      */
      case WM_WINDOWPOSCHANGED:
      {
         WINDOWPOS FAR* lpWinPos = (WINDOWPOS FAR*) lParam;
         LRESULT lResult;

         // guard against recursive call
         if (fInWinPosChged)
            return (LRESULT)0;

         fInWinPosChged = TRUE;
         lResult = CallWindowProc(
               (WNDPROC)lpOutlineApp->m_ListBoxWndProc,
               hWnd,
               Message,
               wParam,
               lParam
         );
         fInWinPosChged = FALSE;

         return lResult;
      }
#endif  // INPLACE_SVR

   }

   return CallWindowProc(
         (WNDPROC)lpOutlineApp->m_ListBoxWndProc,
         hWnd,
         Message,
         wParam,
         lParam
   );

}

// Utility function to count the number of accelerator items in an
//  accelerator table.  A number of OLE APIs need this count, so
//  this can be quite handy.
// (code shamelessly stolen from the Microsoft Foundation Classes)

int GetAccelItemCount(HACCEL hAccel)
{
#if defined( WIN32 )
    return CopyAcceleratorTable(hAccel, NULL, 0);
#else
   #pragma pack(1)
   typedef struct tagACCELERATOR
   {
      BYTE    fFlags;
      WORD    wEvent;
      WORD    wID;
   } ACCELERATOR;
   #pragma pack()

   // attempt to lock down the accelerator resource
   ACCELERATOR FAR* pAccel;
   int cAccelItems = 1;
   if (hAccel == NULL ||
      (pAccel = (ACCELERATOR FAR*)LockResource((HGLOBAL)hAccel)) == NULL)
   {
      // NULL accerator table or LockResource failed on the HACCEL,
      //  no accelerators
      return 0;
   }
   // otherwise, count them -- last entry in accel table has 0x80 bit set
   while ((pAccel->fFlags & 0x80) == 0)
   {
      ++cAccelItems;
      ++pAccel;
   }
   UnlockResource((HGLOBAL)hAccel);
   return cAccelItems;
#endif
}
```

MEMMGR.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    memmgr.c
**
**    This file contains memory management functions.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/


#include "outline.h"


/* New
 * ---
 *
 *      Allocate memory for a new structure
 */
LPVOID New(DWORD lSize)
{
     LPVOID lp = OleStdMalloc((ULONG)lSize);

     return lp;
}


/* Delete
 * ------
 *
 *      Free memory allocated for a structure
 */
void Delete(LPVOID lp)
{
     OleStdFree(lp);
}
```

MESSAGE.H   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Server Sample Code
**
**    message.h
**
**    This file is a user customizable list of status messages associated
**    with menu items.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

// Status bar messages and associated data.

// Message type for status bar messages.
typedef struct {
     UINT wIDItem;
     char *string;
} STATMESG;

/*
 * CUSTOMIZATION NOTE:  Be sure to change NUM_POPUP if you
 *                      change the number of popup messages.
 */

// REVIEW: these messages should be loaded from a string resource

// List of all menu item messages.
STATMESG MesgList[] =
{
     { IDM_F_NEW,        "Creates a new outline" },
     { IDM_F_OPEN,       "Opens an existing outline file"    },
     { IDM_F_SAVE,       "Saves the outline" },
     { IDM_F_SAVEAS,     "Saves the outline with a new name" },
     { IDM_F_PRINT,      "Prints the outline" },
     { IDM_F_PRINTERSETUP, "Changes the printer and the printing options" },
     { IDM_F_EXIT,       "Quits the application, prompting to save
changes" },

     { IDM_E_UNDO,       "Undo not yet implemented" },
     { IDM_E_CUT,        "Cuts the selection and puts it on the
Clipboard" },
     { IDM_E_COPY,       "Copies the selection and puts it on the Clipboard"
},
     { IDM_E_PASTE,      "Inserts the Clipboard contents after current line"
},
     { IDM_E_PASTESPECIAL,"Allows pasting Clipboard data using a special
format" },
     { IDM_E_CLEAR,      "Clears the selection" },
     { IDM_E_SELECTALL,  "Selects the entire outline" },
#if defined( OLE_CNTR )
```

```
     { IDM_E_INSERTOBJECT, "Inserts new object after current line" },
     { IDM_E_EDITLINKS, "Edit and view links contained in the document" },
   { IDM_E_CONVERTVERB, "Converts or activates an object as another type" },
     { IDM_E_OBJECTVERBMIN, "Opens, edits or interacts with an object" },
     { IDM_E_OBJECTVERBMIN+1, "Opens, edits or interacts with an object1" },
     { IDM_E_OBJECTVERBMIN+2, "Opens, edits or interacts with an object2" },
     { IDM_E_OBJECTVERBMIN+3, "Opens, edits or interacts with an object3" },
     { IDM_E_OBJECTVERBMIN+4, "Opens, edits or interacts with an object4" },
     { IDM_E_OBJECTVERBMIN+5, "Opens, edits or interacts with an object5" },
#endif

     { IDM_L_ADDLINE,    "Adds a new line after current line" },
     { IDM_L_EDITLINE,   "Edits the current line" },
     { IDM_L_INDENTLINE, "Indents the selection" },
     { IDM_L_UNINDENTLINE, "Unindents the selection" },
     { IDM_L_SETLINEHEIGHT, "Modify the height of a line" },

     { IDM_N_DEFINENAME, "Assigns a name to the selection" },
     { IDM_N_GOTONAME,   "Jumps to a specified place in the outline" },

     { IDM_H_ABOUT,      "Displays program info, version no., and copyright"
},

     { IDM_D_DEBUGLEVEL,     "Set debug level (0-4)" },
     { IDM_D_INSTALLMSGFILTER,"Install/deinstall the IMessageFilter" },
     { IDM_D_REJECTINCOMING, "Return RETRYLATER to incoming method calls" },

     { IDM_O_BB_TOP, "Position ButtonBar at top of window" },
     { IDM_O_BB_BOTTOM, "Position ButtonBar at botttom of window" },
     { IDM_O_BB_POPUP, "Put ButtonBar in popup pallet" },
     { IDM_O_BB_HIDE, "Hide ButtonBar" },

     { IDM_O_FB_TOP, "Position FormulaBar at top of window" },
     { IDM_O_FB_BOTTOM, "Position FormulaBar at botttom of window" },
     { IDM_O_FB_POPUP, "Put FormulaBar in popup pallet" },

     { IDM_O_HEAD_SHOW, "Show row/column headings" },
     { IDM_O_HEAD_HIDE, "Hide row/column headings" },
     { IDM_O_SHOWOBJECT, "Show border around objects/links" },

     { IDM_V_ZOOM_400, "Set document zoom level" },
     { IDM_V_ZOOM_300, "Set document zoom level" },
     { IDM_V_ZOOM_200, "Set document zoom level" },
     { IDM_V_ZOOM_100, "Set document zoom level" },
     { IDM_V_ZOOM_75, "Set document zoom level" },
     { IDM_V_ZOOM_50, "Set document zoom level" },
     { IDM_V_ZOOM_25, "Set document zoom level" },

     { IDM_V_SETMARGIN_0, "Remove left/right document margins" },
     { IDM_V_SETMARGIN_1, "Set left/right document margins" },
     { IDM_V_SETMARGIN_2, "Set left/right document margins" },
     { IDM_V_SETMARGIN_3, "Set left/right document margins" },
     { IDM_V_SETMARGIN_4, "Set left/right document margins" },

     { IDM_V_ADDTOP_1, "Add top line" },
     { IDM_V_ADDTOP_2, "Add top line" },
     { IDM_V_ADDTOP_3, "Add top line" },
     { IDM_V_ADDTOP_4, "Add top line" }
};

#define NUM_STATS   sizeof(MesgList)/sizeof(MesgList[0])
#define NUM_POPUP   10  // Maximum number of popup messages.
#define MAX_MESSAGE 100 // Maximum characters in a popup message.
```

OLEAPP.C (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    oleapp.c
**
**    This file contains functions and methods that are common to
**    server and the client version of the app. This includes the class
**    factory methods and all OleApp functions.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"
#include <ole2ver.h>

OLEDBGDATA

extern LPOUTLINEAPP             g_lpApp;

extern IUnknownVtbl             g_OleApp_UnknownVtbl;

extern IUnknownVtbl             g_OleDoc_UnknownVtbl;
extern IPersistFileVtbl         g_OleDoc_PersistFileVtbl;
extern IOleItemContainerVtbl    g_OleDoc_OleItemContainerVtbl;
extern IExternalConnectionVtbl  g_OleDoc_ExternalConnectionVtbl;
extern IDataObjectVtbl          g_OleDoc_DataObjectVtbl;

#if defined( USE_DRAGDROP )
extern IDropTargetVtbl          g_OleDoc_DropTargetVtbl;
extern IDropSourceVtbl          g_OleDoc_DropSourceVtbl;
#endif  // USE_DRAGDROP

#if defined( OLE_SERVER )
extern IOleObjectVtbl       g_SvrDoc_OleObjectVtbl;
extern IPersistStorageVtbl  g_SvrDoc_PersistStorageVtbl;

#if defined( SVR_TREATAS )
extern IStdMarshalInfoVtbl  g_SvrDoc_StdMarshalInfoVtbl;
#endif  // SVR_TREATAS

extern IUnknownVtbl         g_PseudoObj_UnknownVtbl;
extern IOleObjectVtbl       g_PseudoObj_OleObjectVtbl;
extern IDataObjectVtbl      g_PseudoObj_DataObjectVtbl;

#if defined( INPLACE_SVR )
extern IOleInPlaceObjectVtbl        g_SvrDoc_OleInPlaceObjectVtbl;
extern IOleInPlaceActiveObjectVtbl  g_SvrDoc_OleInPlaceActiveObjectVtbl;
#endif  // INPLACE_SVR

#endif  // OLE_SERVER
```

```
#if defined( OLE_CNTR )

extern IOleUILinkContainerVtbl  g_CntrDoc_OleUILinkContainerVtbl;
extern IUnknownVtbl             g_CntrLine_UnknownVtbl;
extern IOleClientSiteVtbl       g_CntrLine_OleClientSiteVtbl;
extern IAdviseSinkVtbl          g_CntrLine_AdviseSinkVtbl;

#if defined( INPLACE_CNTR )
extern IOleInPlaceSiteVtbl      g_CntrLine_OleInPlaceSiteVtbl;
extern IOleInPlaceFrameVtbl     g_CntrApp_OleInPlaceFrameVtbl;
extern BOOL g_fInsideOutContainer;
#endif  // INPLACE_CNTR

#endif  // OLE_CNTR

// REVIEW: these are NOT useful end-user messages
static OLECHAR ErrMsgCreateCF[] = OLESTR("Can't create Class Factory!");
static OLECHAR ErrMsgRegCF[] = OLESTR("Can't register Class Factory!");
static OLECHAR ErrMsgRegMF[] = OLESTR("Can't register Message Filter!");

extern UINT g_uMsgHelp;


/* OleApp_InitInstance
 * -------------------
 *
 * Initialize the app instance by creating the main frame window and
 * performing app instance specific initializations
 *  (eg. initializing interface Vtbls).
 *
 * RETURNS: TRUE if the memory could be allocated, and the server app
 *               was properly initialized.
 *          FALSE otherwise
 *
 */
BOOL OleApp_InitInstance(LPOLEAPP lpOleApp, HINSTANCE hInst, int nCmdShow)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)lpOleApp;
   HRESULT hrErr;
   DWORD   dwBuildVersion = OleBuildVersion();
   LPMALLOC lpMalloc = NULL;

   OLEDBG_BEGIN3("OleApp_InitInstance\r\n")

   lpOleApp->m_fOleInitialized = FALSE;

   /* OLE2NOTE: check if the build version of the OLE2 DLL's match
   **    what our application is expecting.
   */
   if (HIWORD(dwBuildVersion) != rmm || LOWORD(dwBuildVersion) < rup) {
      OleDbgAssertSz(0, "ERROR: OLE 2.0 DLL's are NOT compatible!");

#if !defined( _DEBUG )
      return FALSE;   // Wrong version of DLL's
```

```
#endif
   }

   /* OLE2NOTE: the OLE libraries must be properly initialized before
   **    making any calls. OleInitialize automatically calls
   **    CoInitialize. we will use the default task memory allocator
   **    therefore we pass NULL to OleInitialize.
   */
   OLEDBG_BEGIN2("OleInitialize called\r\n")
   hrErr = OleInitialize(NULL);
   OLEDBG_END2

   if (hrErr != NOERROR) {
      OutlineApp_ErrorMessage(lpOutlineApp,OLESTR("OLE initialization
failed!"));
      goto error;
   }

   /*****************************************************************
   ** OLE2NOTE: we must remember the fact that OleInitialize has
   **    been call successfully. the very last thing an app must
   **    be do is properly shut down OLE by calling
   **    OleUninitialize. This call MUST be guarded! it is only
   **    allowable to call OleUninitialize if OleInitialize has
   **    been called SUCCESSFULLY.
   *****************************************************************/

   lpOleApp->m_fOleInitialized = TRUE;

   // Initialize the OLE 2.0 interface method tables.
   if (! OleApp_InitVtbls(lpOleApp))
      goto error;

   // Register OLE 2.0 clipboard formats.
   lpOleApp->m_cfEmbedSource = RegisterClipboardFormat(CF_EMBEDSOURCE);
   lpOleApp->m_cfEmbeddedObject = RegisterClipboardFormat(
         CF_EMBEDDEDOBJECT
   );
   lpOleApp->m_cfLinkSource = RegisterClipboardFormat(CF_LINKSOURCE);
   lpOleApp->m_cfFileName = RegisterClipboardFormat(CF_FILENAME);
   lpOleApp->m_cfObjectDescriptor =
         RegisterClipboardFormat(CF_OBJECTDESCRIPTOR);
   lpOleApp->m_cfLinkSrcDescriptor =
         RegisterClipboardFormat(CF_LINKSRCDESCRIPTOR);

   lpOleApp->m_cRef                    = 0;
   lpOleApp->m_cDoc                    = 0;
   lpOleApp->m_fUserCtrl               = FALSE;
   lpOleApp->m_dwRegClassFac           = 0;
   lpOleApp->m_lpClassFactory          = NULL;
   lpOleApp->m_cModalDlgActive         = 0;

   INIT_INTERFACEIMPL(
         &lpOleApp->m_Unknown,
         &g_OleApp_UnknownVtbl,
         lpOleApp
   );

#if defined( USE_DRAGDROP )

   // delay before dragging should start, in milliseconds
   lpOleApp->m_nDragDelay = GetProfileInt(
         "windows",
         "DragDelay",
         DD_DEFDRAGDELAY
   );

   // minimum distance (radius) before drag should start, in pixels
   lpOleApp->m_nDragMinDist = GetProfileInt(
         "windows",
         "DragMinDist",
         DD_DEFDRAGMINDIST
   );

   // delay before scrolling, in milliseconds
   lpOleApp->m_nScrollDelay = GetProfileInt(
         "windows",
         "DragScrollDelay",
         DD_DEFSCROLLDELAY
   );

   // inset-width of the hot zone, in pixels
   lpOleApp->m_nScrollInset = GetProfileInt(
         "windows",
         "DragScrollInset",
         DD_DEFSCROLLINSET
   );

   // scroll interval, in milliseconds
   lpOleApp->m_nScrollInterval = GetProfileInt(
         "windows",
         "DragScrollInterval",
         DD_DEFSCROLLINTERVAL
   );

#if defined( IF_SPECIAL_DD_CURSORS_NEEDED )
   // This would be used if the app wanted to have custom drag/drop cursors
   lpOleApp->m_hcursorDragNone  = LoadCursor ( hInst, "DragNoneCur" );
   lpOleApp->m_hcursorDragCopy  = LoadCursor ( hInst, "DragCopyCur" );
   lpOleApp->m_hcursorDragMove  = LoadCursor ( hInst, "DragMoveCur" );
   lpOleApp->m_hcursorDragLink  = LoadCursor ( hInst, "DragLinkCur" );
#endif  // IF_SPECIAL_DD_CURSORS_NEEDED

#endif  // USE_DRAGDROP

   lpOleApp->m_lpMsgFilter = NULL;

#if defined( USE_MSGFILTER )
   /* OLE2NOTE: Register our message filter upon app startup. the
   **    message filter is used to handle concurrency.
```

```
   **    we will use a standard implementation of IMessageFilter that
   **    is included as part of the OLE2UI library.
   */
   lpOleApp->m_lpMsgFilter = NULL;
   if (! OleApp_RegisterMessageFilter(lpOleApp))
      goto error;

   /* OLE2NOTE: because our app is initially INVISIBLE, we must
   **    DISABLE the busy dialog. we should NOT put up any dialogs if
   **    our app is invisible. when our app window is made visible,
   **    then the busy dialog will be enabled.
   */
   OleStdMsgFilter_EnableBusyDialog(lpOleApp->m_lpMsgFilter, FALSE);
#endif  // USE_MSGFILTER

#if defined( OLE_SERVER )
   /* OLE2NOTE: perform initialization specific for an OLE server */
   if (! ServerApp_InitInstance((LPSERVERAPP)lpOutlineApp, hInst, nCmdShow))
      goto error;
#endif
#if defined( OLE_CNTR )
   /* OLE2NOTE: perform initialization specific for an OLE container */

   // Register help message
   g_uMsgHelp = RegisterWindowMessage(SZOLEUI_MSG_HELP);

   if (! ContainerApp_InitInstance((LPCONTAINERAPP)lpOutlineApp, hInst,
nCmdShow))
      goto error;
#endif

#if defined( OLE_CNTR )
   lpOleApp->m_hStdPal = OleStdCreateStandardPalette();
#endif

   OLEDBG_END3
   return TRUE;

error:
   OLEDBG_END3
   return FALSE;
}


/*
 * OleApp_TerminateApplication
 * ---------------------------
 *  Perform proper OLE application cleanup before shutting down
 */
void OleApp_TerminateApplication(LPOLEAPP lpOleApp)
{
   OLEDBG_BEGIN3("OleApp_TerminateApplication\r\n")

   /* OLE2NOTE: perform a clean shut down for OLE. at this point our
   **    App refcnt should be 0, or else we should never have reached
   **    this point!
   */
   OleDbgAssertSz(lpOleApp->m_cRef == 0, "App NOT shut down properly");

   if(lpOleApp->m_fOleInitialized) {
      OLEDBG_BEGIN2("OleUninitialize called\r\n")
      OleUninitialize();
      OLEDBG_END2
   }
   OLEDBG_END3
}


/* OleApp_ParseCmdLine
 * -------------------
 *
 * Parse the command line for any execution flags/arguments.
 *      OLE2NOTE: check if "-Embedding" switch is given.
 */
BOOL OleApp_ParseCmdLine(LPOLEAPP lpOleApp, LPSTR lpszCmdLine, int nCmdShow)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)lpOleApp;
   char szFileName[256];   /* buffer for filename in command line */
   BOOL fStatus = TRUE;
   BOOL fEmbedding = FALSE;
   OLECHAR szUniStr[256];

   OLEDBG_BEGIN3("OleApp_ParseCmdLine\r\n")

   szFileName[0] = '\0';
   ParseCmdLine(lpszCmdLine, &fEmbedding, (LPSTR)szFileName);

#if defined( MDI_VERSION )
   /* OLE2NOTE: an MDI app would ALWAYS register its ClassFactory. it
   **    can handle multiple objects at the same time, while an SDI
   **    application can only handle a single embedded or file-based
   **    object at a time.
   */
   fStatus = OleApp_RegisterClassFactory(lpOleApp);
#endif

   if(fEmbedding) {

      if (szFileName[0] == '\0') {

         /**************************************************************
         ** App was launched with /Embedding.
         **    We must register our ClassFactory with OLE, remain hidden
         **    (the app window is initially created not visible), and
         **    wait for OLE to call IClassFactory::CreateInstance
         **    method. We do not automatically create a document as we
         **    do when the app is launched by the user from the
         **    FileManager. We must NOT make our app window visible
         **    until told to do so by our container.
         **
         ** OLE2NOTE: Because we are an SDI app, we only register our
         **    ClassFactory if we are launched with the /Embedding
         **    flag WITHOUT a filename. an MDI app would ALWAYS
         **    register its ClassFactory. it can handle multiple
         **    objects at the same time, while an SDI application
         **    can only handle a single embedded or file-based
         **    object at a time.
         *****************************************************************/

#if defined( SDI_VERSION )
         fStatus = OleApp_RegisterClassFactory(lpOleApp);
#endif
      } else {

         /*****************************************************************
         ** App was launched with /Embedding <Filename>.
         **    We must create a document and load the file and
         **    register it in the RunningObjectTable BEFORE we
         **    enter our GetMessage loop (ie. before we yield).
         **    One way to perform these tasks is to call the same
         **    interface methods that OLE 2.0 calls for linking to a
         **    file:
         **          IClassFactory::CreateInstance
         **          IPersistFile::Load
         **
         **    We must NOT make our app window visible until told to
         **    do so by our container. An application will be
         **    launched in this manner by an OLE 1.0 application
         **    link situation (eg. double clicking a linked object
         **    or OleCreateLinkFromFile called).
         **
         ** OLE2NOTE: Because we are an SDI app, we should NOT
         **    register our ClassFactory when we are launched with the
         **    /Embedding <Filename> flag. our SDI instance can only
         **    handle a single embedded or file-based object.
         **    an MDI app WOULD register its ClassFactory at all
         **    times because it can handle multiple objects.
         *****************************************************************/

         // allocate a new document object
         lpOutlineApp->m_lpDoc = OutlineApp_CreateDoc(lpOutlineApp, FALSE);
         if (! lpOutlineApp->m_lpDoc) {
            OLEDBG_END3
            return FALSE;
         }

         /* OLE2NOTE: initially the Doc object is created with a 0 ref
         **    count. in order to have a stable Doc object during the
         **    process of initializing the new Doc instance,
         **    we intially AddRef the Doc ref cnt and later
         **    Release it. This initial AddRef is artificial; it is simply
         **    done to guarantee that a harmless QueryInterface followed by
         **    a Release does not inadvertantly force our object to destroy
         **    itself prematurely.
         */
```

```
            OleDoc_AddRef((LPOLEDOC)lpOutlineApp->m_lpDoc);

            /* OLE2NOTE: OutlineDoc_LoadFromFile will register our document
            **    in the RunningObjectTable. this registration will
            **    AddRef our document. therefore our document will not
            **    be destroyed when we release the artificial AddRef
            */
            A2W (szFileName, szUniStr, 256);
            fStatus = OutlineDoc_LoadFromFile(
                  lpOutlineApp->m_lpDoc, (LPOLESTR)szUniStr);

            OleDoc_Release((LPOLEDOC)lpOutlineApp->m_lpDoc); // rel AddRef

            OLEDBG_END3
            return fStatus;
         }
      } else {

         /*****************************************************************
         ** App was launched by the user (without /Embedding) and
         **    therefore is marked to be under user control.
         **    In this case, because we are an SDI app, we do NOT
         **    register our ClassFactory with OLE. This app instance can
         **    only manage one document at a time (either a user
         **    document or an embedded object document). An MDI app
         **    would register its ClassFactory here.
         **
         **    We must create a document for the user (either
         **    initialized from a file given on the command line or
         **    initialized as an untitled document. We must also make
         **    our app window visible to the user.
         *****************************************************************/

         // allocate a new document object
         lpOutlineApp->m_lpDoc = OutlineApp_CreateDoc(lpOutlineApp, FALSE);
         if (! lpOutlineApp->m_lpDoc) goto error;

         /* OLE2NOTE: initially the Doc object is created with a 0 ref
         **    count. in order to have a stable Doc object during the
         **    process of initializing the new Doc instance,
         **    we intially AddRef the Doc ref cnt and later
         **    Release it. This initial AddRef is artificial; it is simply
         **    done to guarantee that a harmless QueryInterface followed by
         **    a Release does not inadvertantly force our object to destroy
         **    itself prematurely.
         */
         OleDoc_AddRef((LPOLEDOC)lpOutlineApp->m_lpDoc);

         if(*szFileName) {
            // initialize the document from the specified file
            A2W (szFileName, szUniStr, 256);
            if (! OutlineDoc_LoadFromFile(lpOutlineApp->m_lpDoc, szUniStr))
               goto error;
         } else {
            // set the doc to an (Untitled) doc.
            if (! OutlineDoc_InitNewFile(lpOutlineApp->m_lpDoc))
               goto error;
         }

         // position and size the new doc window
         OutlineApp_ResizeWindows(lpOutlineApp);
         OutlineDoc_ShowWindow(lpOutlineApp->m_lpDoc); // calls OleDoc_Lock
         OleDoc_Release((LPOLEDOC)lpOutlineApp->m_lpDoc);// rel AddRef above

         // show main app window
         ShowWindow(lpOutlineApp->m_hWndApp, nCmdShow);
         UpdateWindow(lpOutlineApp->m_hWndApp);

   #if defined( OLE_CNTR )
         ContainerDoc_UpdateLinks((LPCONTAINERDOC)lpOutlineApp->m_lpDoc);
   #endif

      }

      OLEDBG_END3
      return fStatus;

   error:
      // REVIEW: should load string from string resource
      OutlineApp_ErrorMessage(
            lpOutlineApp,
            OLESTR("Could not create document--Out of Memory")
      );
      if (lpOutlineApp->m_lpDoc)      // rel artificial AddRef above
         OleDoc_Release((LPOLEDOC)lpOutlineApp->m_lpDoc);

      OLEDBG_END3
      return FALSE;
   }


   /* OleApp_CloseAllDocsAndExitCommand
    * ---------------------------------
    *
    *  Close all active documents and exit the app.
    *  Because this is an SDI, there is only one document
    *  If the doc was modified, prompt the user if he wants to save it.
    *
    *  Returns:
    *      TRUE if the app is successfully closed
    *      FALSE if failed or aborted
    *
    * OLE2NOTE: in the OLE version, we can NOT directly
    *     destroy the App object. we can only take all
    *     necessary actions to ensure that our object receives
    *     all of its Releases from clients holding onto
    *     pointers (eg. closing all docs and flushing the
    *     clipboard) and then we must hide our window and wait
    *     actually for our refcnt to reach 0. when it reaches 0,
    *     our destructor (OutlineApp_Destroy) will be called.
    *      each document addref's the app object in order to
    *      guarentee that the app does not shut down while the doc
    *      is still open. closing all docs, will release these
    *      refcnt's. if there are now more open documents AND the
    *      app is not under the control of the user (ie. launched by
    *      OLE) then the app will now shut down. the OleApp_Release
    *      function executes this shut down procedure. after closing
    *      all docs, then releasing the user refcnt will force the
    *      app to shut down.
    */
   BOOL OleApp_CloseAllDocsAndExitCommand(
         LPOLEAPP            lpOleApp,
         BOOL                fForceEndSession
   )
   {
      LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)lpOleApp;
      DWORD dwSaveOption = (fForceEndSession ?
                              OLECLOSE_NOSAVE : OLECLOSE_PROMPTSAVE);

      /* OLE2NOTE: in order to have a stable App object during the
      **    process of closing, we intially AddRef the App ref cnt and
      **    later Release it. This initial AddRef is artificial; it is
      **    simply done to guarantee that our App object does not
      **    destroy itself until the end of this routine.
      */
      OleApp_AddRef(lpOleApp);

      /* Because this is an SDI app, there is only one document.
      ** Close the doc. if it is successfully closed and the app will
      ** not automatically exit, then also exit the app.
      ** if this were an MDI app, we would loop through and close all
      ** open MDI child documents.
      */

   #if defined( OLE_SERVER )
      if (!fForceEndSession &&
            lpOutlineApp->m_lpDoc->m_docInitType == DOCTYPE_EMBEDDED)
         dwSaveOption = OLECLOSE_SAVEIFDIRTY;
   #endif

      if (! OutlineDoc_Close(lpOutlineApp->m_lpDoc, dwSaveOption)) {
         OleApp_Release(lpOleApp);
         return FALSE;     // User Aborted shutdown
      }
   #if defined( _DEBUG )
      OleDbgAssertSz(
            lpOutlineApp->m_lpDoc==NULL,
            "Closed doc NOT properly destroyed"
      );
   #endif

   #if defined( OLE_CNTR )
      /* if we currently have data on the clipboard then we must tell
      **    the clipboard to release our clipboard data object
      **    (document)
      */
      if (lpOutlineApp->m_lpClipboardDoc)
         OleApp_FlushClipboard(lpOleApp);
   #endif

      OleApp_HideWindow(lpOleApp);

      /* OLE2NOTE: this call forces all external connections to our
      **    object to close down and therefore guarantees that we receive
      **    all releases associated with those external connections.
      */
      OLEDBG_BEGIN2("CoDisconnectObject(lpApp) called\r\n")
      CoDisconnectObject((LPUNKNOWN)&lpOleApp->m_Unknown, 0);
      OLEDBG_END2

      OleApp_Release(lpOleApp);       // release artificial AddRef above

      return TRUE;
   }


   /* OleApp_ShowWindow
    * -----------------
    *
    *      Show the window of the app to the user.
    *      make sure app window is visible and bring the app to the top.
    *      IF fGiveUserCtrl == TRUE
    *          THEN give the user the control over the life-time of the app.
    */
   void OleApp_ShowWindow(LPOLEAPP lpOleApp, BOOL fGiveUserCtrl)
   {
      LPOUTLINEAPP    lpOutlineApp = (LPOUTLINEAPP)lpOleApp;

      OLEDBG_BEGIN3("OleApp_ShowWindow\r\n")

      /* OLE2NOTE: while the application is visible and under user
      **    control, we do NOT want it to be prematurely destroyed when
      **    the user closes a document. thus we must inform OLE to hold
      **    an external lock on our application on behalf of the user.
      **    this arranges that OLE holds at least 1 reference to our
      **    application that will NOT be released until we release this
      **    external lock. later, when the application window is hidden, we
      **    will release this external lock.
      */
      if (fGiveUserCtrl && ! lpOleApp->m_fUserCtrl) {
         lpOleApp->m_fUserCtrl = TRUE;
         OleApp_Lock(lpOleApp, TRUE /* fLock */, 0 /* not applicable */);
      }

      // we must show our App window and force it to have input focus
      ShowWindow(lpOutlineApp->m_hWndApp, SW_SHOWNORMAL);
      SetForegroundWindow(lpOutlineApp->m_hWndApp);

      /* OLE2NOTE: because our app is now visible, we can enable the busy
      **    dialog. we should NOT put up any dialogs if our app is
      **    invisible.
      */
      OleApp_EnableBusyDialogs(lpOleApp, TRUE, TRUE);

      OLEDBG_END3
   }


   /* OleApp_HideWindow
    * -----------------
    *
    *      Hide the window of the app from the user.
    *      take away the control of the app by the user.
    */
   void OleApp_HideWindow(LPOLEAPP lpOleApp)
   {
      LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)lpOleApp;

      OLEDBG_BEGIN3("OleApp_HideWindow\r\n")

      /* OLE2NOTE: the application is now being hidden, so we must release
      **    the external lock that was made on behalf of the user.
      **    if this is that last external lock on our application, thus
      **    enabling our application to complete its shutdown operation.
      */
      if (lpOleApp->m_fUserCtrl) {
         lpOleApp->m_fUserCtrl = FALSE;
         OleApp_Lock(lpOleApp, FALSE /*fLock*/, TRUE /*fLastUnlockReleases*/);
      }

      ShowWindow(lpOutlineApp->m_hWndApp, SW_HIDE);

      /* OLE2NOTE: because our app is now INVISIBLE, we must DISABLE the busy
      **    dialog. we should NOT put up any dialogs if our app is
      **    invisible.
      */
      OleApp_EnableBusyDialogs(lpOleApp, FALSE, FALSE);
      OLEDBG_END3
   }


   /* OleApp_Lock
   ** -----------
   **    Lock/Unlock the App object. if the last lock is unlocked and
   **    fLastUnlockReleases == TRUE, then the app object will shut down
   **    (ie. it will recieve its final release and its refcnt will go to 0).
   */
   HRESULT OleApp_Lock(LPOLEAPP lpOleApp, BOOL fLock, BOOL fLastUnlockReleases)
   {
      HRESULT hrErr;

   #if defined( _DEBUG )
      if (fLock) {
         OLEDBG_BEGIN2("CoLockObjectExternal(lpApp,TRUE) called\r\n")
      } else {
         if (fLastUnlockReleases)
            OLEDBG_BEGIN2("CoLockObjectExternal(lpApp,FALSE,TRUE) called\r\n")
         else
            OLEDBG_BEGIN2("CoLockObjectExternal(lpApp,FALSE,FALSE) called\r\n")
      }
   #endif  // _DEBUG

      OleApp_AddRef(lpOleApp);       // artificial AddRef to make object stable

      hrErr = CoLockObjectExternal(
            (LPUNKNOWN)&lpOleApp->m_Unknown, fLock, fLastUnlockReleases);

      OleApp_Release(lpOleApp);       // release artificial AddRef above

      OLEDBG_END2
      return hrErr;
   }


   /* OleApp_Destroy
    * --------------
    *
    *  Free all OLE related resources that had been allocated for the app.
    */
   void OleApp_Destroy(LPOLEAPP lpOleApp)
   {
      // OLE2NOTE: Revoke our message filter upon app shutdown.
      OleApp_RevokeMessageFilter(lpOleApp);

      // OLE2NOTE: Revoke our ClassFactory upon app shutdown.
      OleApp_RevokeClassFactory(lpOleApp);

   #if defined( IF_SPECIAL_DD_CURSORS_NEEDED )
      // This would be used if the app wanted to have custom drag/drop cursors
      DestroyCursor(lpOleApp->m_hcursorDragNone);
      DestroyCursor(lpOleApp->m_hcursorDragCopy);
      DestroyCursor(lpOleApp->m_hcursorDragLink);
      DestroyCursor(lpOleApp->m_hcursorDragMove);
   #endif  // IF_SPECIAL_DD_CURSORS_NEEDED

   #if defined( OLE_CNTR )
      if (lpOleApp->m_hStdPal) {
         DeleteObject(lpOleApp->m_hStdPal);
         lpOleApp->m_hStdPal = NULL;
      }
   #endif
   }


   /* OleApp_DocLockApp
   ** -----------------
   **    Add a lock on the App on behalf of the Doc. the App may not close
   **    while the Doc exists.
   **
   **    when a document is first created, it calls this method to
   **    guarantee that the application stays alive (OleDoc_Init).
   **    when a document is destroyed, it calls
   **    OleApp_DocUnlockApp to release this hold on the app.
   */
   void OleApp_DocLockApp(LPOLEAPP lpOleApp)
   {
      ULONG cDoc;

      OLEDBG_BEGIN3("OleApp_DocLockApp\r\n")

      cDoc = ++lpOleApp->m_cDoc;

      OleDbgOutRefCnt3("OleApp_DocLockApp: cDoc++\r\n", lpOleApp, cDoc);

      OleApp_Lock(lpOleApp, TRUE /* fLock */, 0 /* not applicable */);

      OLEDBG_END3
      return;
   }


   /* OleApp_DocUnlockApp
   ** -------------------
   **    Forget all references to a closed document.
   **    Release the lock on the App on behalf of the Doc. if this was the
   **    last lock on the app, then it will shutdown.
   */
   void OleApp_DocUnlockApp(LPOLEAPP lpOleApp, LPOUTLINEDOC lpOutlineDoc)
   {
      ULONG cDoc;

      OLEDBG_BEGIN3("OleApp_DocUnlockApp\r\n")

      /* OLE2NOTE: when there are no open documents and the app is not
      **    under the control of the user then revoke our ClassFactory to
      **    enable the app to shut down.
      */
      cDoc = --lpOleApp->m_cDoc;

   #if defined( _DEBUG )
      OleDbgAssertSz (
            lpOleApp->m_cDoc >= 0, "DocUnlockApp called with cDoc == 0");

      OleDbgOutRefCnt3(
            "OleApp_DocUnlockApp: cDoc--\r\n", lpOleApp, cDoc);
   #endif

      OleApp_Lock(lpOleApp, FALSE /* fLock */, TRUE /* fLastUnlockReleases */);

      OLEDBG_END3
      return;
   }


   /* OleApp_HideIfNoReasonToStayVisible
```

```
** ----------------------------------
**
** if there are no more documents visible to the user and the app
**    itself is not under user control, then it has no reason to stay
**    visible. we thus should hide the app. we can not directly destroy
**    the app, because it may be validly being used programatically by
**    another client application and should remain running. the app
**    should simply be hidden from the user.
*/
void OleApp_HideIfNoReasonToStayVisible(LPOLEAPP lpOleApp)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)lpOleApp;
   LPOUTLINEDOC lpOutlineDoc;

   OLEDBG_BEGIN3("OleApp_HideIfNoReasonToStayVisible\r\n")

   if (lpOleApp->m_fUserCtrl) {
      OLEDBG_END3
      return;     // remain visible; user in control of app
   }

   /* Because this is an SDI app, there is only one user document.
   ** check if it is visible to the user. an MDI app would loop over
   ** all open MDI child documents to see if any are visible.
   */
   lpOutlineDoc = (LPOUTLINEDOC)lpOutlineApp->m_lpDoc;
   if (lpOutlineDoc && IsWindowVisible(lpOutlineDoc->m_hWndDoc))
      return;     // remain visible; the doc is visible to the user

   // if we reached here, the app should be hidden
   OleApp_HideWindow(lpOleApp);

   OLEDBG_END3
}


/* OleApp_AddRef
** -------------
**
**  increment the ref count of the App object.
**
**    Returns the new ref count on the object
*/
ULONG OleApp_AddRef(LPOLEAPP lpOleApp)
{
   ++lpOleApp->m_cRef;

#if defined( _DEBUG )
   OleDbgOutRefCnt4(
         "OleApp_AddRef: cRef++\r\n",
         lpOleApp,
         lpOleApp->m_cRef
   );
#endif
   return lpOleApp->m_cRef;
```

```
}


/* OleApp_Release
** --------------
**
**  decrement the ref count of the App object.
**    if the ref count goes to 0, then the app object is destroyed.
**
**    Returns the remaining ref count on the object
*/
ULONG OleApp_Release (LPOLEAPP lpOleApp)
{
   ULONG cRef;

   cRef = --lpOleApp->m_cRef;

#if defined( _DEBUG )
   OleDbgAssertSz (lpOleApp->m_cRef >= 0, "Release called with cRef == 0");

   OleDbgOutRefCnt4(
         "OleApp_AddRef: cRef--\r\n", lpOleApp, cRef);
#endif  // _DEBUG
   /*****************************************************************
   ** OLE2NOTE: when the ClassFactory refcnt == 0, then destroy it.    **
   **    otherwise the ClassFactory is still in use.                   **
   ****************************************************************/

   if(cRef == 0)
      OutlineApp_Destroy((LPOUTLINEAPP)lpOleApp);

   return cRef;
}



/* OleApp_QueryInterface
** ---------------------
**
** Retrieve a pointer to an interface on the app object.
**
**    OLE2NOTE: this function will AddRef the ref cnt of the object.
**
**    Returns NOERROR if interface is successfully retrieved.
**             E_NOINTERFACE if the interface is not supported
*/
HRESULT OleApp_QueryInterface (
      LPOLEAPP                lpOleApp,
      REFIID                  riid,
      LPVOID FAR*             lplpvObj
)
{
   SCODE sc;

   /* OLE2NOTE: we must make sure to set all out ptr parameters to NULL. */
   *lplpvObj = NULL;

   if (IsEqualIID(riid, &IID_IUnknown)) {
      OleDbgOut4("OleApp_QueryInterface: IUnknown* RETURNED\r\n");

      *lplpvObj = (LPVOID) &lpOleApp->m_Unknown;
      OleApp_AddRef(lpOleApp);
      sc = S_OK;
   }
   else {
      sc = E_NOINTERFACE;
   }

   OleDbgQueryInterfaceMethod(*lplpvObj);
   return ResultFromScode(sc);
}


/* OleApp_RejectInComingCalls
** -------------------------
**    Reject/Handle in coming OLE (LRPC) calls.
**
**    OLE2NOTE: if the app is in a state when it can NOT handle in
**    coming OLE method calls from an external process (eg. the app has
**    an application modal dialog up), then it should call
**    OleApp_RejectInComingCalls(TRUE). in this state the
**    IMessageFilter::HandleInComingCall method will return
**    SERVERCALL_RETRYLATER. this tells the caller to try again in a
**    little while. normally the calling app will put up a dialog (see
**    OleUIBusy dialog) in this situation informing the user of the
**    situation. the user then is normally given the option to
**    "Switch To..." the busy application, retry, or cancel the
**    operation. when the app is ready to continue processing such
**    calls, it should call OleApp_RejectInComingCalls(FALSE). in this
**    state, SERVERCALL_ISHANDLED is returned by
**    IMessageFilter::HandleInComingCall.
*/
void OleApp_RejectInComingCalls(LPOLEAPP lpOleApp, BOOL fReject)
{
#if defined( _DEBUG )
   if (fReject)
      OleDbgOut3("OleApp_RejectInComingCalls(TRUE)\r\n");
   else
      OleDbgOut3("OleApp_RejectInComingCalls(FALSE)\r\n");
#endif  // _DEBUG

   OleDbgAssert(lpOleApp->m_lpMsgFilter != NULL);
   if (! lpOleApp->m_lpMsgFilter)
      return;

   OleStdMsgFilter_SetInComingCallStatus(
         lpOleApp->m_lpMsgFilter,
         (fReject ? SERVERCALL_RETRYLATER : SERVERCALL_ISHANDLED)
   );
}
```

```
/* OleApp_DisableBusyDialogs
** -------------------------
**    Disable the Busy and NotResponding dialogs.
**
**    Returns previous enable state so that it can be restored by
**    calling OleApp_ReEnableBusyDialogs.
*/
void OleApp_DisableBusyDialogs(
      LPOLEAPP        lpOleApp,
      BOOL FAR*       lpfPrevBusyEnable,
      BOOL FAR*       lpfPrevNREnable
)
{
   if (lpOleApp->m_lpMsgFilter) {
      *lpfPrevNREnable = OleStdMsgFilter_EnableNotRespondingDialog(
            lpOleApp->m_lpMsgFilter, FALSE);
      *lpfPrevBusyEnable = OleStdMsgFilter_EnableBusyDialog(
            lpOleApp->m_lpMsgFilter, FALSE);
   }
}


/* OleApp_EnableBusyDialogs
** ------------------------
**    Set the enable state of the Busy and NotResponding dialogs.
**
**    This function is typically used after a call to
**    OleApp_DisableBusyDialogs in order to restore the previous enable
**    state of the dialogs.
*/
void OleApp_EnableBusyDialogs(
      LPOLEAPP        lpOleApp,
      BOOL            fPrevBusyEnable,
      BOOL            fPrevNREnable
)
{
   if (lpOleApp->m_lpMsgFilter) {
      OleStdMsgFilter_EnableNotRespondingDialog(
            lpOleApp->m_lpMsgFilter, fPrevNREnable);
      OleStdMsgFilter_EnableBusyDialog(
            lpOleApp->m_lpMsgFilter, fPrevBusyEnable);
   }
}


/* OleApp_PreModalDialog
** ---------------------
**    Keep track that a modal dialog is about to be brought up.
**
**    while a modal dialog is up we need to take special actions:
**    1. we do NOT want to initialize our tool bar buttons on
**    WM_ACTIVATEAPP. the tool bar is not accessible.
**    2. we want to reject new top-level, incoming LRPC calls
**       (return SERVERCALL_RETRYLATER from IMessageFilter::
**        HandleInComingCall).
**    3. (IN-PLACE SERVER) tell our in-place container to disable
**    modeless dialogs by calling IOleInPlaceFrame::
**    EnableModeless(FALSE).
**    4. (IN-PLACE CONTAINER) tell our UIActive in-place object to
**    disable modeless dialogs by calling IOleInPlaceActiveObject::
**    EnableModeless(FALSE).
*/
void OleApp_PreModalDialog(LPOLEAPP lpOleApp, LPOLEDOC lpOleDoc)
{
   if (lpOleApp->m_cModalDlgActive == 0) {
      // top-level modal dialog is being brought up

#if defined( USE_FRAMETOOLS )
      LPFRAMETOOLS lptb;

      if (lpOleDoc)
         lptb = ((LPOUTLINEDOC)lpOleDoc)->m_lpFrameTools;
      else
         lptb = OutlineApp_GetFrameTools((LPOUTLINEAPP)lpOleApp);
      if (lptb)
         FrameTools_EnableWindow(lptb, FALSE);
#endif  // USE_FRAMETOOLS

      OleApp_RejectInComingCalls(lpOleApp, TRUE);

#if defined( INPLACE_SVR )
      {
         LPSERVERDOC  lpServerDoc = (LPSERVERDOC)lpOleDoc;

         /* if the document bringing up the modal dialog is
         **    currently a UIActive in-place object, then tell the
         **    top-level in-place frame to disable its modeless
         **    dialogs.
         */
         if (lpServerDoc && lpServerDoc->m_fUIActive &&
               lpServerDoc->m_lpIPData &&
               lpServerDoc->m_lpIPData->lpFrame) {
            OLEDBG_BEGIN2("IOleInPlaceFrame::EnableModless(FALSE)
called\r\n");
            lpServerDoc->m_lpIPData->lpFrame->lpVtbl->EnableModeless(
                  lpServerDoc->m_lpIPData->lpFrame, FALSE);
            OLEDBG_END2
         }
      }
#endif  // INPLACE_SVR
#if defined( INPLACE_CNTR )
      {
         LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)lpOleApp;

         /* if the document bringing up the modal dialog is an
         **    in-place container that has a UIActive object, then
         **    tell the UIActive object to disable its modeless
         **    dialogs.
         */
         if (lpContainerApp->m_lpIPActiveObj) {
            OLEDBG_BEGIN2("IOleInPlaceActiveObject::EnableModless(FALSE)
called\r\n");
            lpContainerApp->m_lpIPActiveObj->lpVtbl->EnableModeless(
                  lpContainerApp->m_lpIPActiveObj, FALSE);
            OLEDBG_END2
         }
      }
#endif  // INPLACE_CNTR
   }

   lpOleApp->m_cModalDlgActive++;
}


/* OleApp_PostModalDialog
** ----------------------
**    Keep track that a modal dialog is being brought down. this call
**    balances the OleApp_PreModalDialog call.
*/
void OleApp_PostModalDialog(LPOLEAPP lpOleApp, LPOLEDOC lpOleDoc)
{
   lpOleApp->m_cModalDlgActive--;

   if (lpOleApp->m_cModalDlgActive == 0) {
      // last modal dialog is being brought down

#if defined( USE_FRAMETOOLS )
      LPFRAMETOOLS lptb;

      if (lpOleDoc)
         lptb = ((LPOUTLINEDOC)lpOleDoc)->m_lpFrameTools;
      else
         lptb = OutlineApp_GetFrameTools((LPOUTLINEAPP)lpOleApp);
      if (lptb) {
         FrameTools_EnableWindow(lptb, TRUE);
         FrameTools_UpdateButtons(lptb, (LPOUTLINEDOC)lpOleDoc);
      }
#endif  // USE_FRAMETOOLS

      OleApp_RejectInComingCalls(lpOleApp, FALSE);

#if defined( INPLACE_SVR )
      {
         LPSERVERDOC  lpServerDoc = (LPSERVERDOC)lpOleDoc;

         /* if the document bringing down the modal dialog is
         **    currently a UIActive in-place object, then tell the
         **    top-level in-place frame it can re-enable its
         **    modeless dialogs.
         */
         if (lpServerDoc && lpServerDoc->m_fUIActive &&
               lpServerDoc->m_lpIPData &&
               lpServerDoc->m_lpIPData->lpFrame) {
```

```
            OLEDBG_BEGIN2("IOleInPlaceFrame::EnableModless(TRUE)
called\r\n");
            lpServerDoc->m_lpIPData->lpFrame->lpVtbl->EnableModeless(
                  lpServerDoc->m_lpIPData->lpFrame, TRUE);
            OLEDBG_END2
         }
      }
#endif  // INPLACE_SVR
#if defined( INPLACE_CNTR )
      {
         LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)lpOleApp;

         /* if the document bringing down the modal dialog is an
         **    in-place container that has a UIActive object, then
         **    tell the UIActive object it can re-enable its
         **    modeless dialogs.
         */
         if (lpContainerApp->m_lpIPActiveObj) {
            OLEDBG_BEGIN2("IOleInPlaceActiveObject::EnableModless(TRUE)
called\r\n");
            lpContainerApp->m_lpIPActiveObj->lpVtbl->EnableModeless(
                  lpContainerApp->m_lpIPActiveObj, TRUE);
            OLEDBG_END2
         }
      }
#endif  // INPLACE_CNTR
   }
}


/* OleApp_InitVtbls
 * ----------------
 *
 * initialize the methods in all of the interface Vtbl's
 *
 * OLE2NOTE: we only need one copy of each Vtbl. When an object which
 *      exposes an interface is instantiated, its lpVtbl is intialized
 *      to point to the single copy of the Vtbl.
 *
 */
BOOL OleApp_InitVtbls (LPOLEAPP lpOleApp)
{
   BOOL fStatus;

   // OleApp::IUnknown method table
   OleStdInitVtbl(&g_OleApp_UnknownVtbl, sizeof(IUnknownVtbl));
   g_OleApp_UnknownVtbl.QueryInterface = OleApp_Unk_QueryInterface;
   g_OleApp_UnknownVtbl.AddRef         = OleApp_Unk_AddRef;
   g_OleApp_UnknownVtbl.Release        = OleApp_Unk_Release;
   fStatus = OleStdCheckVtbl(
         &g_OleApp_UnknownVtbl,
         sizeof(IUnknownVtbl),
         OLESTR("IUnknown")
      );
   if (! fStatus) return FALSE;
   // OleDoc::IUnknown method table
   OleStdInitVtbl(&g_OleDoc_UnknownVtbl, sizeof(IUnknownVtbl));
   g_OleDoc_UnknownVtbl.QueryInterface = OleDoc_Unk_QueryInterface;
   g_OleDoc_UnknownVtbl.AddRef         = OleDoc_Unk_AddRef;
   g_OleDoc_UnknownVtbl.Release        = OleDoc_Unk_Release;
   fStatus = OleStdCheckVtbl(
         &g_OleDoc_UnknownVtbl,
         sizeof(IUnknownVtbl),
         OLESTR("IUnknown")
      );
   if (! fStatus) return FALSE;

   // OleDoc::IPersistFile method table
   OleStdInitVtbl(&g_OleDoc_PersistFileVtbl, sizeof(IPersistFileVtbl));
   g_OleDoc_PersistFileVtbl.QueryInterface = OleDoc_PFile_QueryInterface;
   g_OleDoc_PersistFileVtbl.AddRef         = OleDoc_PFile_AddRef;
   g_OleDoc_PersistFileVtbl.Release        = OleDoc_PFile_Release;
   g_OleDoc_PersistFileVtbl.GetClassID     = OleDoc_PFile_GetClassID;
   g_OleDoc_PersistFileVtbl.IsDirty        = OleDoc_PFile_IsDirty;
   g_OleDoc_PersistFileVtbl.Load           = OleDoc_PFile_Load;
   g_OleDoc_PersistFileVtbl.Save           = OleDoc_PFile_Save;
   g_OleDoc_PersistFileVtbl.SaveCompleted  = OleDoc_PFile_SaveCompleted;
   g_OleDoc_PersistFileVtbl.GetCurFile     = OleDoc_PFile_GetCurFile;
   fStatus = OleStdCheckVtbl(
         &g_OleDoc_PersistFileVtbl,
         sizeof(IPersistFileVtbl),
         OLESTR("IPersistFile")
      );
   if (! fStatus) return FALSE;

   // OleDoc::IOleItemContainer method table
   OleStdInitVtbl(&g_OleDoc_OleItemContainerVtbl,
sizeof(IOleItemContainerVtbl));
   g_OleDoc_OleItemContainerVtbl.QueryInterface =
                                 OleDoc_ItemCont_QueryInterface;
   g_OleDoc_OleItemContainerVtbl.AddRef    = OleDoc_ItemCont_AddRef;
   g_OleDoc_OleItemContainerVtbl.Release   = OleDoc_ItemCont_Release;
   g_OleDoc_OleItemContainerVtbl.ParseDisplayName  =
                                 OleDoc_ItemCont_ParseDisplayName;
   g_OleDoc_OleItemContainerVtbl.EnumObjects= OleDoc_ItemCont_EnumObjects;
   g_OleDoc_OleItemContainerVtbl.LockContainer =
                                 OleDoc_ItemCont_LockContainer;
   g_OleDoc_OleItemContainerVtbl.GetObject = OleDoc_ItemCont_GetObject;
   g_OleDoc_OleItemContainerVtbl.GetObjectStorage =
                                 OleDoc_ItemCont_GetObjectStorage;
   g_OleDoc_OleItemContainerVtbl.IsRunning = OleDoc_ItemCont_IsRunning;
   fStatus = OleStdCheckVtbl(
         &g_OleDoc_OleItemContainerVtbl,
         sizeof(IOleItemContainerVtbl),
         OLESTR("IOleItemContainer")
      );
   if (! fStatus) return FALSE;

   // OleDoc::IExternalConnection method table
```

```
    OleStdInitVtbl(
          &g_OleDoc_ExternalConnectionVtbl,sizeof(IExternalConnectionVtbl));
    g_OleDoc_ExternalConnectionVtbl.QueryInterface =
                                OleDoc_ExtConn_QueryInterface;
    g_OleDoc_ExternalConnectionVtbl.AddRef         = OleDoc_ExtConn_AddRef;
    g_OleDoc_ExternalConnectionVtbl.Release        = OleDoc_ExtConn_Release;
    g_OleDoc_ExternalConnectionVtbl.AddConnection  =
                                OleDoc_ExtConn_AddConnection;
    g_OleDoc_ExternalConnectionVtbl.ReleaseConnection =
                                OleDoc_ExtConn_ReleaseConnection;
    fStatus = OleStdCheckVtbl(
          &g_OleDoc_ExternalConnectionVtbl,
          sizeof(IExternalConnectionVtbl),
          OLESTR("IExternalConnection")
    );
    if (! fStatus) return FALSE;

    // OleDoc::IDataObject method table
    OleStdInitVtbl(&g_OleDoc_DataObjectVtbl, sizeof(IDataObjectVtbl));
    g_OleDoc_DataObjectVtbl.QueryInterface  = OleDoc_DataObj_QueryInterface;
    g_OleDoc_DataObjectVtbl.AddRef          = OleDoc_DataObj_AddRef;
    g_OleDoc_DataObjectVtbl.Release         = OleDoc_DataObj_Release;
    g_OleDoc_DataObjectVtbl.GetData         = OleDoc_DataObj_GetData;
    g_OleDoc_DataObjectVtbl.GetDataHere     = OleDoc_DataObj_GetDataHere;
    g_OleDoc_DataObjectVtbl.QueryGetData    = OleDoc_DataObj_QueryGetData;
    g_OleDoc_DataObjectVtbl.GetCanonicalFormatEtc =
                             OleDoc_DataObj_GetCanonicalFormatEtc;
    g_OleDoc_DataObjectVtbl.SetData         = OleDoc_DataObj_SetData;
    g_OleDoc_DataObjectVtbl.EnumFormatEtc   = OleDoc_DataObj_EnumFormatEtc;
    g_OleDoc_DataObjectVtbl.DAdvise          = OleDoc_DataObj_DAdvise;
    g_OleDoc_DataObjectVtbl.DUnadvise        = OleDoc_DataObj_DUnadvise;
    g_OleDoc_DataObjectVtbl.EnumDAdvise      = OleDoc_DataObj_EnumDAdvise;

    fStatus = OleStdCheckVtbl(
          &g_OleDoc_DataObjectVtbl,
          sizeof(IDataObjectVtbl),
          OLESTR("IDataObject")
       );
    if (! fStatus) return FALSE;

#if defined( USE_DRAGDROP )

    // OleDoc::IDropTarget method table
    OleStdInitVtbl(&g_OleDoc_DropTargetVtbl, sizeof(IDropTargetVtbl));
    g_OleDoc_DropTargetVtbl.QueryInterface= OleDoc_DropTarget_QueryInterface;
    g_OleDoc_DropTargetVtbl.AddRef      = OleDoc_DropTarget_AddRef;
    g_OleDoc_DropTargetVtbl.Release     = OleDoc_DropTarget_Release;

    g_OleDoc_DropTargetVtbl.DragEnter   = OleDoc_DropTarget_DragEnter;
    g_OleDoc_DropTargetVtbl.DragOver    = OleDoc_DropTarget_DragOver;
    g_OleDoc_DropTargetVtbl.DragLeave   = OleDoc_DropTarget_DragLeave;
    g_OleDoc_DropTargetVtbl.Drop        = OleDoc_DropTarget_Drop;

    fStatus = OleStdCheckVtbl(
          &g_OleDoc_DropTargetVtbl,
          sizeof(IDropTargetVtbl),
          OLESTR("IDropTarget")
    );
    if (! fStatus)
       return FALSE;

    // OleDoc::IDropSource method table
    OleStdInitVtbl(&g_OleDoc_DropSourceVtbl, sizeof(IDropSourceVtbl));
    g_OleDoc_DropSourceVtbl.QueryInterface  =
                               OleDoc_DropSource_QueryInterface;
    g_OleDoc_DropSourceVtbl.AddRef          = OleDoc_DropSource_AddRef;
    g_OleDoc_DropSourceVtbl.Release         = OleDoc_DropSource_Release;

    g_OleDoc_DropSourceVtbl.QueryContinueDrag =
                               OleDoc_DropSource_QueryContinueDrag;
    g_OleDoc_DropSourceVtbl.GiveFeedback    = OleDoc_DropSource_GiveFeedback;

    fStatus = OleStdCheckVtbl(
          &g_OleDoc_DropSourceVtbl,
          sizeof(IDropSourceVtbl),
          OLESTR("IDropSource")
    );
    if (! fStatus) return FALSE;
 #endif  // USE_DRAGDROP

 #if defined( OLE_SERVER )

    // Initialize the server specific interface method tables.
    if (! ServerApp_InitVtbls((LPSERVERAPP)lpOleApp))
       return FALSE;
 #endif
 #if defined( OLE_CNTR )

    // Initialize the container specific interface method tables.
    if (! ContainerApp_InitVtbls((LPCONTAINERAPP)lpOleApp))
       return FALSE;
 #endif
    return TRUE;
 };



 /* OleApp_InitMenu
  * ---------------
  *
  *      Enable or Disable menu items depending on the state of
  * the appliation.
  * The OLE versions of the Outline sample app add a PasteSpecial command.
  * Also, the container version add InsertObject and ObjectVerb menu items.
  */
 void OleApp_InitMenu(
       LPOLEAPP                lpOleApp,
       LPOLEDOC                lpOleDoc,
       HMENU                   hMenu
 )
```

```
{
   BOOL bMsgFilterInstalled = FALSE;
   BOOL bRejecting = FALSE;

   if (!lpOleApp || !hMenu)
      return;

   OLEDBG_BEGIN3("OleApp_InitMenu\r\n")

   /*
   ** Enable/disable menu items for Message Filter
   */
   bMsgFilterInstalled = (lpOleApp->m_lpMsgFilter != NULL);
   bRejecting = bMsgFilterInstalled &&
      OleStdMsgFilter_GetInComingCallStatus(lpOleApp->m_lpMsgFilter) !=
SERVERCALL_ISHANDLED;

   CheckMenuItem(hMenu,
      IDM_D_INSTALLMSGFILTER,
      bMsgFilterInstalled ? MF_CHECKED : MF_UNCHECKED);

   EnableMenuItem(hMenu,
      IDM_D_REJECTINCOMING,
      bMsgFilterInstalled ? MF_ENABLED : MF_GRAYED);

   CheckMenuItem(hMenu,
      IDM_D_REJECTINCOMING,
      bRejecting ? MF_CHECKED : MF_UNCHECKED);

#if defined( OLE_CNTR )
   {
      LPCONTAINERDOC lpContainerDoc = (LPCONTAINERDOC)lpOleDoc;
      BOOL fShowObject;

      fShowObject = ContainerDoc_GetShowObjectFlag(lpContainerDoc);
      CheckMenuItem(
            hMenu,
            IDM_O_SHOWOBJECT,
            (fShowObject ? MF_CHECKED : MF_UNCHECKED)
      );

#if defined( INPLACE_CNTR ) && defined( _DEBUG )
      CheckMenuItem(
            hMenu,
            IDM_D_INSIDEOUT,
            g_fInsideOutContainer ? MF_CHECKED:MF_UNCHECKED);
#endif  // INPLACE_CNTR && _DEBUG

   }
#endif  // OLE_CNTR

   OLEDBG_END3
}
```

```
/* OleApp_UpdateEditMenu
 * ---------------------
 *
 *  Purpose:
 *      Update the Edit menuitems of the App according to the state of
 *      OutlineDoc
 *
 *  Parameter:
 *      lpOutlineDoc        pointer to the document
 *      hMenuEdit           edit menu handle
 */
void OleApp_UpdateEditMenu(
      LPOLEAPP                lpOleApp,
      LPOUTLINEDOC            lpOutlineDoc,
      HMENU                   hMenuEdit
)
{
   int              nFmtEtc;
   UINT             uEnablePaste = MF_GRAYED;
   UINT             uEnablePasteLink = MF_GRAYED;
   LPDATAOBJECT    lpClipboardDataObj;
   LPOLEDOC         lpOleDoc = (LPOLEDOC)lpOutlineDoc;
   HRESULT          hrErr;
   BOOL             fPrevEnable1;
   BOOL             fPrevEnable2;

   if (!lpOleApp || !lpOutlineDoc || !hMenuEdit)
      return;

   if (!OleDoc_GetUpdateEditMenuFlag(lpOleDoc))
      /* OLE2NOTE: if the flag is not set, we don't have to update
      **    the edit menu again. This blocks repetitive updating when
      **    the user move the mouse across Edit menu while holding
      **    down the button
      */
      return;

   OLEDBG_BEGIN3("OleApp_InitEditMenu\r\n")

   /* OLE2NOTE: we do not want to ever give the busy dialog when we
   **    are trying to put up our menus. eg. even if the source of
   **    data on the clipboard is busy, we do not want put up the busy
   **    dialog. thus we will disable the dialog and at the end
   **    re-enable it.
   */
   OleApp_DisableBusyDialogs(lpOleApp, &fPrevEnable1, &fPrevEnable2);

   // check if there is data on the clipboard that we can paste/paste link

   OLEDBG_BEGIN2("OleGetClipboard called\r\n")
   hrErr = OleGetClipboard((LPDATAOBJECT FAR*)&lpClipboardDataObj);
   OLEDBG_END2

   if (hrErr == NOERROR) {
      nFmtEtc = OleStdGetPriorityClipboardFormat(
            lpClipboardDataObj,
            lpOleApp->m_arrPasteEntries,
            lpOleApp->m_nPasteEntries
      );

      if (nFmtEtc >= 0)
         uEnablePaste = MF_ENABLED;  // there IS a format we like

      OLEDBG_BEGIN2("OleQueryLinkFromData called\r\n")
      hrErr = OleQueryLinkFromData(lpClipboardDataObj);
      OLEDBG_END2

      if(hrErr == NOERROR)
         uEnablePasteLink = MF_ENABLED;

      OleStdRelease((LPUNKNOWN)lpClipboardDataObj);
   }

   EnableMenuItem(hMenuEdit, IDM_E_PASTE, uEnablePaste);
   EnableMenuItem(hMenuEdit, IDM_E_PASTESPECIAL, uEnablePaste);


#if defined( OLE_CNTR )
   if (ContainerDoc_GetNextLink((LPCONTAINERDOC)lpOutlineDoc, NULL))
      EnableMenuItem(hMenuEdit, IDM_E_EDITLINKS, MF_ENABLED);
   else
      EnableMenuItem(hMenuEdit, IDM_E_EDITLINKS, MF_GRAYED);


   {
      LPCONTAINERAPP  lpContainerApp = (LPCONTAINERAPP)lpOleApp;
      HMENU           hMenuVerb = NULL;
      LPOLEOBJECT     lpOleObj = NULL;
      LPCONTAINERLINE lpContainerLine = NULL;
      BOOL            fSelIsOleObject;

      EnableMenuItem(hMenuEdit, IDM_E_PASTELINK, uEnablePasteLink);

      /* check if selection is a single line that contains an OleObject */

      fSelIsOleObject = ContainerDoc_IsSelAnOleObject(
            (LPCONTAINERDOC)lpOutlineDoc,
            &IID_IOleObject,
            (LPUNKNOWN FAR*)&lpOleObj,
            NULL,    /* we don't need the line index */
            (LPCONTAINERLINE FAR*)&lpContainerLine
      );

      if (hMenuEdit != NULL) {

         /* If the current line is an ContainerLine, add the object
         **    verb sub menu to the Edit menu. if the line is not an
         **    ContainerLine, (lpOleObj==NULL) then disable the
         **    Edit.Object command. this helper API takes care of
         **    building the verb menu as appropriate.
         */
         OleUIAddVerbMenu(
               (LPOLEOBJECT)lpOleObj,
               (lpContainerLine ? lpContainerLine->m_lpszShortType:NULL),
               hMenuEdit,
               POS_OBJECT,
               IDM_E_OBJECTVERBMIN,
               0,                     // no uIDVerbMax enforced
               TRUE,                  // Add Convert menu item
               IDM_E_CONVERTVERB,     // ID for Convert menu item
               (HMENU FAR*) &hMenuVerb
         );

#if defined( USE_STATUSBAR_LATER )
         /* setup status messages for the object verb menu */
         if (hMenuVerb) {
            // REVIEW: this string should come from a string resource.
            // REVIEW: this doesn't work for dynamically created menus
            AssignPopupMessage(
                  hMenuVerb,
                  "Open, edit or interact with an object"
            );
         }
#endif  // USE_STATUSBAR_LATER
      }

      if (lpOleObj)
         OleStdRelease((LPUNKNOWN)lpOleObj);
   }

#endif  // OLE_CNTR

   // re-enable the Busy/NotResponding dialogs
   OleApp_EnableBusyDialogs(lpOleApp, fPrevEnable1, fPrevEnable2);

   OleDoc_SetUpdateEditMenuFlag(lpOleDoc, FALSE);

   OLEDBG_END3
}


/* OleApp_RegisterClassFactory
 * ---------------------------
 *
 * Register our app's ClassFactory with OLE.
 *
 */
BOOL OleApp_RegisterClassFactory(LPOLEAPP lpOleApp)
{
   HRESULT hrErr;

   if (lpOleApp->m_lpClassFactory)
      return TRUE;    // already registered
   OLEDBG_BEGIN3("OleApp_RegisterClassFactory\r\n")

   /*****************************************************************
   ** An SDI app must register its ClassFactory if it is launched
   **    for embedding (/Embedding command line option specified).
   ** An MDI app must register its ClassFactory in all cases,
   *****************************************************************/

   lpOleApp->m_lpClassFactory = AppClassFactory_Create();
   if (! lpOleApp->m_lpClassFactory) {
      OutlineApp_ErrorMessage(g_lpApp, ErrMsgCreateCF);
      goto error;
   }

   OLEDBG_BEGIN2("CoRegisterClassObject called\r\n")
   hrErr = CoRegisterClassObject(
            &CLSID_APP,
            (LPUNKNOWN)lpOleApp->m_lpClassFactory,
            CLSCTX_LOCAL_SERVER,
            REGCLS_SINGLEUSE,
            &lpOleApp->m_dwRegClassFac
   );
   OLEDBG_END2

   if(hrErr != NOERROR) {
      OleDbgOutHResult("CoRegisterClassObject returned", hrErr);
      OutlineApp_ErrorMessage(g_lpApp, ErrMsgRegCF);
      goto error;
   }

   OLEDBG_END3
   return TRUE;

error:

   if (lpOleApp->m_lpClassFactory) {
      OleStdRelease((LPUNKNOWN)lpOleApp->m_lpClassFactory);
      lpOleApp->m_lpClassFactory = NULL;
   }
   OLEDBG_END3
   return FALSE;
}


/* OleApp_RevokeClassFactory
 * -------------------------
 *
 * Revoke our app's ClassFactory.
 *
 */
void OleApp_RevokeClassFactory(LPOLEAPP lpOleApp)
{
   HRESULT hrErr;

   if (lpOleApp->m_lpClassFactory) {
      OLEDBG_BEGIN2("CoRevokeClassObject called\r\n")
      hrErr = CoRevokeClassObject(lpOleApp->m_dwRegClassFac);
      OLEDBG_END2

#if defined( _DEBUG )
      if (hrErr != NOERROR) {
         OleDbgOutHResult("CoRevokeClassObject returned", hrErr);
      }
#endif

      OleStdVerifyRelease(
            (LPUNKNOWN)lpOleApp->m_lpClassFactory,
            OLESTR("ClassFactory NOT released properly!")
      );
      lpOleApp->m_lpClassFactory = NULL;
   }
}


#if defined( USE_MSGFILTER )

/* OleApp_RegisterMessageFilter
 * ----------------------------
 *  Register our IMessageFilter*. the message filter is used to handle
 *  concurrency. we will use a standard implementation of IMessageFilter
 *  that is included as part of the OLE2UI library.
 */
BOOL OleApp_RegisterMessageFilter(LPOLEAPP lpOleApp)
{
   HRESULT hrErr;
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)lpOleApp;

   if (lpOleApp->m_lpMsgFilter == NULL) {
      // Register our message filter.
      lpOleApp->m_lpfnMsgPending = (MSGPENDINGPROC)MessagePendingProc;
      lpOleApp->m_lpMsgFilter = OleStdMsgFilter_Create(
            g_lpApp->m_hWndApp,
            (LPOLESTR)APPNAME,
            lpOleApp->m_lpfnMsgPending,
            NULL    /* Busy dialog callback hook function */
      );

      OLEDBG_BEGIN2("CoRegisterMessageFilter called\r\n")
      hrErr = CoRegisterMessageFilter(
               lpOleApp->m_lpMsgFilter,
               NULL    /* don't need previous message filter */
      );
      OLEDBG_END2

      if(hrErr != NOERROR) {
         OutlineApp_ErrorMessage(lpOutlineApp, ErrMsgRegMF);
         return FALSE;
      }
   }
   return TRUE;
}


/* OleApp_RevokeMessageFilter
 * --------------------------
 *  Revoke our IMessageFilter*. the message filter is used to handle
 *  concurrency. we will use a standard implementation of IMessageFilter
 *  that is included as part of the OLE2UI library.
 */
void OleApp_RevokeMessageFilter(LPOLEAPP lpOleApp)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;

   if (lpOleApp->m_lpMsgFilter != NULL) {
      // Revoke our message filter
      OLEDBG_BEGIN2("CoRegisterMessageFilter(NULL) called\r\n")
      CoRegisterMessageFilter(NULL, NULL);
      OLEDBG_END2

      if (lpOleApp->m_lpfnMsgPending) {
         lpOleApp->m_lpfnMsgPending = NULL;
      }

      OleStdVerifyRelease(
            (LPUNKNOWN)lpOleApp->m_lpMsgFilter,
            OLESTR("Release MessageFilter FAILED!")
      );
      lpOleApp->m_lpMsgFilter = NULL;
   }
}


/* MessagePendingProc
 * ------------------
 *
 * Callback function for the IMessageFilter::MessagePending procedure.  This
 * function is called when a message is received by our application while
 * we are waiting for an OLE call to complete.  We are essentially
 * blocked at this point, waiting for a response from the other OLE
application.
 * We should not process any messages which might cause another OLE call
 * to become blocked, or any other call which might cause re-entrancy
problems.
 *
 * For this application, only process WM_PAINT messages.  A more
sophisticated
 * application might allow certain menu messages and menu items to be
processed
 * also.
 *
 * RETURNS: TRUE if we processed the message, FALSE if we did not.
 */

BOOL FAR PASCAL EXPORT MessagePendingProc(MSG FAR *lpMsg)
```

```
{
   // Our application is only handling WM_PAINT messages when we are blocked
   switch (lpMsg->message) {
      case WM_PAINT:
         OleDbgOut2("WM_PAINT dispatched while blocked\r\n");

         DispatchMessage(lpMsg);
         break;
   }

   return FALSE;   // return PENDINGMSG_WAITDEFPROCESS from MessagePending
}
#endif  // USE_MSGFILTER


/* OleApp_FlushClipboard
 * ---------------------
 *
 *  Force the Windows clipboard to release our clipboard DataObject.
 */
void OleApp_FlushClipboard(LPOLEAPP lpOleApp)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)lpOleApp;
   LPOLEDOC lpClipboardDoc = (LPOLEDOC)lpOutlineApp->m_lpClipboardDoc;
   OLEDBG_BEGIN3("OleApp_FlushClipboard\r\n")

   /* OLE2NOTE: if for some reason our clipboard data transfer
   **    document is still held on to by an external client, we want
   **    to forceably break all external connections.
   */
   OLEDBG_BEGIN2("CoDisconnectObject called\r\n")
   CoDisconnectObject((LPUNKNOWN)&lpClipboardDoc->m_Unknown, 0);
   OLEDBG_END2

   OLEDBG_BEGIN2("OleFlushClipboard called\r\n")
   OleFlushClipboard();
   OLEDBG_END2

   lpOutlineApp->m_lpClipboardDoc = NULL;

   OLEDBG_END3
}


/* OleApp_NewCommand
 * -----------------
 *
 *  Start a new untitled document (File.New command).
 */
void OleApp_NewCommand(LPOLEAPP lpOleApp)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)lpOleApp;
   LPOUTLINEDOC lpOutlineDoc = lpOutlineApp->m_lpDoc;

   if (! OutlineDoc_Close(lpOutlineDoc, OLECLOSE_PROMPTSAVE))
```

```
      return;

   OleDbgAssertSz(lpOutlineApp->m_lpDoc==NULL,"Closed doc NOT properly
destroyed");
   lpOutlineApp->m_lpDoc = OutlineApp_CreateDoc(lpOutlineApp, FALSE);
   if (! lpOutlineApp->m_lpDoc) goto error;

   /* OLE2NOTE: initially the Doc object is created with a 0 ref
   **    count. in order to have a stable Doc object during the
   **    process of initializing the new Doc instance,
   **    we intially AddRef the Doc ref cnt and later
   **    Release it. This initial AddRef is artificial; it is simply
   **    done to guarantee that a harmless QueryInterface followed by
   **    a Release does not inadvertantly force our object to destroy
   **    itself prematurely.
   */
   OleDoc_AddRef((LPOLEDOC)lpOutlineApp->m_lpDoc);

   // set the doc to an (Untitled) doc.
   if (! OutlineDoc_InitNewFile(lpOutlineApp->m_lpDoc))
      goto error;

   // position and size the new doc window
   OutlineApp_ResizeWindows(lpOutlineApp);
   OutlineDoc_ShowWindow(lpOutlineApp->m_lpDoc); // calls OleDoc_Lock

   OleDoc_Release((LPOLEDOC)lpOutlineApp->m_lpDoc);  // rel artificial
AddRef

   return;

error:
   // REVIEW: should load string from string resource
   OutlineApp_ErrorMessage(lpOutlineApp, OLESTR("Could not create new
document"));

   if (lpOutlineApp->m_lpDoc) {
      // releasing the artificial AddRef above will destroy the document
      OleDoc_Release((LPOLEDOC)lpOutlineApp->m_lpDoc);
      lpOutlineApp->m_lpDoc = NULL;
   }

   return;
}


/* OleApp_OpenCommand
 * ------------------
 *
 *  Load a document from file (File.Open command).
 */
void OleApp_OpenCommand(LPOLEAPP lpOleApp)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)lpOleApp;
   LPOUTLINEDOC lpOutlineDoc = lpOutlineApp->m_lpDoc;
   OPENFILENAME ofn;
   char szFilter[]=APPFILENAMEFILTER;
   char szFileName[256];
   UINT i;
   DWORD dwSaveOption = OLECLOSE_PROMPTSAVE;
   BOOL fStatus = TRUE;
   OLECHAR szUniStr[256];

   if (! OutlineDoc_CheckSaveChanges(lpOutlineDoc, &dwSaveOption))
      return;           // abort opening new doc

   for(i=0; szFilter[i]; i++)
      if(szFilter[i]=='|') szFilter[i]='\0';

   _fmemset((LPOPENFILENAME)&ofn,0,sizeof(OPENFILENAME));

   szFileName[0]='\0';

   ofn.lStructSize=sizeof(OPENFILENAME);
   ofn.hwndOwner=lpOutlineApp->m_hWndApp;
   ofn.lpstrFilter=(LPSTR)szFilter;
   ofn.lpstrFile=(LPSTR)szFileName;
   ofn.nMaxFile=256;
   ofn.Flags=OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;
   ofn.lpstrDefExt=DEFEXTENSION;

   OleApp_PreModalDialog(lpOleApp, (LPOLEDOC)lpOutlineApp->m_lpDoc);

   fStatus = GetOpenFileName((LPOPENFILENAME)&ofn);

   OleApp_PostModalDialog(lpOleApp, (LPOLEDOC)lpOutlineApp->m_lpDoc);

   if(! fStatus)
      return;         // user canceled file open dialog

   OutlineDoc_Close(lpOutlineDoc, OLECLOSE_NOSAVE);
   OleDbgAssertSz(lpOutlineApp->m_lpDoc==NULL,"Closed doc NOT properly
destroyed");

   lpOutlineApp->m_lpDoc = OutlineApp_CreateDoc(lpOutlineApp, FALSE);
   if (! lpOutlineApp->m_lpDoc) goto error;

   /* OLE2NOTE: initially the Doc object is created with a 0 ref
   **    count. in order to have a stable Doc object during the
   **    process of initializing the new Doc instance,
   **    we intially AddRef the Doc ref cnt and later
   **    Release it. This initial AddRef is artificial; it is simply
   **    done to guarantee that a harmless QueryInterface followed by
   **    a Release does not inadvertantly force our object to destroy
   **    itself prematurely.
   */
   OleDoc_AddRef((LPOLEDOC)lpOutlineApp->m_lpDoc);

   A2W (szFileName, szUniStr, 256);
   fStatus=OutlineDoc_LoadFromFile(lpOutlineApp->m_lpDoc,
(LPOLESTR)szUniStr);

   if (! fStatus) {
      // loading the doc failed; create an untitled instead

      // releasing the artificial AddRef above will destroy the document
      OleDoc_Release((LPOLEDOC)lpOutlineApp->m_lpDoc);

      lpOutlineApp->m_lpDoc = OutlineApp_CreateDoc(lpOutlineApp, FALSE);
      if (! lpOutlineApp->m_lpDoc) goto error;
      OleDoc_AddRef((LPOLEDOC)lpOutlineApp->m_lpDoc);

      if (! OutlineDoc_InitNewFile(lpOutlineApp->m_lpDoc))
         goto error;
   }

   // position and size the new doc window
   OutlineApp_ResizeWindows(lpOutlineApp);
   OutlineDoc_ShowWindow(lpOutlineApp->m_lpDoc);

#if defined( OLE_CNTR )
   UpdateWindow(lpOutlineApp->m_hWndApp);
   ContainerDoc_UpdateLinks((LPCONTAINERDOC)lpOutlineApp->m_lpDoc);
#endif

   OleDoc_Release((LPOLEDOC)lpOutlineApp->m_lpDoc);  // rel artificial
AddRef

   return;

error:
   // REVIEW: should load string from string resource
   OutlineApp_ErrorMessage(lpOutlineApp, OLESTR("Could not create new
document"));

   if (lpOutlineApp->m_lpDoc) {
      // releasing the artificial AddRef above will destroy the document
      OleDoc_Release((LPOLEDOC)lpOutlineApp->m_lpDoc);
      lpOutlineApp->m_lpDoc = NULL;
   }

   return;
}



#if defined( OLE_CNTR )

/* OLE2NOTE: forward the WM_QUERYNEWPALETTE message (via
**    SendMessage) to UIActive in-place object if there is one.
**    this gives the UIActive object the opportunity to select
**    and realize its color palette as the FOREGROUND palette.
**    this is optional for in-place containers. if a container
**    prefers to force its color palette as the foreground
**    palette then it should NOT forward the this message. or
**    the container can give the UIActive object priority; if
**    the UIActive object returns 0 from the WM_QUERYNEWPALETTE
**    message (ie. it did not realize its own palette), then
**    the container can realize its palette.
**    (see ContainerDoc_ForwardPaletteChangedMsg for more info)
**
**    (It is a good idea for containers to use the standard
**    palette even if they do not use colors themselves. this
**    will allow embedded object to get a good distribution of
**    colors when they are being drawn by the container)
**
*/

LRESULT OleApp_QueryNewPalette(LPOLEAPP lpOleApp)
{
#if defined( INPLACE_CNTR )
   LPCONTAINERAPP lpContainerApp = (LPCONTAINERAPP)lpOleApp;

   if (lpContainerApp && lpContainerApp->m_hWndUIActiveObj) {
      if (SendMessage(lpContainerApp->m_hWndUIActiveObj, WM_QUERYNEWPALETTE,
            (WPARAM)0, (LPARAM)0)) {
         /* Object selected its palette as foreground palette */
         return (LRESULT)1;
      }
   }
#endif  // INPLACE_CNTR


   return wSelectPalette(((LPOUTLINEAPP)lpOleApp)->m_hWndApp,
      lpOleApp->m_hStdPal, FALSE/*fBackground*/);
}

#endif // OLE_CNTR



/* This is just a helper routine */

LRESULT wSelectPalette(HWND hWnd, HPALETTE hPal, BOOL fBackground)
{
   HDC hdc;
   HPALETTE hOldPal;
   UINT iPalChg = 0;

   if (hPal == 0)
      return (LRESULT)0;

   hdc = GetDC(hWnd);
   hOldPal = SelectPalette(hdc, hPal, fBackground);
   iPalChg = RealizePalette(hdc);
   SelectPalette(hdc, hOldPal, TRUE /*fBackground*/);
   ReleaseDC(hWnd, hdc);

   if (iPalChg > 0)
```

```
      InvalidateRect(hWnd, NULL, TRUE);

   return (LRESULT)1;
}




/*************************************************************************
** OleApp::IUnknown interface implementation
*************************************************************************/

STDMETHODIMP OleApp_Unk_QueryInterface(
      LPUNKNOWN           lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
)
{
   LPOLEAPP lpOleApp = ((struct CAppUnknownImpl FAR*)lpThis)->lpOleApp;

   return OleApp_QueryInterface(lpOleApp, riid, lplpvObj);
}


STDMETHODIMP_(ULONG) OleApp_Unk_AddRef(LPUNKNOWN lpThis)
{
   LPOLEAPP lpOleApp = ((struct CAppUnknownImpl FAR*)lpThis)->lpOleApp;

   OleDbgAddRefMethod(lpThis, "IUnknown");

   return OleApp_AddRef(lpOleApp);
}


STDMETHODIMP_(ULONG) OleApp_Unk_Release (LPUNKNOWN lpThis)
{
   LPOLEAPP lpOleApp = ((struct CAppUnknownImpl FAR*)lpThis)->lpOleApp;

   OleDbgReleaseMethod(lpThis, "IUnknown");

   return OleApp_Release(lpOleApp);
}




#if defined( OLE_SERVER )

/*************************************************************************
** ServerDoc Supprt Functions Used by Server versions
*************************************************************************/

/* ServerApp_InitInstance
 * ----------------------
 *
 * Initialize the app instance by creating the main frame window and
 * performing app instance specific initializations
 *  (eg. initializing interface Vtbls).
 *
 * RETURNS: TRUE if the memory could be allocated, and the server app
 *                was properly initialized.
 *          FALSE otherwise
 *
 */

BOOL ServerApp_InitInstance(
      LPSERVERAPP            lpServerApp,
      HINSTANCE              hInst,
      int                    nCmdShow
)
{
   LPOLEAPP lpOleApp = (LPOLEAPP)lpServerApp;
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)lpServerApp;

   /* Setup arrays used by IDataObject::EnumFormatEtc.
   **
   ** OLE2NOTE: The order that the formats are listed for GetData is very
   **    significant. It should be listed in order of highest fidelity
   **    formats to least fidelity formats. A common ordering will be:
   **                 1. private app formats
   **                 2. EmbedSource
   **                 3. lower fidelity interchange formats
   **                 4. pictures (metafile, dib, etc.)
   **                     (graphic-related apps offer pictures 1st!)
   **                 5. LinkSource
   */

   /* m_arrDocGetFmts array enumerates the formats that a ServerDoc
   **    DataTransferDoc object can offer (give) through a
   **    IDataObject::GetData call. a ServerDoc DataTransferDoc offers
   **    data formats in the following order:
   **                 1. CF_OUTLINE
   **                 2. CF_EMBEDSOURCE
   **                 3. CF_OBJECTDESCRIPTOR
   **                 4. CF_TEXT
   **                 5. CF_METAFILEPICT
   **                 6. CF_LINKSOURCE *
   **                 7. CF_LINKSRCDESCRIPTOR *
   **
   **    * NOTE: CF_LINKSOURCE and CF_LINKSRCDESCRIPTOR is only
   **    offered if the doc is able to give
   **    a Moniker which references the data. CF_LINKSOURCE is
   **    deliberately listed last in this array of possible formats.
   **    if the doc does not have a Moniker then the last element of
   **    this array is not used. (see SvrDoc_DataObj_EnumFormatEtc).
   **
   **    NOTE: The list of formats that a USER ServerDoc document can
   **    offer is a static list and is registered in the registration
   **    database for the SVROUTL class. The
   **    IDataObject::EnumFormatEtc method returns OLE_S_USEREG in the
   **    case the document is a user docuemt (ie. created via
   **    File.New, File.Open, InsertObject in a container, or
   **    IPersistFile::Load during binding a link source). this tells
   **    OLE to enumerate the formats automatically using the data the
   **    the REGDB.
   */

   lpOleApp->m_arrDocGetFmts[0].cfFormat   = lpOutlineApp->m_cfOutline;
   lpOleApp->m_arrDocGetFmts[0].ptd        = NULL;
   lpOleApp->m_arrDocGetFmts[0].dwAspect   = DVASPECT_CONTENT;
   lpOleApp->m_arrDocGetFmts[0].tymed      = TYMED_HGLOBAL;
   lpOleApp->m_arrDocGetFmts[0].lindex     = -1;

   lpOleApp->m_arrDocGetFmts[1].cfFormat   = lpOleApp->m_cfEmbedSource;
   lpOleApp->m_arrDocGetFmts[1].ptd        = NULL;
   lpOleApp->m_arrDocGetFmts[1].dwAspect   = DVASPECT_CONTENT;
   lpOleApp->m_arrDocGetFmts[1].tymed      = TYMED_ISTORAGE;
   lpOleApp->m_arrDocGetFmts[1].lindex     = -1;

   lpOleApp->m_arrDocGetFmts[2].cfFormat   = CF_TEXT;
   lpOleApp->m_arrDocGetFmts[2].ptd        = NULL;
   lpOleApp->m_arrDocGetFmts[2].dwAspect   = DVASPECT_CONTENT;
   lpOleApp->m_arrDocGetFmts[2].tymed      = TYMED_HGLOBAL;
   lpOleApp->m_arrDocGetFmts[2].lindex     = -1;

   lpOleApp->m_arrDocGetFmts[3].cfFormat   = CF_METAFILEPICT;
   lpOleApp->m_arrDocGetFmts[3].ptd        = NULL;
   lpOleApp->m_arrDocGetFmts[3].dwAspect   = DVASPECT_CONTENT;
   lpOleApp->m_arrDocGetFmts[3].tymed      = TYMED_MFPICT;
   lpOleApp->m_arrDocGetFmts[3].lindex     = -1;

   lpOleApp->m_arrDocGetFmts[4].cfFormat   = lpOleApp->m_cfObjectDescriptor;
   lpOleApp->m_arrDocGetFmts[4].ptd        = NULL;
   lpOleApp->m_arrDocGetFmts[4].dwAspect   = DVASPECT_CONTENT;
   lpOleApp->m_arrDocGetFmts[4].tymed      = TYMED_HGLOBAL;
   lpOleApp->m_arrDocGetFmts[4].lindex     = -1;

   lpOleApp->m_arrDocGetFmts[5].cfFormat   = lpOleApp->m_cfLinkSource;
   lpOleApp->m_arrDocGetFmts[5].ptd        = NULL;
   lpOleApp->m_arrDocGetFmts[5].dwAspect   = DVASPECT_CONTENT;
   lpOleApp->m_arrDocGetFmts[5].tymed      = TYMED_ISTREAM;
   lpOleApp->m_arrDocGetFmts[5].lindex     = -1;

   lpOleApp->m_arrDocGetFmts[6].cfFormat   = lpOleApp-
>m_cfLinkSrcDescriptor;
   lpOleApp->m_arrDocGetFmts[6].ptd        = NULL;
   lpOleApp->m_arrDocGetFmts[6].dwAspect   = DVASPECT_CONTENT;
   lpOleApp->m_arrDocGetFmts[6].tymed      = TYMED_HGLOBAL;
   lpOleApp->m_arrDocGetFmts[6].lindex     = -1;

   lpOleApp->m_nDocGetFmts = 7;

   /* m_arrPasteEntries array enumerates the formats that a ServerDoc
   **    object can accept (get) from the clipboard.
   **    The formats are listed in priority order.
   **    ServerDoc accept data formats in the following order:
   **                  1. CF_OUTLINE
   **                  2. CF_TEXT
   */
   // REVIEW: strings should be loaded from string resource
   lpOleApp->m_arrPasteEntries[0].fmtetc.cfFormat =lpOutlineApp-
>m_cfOutline;
   lpOleApp->m_arrPasteEntries[0].fmtetc.ptd      = NULL;
   lpOleApp->m_arrPasteEntries[0].fmtetc.dwAspect = DVASPECT_CONTENT;
   lpOleApp->m_arrPasteEntries[0].fmtetc.tymed    = TYMED_HGLOBAL;
   lpOleApp->m_arrPasteEntries[0].fmtetc.lindex   = -1;
   lpOleApp->m_arrPasteEntries[0].lpstrFormatName = "Outline Data";
   lpOleApp->m_arrPasteEntries[0].lpstrResultText = "Outline Data";
   lpOleApp->m_arrPasteEntries[0].dwFlags         = OLEUIPASTE_PASTEONLY;

   lpOleApp->m_arrPasteEntries[1].fmtetc.cfFormat = CF_TEXT;
   lpOleApp->m_arrPasteEntries[1].fmtetc.ptd      = NULL;
   lpOleApp->m_arrPasteEntries[1].fmtetc.dwAspect = DVASPECT_CONTENT;
   lpOleApp->m_arrPasteEntries[1].fmtetc.tymed    = TYMED_HGLOBAL;
   lpOleApp->m_arrPasteEntries[1].fmtetc.lindex   = -1;
   lpOleApp->m_arrPasteEntries[1].lpstrFormatName = "Text";
   lpOleApp->m_arrPasteEntries[1].lpstrResultText = "text";
   lpOleApp->m_arrPasteEntries[1].dwFlags         = OLEUIPASTE_PASTEONLY;

   lpOleApp->m_nPasteEntries = 2;

   /**    m_arrLinkTypes array enumerates the link types that a ServerDoc
   **    object can accept from the clipboard. ServerDoc does NOT
   **    accept any type of link from the clipboard. ServerDoc can
   **    only be the source of a link. it can not contain links.
   */

   lpOleApp->m_nLinkTypes = 0;

#if defined( INPLACE_SVR )

   lpServerApp->m_hAccelBaseApp = NULL;
   lpServerApp->m_hAccelIPSvr = LoadAccelerators(
         hInst,
         "InPlaceSvrOutlAccel"
   );

   lpServerApp->m_lpIPData = NULL;

   lpServerApp->m_hMenuEdit = GetSubMenu (
         lpOutlineApp->m_hMenuApp,
         POS_EDITMENU
   );
   lpServerApp->m_hMenuLine = GetSubMenu (
         lpOutlineApp->m_hMenuApp,
         POS_LINEMENU
   );
   lpServerApp->m_hMenuName = GetSubMenu (
         lpOutlineApp->m_hMenuApp,
         POS_NAMEMENU
   );
   lpServerApp->m_hMenuOptions = GetSubMenu (
         lpOutlineApp->m_hMenuApp,
         POS_OPTIONSMENU
   );
   lpServerApp->m_hMenuDebug = GetSubMenu (
         lpOutlineApp->m_hMenuApp,
         POS_DEBUGMENU
   );
   lpServerApp->m_hMenuHelp = GetSubMenu (
         lpOutlineApp->m_hMenuApp,
         POS_HELPMENU
   );

#endif  // INPLACE_SVR

   return TRUE;
}


/* ServerApp_InitVtbls
 * -------------------
 *
 * initialize the methods in all of the interface Vtbl's
 *
 * OLE2NOTE: we only need one copy of each Vtbl. When an object which
 *      exposes an interface is instantiated, its lpVtbl is intialized
 *      to point to the single copy of the Vtbl.
 *
 */
BOOL ServerApp_InitVtbls (LPSERVERAPP lpServerApp)
{
   BOOL fStatus;

   // ServerDoc::IOleObject method table
   OleStdInitVtbl(&g_SvrDoc_OleObjectVtbl, sizeof(IOleObjectVtbl));
   g_SvrDoc_OleObjectVtbl.QueryInterface   = SvrDoc_OleObj_QueryInterface;
   g_SvrDoc_OleObjectVtbl.AddRef           = SvrDoc_OleObj_AddRef;
   g_SvrDoc_OleObjectVtbl.Release          = SvrDoc_OleObj_Release;
   g_SvrDoc_OleObjectVtbl.SetClientSite    = SvrDoc_OleObj_SetClientSite;
   g_SvrDoc_OleObjectVtbl.GetClientSite    = SvrDoc_OleObj_GetClientSite;
   g_SvrDoc_OleObjectVtbl.SetHostNames     = SvrDoc_OleObj_SetHostNames;
   g_SvrDoc_OleObjectVtbl.Close            = SvrDoc_OleObj_Close;
   g_SvrDoc_OleObjectVtbl.SetMoniker       = SvrDoc_OleObj_SetMoniker;
   g_SvrDoc_OleObjectVtbl.GetMoniker       = SvrDoc_OleObj_GetMoniker;
   g_SvrDoc_OleObjectVtbl.InitFromData     = SvrDoc_OleObj_InitFromData;
   g_SvrDoc_OleObjectVtbl.GetClipboardData = SvrDoc_OleObj_GetClipboardData;
   g_SvrDoc_OleObjectVtbl.DoVerb           = SvrDoc_OleObj_DoVerb;
   g_SvrDoc_OleObjectVtbl.EnumVerbs        = SvrDoc_OleObj_EnumVerbs;
   g_SvrDoc_OleObjectVtbl.Update           = SvrDoc_OleObj_Update;
   g_SvrDoc_OleObjectVtbl.IsUpToDate       = SvrDoc_OleObj_IsUpToDate;
   g_SvrDoc_OleObjectVtbl.GetUserClassID   = SvrDoc_OleObj_GetUserClassID;
   g_SvrDoc_OleObjectVtbl.GetUserType      = SvrDoc_OleObj_GetUserType;
   g_SvrDoc_OleObjectVtbl.SetExtent        = SvrDoc_OleObj_SetExtent;
   g_SvrDoc_OleObjectVtbl.GetExtent        = SvrDoc_OleObj_GetExtent;
   g_SvrDoc_OleObjectVtbl.Advise           = SvrDoc_OleObj_Advise;
   g_SvrDoc_OleObjectVtbl.Unadvise         = SvrDoc_OleObj_Unadvise;
   g_SvrDoc_OleObjectVtbl.EnumAdvise       = SvrDoc_OleObj_EnumAdvise;
   g_SvrDoc_OleObjectVtbl.GetMiscStatus    = SvrDoc_OleObj_GetMiscStatus;
   g_SvrDoc_OleObjectVtbl.SetColorScheme   = SvrDoc_OleObj_SetColorScheme;
   fStatus = OleStdCheckVtbl(
         &g_SvrDoc_OleObjectVtbl,
         sizeof(IOleObjectVtbl),
         OLESTR("IOleObject")
      );
   if (! fStatus) return FALSE;

   // ServerDoc::IPersistStorage method table
   OleStdInitVtbl(&g_SvrDoc_PersistStorageVtbl,
sizeof(IPersistStorageVtbl));
   g_SvrDoc_PersistStorageVtbl.QueryInterface  = SvrDoc_PStg_QueryInterface;
   g_SvrDoc_PersistStorageVtbl.AddRef          = SvrDoc_PStg_AddRef;
   g_SvrDoc_PersistStorageVtbl.Release         = SvrDoc_PStg_Release;
   g_SvrDoc_PersistStorageVtbl.GetClassID      = SvrDoc_PStg_GetClassID;
   g_SvrDoc_PersistStorageVtbl.IsDirty         = SvrDoc_PStg_IsDirty;
   g_SvrDoc_PersistStorageVtbl.InitNew         = SvrDoc_PStg_InitNew;
   g_SvrDoc_PersistStorageVtbl.Load            = SvrDoc_PStg_Load;
   g_SvrDoc_PersistStorageVtbl.Save            = SvrDoc_PStg_Save;
   g_SvrDoc_PersistStorageVtbl.SaveCompleted   = SvrDoc_PStg_SaveCompleted;
   g_SvrDoc_PersistStorageVtbl.HandsOffStorage =
SvrDoc_PStg_HandsOffStorage;
   fStatus = OleStdCheckVtbl(
         &g_SvrDoc_PersistStorageVtbl,
         sizeof(IPersistStorageVtbl),
         OLESTR("IPersistStorage")
      );
   if (! fStatus) return FALSE;

#if defined( SVR_TREATAS )
   // ServerDoc::IStdMarshalInfo method table
   OleStdInitVtbl(
         &g_SvrDoc_StdMarshalInfoVtbl, sizeof(IStdMarshalInfoVtbl));
   g_SvrDoc_StdMarshalInfoVtbl.QueryInterface  =
                              SvrDoc_StdMshl_QueryInterface;
   g_SvrDoc_StdMarshalInfoVtbl.AddRef          = SvrDoc_StdMshl_AddRef;
   g_SvrDoc_StdMarshalInfoVtbl.Release         = SvrDoc_StdMshl_Release;
   g_SvrDoc_StdMarshalInfoVtbl.GetClassForHandler =
                              SvrDoc_StdMshl_GetClassForHandler;
   fStatus = OleStdCheckVtbl(
         &g_SvrDoc_StdMarshalInfoVtbl,
         sizeof(IStdMarshalInfoVtbl),
         OLESTR("IStdMarshalInfo")
      );
   if (! fStatus) return FALSE;
#endif  // SVR_TREATAS

#if defined( INPLACE_SVR )
   // ServerDoc::IOleInPlaceObject method table
   OleStdInitVtbl(
      &g_SvrDoc_OleInPlaceObjectVtbl,
      sizeof(IOleInPlaceObjectVtbl)
   );
   g_SvrDoc_OleInPlaceObjectVtbl.QueryInterface
                  = SvrDoc_IPObj_QueryInterface;
   g_SvrDoc_OleInPlaceObjectVtbl.AddRef
                  = SvrDoc_IPObj_AddRef;
   g_SvrDoc_OleInPlaceObjectVtbl.Release
                  = SvrDoc_IPObj_Release;
   g_SvrDoc_OleInPlaceObjectVtbl.GetWindow
                  = SvrDoc_IPObj_GetWindow;
   g_SvrDoc_OleInPlaceObjectVtbl.ContextSensitiveHelp
                  = SvrDoc_IPObj_ContextSensitiveHelp;
   g_SvrDoc_OleInPlaceObjectVtbl.InPlaceDeactivate
                  = SvrDoc_IPObj_InPlaceDeactivate;
   g_SvrDoc_OleInPlaceObjectVtbl.UIDeactivate
                  = SvrDoc_IPObj_UIDeactivate;
   g_SvrDoc_OleInPlaceObjectVtbl.SetObjectRects
                  = SvrDoc_IPObj_SetObjectRects;
   g_SvrDoc_OleInPlaceObjectVtbl.ReactivateAndUndo
                  = SvrDoc_IPObj_ReactivateAndUndo;
   fStatus = OleStdCheckVtbl(
         &g_SvrDoc_OleInPlaceObjectVtbl,
         sizeof(IOleInPlaceObjectVtbl),
         OLESTR("IOleInPlaceObject")
      );
   if (! fStatus) return FALSE;

   // ServerDoc::IOleInPlaceActiveObject method table
   OleStdInitVtbl(
      &g_SvrDoc_OleInPlaceActiveObjectVtbl,
      sizeof(IOleInPlaceActiveObjectVtbl)
   );
   g_SvrDoc_OleInPlaceActiveObjectVtbl.QueryInterface
                  = SvrDoc_IPActiveObj_QueryInterface;
   g_SvrDoc_OleInPlaceActiveObjectVtbl.AddRef
                  = SvrDoc_IPActiveObj_AddRef;
   g_SvrDoc_OleInPlaceActiveObjectVtbl.Release
                  = SvrDoc_IPActiveObj_Release;
   g_SvrDoc_OleInPlaceActiveObjectVtbl.GetWindow
                  = SvrDoc_IPActiveObj_GetWindow;
   g_SvrDoc_OleInPlaceActiveObjectVtbl.ContextSensitiveHelp
                  = SvrDoc_IPActiveObj_ContextSensitiveHelp;
   g_SvrDoc_OleInPlaceActiveObjectVtbl.TranslateAccelerator
                  = SvrDoc_IPActiveObj_TranslateAccelerator;
   g_SvrDoc_OleInPlaceActiveObjectVtbl.OnFrameWindowActivate
                  = SvrDoc_IPActiveObj_OnFrameWindowActivate;
   g_SvrDoc_OleInPlaceActiveObjectVtbl.OnDocWindowActivate
                  = SvrDoc_IPActiveObj_OnDocWindowActivate;
   g_SvrDoc_OleInPlaceActiveObjectVtbl.ResizeBorder
                  = SvrDoc_IPActiveObj_ResizeBorder;
   g_SvrDoc_OleInPlaceActiveObjectVtbl.EnableModeless
                  = SvrDoc_IPActiveObj_EnableModeless;
   fStatus = OleStdCheckVtbl(
         &g_SvrDoc_OleInPlaceActiveObjectVtbl,
         sizeof(IOleInPlaceActiveObjectVtbl),
        OLESTR("IOleInPlaceActiveObject")
      );
   if (! fStatus) return FALSE;

#endif


   // PseudoObj::IUnknown method table
   OleStdInitVtbl(&g_PseudoObj_UnknownVtbl, sizeof(IUnknownVtbl));
   g_PseudoObj_UnknownVtbl.QueryInterface  = PseudoObj_Unk_QueryInterface;
   g_PseudoObj_UnknownVtbl.AddRef          = PseudoObj_Unk_AddRef;
   g_PseudoObj_UnknownVtbl.Release         = PseudoObj_Unk_Release;
   fStatus = OleStdCheckVtbl(
         &g_PseudoObj_UnknownVtbl,
         sizeof(IUnknownVtbl),
         OLESTR("IUnknown")
      );
   if (! fStatus) return FALSE;

   // PseudoObj::IOleObject method table
   OleStdInitVtbl(&g_PseudoObj_OleObjectVtbl, sizeof(IOleObjectVtbl));
   g_PseudoObj_OleObjectVtbl.QueryInterface=
PseudoObj_OleObj_QueryInterface;
   g_PseudoObj_OleObjectVtbl.AddRef        = PseudoObj_OleObj_AddRef;
   g_PseudoObj_OleObjectVtbl.Release       = PseudoObj_OleObj_Release;
   g_PseudoObj_OleObjectVtbl.SetClientSite = PseudoObj_OleObj_SetClientSite;
   g_PseudoObj_OleObjectVtbl.GetClientSite = PseudoObj_OleObj_GetClientSite;
   g_PseudoObj_OleObjectVtbl.SetHostNames  = PseudoObj_OleObj_SetHostNames;
   g_PseudoObj_OleObjectVtbl.Close         = PseudoObj_OleObj_Close;
   g_PseudoObj_OleObjectVtbl.SetMoniker    = PseudoObj_OleObj_SetMoniker;
   g_PseudoObj_OleObjectVtbl.GetMoniker    = PseudoObj_OleObj_GetMoniker;
   g_PseudoObj_OleObjectVtbl.InitFromData  = PseudoObj_OleObj_InitFromData;
   g_PseudoObj_OleObjectVtbl.GetClipboardData =
                              PseudoObj_OleObj_GetClipboardData;
   g_PseudoObj_OleObjectVtbl.DoVerb        = PseudoObj_OleObj_DoVerb;
   g_PseudoObj_OleObjectVtbl.EnumVerbs     = PseudoObj_OleObj_EnumVerbs;
   g_PseudoObj_OleObjectVtbl.Update        = PseudoObj_OleObj_Update;
   g_PseudoObj_OleObjectVtbl.IsUpToDate    = PseudoObj_OleObj_IsUpToDate;
   g_PseudoObj_OleObjectVtbl.GetUserType   = PseudoObj_OleObj_GetUserType;
   g_PseudoObj_OleObjectVtbl.GetUserClassID=
PseudoObj_OleObj_GetUserClassID;
   g_PseudoObj_OleObjectVtbl.SetExtent     = PseudoObj_OleObj_SetExtent;
   g_PseudoObj_OleObjectVtbl.GetExtent     = PseudoObj_OleObj_GetExtent;
   g_PseudoObj_OleObjectVtbl.Advise        = PseudoObj_OleObj_Advise;
   g_PseudoObj_OleObjectVtbl.Unadvise      = PseudoObj_OleObj_Unadvise;
   g_PseudoObj_OleObjectVtbl.EnumAdvise    = PseudoObj_OleObj_EnumAdvise;
   g_PseudoObj_OleObjectVtbl.GetMiscStatus = PseudoObj_OleObj_GetMiscStatus;
   g_PseudoObj_OleObjectVtbl.SetColorScheme=
PseudoObj_OleObj_SetColorScheme;
   fStatus = OleStdCheckVtbl(
         &g_PseudoObj_OleObjectVtbl,
         sizeof(IOleObjectVtbl),
         OLESTR("IOleObject")
      );
   if (! fStatus) return FALSE;
   // ServerDoc::IDataObject method table
   OleStdInitVtbl(&g_PseudoObj_DataObjectVtbl, sizeof(IDataObjectVtbl));
   g_PseudoObj_DataObjectVtbl.QueryInterface =
                        PseudoObj_DataObj_QueryInterface;
   g_PseudoObj_DataObjectVtbl.AddRef       = PseudoObj_DataObj_AddRef;
   g_PseudoObj_DataObjectVtbl.Release      = PseudoObj_DataObj_Release;
   g_PseudoObj_DataObjectVtbl.GetData      = PseudoObj_DataObj_GetData;
   g_PseudoObj_DataObjectVtbl.GetDataHere  = PseudoObj_DataObj_GetDataHere;
   g_PseudoObj_DataObjectVtbl.QueryGetData = PseudoObj_DataObj_QueryGetData;
   g_PseudoObj_DataObjectVtbl.GetCanonicalFormatEtc =
                        PseudoObj_DataObj_GetCanonicalFormatEtc;
   g_PseudoObj_DataObjectVtbl.SetData      = PseudoObj_DataObj_SetData;
   g_PseudoObj_DataObjectVtbl.EnumFormatEtc=
PseudoObj_DataObj_EnumFormatEtc;
   g_PseudoObj_DataObjectVtbl.DAdvise       = PseudoObj_DataObj_DAdvise;
   g_PseudoObj_DataObjectVtbl.DUnadvise     = PseudoObj_DataObj_DUnadvise;
   g_PseudoObj_DataObjectVtbl.EnumDAdvise   = PseudoObj_DataObj_EnumAdvise;

   fStatus = OleStdCheckVtbl(
         &g_PseudoObj_DataObjectVtbl,
         sizeof(IDataObjectVtbl),
         OLESTR("IDataObject")
      );
   if (! fStatus) return FALSE;

   return TRUE;
}

#endif  // OLE_SERVER



#if defined( OLE_CNTR )

/*************************************************************************
** ContainerDoc Supprt Functions Used by Container versions
*************************************************************************/


/* ContainerApp_InitInstance
 * -------------------------
 *
 * Initialize the app instance by creating the main frame window and
 * performing app instance specific initializations
 *  (eg. initializing interface Vtbls).
 *
 * RETURNS: TRUE if the memory could be allocated, and the server app
 *               was properly initialized.
 *          FALSE otherwise
 *
 */

BOOL ContainerApp_InitInstance(
      LPCONTAINERAPP          lpContainerApp,
      HINSTANCE               hInst,
      int                     nCmdShow
)
{
   LPOLEAPP lpOleApp = (LPOLEAPP)lpContainerApp;
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)lpContainerApp;
   char  szAnsiString[256];

   W2A (CONTAINERDOCFORMAT, szAnsiString, 256);
   lpContainerApp->m_cfCntrOutl=RegisterClipboardFormat(szAnsiString);
   if(! lpContainerApp->m_cfCntrOutl) {
      // REVIEW: should load string from string resource
      OutlineApp_ErrorMessage(lpOutlineApp, OLESTR("Can't register clipboard 
format!"));
      return FALSE;
   }

#if defined( INPLACE_CNTR )

   lpContainerApp->m_fPendingUIDeactivate  = FALSE;
   lpContainerApp->m_fMustResizeClientArea = FALSE;
   lpContainerApp->m_lpIPActiveObj         = NULL;
   lpContainerApp->m_hWndUIActiveObj       = NULL;
   lpContainerApp->m_hAccelIPCntr = LoadAccelerators(
         hInst,
         "InPlaceCntrOutlAccel"
   );
   lpContainerApp->m_hMenuFile = GetSubMenu (
         lpOutlineApp->m_hMenuApp,
         POS_FILEMENU
   );
   lpContainerApp->m_hMenuView = GetSubMenu (
         lpOutlineApp->m_hMenuApp,
         POS_VIEWMENU
   );
   lpContainerApp->m_hMenuDebug = GetSubMenu (
         lpOutlineApp->m_hMenuApp,
         POS_DEBUGMENU
   );

   INIT_INTERFACEIMPL(
         &lpContainerApp->m_OleInPlaceFrame,
         &g_CntrApp_OleInPlaceFrameVtbl,
         lpContainerApp
   );

#endif

   /* Setup arrays used by IDataObject::EnumFormatEtc. This is used to
   **    support copy/paste and drag/drop operations.
   **
   ** OLE2NOTE: The order that the formats are listed for GetData is very
   **    significant. It should be listed in order of highest fidelity
   **    formats to least fidelity formats. A common ordering will be:
   **                  1. private app formats
   **                   2. CF_EMBEDSOURCE or CF_EMBEDOBJECT (as appropriate)
   **                   3. lower fidelity interchange formats
   **                   4. CF_METAFILEPICT
   **                       (graphic-related apps might offer picture 1st!)
   **                   5. CF_OBJECTDESCRIPTOR
   **                   6. CF_LINKSOURCE
   **                   6. CF_LINKSRCDESCRIPTOR
   */

   /* m_arrDocGetFmts array enumerates the formats that a ContainerDoc
   **    object can offer (give) through a IDataObject::GetData call
   **    when the selection copied is NOT a single embedded object.
   **    when a single embedded object this list of formats available
   **    is built dynamically depending on the object copied. (see
   **    ContainerDoc_SetupDocGetFmts).
   **    The formats are listed in priority order.
   **    ContainerDoc objects accept data formats in the following order:
   **                   1. CF_CNTROUTL
   **                   2. CF_OUTLINE
   **                   3. CF_TEXT
   **                   4. CF_OBJECTDESCRIPTOR
   **
   **    OLE2NOTE: CF_OBJECTDESCRIPTOR format is used to describe the
   **    data on the clipboard. this information is intended to be
   **    used, for example, to drive the PasteSpecial dialog. it is
   **    useful to render CF_OBJECTDESCRIPTOR format even when the
   **    data on the clipboard does NOT include CF_EMBEDDEDOBJECT
   **    format or CF_EMBEDSOURCE format as when a selection that is
   **    not a single OLE object is copied from the container only
   **    version CNTROUTL. by rendering CF_OBJECTDESCRIPTOR format the
   **    app can indicate a useful string to identifiy the source of
   **    the copy to the user.
   */

   lpOleApp->m_arrDocGetFmts[0].cfFormat = lpContainerApp->m_cfCntrOutl;
   lpOleApp->m_arrDocGetFmts[0].ptd      = NULL;
   lpOleApp->m_arrDocGetFmts[0].dwAspect = DVASPECT_CONTENT;
   lpOleApp->m_arrDocGetFmts[0].tymed    = TYMED_ISTORAGE;
   lpOleApp->m_arrDocGetFmts[0].lindex   = -1;

   lpOleApp->m_arrDocGetFmts[1].cfFormat = lpOutlineApp->m_cfOutline;
   lpOleApp->m_arrDocGetFmts[1].ptd      = NULL;
   lpOleApp->m_arrDocGetFmts[1].dwAspect = DVASPECT_CONTENT;
   lpOleApp->m_arrDocGetFmts[1].tymed    = TYMED_HGLOBAL;
   lpOleApp->m_arrDocGetFmts[1].lindex   = -1;

   lpOleApp->m_arrDocGetFmts[2].cfFormat = CF_TEXT;
   lpOleApp->m_arrDocGetFmts[2].ptd      = NULL;
   lpOleApp->m_arrDocGetFmts[2].dwAspect = DVASPECT_CONTENT;
   lpOleApp->m_arrDocGetFmts[2].tymed    = TYMED_HGLOBAL;
   lpOleApp->m_arrDocGetFmts[2].lindex   = -1;

   lpOleApp->m_arrDocGetFmts[3].cfFormat = lpOleApp->m_cfObjectDescriptor;
   lpOleApp->m_arrDocGetFmts[3].ptd      = NULL;
   lpOleApp->m_arrDocGetFmts[3].dwAspect = DVASPECT_CONTENT;
```

```
lpOleApp->m_arrDocGetFmts[3].tymed    = TYMED_HGLOBAL;
lpOleApp->m_arrDocGetFmts[3].lindex   = -1;

lpOleApp->m_nDocGetFmts = 4;

/* m_arrSingleObjGetFmts array enumerates the formats that a
**    ContainerDoc object can offer (give) through a
**    IDataObject::GetData call when the selection copied IS a
**    single OLE object.
**    ContainerDoc objects accept data formats in the following order:
**                 1. CF_CNTROUTL
**                 2. CF_EMBEDDEDOBJECT
**                 3. CF_OBJECTDESCRIPTOR
**                 4. CF_METAFILEPICT  (note DVASPECT will vary)
**                 5. CF_LINKSOURCE *
**                 6. CF_LINKSRCDESCRIPTOR *
**
**    * OLE2NOTE: CF_LINKSOURCE and CF_LINKSRCDESCRIPTOR is only
**    offered if the OLE object is allowed to be linked to from the
**    inside (ie. we are allowed to give out a moniker which binds
**    to the running OLE object), then we want to offer
**    CF_LINKSOURCE format. if the object is an OLE 2.0 embedded
**    object then it is allowed to be linked to from the inside. if
**    the object is either an OleLink or an OLE 1.0 embedding then
**    it can not be linked to from the inside. if we were a
**    container/server app then we could offer linking to the
**    outside of the object (ie. a pseudo object within our
**    document). we are a container only app that does not support
**    linking to ranges of its data.
**    the simplest way to determine if an object can be linked to
**    on the inside is to call IOleObject::GetMiscStatus and test
**    to see if the OLEMISC_CANTLINKINSIDE bit is NOT set.
**
**    OLE2NOTE: optionally, a container that wants to have a
**    potentially richer data transfer, can enumerate the data
**    formats from the OLE object's cache and offer them too. if
**    the object has a special handler, then it might be able to
**    render additional data formats.
*/
lpContainerApp->m_arrSingleObjGetFmts[0].cfFormat =
                                 lpContainerApp->m_cfCntrOutl;
lpContainerApp->m_arrSingleObjGetFmts[0].ptd      = NULL;
lpContainerApp->m_arrSingleObjGetFmts[0].dwAspect = DVASPECT_CONTENT;
lpContainerApp->m_arrSingleObjGetFmts[0].tymed    = TYMED_ISTORAGE;
lpContainerApp->m_arrSingleObjGetFmts[0].lindex   = -1;

lpContainerApp->m_arrSingleObjGetFmts[1].cfFormat =
                                 lpOleApp->m_cfEmbeddedObject;
lpContainerApp->m_arrSingleObjGetFmts[1].ptd      = NULL;
lpContainerApp->m_arrSingleObjGetFmts[1].dwAspect = DVASPECT_CONTENT;
lpContainerApp->m_arrSingleObjGetFmts[1].tymed    = TYMED_ISTORAGE;
lpContainerApp->m_arrSingleObjGetFmts[1].lindex   = -1;

lpContainerApp->m_arrSingleObjGetFmts[2].cfFormat =
                                 lpOleApp->m_cfObjectDescriptor;
```

```
lpContainerApp->m_arrSingleObjGetFmts[2].ptd      = NULL;
lpContainerApp->m_arrSingleObjGetFmts[2].dwAspect = DVASPECT_CONTENT;
lpContainerApp->m_arrSingleObjGetFmts[2].tymed    = TYMED_HGLOBAL;
lpContainerApp->m_arrSingleObjGetFmts[2].lindex   = -1;

lpContainerApp->m_arrSingleObjGetFmts[3].cfFormat = CF_METAFILEPICT;
lpContainerApp->m_arrSingleObjGetFmts[3].ptd      = NULL;
lpContainerApp->m_arrSingleObjGetFmts[3].dwAspect = DVASPECT_CONTENT;
lpContainerApp->m_arrSingleObjGetFmts[3].tymed    = TYMED_MFPICT;
lpContainerApp->m_arrSingleObjGetFmts[3].lindex   = -1;

lpContainerApp->m_arrSingleObjGetFmts[4].cfFormat =
                                    lpOleApp->m_cfLinkSource;
lpContainerApp->m_arrSingleObjGetFmts[4].ptd      = NULL;
lpContainerApp->m_arrSingleObjGetFmts[4].dwAspect = DVASPECT_CONTENT;
lpContainerApp->m_arrSingleObjGetFmts[4].tymed    = TYMED_ISTREAM;
lpContainerApp->m_arrSingleObjGetFmts[4].lindex   = -1;

lpContainerApp->m_arrSingleObjGetFmts[5].cfFormat =
                                lpOleApp->m_cfLinkSrcDescriptor;
lpContainerApp->m_arrSingleObjGetFmts[5].ptd      = NULL;
lpContainerApp->m_arrSingleObjGetFmts[5].dwAspect = DVASPECT_CONTENT;
lpContainerApp->m_arrSingleObjGetFmts[5].tymed    = TYMED_HGLOBAL;
lpContainerApp->m_arrSingleObjGetFmts[5].lindex   = -1;

lpContainerApp->m_nSingleObjGetFmts = 6;

/* NOTE: the Container-Only version of Outline does NOT offer
**    IDataObject interface from its User documents and the
**    IDataObject interface available from DataTransferDoc's do NOT
**    support SetData. IDataObject interface is required by objects
**    which can be embedded or linked. the Container-only app only
**    allows linking to its contained objects, NOT the data of the
**    container itself.
*/

/*    m_arrPasteEntries array enumerates the formats that a ContainerDoc
**    object can accept from the clipboard. this array is used to
**    support the PasteSpecial dialog.
**    The formats are listed in priority order.
**    ContainerDoc objects accept data formats in the following order:
**                  1. CF_CNTROUTL
**                  2. CF_OUTLINE
**                  3. CF_EMBEDDEDOBJECT
**                  4. CF_TEXT
**                  5. CF_METAFILEPICT
**                  6. CF_DIB
**                  7. CF_BITMAP
**                  8. CF_LINKSOURCE
**
**    NOTE: specifying CF_EMBEDDEDOBJECT in the PasteEntry array
**    indicates that the caller is interested in pasting OLE
**    objects (ie. the caller calls OleCreateFromData). the
**    OleUIPasteSpecial dialog and OleStdGetPriorityClipboardFormat
**    call OleQueryCreateFromData to see if an OLE object format is
```

```
   **    available. thus, in fact if CF_EMBEDSOURCE or CF_FILENAME are
   **    available from the data source then and OLE object can be
   **    created and this entry will be matched. the caller should
   **    only specify one object type format.
   **    CF_FILENAME format (as generated by copying a file to
   **    the clipboard from the FileManager) is considered an object
   **    format; OleCreatFromData creates an object if the file has an
   **    associated class (see GetClassFile API) or if no class it
   **    creates an OLE 1.0 Package object. this format can also be
   **    paste linked by calling OleCreateLinkFromData.
   */
   // REVIEW: strings should be loaded from string resource

   lpOleApp->m_arrPasteEntries[0].fmtetc.cfFormat =
                        lpContainerApp->m_cfCntrOutl;
   lpOleApp->m_arrPasteEntries[0].fmtetc.ptd      = NULL;
   lpOleApp->m_arrPasteEntries[0].fmtetc.dwAspect = DVASPECT_CONTENT;
   lpOleApp->m_arrPasteEntries[0].fmtetc.tymed    = TYMED_ISTORAGE;
   lpOleApp->m_arrPasteEntries[0].fmtetc.lindex   = -1;
   lpOleApp->m_arrPasteEntries[0].lpstrFormatName = "Container Outline
Data";
   lpOleApp->m_arrPasteEntries[0].lpstrResultText = "Container Outline
Data";
   lpOleApp->m_arrPasteEntries[0].dwFlags         = OLEUIPASTE_PASTEONLY;

   lpOleApp->m_arrPasteEntries[1].fmtetc.cfFormat =lpOutlineApp-
>m_cfOutline;
   lpOleApp->m_arrPasteEntries[1].fmtetc.ptd      = NULL;
   lpOleApp->m_arrPasteEntries[1].fmtetc.dwAspect = DVASPECT_CONTENT;
   lpOleApp->m_arrPasteEntries[1].fmtetc.tymed    = TYMED_HGLOBAL;
   lpOleApp->m_arrPasteEntries[1].fmtetc.lindex   = -1;
   lpOleApp->m_arrPasteEntries[1].lpstrFormatName = "Outline Data";
   lpOleApp->m_arrPasteEntries[1].lpstrResultText = "Outline Data";
   lpOleApp->m_arrPasteEntries[1].dwFlags         = OLEUIPASTE_PASTEONLY;

   lpOleApp->m_arrPasteEntries[2].fmtetc.cfFormat =
                        lpOleApp->m_cfEmbeddedObject;
   lpOleApp->m_arrPasteEntries[2].fmtetc.ptd      = NULL;
   lpOleApp->m_arrPasteEntries[2].fmtetc.dwAspect = DVASPECT_CONTENT;
   lpOleApp->m_arrPasteEntries[2].fmtetc.tymed    = TYMED_ISTORAGE;
   lpOleApp->m_arrPasteEntries[2].fmtetc.lindex   = -1;
   lpOleApp->m_arrPasteEntries[2].lpstrFormatName = "%s";
   lpOleApp->m_arrPasteEntries[2].lpstrResultText = "%s";
   lpOleApp->m_arrPasteEntries[2].dwFlags         =
                        OLEUIPASTE_PASTE | OLEUIPASTE_ENABLEICON;

   lpOleApp->m_arrPasteEntries[3].fmtetc.cfFormat = CF_TEXT;
   lpOleApp->m_arrPasteEntries[3].fmtetc.ptd      = NULL;
   lpOleApp->m_arrPasteEntries[3].fmtetc.dwAspect = DVASPECT_CONTENT;
   lpOleApp->m_arrPasteEntries[3].fmtetc.tymed    = TYMED_HGLOBAL;
   lpOleApp->m_arrPasteEntries[3].fmtetc.lindex   = -1;
   lpOleApp->m_arrPasteEntries[3].lpstrFormatName = "Text";
   lpOleApp->m_arrPasteEntries[3].lpstrResultText = "text";
   lpOleApp->m_arrPasteEntries[3].dwFlags         = OLEUIPASTE_PASTEONLY;
   lpOleApp->m_arrPasteEntries[4].fmtetc.cfFormat = CF_METAFILEPICT;
   lpOleApp->m_arrPasteEntries[4].fmtetc.ptd      = NULL;
   lpOleApp->m_arrPasteEntries[4].fmtetc.dwAspect = DVASPECT_CONTENT;
   lpOleApp->m_arrPasteEntries[4].fmtetc.tymed    = TYMED_MFPICT;
   lpOleApp->m_arrPasteEntries[4].fmtetc.lindex   = -1;
   lpOleApp->m_arrPasteEntries[4].lpstrFormatName = "Picture (Metafile)";
   lpOleApp->m_arrPasteEntries[4].lpstrResultText = "a static picture";
   lpOleApp->m_arrPasteEntries[4].dwFlags         = OLEUIPASTE_PASTEONLY;

   lpOleApp->m_arrPasteEntries[5].fmtetc.cfFormat = CF_DIB;
   lpOleApp->m_arrPasteEntries[5].fmtetc.ptd      = NULL;
   lpOleApp->m_arrPasteEntries[5].fmtetc.dwAspect = DVASPECT_CONTENT;
   lpOleApp->m_arrPasteEntries[5].fmtetc.tymed    = TYMED_HGLOBAL;
   lpOleApp->m_arrPasteEntries[5].fmtetc.lindex   = -1;
   lpOleApp->m_arrPasteEntries[5].lpstrFormatName = "Picture (DIB)";
   lpOleApp->m_arrPasteEntries[5].lpstrResultText = "a static picture";
   lpOleApp->m_arrPasteEntries[5].dwFlags         = OLEUIPASTE_PASTEONLY;

   lpOleApp->m_arrPasteEntries[6].fmtetc.cfFormat = CF_BITMAP;
   lpOleApp->m_arrPasteEntries[6].fmtetc.ptd      = NULL;
   lpOleApp->m_arrPasteEntries[6].fmtetc.dwAspect = DVASPECT_CONTENT;
   lpOleApp->m_arrPasteEntries[6].fmtetc.tymed    = TYMED_GDI;
   lpOleApp->m_arrPasteEntries[6].fmtetc.lindex   = -1;
   lpOleApp->m_arrPasteEntries[6].lpstrFormatName = "Picture (Bitmap)";
   lpOleApp->m_arrPasteEntries[6].lpstrResultText = "a static picture";
   lpOleApp->m_arrPasteEntries[6].dwFlags         = OLEUIPASTE_PASTEONLY;

   lpOleApp->m_arrPasteEntries[7].fmtetc.cfFormat = lpOleApp-
>m_cfLinkSource;
   lpOleApp->m_arrPasteEntries[7].fmtetc.ptd      = NULL;
   lpOleApp->m_arrPasteEntries[7].fmtetc.dwAspect = DVASPECT_CONTENT;
   lpOleApp->m_arrPasteEntries[7].fmtetc.tymed    = TYMED_ISTREAM;
   lpOleApp->m_arrPasteEntries[7].fmtetc.lindex   = -1;
   lpOleApp->m_arrPasteEntries[7].lpstrFormatName = "%s";
   lpOleApp->m_arrPasteEntries[7].lpstrResultText = "%s";
   lpOleApp->m_arrPasteEntries[7].dwFlags         =
                        OLEUIPASTE_LINKTYPE1 | OLEUIPASTE_ENABLEICON;

   lpOleApp->m_nPasteEntries = 8;

   /*    m_arrLinkTypes array enumerates the link types that a ContainerDoc
   **    object can accept from the clipboard
   */

   lpOleApp->m_arrLinkTypes[0] = lpOleApp->m_cfLinkSource;
   lpOleApp->m_nLinkTypes = 1;

   return TRUE;
}


/* ContainerApp_InitVtbls
** ----------------------
**
**    initialize the interface Vtbl's used to support the OLE 2.0
**    Container functionality.
*/

BOOL ContainerApp_InitVtbls(LPCONTAINERAPP lpApp)
{
   BOOL fStatus;

   // ContainerDoc::IOleUILinkContainer method table
   OleStdInitVtbl(
         &g_CntrDoc_OleUILinkContainerVtbl,
         sizeof(IOleUILinkContainerVtbl)
   );
   g_CntrDoc_OleUILinkContainerVtbl.QueryInterface =
                                 CntrDoc_LinkCont_QueryInterface;
   g_CntrDoc_OleUILinkContainerVtbl.AddRef     = CntrDoc_LinkCont_AddRef;
   g_CntrDoc_OleUILinkContainerVtbl.Release    = CntrDoc_LinkCont_Release;
   g_CntrDoc_OleUILinkContainerVtbl.GetNextLink =
                              CntrDoc_LinkCont_GetNextLink;
   g_CntrDoc_OleUILinkContainerVtbl.SetLinkUpdateOptions =
                              CntrDoc_LinkCont_SetLinkUpdateOptions;
   g_CntrDoc_OleUILinkContainerVtbl.GetLinkUpdateOptions =
                              CntrDoc_LinkCont_GetLinkUpdateOptions;
   g_CntrDoc_OleUILinkContainerVtbl.SetLinkSource =
                              CntrDoc_LinkCont_SetLinkSource;
   g_CntrDoc_OleUILinkContainerVtbl.GetLinkSource =
                              CntrDoc_LinkCont_GetLinkSource;
   g_CntrDoc_OleUILinkContainerVtbl.OpenLinkSource =
                              CntrDoc_LinkCont_OpenLinkSource;
   g_CntrDoc_OleUILinkContainerVtbl.UpdateLink =
                              CntrDoc_LinkCont_UpdateLink;
   g_CntrDoc_OleUILinkContainerVtbl.CancelLink =
                              CntrDoc_LinkCont_CancelLink;
   fStatus = OleStdCheckVtbl(
         &g_CntrDoc_OleUILinkContainerVtbl,
         sizeof(IOleUILinkContainerVtbl),
         OLESTR("IOleUILinkContainer")
      );
   if (! fStatus) return FALSE;

#if defined( INPLACE_CNTR )

   // ContainerApp::IOleInPlaceFrame interface method table
   OleStdInitVtbl(
         &g_CntrApp_OleInPlaceFrameVtbl,
         sizeof(g_CntrApp_OleInPlaceFrameVtbl)
   );

   g_CntrApp_OleInPlaceFrameVtbl.QueryInterface
                  = CntrApp_IPFrame_QueryInterface;
   g_CntrApp_OleInPlaceFrameVtbl.AddRef
                  = CntrApp_IPFrame_AddRef;
   g_CntrApp_OleInPlaceFrameVtbl.Release
                  = CntrApp_IPFrame_Release;
   g_CntrApp_OleInPlaceFrameVtbl.GetWindow
                  = CntrApp_IPFrame_GetWindow;
```

```
    g_CntrApp_OleInPlaceFrameVtbl.ContextSensitiveHelp
                    = CntrApp_IPFrame_ContextSensitiveHelp;

    g_CntrApp_OleInPlaceFrameVtbl.GetBorder
                    = CntrApp_IPFrame_GetBorder;
    g_CntrApp_OleInPlaceFrameVtbl.RequestBorderSpace
                    = CntrApp_IPFrame_RequestBorderSpace;
    g_CntrApp_OleInPlaceFrameVtbl.SetBorderSpace
                    = CntrApp_IPFrame_SetBorderSpace;
    g_CntrApp_OleInPlaceFrameVtbl.SetActiveObject
                    = CntrApp_IPFrame_SetActiveObject;
    g_CntrApp_OleInPlaceFrameVtbl.InsertMenus
                    = CntrApp_IPFrame_InsertMenus;
    g_CntrApp_OleInPlaceFrameVtbl.SetMenu
                    = CntrApp_IPFrame_SetMenu;
    g_CntrApp_OleInPlaceFrameVtbl.RemoveMenus
                    = CntrApp_IPFrame_RemoveMenus;
    g_CntrApp_OleInPlaceFrameVtbl.SetStatusText
                    = CntrApp_IPFrame_SetStatusText;
    g_CntrApp_OleInPlaceFrameVtbl.EnableModeless
                    = CntrApp_IPFrame_EnableModeless;
    g_CntrApp_OleInPlaceFrameVtbl.TranslateAccelerator
                    = CntrApp_IPFrame_TranslateAccelerator;

    fStatus = OleStdCheckVtbl(
          &g_CntrApp_OleInPlaceFrameVtbl,
          sizeof(g_CntrApp_OleInPlaceFrameVtbl),
          OLESTR("IOleInPlaceFrame")
       );
    if (! fStatus) return FALSE;

#endif  // INPLACE_CNTR


    // ContainerLine::IUnknown interface method table
    OleStdInitVtbl(
          &g_CntrLine_UnknownVtbl,
          sizeof(g_CntrLine_UnknownVtbl)
       );
    g_CntrLine_UnknownVtbl.QueryInterface   = CntrLine_Unk_QueryInterface;
    g_CntrLine_UnknownVtbl.AddRef           = CntrLine_Unk_AddRef;
    g_CntrLine_UnknownVtbl.Release          = CntrLine_Unk_Release;
    fStatus = OleStdCheckVtbl(
          &g_CntrLine_UnknownVtbl,
          sizeof(g_CntrLine_UnknownVtbl),
          OLESTR("IUnknown")
       );
    if (! fStatus) return FALSE;

    // ContainerLine::IOleClientSite interface method table
    OleStdInitVtbl(
          &g_CntrLine_OleClientSiteVtbl,
          sizeof(g_CntrLine_OleClientSiteVtbl)
       );
    g_CntrLine_OleClientSiteVtbl.QueryInterface =
                                  CntrLine_CliSite_QueryInterface;
    g_CntrLine_OleClientSiteVtbl.AddRef       = CntrLine_CliSite_AddRef;
    g_CntrLine_OleClientSiteVtbl.Release      = CntrLine_CliSite_Release;
    g_CntrLine_OleClientSiteVtbl.SaveObject   = CntrLine_CliSite_SaveObject;
    g_CntrLine_OleClientSiteVtbl.GetMoniker   = CntrLine_CliSite_GetMoniker;
    g_CntrLine_OleClientSiteVtbl.GetContainer =
 CntrLine_CliSite_GetContainer;
    g_CntrLine_OleClientSiteVtbl.ShowObject   = CntrLine_CliSite_ShowObject;
    g_CntrLine_OleClientSiteVtbl.OnShowWindow =
 CntrLine_CliSite_OnShowWindow;
    g_CntrLine_OleClientSiteVtbl.RequestNewObjectLayout =
                            CntrLine_CliSite_RequestNewObjectLayout;
    fStatus = OleStdCheckVtbl(
          &g_CntrLine_OleClientSiteVtbl,
          sizeof(g_CntrLine_OleClientSiteVtbl),
          OLESTR("IOleClientSite")
       );
    if (! fStatus) return FALSE;

    // ContainerLine::IAdviseSink interface method table
    OleStdInitVtbl(
          &g_CntrLine_AdviseSinkVtbl,
          sizeof(g_CntrLine_AdviseSinkVtbl)
    );
    g_CntrLine_AdviseSinkVtbl.QueryInterface=
 CntrLine_AdvSink_QueryInterface;
    g_CntrLine_AdviseSinkVtbl.AddRef        = CntrLine_AdvSink_AddRef;
    g_CntrLine_AdviseSinkVtbl.Release       = CntrLine_AdvSink_Release;
    g_CntrLine_AdviseSinkVtbl.OnDataChange  = CntrLine_AdvSink_OnDataChange;
    g_CntrLine_AdviseSinkVtbl.OnViewChange  = CntrLine_AdvSink_OnViewChange;
    g_CntrLine_AdviseSinkVtbl.OnRename      = CntrLine_AdvSink_OnRename;
    g_CntrLine_AdviseSinkVtbl.OnSave        = CntrLine_AdvSink_OnSave;
    g_CntrLine_AdviseSinkVtbl.OnClose       = CntrLine_AdvSink_OnClose;
    fStatus = OleStdCheckVtbl(
          &g_CntrLine_AdviseSinkVtbl,
          sizeof(g_CntrLine_AdviseSinkVtbl),
          OLESTR("IAdviseSink")
       );
    if (! fStatus) return FALSE;


 #if defined( INPLACE_CNTR )

    // ContainerLine::IOleInPlaceSite interface method table
    OleStdInitVtbl(
          &g_CntrLine_OleInPlaceSiteVtbl,
          sizeof(g_CntrLine_OleInPlaceSiteVtbl)
    );

    g_CntrLine_OleInPlaceSiteVtbl.QueryInterface
                   = CntrLine_IPSite_QueryInterface;
    g_CntrLine_OleInPlaceSiteVtbl.AddRef
                   = CntrLine_IPSite_AddRef;
    g_CntrLine_OleInPlaceSiteVtbl.Release
                   = CntrLine_IPSite_Release;
    g_CntrLine_OleInPlaceSiteVtbl.GetWindow
                   = CntrLine_IPSite_GetWindow;
    g_CntrLine_OleInPlaceSiteVtbl.ContextSensitiveHelp
                   = CntrLine_IPSite_ContextSensitiveHelp;
    g_CntrLine_OleInPlaceSiteVtbl.CanInPlaceActivate
                   = CntrLine_IPSite_CanInPlaceActivate;
    g_CntrLine_OleInPlaceSiteVtbl.OnInPlaceActivate
                   = CntrLine_IPSite_OnInPlaceActivate;
    g_CntrLine_OleInPlaceSiteVtbl.OnUIActivate
                   = CntrLine_IPSite_OnUIActivate;
    g_CntrLine_OleInPlaceSiteVtbl.GetWindowContext
                   = CntrLine_IPSite_GetWindowContext;
    g_CntrLine_OleInPlaceSiteVtbl.Scroll
                   = CntrLine_IPSite_Scroll;
    g_CntrLine_OleInPlaceSiteVtbl.OnUIDeactivate
                   = CntrLine_IPSite_OnUIDeactivate;

    g_CntrLine_OleInPlaceSiteVtbl.OnInPlaceDeactivate
                   = CntrLine_IPSite_OnInPlaceDeactivate;
    g_CntrLine_OleInPlaceSiteVtbl.DiscardUndoState
                   = CntrLine_IPSite_DiscardUndoState;
    g_CntrLine_OleInPlaceSiteVtbl.DeactivateAndUndo
                   = CntrLine_IPSite_DeactivateAndUndo;
    g_CntrLine_OleInPlaceSiteVtbl.OnPosRectChange
                   = CntrLine_IPSite_OnPosRectChange;

    fStatus = OleStdCheckVtbl(
          &g_CntrLine_OleInPlaceSiteVtbl,
          sizeof(g_CntrLine_OleInPlaceSiteVtbl),
          OLESTR("IOleInPlaceSite")
       );
    if (! fStatus) return FALSE;

 #endif  // INPLACE_CNTR

    return TRUE;
 }


 #endif  // OLE_CNTR
```

OLEDOC.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Server Sample Code
**
**    oledoc.c
**
**    This file contains general OleDoc methods and related support
**    functions. OleDoc implementation is used by both the Container
**    versions and the Server (Object) versions of the Outline Sample.
**
**    This file includes general support for the following:
**    1. show/hide doc window
**    2. QueryInterface, AddRef, Release
**    3. document locking (calls CoLockObjectExternal)
**    4. document shutdown (Close, Destroy)
**    5. clipboard support
**
**    OleDoc Object
**      exposed interfaces:
**          IUnknown
**          IPersistFile
**          IOleItemContainer
**          IDataObject
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/


#include "outline.h"

OLEDBGDATA

extern LPOUTLINEAPP             g_lpApp;

extern IUnknownVtbl             g_OleDoc_UnknownVtbl;
extern IPersistFileVtbl         g_OleDoc_PersistFileVtbl;
extern IOleItemContainerVtbl    g_OleDoc_OleItemContainerVtbl;
extern IExternalConnectionVtbl  g_OleDoc_ExternalConnectionVtbl;
extern IDataObjectVtbl          g_OleDoc_DataObjectVtbl;

#if defined( USE_DRAGDROP )
extern IDropTargetVtbl          g_OleDoc_DropTargetVtbl;
extern IDropSourceVtbl          g_OleDoc_DropSourceVtbl;
#endif  // USE_DRAGDROP

#if defined( INPLACE_CNTR )
extern BOOL g_fInsideOutContainer;
#endif


/* OleDoc_Init
```

```
 * -----------
 *
 *  Initialize the fields of a new OleDoc object. The object is initially
 *  not associated with a file or an (Untitled) document. This function sets
 *  the docInitType to DOCTYPE_UNKNOWN. After calling this function the
 *  caller should call:
 *      1.) Doc_InitNewFile to set the OleDoc to (Untitled)
 *      2.) Doc_LoadFromFile to associate the OleDoc with a file.
 *  This function creates a new window for the document.
 *
 *  NOTE: the window is initially created with a NIL size. it must be
 *        sized and positioned by the caller. also the document is initially
 *        created invisible. the caller must call OutlineDoc_ShowWindow
 *        after sizing it to make the document window visible.
 */
BOOL OleDoc_Init(LPOLEDOC lpOleDoc, BOOL fDataTransferDoc)
{
   LPOLEAPP   lpOleApp = (LPOLEAPP)g_lpApp;
   LPLINELIST lpLL     = (LPLINELIST)&((LPOUTLINEDOC)lpOleDoc)->m_LineList;

   lpOleDoc->m_cRef                        = 0;
   lpOleDoc->m_dwStrongExtConn             = 0;
#if defined( _DEBUG )
   lpOleDoc->m_cCntrLock                   = 0;
#endif
   lpOleDoc->m_lpStg                       = NULL;
   lpOleDoc->m_lpLLStm                     = NULL;
   lpOleDoc->m_lpNTStm                     = NULL;
   lpOleDoc->m_dwRegROT                    = 0;
   lpOleDoc->m_lpFileMoniker               = NULL;
   lpOleDoc->m_fLinkSourceAvail            = FALSE;
   lpOleDoc->m_lpSrcDocOfCopy              = NULL;
   lpOleDoc->m_fObjIsClosing               = FALSE;
   lpOleDoc->m_fObjIsDestroying            = FALSE;
   lpOleDoc->m_fUpdateEditMenu             = FALSE;

#if defined( USE_DRAGDROP )
   lpOleDoc->m_dwTimeEnterScrollArea       = 0L;
   lpOleDoc->m_dwNextScrollTime            = 0L;
   lpOleDoc->m_dwLastScrollDir             = SCROLLDIR_NULL;
   lpOleDoc->m_fRegDragDrop                = FALSE;
   lpOleDoc->m_fLocalDrag                  = FALSE;
   lpOleDoc->m_fCanDropCopy                = FALSE;
   lpOleDoc->m_fCanDropLink                = FALSE;
   lpOleDoc->m_fLocalDrop                  = FALSE;
   lpOleDoc->m_fDragLeave                  = FALSE;
   lpOleDoc->m_fPendingDrag                = FALSE;
#endif
#if defined( INPLACE_SVR ) || defined( INPLACE_CNTR )
   lpOleDoc->m_fCSHelpMode                 = FALSE;    // Shift-F1 context
                                          // sensitive help mode
#endif

   INIT_INTERFACEIMPL(
         &lpOleDoc->m_Unknown,
            &g_OleDoc_UnknownVtbl,
            lpOleDoc
    );

    INIT_INTERFACEIMPL(
            &lpOleDoc->m_PersistFile,
            &g_OleDoc_PersistFileVtbl,
            lpOleDoc
    );

    INIT_INTERFACEIMPL(
            &lpOleDoc->m_OleItemContainer,
            &g_OleDoc_OleItemContainerVtbl,
            lpOleDoc
    );

    INIT_INTERFACEIMPL(
            &lpOleDoc->m_ExternalConnection,
            &g_OleDoc_ExternalConnectionVtbl,
            lpOleDoc
    );

    INIT_INTERFACEIMPL(
            &lpOleDoc->m_DataObject,
            &g_OleDoc_DataObjectVtbl,
            lpOleDoc
    );

#if defined( USE_DRAGDROP )
    INIT_INTERFACEIMPL(
            &lpOleDoc->m_DropSource,
            &g_OleDoc_DropSourceVtbl,
            lpOleDoc
    );

    INIT_INTERFACEIMPL(
            &lpOleDoc->m_DropTarget,
            &g_OleDoc_DropTargetVtbl,
            lpOleDoc
    );
#endif  // USE_DRAGDROP

    /*
    ** OLE2NOTE: each user level document addref's the app object in
    **    order to guarentee that the app does not shut down while the
    **    doc is still open.
    */

    // OLE2NOTE: data transfer documents should not hold the app alive
    if (! fDataTransferDoc)
       OleApp_DocLockApp(lpOleApp);

#if defined( OLE_SERVER )
    /* OLE2NOTE: perform initialization specific for an OLE server */
    if (! ServerDoc_Init((LPSERVERDOC)lpOleDoc, fDataTransferDoc))
```

```
      return FALSE;
#endif
#if defined( OLE_CNTR )

   /* OLE2NOTE: perform initialization specific for an OLE container */
   if (! ContainerDoc_Init((LPCONTAINERDOC)lpOleDoc, fDataTransferDoc))
      return FALSE;
#endif

   return TRUE;
}



/* OleDoc_InitNewFile
 * ------------------
 *
 *  Initialize the document to be a new (Untitled) document.
 *  This function sets the docInitType to DOCTYPE_NEW.
 *
 *  OLE2NOTE: if this is a visible user document then generate a unique
 *  untitled name that we can use to register in the RunningObjectTable.
 *  We need a unique name so that clients can link to data in this document
 *  even when the document is in the un-saved (untitled) state. it would be
 *  ambiguous to register two documents titled "Outline1" in the ROT. we
 *  thus generate the lowest numbered document that is not already
 *  registered in the ROT.
 */
BOOL OleDoc_InitNewFile(LPOLEDOC lpOleDoc)
{
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpOleDoc;

   static UINT uUnique = 1;

   OleDbgAssert(lpOutlineDoc->m_docInitType == DOCTYPE_UNKNOWN);

#if defined( OLE_CNTR )
   {
      LPCONTAINERDOC lpContainerDoc = (LPCONTAINERDOC)lpOleDoc;
#if defined( _DEBUG )
      OleDbgAssertSz(lpOleDoc->m_lpStg == NULL,
            "Setting to untitled with current file open"
      );
#endif

      /* Create a temp, (delete-on-release) file base storage
      **  for the untitled document.
      */
      lpOleDoc->m_lpStg = OleStdCreateRootStorage(
            NULL,
            STGM_SHARE_EXCLUSIVE
      );
      if (! lpOleDoc->m_lpStg) return FALSE;
   }
#endif
```

```
   lpOutlineDoc->m_docInitType = DOCTYPE_NEW;

   if (! lpOutlineDoc->m_fDataTransferDoc) {
      /* OLE2NOTE: choose a unique name for a Moniker so that
      **    potential clients can link to our new, untitled document.
      **    if links are established (and currently are connected),
      **    then they will be notified that we have been renamed when
      **    this document is saved to a file.
      */

      lpOleDoc->m_fLinkSourceAvail = TRUE;

      // REVIEW: should load UNTITLED string from string resource
      OleStdCreateTempFileMoniker(
            UNTITLED,
            (UINT FAR*)&uUnique,
            lpOutlineDoc->m_szFileName,
            &lpOleDoc->m_lpFileMoniker
      );

      OLEDBG_BEGIN3("OleStdRegisterAsRunning called\r\n")
      OleStdRegisterAsRunning(
            (LPUNKNOWN)&lpOleDoc->m_PersistFile,
            (LPMONIKER)lpOleDoc->m_lpFileMoniker,
            &lpOleDoc->m_dwRegROT
      );
      OLEDBG_END3

      lpOutlineDoc->m_lpszDocTitle = lpOutlineDoc->m_szFileName;
      OutlineDoc_SetTitle(lpOutlineDoc, FALSE /*fMakeUpperCase*/);
   } else {
      OLESTRCPY(lpOutlineDoc->m_szFileName, UNTITLED);
      lpOutlineDoc->m_lpszDocTitle = lpOutlineDoc->m_szFileName;
   }

   return TRUE;
}


/* OleDoc_ShowWindow
 * -----------------
 *
 *      Show the window of the document to the user.
 *      make sure app window is visible and bring the document to the top.
 *      if the document is a file-based document or a new untitled
 *      document, give the user the control over the life-time of the doc.
 */
void OleDoc_ShowWindow(LPOLEDOC lpOleDoc)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpOleDoc;
   LPLINELIST lpLL     = (LPLINELIST)&((LPOUTLINEDOC)lpOleDoc)->m_LineList;
#if defined( OLE_SERVER )
```

```
   LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOleDoc;
#endif // OLE_SERVER

   OLEDBG_BEGIN3("OleDoc_ShowWindow\r\n")

   /* OLE2NOTE: while the document is visible, we do NOT want it to be
   **    prematurely destroyed when a linking client disconnects. thus
   **    we must inform OLE to hold an external lock on our document.
   **    this arranges that OLE holds at least 1 reference to our
   **    document that will NOT be released until we release this
   **    external lock. later, when the document window is hidden, we
   **    will release this external lock.
   */
   if (! IsWindowVisible(lpOutlineDoc->m_hWndDoc))
      OleDoc_Lock(lpOleDoc, TRUE /* fLock */, 0 /* not applicable */);

#if defined( USE_DRAGDROP )
   /* OLE2NOTE: since our window is now being made visible, we will
   **    register our window as a potential drop target. when the
   **    window is hidden there is no reason to be registered as a
   **    drop target.
   */
   if (! lpOleDoc->m_fRegDragDrop) {
      OLEDBG_BEGIN2("RegisterDragDrop called\r\n")
      RegisterDragDrop(
            LineList_GetWindow(lpLL),
            (LPDROPTARGET)&lpOleDoc->m_DropTarget
      );
      OLEDBG_END2
      lpOleDoc->m_fRegDragDrop = TRUE;
   }
#endif  // USE_DRAGDROP

#if defined( USE_FRAMETOOLS )
   {
      /* OLE2NOTE: we need to enable our frame level tools
      */
      FrameTools_Enable(lpOutlineDoc->m_lpFrameTools, TRUE);
   }
#endif // USE_FRAMETOOLS

#if defined( OLE_SERVER )

   if (lpOutlineDoc->m_docInitType == DOCTYPE_EMBEDDED &&
         lpServerDoc->m_lpOleClientSite != NULL) {

      /* OLE2NOTE: we must also ask our container to show itself if
      **    it is not already visible and to scroll us into view. we
      **    must make sure to call this BEFORE showing our server's
      **    window and taking focus. we do not want our container's
      **    window to end up on top.
      */
      OLEDBG_BEGIN2("IOleClientSite::ShowObject called\r\n");
      lpServerDoc->m_lpOleClientSite->lpVtbl->ShowObject(
            lpServerDoc->m_lpOleClientSite
      );
      OLEDBG_END2

      /* OLE2NOTE: if we are an embedded object and we are not
      **    in-place active in our containers window, we must inform our
      **    embedding container that our window is opening.
      **    the container must now hatch our object.
      */

#if defined( INPLACE_SVR )
      if (! lpServerDoc->m_fInPlaceActive)
#endif
      {
         OLEDBG_BEGIN2("IOleClientSite::OnShowWindow(TRUE) called\r\n");
         lpServerDoc->m_lpOleClientSite->lpVtbl->OnShowWindow(
               lpServerDoc->m_lpOleClientSite,
               TRUE
         );
         OLEDBG_END2
      }

      /* OLE2NOTE: the life-time of our document is controlled by our
      **    client and NOT by the user. we are not an independent
      **    file-level object. we simply want to show our window here.
      **
      **    if we are not in-place active (ie. we are opening
      **    our own window), we must make sure our main app window is
      **    visible. we do not, however, want to give the user
      **    control of the App window; we do not want OleApp_ShowWindow
      **    to call OleApp_Lock on behalf of the user.
      */
      if (! IsWindowVisible(lpOutlineApp->m_hWndApp) ||
            IsIconic(lpOutlineApp->m_hWndApp)) {
#if defined( INPLACE_SVR )
         if (! ((LPSERVERDOC)lpOleDoc)->m_fInPlaceActive)
#endif
            OleApp_ShowWindow(lpOleApp, FALSE /* fGiveUserCtrl */);
         SetFocus(lpOutlineDoc->m_hWndDoc);
      }

   } else
#endif  // OLE_SERVER

   {    // DOCTYPE_NEW || DOCTYPE_FROMFILE

      // we must make sure our app window is visible
      OleApp_ShowWindow(lpOleApp, TRUE /* fGiveUserCtrl */);
   }

   // make document window visible and make sure it is not minimized
   ShowWindow(lpOutlineDoc->m_hWndDoc, SW_SHOWNORMAL);
   SetForegroundWindow(lpOutlineDoc->m_hWndDoc);

   OLEDBG_END3
}
```

```
/* OleDoc_HideWindow
 * -----------------
 *
 *      Hide the window of the document from the user.
 *      take away the control of the document by the user.
 */
void OleDoc_HideWindow(LPOLEDOC lpOleDoc, BOOL fShutdown)
{
   LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpOleDoc;
   LPLINELIST lpLL     = (LPLINELIST)&((LPOUTLINEDOC)lpOleDoc)->m_LineList;

   if (! IsWindowVisible(lpOutlineDoc->m_hWndDoc))
      return;     // already visible

   OLEDBG_BEGIN3("OleDoc_HideWindow\r\n")

#if defined( USE_DRAGDROP )
   // The document's window is being hidden, revoke it as a DropTarget
   if (lpOleDoc->m_fRegDragDrop) {
      OLEDBG_BEGIN2("RevokeDragDrop called\r\n");
      RevokeDragDrop(LineList_GetWindow(lpLL));
      OLEDBG_END2

      lpOleDoc->m_fRegDragDrop = FALSE ;
   }
#endif  // USE_DRAGDROP

   /* OLE2NOTE: the document is now being hidden, so we must release
   **    the external lock made when the document was made visible.
   **    if this is a shutdown situation (fShutdown==TRUE), then OLE
   **    is instructed to release our document. if this is that last
   **    external lock on our document, thus enabling our document to
   **    complete its shutdown operation. If This is not a shutdown
   **    situation (eg. in-place server hiding its window when
   **    UIDeactivating or IOleObject::DoVerb(OLEVERB_HIDE) is called),
   **    then OLE is told to NOT immediately release the document.
   **    this leaves the document in an unstable state where the next
   **    Lock/Unlock sequence will shut the document down (eg. a
   **    linking client connecting and disconnecting).
   */
   if (IsWindowVisible(lpOutlineDoc->m_hWndDoc))
      OleDoc_Lock(lpOleDoc, FALSE /* fLock */, fShutdown);

   ShowWindow(((LPOUTLINEDOC)lpOleDoc)->m_hWndDoc, SW_HIDE);

#if defined( OLE_SERVER )
   {
      LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOleDoc;

      /* OLE2NOTE: if we are an embedded object and we are not
      **    in-place active, we must inform our
      **    embedding container that our window is hiding (closing
      **    from the user's perspective). the container must now
      **    un-hatch our object.
      */
      if (lpServerDoc->m_lpOleClientSite != NULL
#if defined( INPLACE_SVR )
         && !lpServerDoc->m_fInPlaceVisible
#endif
      ) {
         OLEDBG_BEGIN2("IOleClientSite::OnShowWindow(FALSE) called\r\n");
         lpServerDoc->m_lpOleClientSite->lpVtbl->OnShowWindow(
               lpServerDoc->m_lpOleClientSite,
               FALSE
         );
         OLEDBG_END2
      }
   }
#endif

   /* OLE2NOTE: if there are no more documents visible to the user.
   **    and the app itself is not under user control, then
   **    it has no reason to stay visible. we thus should hide the
   **    app. we can not directly destroy the app, because it may be
   **    validly being used programatically by another client
   **    application and should remain running. it should simply be
   **    hidded from the user.
   */
   OleApp_HideIfNoReasonToStayVisible(lpOleApp);
   OLEDBG_END3
}


/* OleDoc_Lock
** -----------
**    Lock/Unlock the Doc object. if the last lock is unlocked and
**    fLastUnlockReleases == TRUE, then the Doc object will shut down
**    (ie. it will recieve its final release and its refcnt will go to 0).
*/
HRESULT OleDoc_Lock(LPOLEDOC lpOleDoc, BOOL fLock, BOOL fLastUnlockReleases)
{
   HRESULT hrErr;

#if defined( _DEBUG )
   if (fLock) {
      OLEDBG_BEGIN2("CoLockObjectExternal(lpDoc,TRUE) called\r\n")
   } else {
      if (fLastUnlockReleases)
         OLEDBG_BEGIN2("CoLockObjectExternal(lpDoc,FALSE,TRUE) called\r\n")
      else
         OLEDBG_BEGIN2("CoLockObjectExternal(lpDoc,FALSE,FALSE) called\r\n")
   }
#endif  // _DEBUG

   hrErr = CoLockObjectExternal(
         (LPUNKNOWN)&lpOleDoc->m_Unknown, fLock, fLastUnlockReleases);
   OLEDBG_END2
   return hrErr;
}


/* OleDoc_AddRef
** -------------
**
**  increment the ref count of the document object.
**
**    Returns the new ref count on the object
*/
ULONG OleDoc_AddRef(LPOLEDOC lpOleDoc)
{
   ++lpOleDoc->m_cRef;

#if defined( _DEBUG )
   OleDbgOutRefCnt4(
         "OleDoc_AddRef: cRef++\r\n",
         lpOleDoc,
         lpOleDoc->m_cRef
   );
#endif
   return lpOleDoc->m_cRef;
}


/* OleDoc_Release
** --------------
**
**  decrement the ref count of the document object.
**    if the ref count goes to 0, then the document is destroyed.
**
**    Returns the remaining ref count on the object
*/
ULONG OleDoc_Release (LPOLEDOC lpOleDoc)
{
   ULONG cRef;
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;

   /*****************************************************************
   ** OLE2NOTE: when the obj refcnt == 0, then destroy the object.     **
   **     otherwise the object is still in use.                        **
   *****************************************************************/

   cRef = --lpOleDoc->m_cRef;

#if defined( _DEBUG )
   OleDbgAssertSz (lpOleDoc->m_cRef >= 0, "Release called with cRef == 0");

   OleDbgOutRefCnt4(
         "OleDoc_Release: cRef--\r\n", lpOleDoc, cRef);
#endif
   if (cRef == 0)
```

```
      OutlineDoc_Destroy((LPOUTLINEDOC)lpOleDoc);

   return cRef;
}


/* OleDoc_QueryInterface
** ---------------------
**
** Retrieve a pointer to an interface on the document object.
**
**    OLE2NOTE: this function will AddRef the ref cnt of the object.
**
**    Returns S_OK if interface is successfully retrieved.
**            E_NOINTERFACE if the interface is not supported
*/
HRESULT OleDoc_QueryInterface(
      LPOLEDOC          lpOleDoc,
      REFIID            riid,
      LPVOID FAR*       lplpvObj
)
{
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpOleDoc;
   SCODE sc = E_NOINTERFACE;

   /* OLE2NOTE: we must make sure to set all out ptr parameters to NULL. */
   *lplpvObj = NULL;

   if (IsEqualIID(riid, &IID_IUnknown)) {
      OleDbgOut4("OleDoc_QueryInterface: IUnknown* RETURNED\r\n");

      *lplpvObj = (LPVOID) &lpOleDoc->m_Unknown;
      OleDoc_AddRef(lpOleDoc);
      sc = S_OK;
   }
   else if(lpOutlineDoc->m_fDataTransferDoc
         && IsEqualIID(riid, &IID_IDataObject)) {
      OleDbgOut4("OleDoc_QueryInterface: IDataObject* RETURNED\r\n");

      *lplpvObj = (LPVOID) &lpOleDoc->m_DataObject;
      OleDoc_AddRef(lpOleDoc);
      sc = S_OK;
   }

   /* OLE2NOTE: if this document is a DataTransferDocument used to
   **    support a clipboard or drag/drop operation, then it should
   **    only expose IUnknown, IDataObject, and IDropSource
   **    interfaces. if the document is a normal user document, then
   **    we will also continue to consider our other interfaces.
   */
   if (lpOutlineDoc->m_fDataTransferDoc)
      goto done;

   if(IsEqualIID(riid,&IID_IPersist) || IsEqualIID(riid,&IID_IPersistFile))
{
      OleDbgOut4("OleDoc_QueryInterface: IPersistFile* RETURNED\r\n");

      *lplpvObj = (LPVOID) &lpOleDoc->m_PersistFile;
      OleDoc_AddRef(lpOleDoc);
      sc = S_OK;
   }
   else if(IsEqualIID(riid, &IID_IOleItemContainer) ||
         IsEqualIID(riid, &IID_IOleContainer) ||
         IsEqualIID(riid, &IID_IParseDisplayName) ) {
      OleDbgOut4("OleDoc_QueryInterface: IOleItemContainer* RETURNED\r\n");

      *lplpvObj = (LPVOID) &lpOleDoc->m_OleItemContainer;
      OleDoc_AddRef(lpOleDoc);
      sc = S_OK;
   }
   else if(IsEqualIID(riid, &IID_IExternalConnection)) {
      OleDbgOut4("OleDoc_QueryInterface: IExternalConnection*
RETURNED\r\n");

      *lplpvObj = (LPVOID) &lpOleDoc->m_ExternalConnection;
      OleDoc_AddRef(lpOleDoc);
      sc = S_OK;
   }

#if defined( USE_DRAGDROP )
   else if(IsEqualIID(riid, &IID_IDropTarget)) {
      OleDbgOut4("OleDoc_QueryInterface: IDropTarget* RETURNED\r\n");

      *lplpvObj = (LPVOID) &lpOleDoc->m_DropTarget;
      OleDoc_AddRef(lpOleDoc);
      sc = S_OK;
   }
   else if(IsEqualIID(riid, &IID_IDropSource)) {
      OleDbgOut4("OleDoc_QueryInterface: IDropSource* RETURNED\r\n");

      *lplpvObj = (LPVOID) &lpOleDoc->m_DropSource;
      OleDoc_AddRef(lpOleDoc);
      sc = S_OK;
   }
#endif

#if defined( OLE_CNTR )
   else if (IsEqualIID(riid, &IID_IOleUILinkContainer)) {
      OleDbgOut4("OleDoc_QueryInterface: IOleUILinkContainer*
RETURNED\r\n");

      *lplpvObj=(LPVOID)&((LPCONTAINERDOC)lpOleDoc)->m_OleUILinkContainer;
      OleDoc_AddRef(lpOleDoc);
      sc = S_OK;
   }
#endif

#if defined( OLE_SERVER )

   /* OLE2NOTE: if OLE server version, than also offer the server
   **    specific interfaces: IOleObject and IPersistStorage.
   */
   else if (IsEqualIID(riid, &IID_IOleObject)) {
      OleDbgOut4("OleDoc_QueryInterface: IOleObject* RETURNED\r\n");

      *lplpvObj = (LPVOID) &((LPSERVERDOC)lpOleDoc)->m_OleObject;
      OleDoc_AddRef(lpOleDoc);
      sc = S_OK;
   }
   else if(IsEqualIID(riid, &IID_IPersistStorage)) {
      OleDbgOut4("OleDoc_QueryInterface: IPersistStorage* RETURNED\r\n");

      *lplpvObj = (LPVOID) &((LPSERVERDOC)lpOleDoc)->m_PersistStorage;
      OleDoc_AddRef(lpOleDoc);
      sc = S_OK;
   }
   else if(IsEqualIID(riid, &IID_IDataObject)) {
      OleDbgOut4("OleDoc_QueryInterface: IDataObject* RETURNED\r\n");

      *lplpvObj = (LPVOID) &lpOleDoc->m_DataObject;
      OleDoc_AddRef(lpOleDoc);
      sc = S_OK;
   }

#if defined( SVR_TREATAS )
   else if(IsEqualIID(riid, &IID_IStdMarshalInfo)) {
      OleDbgOut4("OleDoc_QueryInterface: IStdMarshalInfo* RETURNED\r\n");

      *lplpvObj = (LPVOID) &((LPSERVERDOC)lpOleDoc)->m_StdMarshalInfo;
      OleDoc_AddRef(lpOleDoc);
      sc = S_OK;
   }
#endif  // SVR_TREATAS

#if defined( INPLACE_SVR )
   else if (IsEqualIID(riid, &IID_IOleWindow) ||
          IsEqualIID(riid, &IID_IOleInPlaceObject)) {
      OleDbgOut4("OleDoc_QueryInterface: IOleInPlaceObject* RETURNED\r\n");

      *lplpvObj = (LPVOID) &((LPSERVERDOC)lpOleDoc)->m_OleInPlaceObject;
      OleDoc_AddRef(lpOleDoc);
      sc = S_OK;
   }
#endif // INPLACE_SVR
#endif // OLE_SERVER

done:
   OleDbgQueryInterfaceMethod(*lplpvObj);

   return ResultFromScode(sc);
}


/* OleDoc_Close
 * ------------
 *
 *  Close the document.
 *      This functions performs the actions that are in common to all
 *      document types which derive from OleDoc (eg. ContainerDoc and
 *      ServerDoc) which are required to close a document.
 *
 *  Returns:
 *      FALSE -- user canceled the closing of the doc.
 *      TRUE -- the doc was successfully closed
 */

BOOL OleDoc_Close(LPOLEDOC lpOleDoc, DWORD dwSaveOption)
{
   LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPOLEDOC lpClipboardDoc;
   LPLINELIST lpLL     = (LPLINELIST)&((LPOUTLINEDOC)lpOleDoc)->m_LineList;
   BOOL fAbortIfSaveCanceled = (dwSaveOption == OLECLOSE_PROMPTSAVE);

   if (! lpOleDoc)
      return TRUE;    // active doc's are already destroyed

   if (lpOleDoc->m_fObjIsClosing)
      return TRUE;    // Closing is already in progress

   OLEDBG_BEGIN3("OleDoc_Close\r\n")

   if (! OutlineDoc_CheckSaveChanges((LPOUTLINEDOC)lpOleDoc,&dwSaveOption)
         && fAbortIfSaveCanceled) {
      OLEDBG_END3
      return FALSE;            // cancel closing the doc
   }

   lpOleDoc->m_fObjIsClosing = TRUE;   // guard against recursive call

   /* OLE2NOTE: in order to have a stable app and doc during the
   **    process of closing, we intially AddRef the App and Doc ref
   **    cnts and later Release them. These initial AddRefs are
   **    artificial; they simply guarantee that these objects do not
   **    get destroyed until the end of this routine.
   */
   OleApp_AddRef(lpOleApp);
   OleDoc_AddRef(lpOleDoc);

#if defined( OLE_CNTR )
   {
      LPCONTAINERDOC lpContainerDoc = (LPCONTAINERDOC)lpOleDoc;

      /* OLE2NOTE: force all OLE objects to close. this forces all
      **    OLE object to transition from running to loaded. we can
      **    NOT exit if any embeddings are still running.
      **    if an object can't be closed and this close operation was
      **    started by the user, then we will abort closing our document.
      */
      if (! ContainerDoc_CloseAllOleObjects(lpContainerDoc, OLECLOSE_NOSAVE)
            && fAbortIfSaveCanceled) {
         OleDoc_Release(lpOleDoc);       // release artificial AddRef above
         OleApp_Release(lpOleApp);       // release artificial AddRef above
         lpOleDoc->m_fObjIsClosing = FALSE; // clear recursion guard

         OLEDBG_END3
         return FALSE;   // Closing is aborted
      }
   }
#endif

#if defined( INPLACE_SVR )
   /* OLE2NOTE: if the server is currently in-place active we must
   **    deactivate it now before closing
   */
   ServerDoc_DoInPlaceDeactivate((LPSERVERDOC)lpOleDoc);
#endif

   /* OLE2NOTE: if this document is the source of data for the
   **    clipboard, then flush the clipboard. it is important to flush
   **    the clipboard BEFORE calling sending any notifications to
   **    clients (eg. IOleClientSite::OnShowWindow(FALSE)) which could
   **    give them a chance to run and try to get our clipboard data
   **    object that we want to destroy. (eg. our app tries to
   **    update the paste button of the toolbar when
   **    WM_ACTIVATEAPP is received.)
   */
   lpClipboardDoc = (LPOLEDOC)lpOutlineApp->m_lpClipboardDoc;
   if (lpClipboardDoc &&
      lpClipboardDoc->m_lpSrcDocOfCopy == lpOleDoc) {
      OleApp_FlushClipboard(lpOleApp);
   }

   /* OLE2NOTE: Revoke the object from the Running Object Table. it is
   **    best if the object is revoke prior to calling
   **    COLockObjectExternal(FALSE,TRUE) which is called when the
   **    document window is hidden from the user.
   */
   OLEDBG_BEGIN3("OleStdRevokeAsRunning called\r\n")
   OleStdRevokeAsRunning(&lpOleDoc->m_dwRegROT);
   OLEDBG_END3

   /* OLE2NOTE: if the user is in control of the document, the user
   **    accounts for one refcnt on the document. Closing the
   **    document is achieved by releasing the object on behalf of
   **    the user. if the document is not referenced by any other
   **    clients, then the document will also be destroyed. if it
   **    is referenced by other clients, then it will remain until
   **    they release it. it is important to hide the window and call
   **    IOleClientSite::OnShowWindow(FALSE) BEFORE sending OnClose
   **    notification.
   */
   OleDoc_HideWindow(lpOleDoc, TRUE);

#if defined( OLE_SERVER )
```

```
{
   LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOleDoc;
   LPSERVERNAMETABLE lpServerNameTable =
      (LPSERVERNAMETABLE)((LPOUTLINEDOC)lpOleDoc)->m_lpNameTable;

   /* OLE2NOTE: force all pseudo objects to close. this informs all
   **    linking clients of pseudo objects to release their PseudoObj.
   */
   ServerNameTable_CloseAllPseudoObjs(lpServerNameTable);

   /* OLE2NOTE: send last OnDataChange notification to clients
   **    that have registered for data notifications when object
   **    stops running (ADVF_DATAONSTOP), if the data in our
   **    object has ever changed. it is best to only send this
   **    notification if necessary.
   */
   if (lpServerDoc->m_lpDataAdviseHldr) {
      if (lpServerDoc->m_fSendDataOnStop) {
         ServerDoc_SendAdvise(
               (LPSERVERDOC)lpOleDoc,
               OLE_ONDATACHANGE,
               NULL,   /* lpmkDoc -- not relevant here */
               ADVF_DATAONSTOP
         );
      }
      /* OLE2NOTE: we just sent the last data notification that we
      **    need to send; release our DataAdviseHolder. we SHOULD be
      **    the only one using it.
      */

      OleStdVerifyRelease(
            (LPUNKNOWN)lpServerDoc->m_lpDataAdviseHldr,
            OLESTR("DataAdviseHldr not released properly")
      );
      lpServerDoc->m_lpDataAdviseHldr = NULL;
   }

   // OLE2NOTE: inform all of our linking clients that we are closing.


   if (lpServerDoc->m_lpOleAdviseHldr) {
      ServerDoc_SendAdvise(
            (LPSERVERDOC)lpOleDoc,
            OLE_ONCLOSE,
            NULL,   /* lpmkDoc -- not relevant here */
            0       /* advf -- not relevant here */
      );

      /* OLE2NOTE: OnClose is the last notification that we need to
      **    send; release our OleAdviseHolder. we SHOULD be the only
      **    one using it. this will make our destructor realize that
      **    OnClose notification has already been sent.
      */
      OleStdVerifyRelease(
            (LPUNKNOWN)lpServerDoc->m_lpOleAdviseHldr,
                OLESTR("OleAdviseHldr not released properly")
          );
          lpServerDoc->m_lpOleAdviseHldr = NULL;
       }

       /* release our Container's ClientSite. */
       if(lpServerDoc->m_lpOleClientSite) {
          OleStdRelease((LPUNKNOWN)lpServerDoc->m_lpOleClientSite);
          lpServerDoc->m_lpOleClientSite = NULL;
       }
    }
#endif

    if (lpOleDoc->m_lpLLStm) {
       /* release our LineList stream. */
       OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpLLStm);
       lpOleDoc->m_lpLLStm = NULL;
    }

    if (lpOleDoc->m_lpNTStm) {
       /* release our NameTable stream. */
       OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpNTStm);
       lpOleDoc->m_lpNTStm = NULL;
    }

    if (lpOleDoc->m_lpStg) {
       /* release our doc storage. */
       OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpStg);
       lpOleDoc->m_lpStg = NULL;
    }

    if (lpOleDoc->m_lpFileMoniker) {
       OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpFileMoniker);
       lpOleDoc->m_lpFileMoniker = NULL;
    }

    /* OLE2NOTE: this call forces all external connections to our
    **    object to close down and therefore guarantees that we receive
    **    all releases associated with those external connections.
    */
    OLEDBG_BEGIN2("CoDisconnectObject(lpDoc) called\r\n")
    CoDisconnectObject((LPUNKNOWN)&lpOleDoc->m_Unknown, 0);
    OLEDBG_END2

    OleDoc_Release(lpOleDoc);       // release artificial AddRef above
    OleApp_Release(lpOleApp);       // release artificial AddRef above

    OLEDBG_END3
    return TRUE;
}


/* OleDoc_Destroy
 * --------------
 *
 *  Free all OLE related resources that had been allocated for a document.
 */
void OleDoc_Destroy(LPOLEDOC lpOleDoc)
{
    LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
    LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpOleDoc;

    if (lpOleDoc->m_fObjIsDestroying)
       return;     // Doc destruction is already in progress

    lpOleDoc->m_fObjIsDestroying = TRUE;    // guard against recursive call

#if defined( OLE_SERVER )

    /* OLE2NOTE: it is ALWAYS necessary to make sure that the work we
    **    do in our OleDoc_Close function is performed before we
    **    destroy our document object. this includes revoking from the
    **    Running Object Table (ROT), sending OnClose notification,
    **    revoking from Drag/Drop, closing all pseudo objects, etc.
    **    There are some tricky scenarios involving linking and
    **    when IOleObject::Close is called versus when we get our
    **    final release causing us to call our OleDoc_Destroy
    **    (destructor) function.
    **
    **    SCENARIO 1 -- closing from server (File.Exit or File.Close)
    **                  OleDoc_Close function is called directly by the
    **                  server in response to the menu command
    **                  (WM_COMMAND processing).
    **
    **    SCENARIO 2 -- closed by embedding container
    **                  our embedding container calls IOleObject::Close
    **                  directly.
    **
    **    SCENARIO 3 -- silent-update final release
    **                  THIS IS THE TRICKY ONE!!!
    **                  in the case that our object is launched because
    **                  a linking client calls IOleObject::Update on
    **                  its link, then our object will be run
    **                  invisibly, typically GetData will be called,
    **                  and then the connection from the linking client
    **                  will be released. the release of this last
    **                  linking connection should cause our object to
    **                  shut down.
    **                  there are 2 strategies to deal with this scenario:
    **
    **                  STRATEGY 1 -- implement IExternalConnection.
    **                  IExternalConnection::AddConnection will be
    **                  called (by the StubManager) every time that an
    **                  external (linking) connection is created or
    **                  CoLockObjectExternal is called. the object
    **                  should maintain a count of strong connections
    **                  (m_dwStrongExtConn). IExternalConnection::
    **                  ReleaseConnection will be called when these
    **                  connections are released. when the
    **                  m_dwStrongExtConn transistions to 0, the object
    **                   should call its IOleObject::Close function.
    **                   this assumes that CoLockObjectExternal is used
    **                   to manage locks by the object itself (eg. when
    **                   the object is visible to the user--fUserCtrl,
    **                   and when PseudoObjects are created, etc.)
    **                   this is the strategy implemented by SVROUTL.
    **
    **                   STRATEGY 2 -- guard both the destructor
    **                   function and the Close function. if the
    **                   destructor is called directly without Close
    **                   first being called, then call Close before
    **                   proceeding with the destruction code.
    **                   previously SVROUTL was organized in this
    **                   manner. that old code is conditionaly compiled
    **                   away with "#ifdef OBSOLETE" below. this
    **                   method has the disadvantage that external
    **                   remoting is no longer possible by the time the
    **                   Close is called making it impossible for
    **                   the object to ask its container to save the
    **                   object if the object is dirty. this can result
    **                   in data loss. thus STRATEGY 1 is safer.
    **                   consider the scenario where an in-place
    **                   container UIDeactivates an object but does NOT
    **                   keep the object locked running (this is
    **                   required--see CntrLine_IPSite_OnInPlaceActivate
    **                   in cntrline.c), then, if a linking client binds
    **                   and unbinds from the object, the object will be
    **                   destroyed and will NOT have an opportunity to
    **                   be saved. by implementing IExternalConnection,
    **                   a server can insulate itself from a poorly
    **                   written container.
    */
 #if defined( _DEBUG )
    OleDbgAssertSz(
          (lpOutlineDoc->m_fDataTransferDoc || lpOleDoc->m_fObjIsClosing),
          "Destroy called without Close being called\r\n"
    );
 #endif  // _DEBUG
 #if defined( OBSOLETE )
    /* OLE2NOTE: if the document destructor is called directly because
    **    the object's refcnt went to 0 (ie. without OleDoc_Close first
    **    being called), then we need to make sure that the document is
    **    properly closed before destroying the object. this scenario
    **    could arise during a silent-update of a link. calling
    **    OleDoc_Close here guarantees that the clipboard will be
    **    properly flushed, the doc's moniker will be properly revoked,
    **    the document will be saved if necessary, etc.
    */
    if (!lpOutlineDoc->m_fDataTransferDoc && !lpOleDoc->m_fObjIsClosing)
       OleDoc_Close(lpOleDoc, OLECLOSE_NOSAVE);
 #endif

    {
       LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOleDoc;
       /* OLE2NOTE: perform processing specific for an OLE server */
 #if defined( SVR_TREATAS )
       if (lpServerDoc->m_lpszTreatAsType) {
          OleStdFreeString(lpServerDoc->m_lpszTreatAsType, NULL);
          lpServerDoc->m_lpszTreatAsType = NULL;
       }
 #endif  // SVR_TREATAS

 #if defined( INPLACE_SVR )
       if (IsWindow(lpServerDoc->m_hWndHatch))
          DestroyWindow(lpServerDoc->m_hWndHatch);
 #endif  // INPLACE_SVR
    }
 #endif  // OLE_SERVER

    if (lpOleDoc->m_lpLLStm) {
       /* release our LineList stream. */
       OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpLLStm);
       lpOleDoc->m_lpLLStm = NULL;
    }

    if (lpOleDoc->m_lpNTStm) {
       /* release our NameTable stream. */
       OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpNTStm);
       lpOleDoc->m_lpNTStm = NULL;
    }

    if (lpOleDoc->m_lpStg) {
       /* release our doc storage. */
       OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpStg);
       lpOleDoc->m_lpStg = NULL;
    }

    if (lpOleDoc->m_lpFileMoniker) {
       OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpFileMoniker);
       lpOleDoc->m_lpFileMoniker = NULL;
    }

    /******************************************************************
    ** OLE2NOTE: each document addref's the app object in order to   **
    **    guarentee that the app does not shut down while the doc    **
    **    is still open. since this doc is now destroyed, we will    **
    **    release this refcnt now. if there are now more open        **
    **    documents AND the app is not under the control of the      **
    **    user (ie. launched by OLE) then the app will revoke its    **
    **    ClassFactory. if there are no more references to the       **
    **    ClassFactory after it is revoked, then the app will shut   **
    **    down. this whole procedure is triggered by calling         **
    **    OutlineApp_DocUnlockApp.                                   **
    ******************************************************************/

    OutlineApp_DocUnlockApp(lpOutlineApp, lpOutlineDoc);
 }
```

```
/* OleDoc_SetUpdateEditMenuFlag
 * ----------------------------
 *
 *  Purpose:
 *      Set/clear the UpdateEditMenuFlag in OleDoc.
 *
 *  Parameters:
 *      fUpdate     new value of the flag
 *
 *  Returns:
 */
void OleDoc_SetUpdateEditMenuFlag(LPOLEDOC lpOleDoc, BOOL fUpdate)
{
   if (!lpOleDoc)
      return;

   lpOleDoc->m_fUpdateEditMenu = fUpdate;
}


/* OleDoc_GetUpdateEditMenuFlag
 * ----------------------------
 *
 *  Purpose:
 *      Get the value of the UpdateEditMenuFlag in OleDoc
 *
 *  Parameters:
 *
 *  Returns:
 *      value of the flag
 */
BOOL OleDoc_GetUpdateEditMenuFlag(LPOLEDOC lpOleDoc)
{
   if (!lpOleDoc)
      return FALSE;

   return lpOleDoc->m_fUpdateEditMenu;
}



/*************************************************************************
** OleDoc::IUnknown interface implementation
*************************************************************************/

STDMETHODIMP OleDoc_Unk_QueryInterface(
      LPUNKNOWN           lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
)
{
   LPOLEDOC lpOleDoc = ((struct CDocUnknownImpl FAR*)lpThis)->lpOleDoc;

   return OleDoc_QueryInterface(lpOleDoc, riid, lplpvObj);
}
```

```
STDMETHODIMP_(ULONG) OleDoc_Unk_AddRef(LPUNKNOWN lpThis)
{
   LPOLEDOC lpOleDoc = ((struct CDocUnknownImpl FAR*)lpThis)->lpOleDoc;

   OleDbgAddRefMethod(lpThis, "IUnknown");

   return OleDoc_AddRef(lpOleDoc);
}


STDMETHODIMP_(ULONG) OleDoc_Unk_Release (LPUNKNOWN lpThis)
{
   LPOLEDOC lpOleDoc = ((struct CDocUnknownImpl FAR*)lpThis)->lpOleDoc;

   OleDbgReleaseMethod(lpThis, "IUnknown");

   return OleDoc_Release(lpOleDoc);
}
```

OLEOUTL.H   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Sample Code
**
**    oleoutl.h
**
**    This file contains file contains data structure defintions,
**    function prototypes, constants, etc. which are common to the
**    server version and the container version of the app.
**    app version of the Outline series of sample applications:
**          Outline -- base version of the app (without OLE functionality)
**          SvrOutl -- OLE 2.0 Server sample app
**          CntrOutl -- OLE 2.0 Containter sample app
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#if !defined( _OLEOUTL_H_ )
#define _OLEOUTL_H_

#ifndef RC_INVOKED
#pragma message ("INCLUDING OLEOUTL.H from " __FILE__)
// make 'different levels of inderection' considered an error
#pragma warning (error:4047)
#endif  /* RC_INVOKED */

#if defined( USE_MSGFILTER )
//#include "..\include\msgfiltr.h"
#include "msgfiltr.h"
#endif  // USE_MSGFILTER

#include "defguid.h"
#include "ansiapi.h"

/* Defines */

/* OLE2NOTE: these strings should correspond to the strings registered
**    in the registration database.
*/
// REVIEW: should load strings from resource file
#if defined( INPLACE_SVR )

#ifdef TEST32

#define CLSID_APP   CLSID_ISvrOu32
#define FULLUSERTYPENAME    OLESTR("Ole 2.0 In-Place Server Outline 32")
#define SHORTUSERTYPENAME   OLESTR("Outline32")   // max 15 chars
#undef  APPFILENAMEFILTER
#define APPFILENAMEFILTER   "Outline Files (*.OLT)|*.olt|All files (*.*)|
*.*|"
#undef  DEFEXTENSION
```

```
#define DEFEXTENSION    "olt"           // Default file extension

#else

#define CLSID_APP   CLSID_ISvrOtl
#define FULLUSERTYPENAME    OLESTR("Ole 2.0 In-Place Server Outline")
#define SHORTUSERTYPENAME   OLESTR("Outline")   // max 15 chars
#undef  APPFILENAMEFILTER
#define APPFILENAMEFILTER   "Outline Files (*.OLN)|*.oln|All files (*.*)|
*.*|"
#undef  DEFEXTENSION
#define DEFEXTENSION    "oln"           // Default file extension

#endif  // TEST32
#endif  // INPLACE_SVR

#if defined( INPLACE_CNTR )

#ifdef TEST32

#define CLSID_APP   CLSID_ICntrO32
#define FULLUSERTYPENAME    OLESTR("Ole 2.0 In-Place Container Outline 32")
// #define SHORTUSERTYPENAME    "Outline32"   // max 15 chars
#undef  APPFILENAMEFILTER
#define APPFILENAMEFILTER   "CntrOutl Files (*.OLC)|*.olc|Outline Files
(*.OLT)|*.olt|All files (*.*)|*.*|"
#undef  DEFEXTENSION
#define DEFEXTENSION    "olc"           // Default file extension

#else

#define CLSID_APP   CLSID_ICntrOtl
#define FULLUSERTYPENAME    OLESTR("Ole 2.0 In-Place Container Outline")
// #define SHORTUSERTYPENAME    "Outline"   // max 15 chars
#undef  APPFILENAMEFILTER
#define APPFILENAMEFILTER   "CntrOutl Files (*.OLC)|*.olc|Outline Files
(*.OLN)|*.oln|All files (*.*)|*.*|"
#undef  DEFEXTENSION
#define DEFEXTENSION    "olc"           // Default file extension

#endif TEST32
#endif  // INPLACE_CNTR

#if defined( OLE_SERVER ) && !defined( INPLACE_SVR )

#ifdef TEST32

#define CLSID_APP   CLSID_SvrOut32
#define FULLUSERTYPENAME    OLESTR("Ole 2.0 Server Sample Outline 32")
#define SHORTUSERTYPENAME   OLESTR("Outline32")
#undef  APPFILENAMEFILTER
#define APPFILENAMEFILTER   "Outline Files (*.OLT)|*.olt|All files (*.*)|
*.*|"
#undef  DEFEXTENSION
#define DEFEXTENSION    "olt"           // Default file extension
```

```
#else

#define CLSID_APP   CLSID_SvrOutl
#define FULLUSERTYPENAME    OLESTR("Ole 2.0 Server Sample Outline")
#define SHORTUSERTYPENAME   OLESTR("Outline")
#undef  APPFILENAMEFILTER
#define APPFILENAMEFILTER   "Outline Files (*.OLN)|*.oln|All files (*.*)|
*.*|"
#undef  DEFEXTENSION
#define DEFEXTENSION    "oln"           // Default file extension

#endif  //TEST32
#endif  // OLE_SERVER && ! INPLACE_SVR

#if defined( OLE_CNTR ) && !defined( INPLACE_CNTR )
#ifdef TEST32

#define CLSID_APP   CLSID_CntrOu32
#define FULLUSERTYPENAME    OLESTR("Ole 2.0 Container Sample Outline 32")
// #define SHORTUSERTYPENAME    "Outline32"  // max 15 chars
#undef  APPFILENAMEFILTER
#define APPFILENAMEFILTER   "CntrOutl Files (*.OLC)|*.olc|Outline Files
(*.OLT)|*.olt|All files (*.*)|*.*|"
#undef  DEFEXTENSION
#define DEFEXTENSION    "olc"           // Default file extension

#else

#define CLSID_APP   CLSID_CntrOutl
#define FULLUSERTYPENAME    OLESTR("Ole 2.0 Container Sample Outline")
// #define SHORTUSERTYPENAME    "Outline"  // max 15 chars
#undef  APPFILENAMEFILTER
#define APPFILENAMEFILTER   "CntrOutl Files (*.OLC)|*.olc|Outline Files
(*.OLN)|*.oln|All files (*.*)|*.*|"
#undef  DEFEXTENSION
#define DEFEXTENSION    "olc"           // Default file extension

#endif  //TEST32
#endif  // OLE_CNTR && ! INPLACE_CNTR

// Maximum number of formats offered by IDataObject::GetData/SetData
#define MAXNOFMTS       10
#define MAXNOLINKTYPES   3

#if defined( USE_DRAGDROP )
#define DD_SEL_THRESH       HITTESTDELTA    // Border threshold to start
drag
#define MAX_SEL_ITEMS       0x0080
#endif  // USE_DRAGDROP

/* Positions of the various menus */
#define POS_FILEMENU        0
#define POS_EDITMENU        1
#define POS_VIEWMENU        2
```

```
#define POS_LINEMENU        3
#define POS_NAMEMENU        4
#define POS_OPTIONSMENU     5
#define POS_DEBUGMENU       6
#define POS_HELPMENU        7


#define POS_OBJECT      11


/* Types */

// Document initialization type
#define DOCTYPE_EMBEDDED    3   // init from an IStorage* of an embedded obj
#define DOCTYPE_FROMSTG     4   // init from an IStorage* with doc bit set

/* Forward type definitions */
typedef struct tagOLEAPP FAR* LPOLEAPP;
typedef struct tagOLEDOC FAR* LPOLEDOC;

/* Flags to control Moniker assignment for OleDoc_GetFullMoniker */
// REVIEW: should use official OLEGETMONIKER type for final version
typedef enum tagGETMONIKERTYPE {
   GETMONIKER_ONLYIFTHERE  = 1,
   GETMONIKER_FORCEASSIGN  = 2,
   GETMONIKER_UNASSIGN     = 3,
   GETMONIKER_TEMPFORUSER  = 4
} GETMONIKERTYPE;

/* Flags to control direction for drag scrolling */
typedef enum tagSCROLLDIR {
   SCROLLDIR_NULL          = 0,
   SCROLLDIR_UP            = 1,
   SCROLLDIR_DOWN          = 2,
   SCROLLDIR_RIGHT         = 3,    // currently not used
   SCROLLDIR_LEFT          = 4     // currently not used
} SCROLLDIR;


/****************************************************************************
** class OLEDOC : OUTLINEDOC
**    OLEDOC is an extention to the base OUTLINEDOC object (structure)
**    that adds common OLE 2.0 functionality used by both the server
**    and container versions. This is an abstract class. You do not
**    instantiate an instance of OLEDOC directly but instead
**    instantiate one of its concrete subclasses: SERVERDOC or
**    CONTAINERDOC. There is one instance of an document
**    object created per document open in the app. The SDI
**    version of the app supports one ServerDoc at a time. The MDI
**    version of the app can manage multiple documents at one time.
**    The OLEDOC class inherits all fields from the OUTLINEDOC class.
**    This inheritance is achieved by including a member variable of
**    type OUTLINEDOC as the first field in the OLEDOC
**    structure. Thus a pointer to an OLEDOC object can be cast to be
**    a pointer to a OUTLINEDOC object.
```

```
*************************************************************************/

typedef struct tagOLEDOC {
   OUTLINEDOC      m_OutlineDoc;       // ServerDoc inherits from OutlineDoc
   ULONG           m_cRef;             // total ref count for document
   ULONG           m_dwStrongExtConn;  // total strong connection count
                            //  (from IExternalConnection)
                            //  when this count transitions to 0
                            //  and fLastUnlockCloses==TRUE, then
                            //  IOleObject::Close is called to
                            //  close the document.
#if defined( _DEBUG )
   ULONG           m_cCntrLock;        // total count of LockContainer locks
                            //  (for debugging purposes only)
#endif
   LPSTORAGE       m_lpStg;            // OleDoc must keep its stg open
                            //  even in-memory server doc should
                            //  keep Stg open for low memory save
   LPSTREAM        m_lpLLStm;          // Hold LineList IStream* open for
                            //  low memory save
   LPSTREAM        m_lpNTStm;          // Hold NameTable IStream* open for
                            //  low memory save
   BOOL            m_fObjIsClosing;    // flag to guard recursive close call
   BOOL            m_fObjIsDestroying; // flag to guard recursiv destroy
call
   DWORD           m_dwRegROT;         // key if doc registered as running
   LPMONIKER       m_lpFileMoniker;    // moniker if file-based/untitled doc
   BOOL            m_fLinkSourceAvail; // can doc offer CF_LINKSOURCE
   LPOLEDOC        m_lpSrcDocOfCopy;   // src doc if doc created for copy
   BOOL            m_fUpdateEditMenu;  // need to update edit menu??

#if defined( USE_DRAGDROP )
   DWORD           m_dwTimeEnterScrollArea;  // time entering scroll region
   DWORD           m_dwLastScrollDir;  // current dir for drag scroll
   DWORD           m_dwNextScrollTime; // time for next scroll
   BOOL            m_fRegDragDrop;     // is doc registered as drop target?
   BOOL            m_fLocalDrag;       // is doc source of the drag
   BOOL            m_fLocalDrop;       // was doc target of the drop
   BOOL            m_fCanDropCopy;     // is Drag/Drop copy/move possible?
   BOOL            m_fCanDropLink;     // is Drag/Drop link possible?
   BOOL            m_fDragLeave;       // has drag left
   BOOL            m_fPendingDrag;     // LButtonDown--possible drag pending
   POINT           m_ptButDown;        // LButtonDown coordinates
#endif  // USE_DRAGDROP

#if defined( INPLACE_SVR ) || defined( INPLACE_CNTR )
   BOOL            m_fCSHelpMode;      // Shift-F1 context help mode
#endif

   struct CDocUnknownImpl {
      IUnknownVtbl FAR*       lpVtbl;
      LPOLEDOC                lpOleDoc;
      int                     cRef;   // interface specific ref count.
   } m_Unknown;
   struct CDocPersistFileImpl {
      IPersistFileVtbl FAR*   lpVtbl;
      LPOLEDOC                lpOleDoc;
      int                     cRef;   // interface specific ref count.
   } m_PersistFile;

   struct CDocOleItemContainerImpl {
      IOleItemContainerVtbl FAR*  lpVtbl;
      LPOLEDOC                    lpOleDoc;
      int                         cRef;   // interface specific ref count.
   } m_OleItemContainer;

   struct CDocExternalConnectionImpl {
      IExternalConnectionVtbl FAR* lpVtbl;
      LPOLEDOC                lpOleDoc;
      int                     cRef;   // interface specific ref count.
   } m_ExternalConnection;

   struct CDocDataObjectImpl {
      IDataObjectVtbl FAR*        lpVtbl;
      LPOLEDOC                    lpOleDoc;
      int                         cRef;   // interface specific ref count.
   } m_DataObject;

#ifdef USE_DRAGDROP
   struct CDocDropSourceImpl {
      IDropSourceVtbl FAR*    lpVtbl;
      LPOLEDOC                lpOleDoc;
      int                     cRef;   // interface specific ref count.
   } m_DropSource;

   struct CDocDropTargetImpl {
      IDropTargetVtbl FAR*    lpVtbl;
      LPOLEDOC                lpOleDoc;
      int                     cRef;   // interface specific ref count.
   } m_DropTarget;
#endif  // USE_DRAGDROP

} OLEDOC;

/* OleDoc methods (functions) */
BOOL OleDoc_Init(LPOLEDOC lpOleDoc, BOOL fDataTransferDoc);
BOOL OleDoc_InitNewFile(LPOLEDOC lpOleDoc);
void OleDoc_ShowWindow(LPOLEDOC lpOleDoc);
void OleDoc_HideWindow(LPOLEDOC lpOleDoc, BOOL fShutDown);
HRESULT OleDoc_Lock(LPOLEDOC lpOleDoc, BOOL fLock, BOOL
fLastUnlockReleases);
ULONG OleDoc_AddRef(LPOLEDOC lpOleDoc);
ULONG OleDoc_Release (LPOLEDOC lpOleDoc);
HRESULT OleDoc_QueryInterface(
      LPOLEDOC          lpOleDoc,
      REFIID            riid,
      LPVOID FAR*       lplpUnk
);
BOOL OleDoc_Close(LPOLEDOC lpOleDoc, DWORD dwSaveOption);
```

```
void OleDoc_Destroy(LPOLEDOC lpOleDoc);
void OleDoc_SetUpdateEditMenuFlag(LPOLEDOC lpOleDoc, BOOL fUpdate);
BOOL OleDoc_GetUpdateEditMenuFlag(LPOLEDOC lpOleDoc);
void OleDoc_GetExtent(LPOLEDOC lpOleDoc, LPSIZEL lpsizel);
HGLOBAL OleDoc_GetObjectDescriptorData(
      LPOLEDOC            lpOleDoc,
      LPLINERANGE         lplrSel
);
LPMONIKER OleDoc_GetFullMoniker(LPOLEDOC lpOleDoc, DWORD dwAssign);
void OleDoc_GetExtent(LPOLEDOC lpOleDoc, LPSIZEL lpsizel);
void OleDoc_DocRenamedUpdate(LPOLEDOC lpOleDoc, LPMONIKER lpmkDoc);
void OleDoc_CopyCommand(LPOLEDOC lpSrcOleDoc);
void OleDoc_PasteCommand(LPOLEDOC lpOleDoc);
void OleDoc_PasteSpecialCommand(LPOLEDOC lpOleDoc);
LPOUTLINEDOC OleDoc_CreateDataTransferDoc(LPOLEDOC lpSrcOleDoc);
BOOL OleDoc_PasteFromData(
      LPOLEDOC            lpOleDoc,
      LPDATAOBJECT        lpSrcDataObj,
      BOOL                fLocalDataObj,
      BOOL                fLink
);
BOOL OleDoc_PasteFormatFromData(
      LPOLEDOC            lpOleDoc,
      CLIPFORMAT          cfFormat,
      LPDATAOBJECT        lpSrcDataObj,
      BOOL                fLocalDataObj,
      BOOL                fLink,
      BOOL                fDisplayAsIcon,
      HGLOBAL             hMetaPict,
      LPSIZEL             lpSizelInSrc
);
BOOL OleDoc_QueryPasteFromData(
      LPOLEDOC            lpOleDoc,
      LPDATAOBJECT        lpSrcDataObj,
      BOOL                fLink
);

#if defined( USE_DRAGDROP )

BOOL OleDoc_QueryDrag( LPOLEDOC lpOleDoc, int y );
BOOL OleDoc_QueryDrop (
   LPOLEDOC        lpOleDoc,
   DWORD           grfKeyState,
   POINTL          pointl,
   BOOL            fDragScroll,
   LPDWORD         lpdwEffect
);
DWORD OleDoc_DoDragDrop (LPOLEDOC lpSrcOleDoc);
BOOL OleDoc_DoDragScroll(LPOLEDOC lpOleDoc, POINTL pointl);

#endif  // USE_DRAGDROP

/* OleDoc::IUnknown methods (functions) */
STDMETHODIMP OleDoc_Unk_QueryInterface(
      LPUNKNOWN           lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
);
STDMETHODIMP_(ULONG) OleDoc_Unk_AddRef(LPUNKNOWN lpThis);
STDMETHODIMP_(ULONG) OleDoc_Unk_Release (LPUNKNOWN lpThis);

/* OleDoc::IPersistFile methods (functions) */
STDMETHODIMP OleDoc_PFile_QueryInterface(
      LPPERSISTFILE       lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
);
STDMETHODIMP_(ULONG) OleDoc_PFile_AddRef(LPPERSISTFILE lpThis);
STDMETHODIMP_(ULONG) OleDoc_PFile_Release (LPPERSISTFILE lpThis);
STDMETHODIMP OleDoc_PFile_GetClassID (
      LPPERSISTFILE       lpThis,
      CLSID FAR*          lpclsid
);
STDMETHODIMP  OleDoc_PFile_IsDirty(LPPERSISTFILE lpThis);
STDMETHODIMP OleDoc_PFile_Load (
      LPPERSISTFILE       lpThis,
      LPCOLESTR           lpszFileName,
      DWORD               grfMode
);
STDMETHODIMP OleDoc_PFile_Save (
      LPPERSISTFILE       lpThis,
      LPCOLESTR           lpszFileName,
      BOOL                fRemember
);
STDMETHODIMP OleDoc_PFile_SaveCompleted (
      LPPERSISTFILE       lpThis,
      LPCOLESTR           lpszFileName
);
STDMETHODIMP OleDoc_PFile_GetCurFile (
      LPPERSISTFILE   lpThis,
      LPOLESTR FAR*   lplpszFileName
);

/* OleDoc::IOleItemContainer methods (functions) */
STDMETHODIMP OleDoc_ItemCont_QueryInterface(
      LPOLEITEMCONTAINER  lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
);
STDMETHODIMP_(ULONG) OleDoc_ItemCont_AddRef(LPOLEITEMCONTAINER lpThis);
STDMETHODIMP_(ULONG) OleDoc_ItemCont_Release(LPOLEITEMCONTAINER lpThis);
STDMETHODIMP OleDoc_ItemCont_ParseDisplayName(
      LPOLEITEMCONTAINER  lpThis,
      LPBC                lpbc,
      LPOLESTR            lpszDisplayName,
      ULONG FAR*          lpchEaten,
      LPMONIKER FAR*      lplpmkOut
);

STDMETHODIMP OleDoc_ItemCont_EnumObjects(
      LPOLEITEMCONTAINER  lpThis,
      DWORD               grfFlags,
      LPENUMUNKNOWN FAR*  lplpenumUnknown
);
STDMETHODIMP OleDoc_ItemCont_LockContainer(
      LPOLEITEMCONTAINER  lpThis,
      BOOL                fLock
);
STDMETHODIMP OleDoc_ItemCont_GetObject(
      LPOLEITEMCONTAINER  lpThis,
      LPOLESTR            lpszItem,
      DWORD               dwSpeedNeeded,
      LPBINDCTX           lpbc,
      REFIID              riid,
      LPVOID FAR*         lplpvObject
);
STDMETHODIMP OleDoc_ItemCont_GetObjectStorage(
      LPOLEITEMCONTAINER  lpThis,
      LPOLESTR            lpszItem,
      LPBINDCTX           lpbc,
      REFIID              riid,
      LPVOID FAR*         lplpvStorage
);
STDMETHODIMP OleDoc_ItemCont_IsRunning(
      LPOLEITEMCONTAINER  lpThis,
      LPOLESTR            lpszItem
);

/* OleDoc::IPersistFile methods (functions) */
STDMETHODIMP OleDoc_ExtConn_QueryInterface(
      LPEXTERNALCONNECTION    lpThis,
      REFIID                  riid,
      LPVOID FAR*             lplpvObj
);
STDMETHODIMP_(ULONG) OleDoc_ExtConn_AddRef(LPEXTERNALCONNECTION lpThis);
STDMETHODIMP_(ULONG) OleDoc_ExtConn_Release (LPEXTERNALCONNECTION lpThis);
STDMETHODIMP_(DWORD) OleDoc_ExtConn_AddConnection(
      LPEXTERNALCONNECTION    lpThis,
      DWORD                   extconn,
      DWORD                   reserved
);
STDMETHODIMP_(DWORD) OleDoc_ExtConn_ReleaseConnection(
      LPEXTERNALCONNECTION    lpThis,
      DWORD                   extconn,
      DWORD                   reserved,
      BOOL                    fLastReleaseCloses
);

/* OleDoc::IDataObject methods (functions) */
STDMETHODIMP OleDoc_DataObj_QueryInterface (
      LPDATAOBJECT        lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
);
STDMETHODIMP_(ULONG) OleDoc_DataObj_AddRef(LPDATAOBJECT lpThis);
STDMETHODIMP_(ULONG) OleDoc_DataObj_Release (LPDATAOBJECT lpThis);
STDMETHODIMP OleDoc_DataObj_GetData (
      LPDATAOBJECT        lpThis,
      LPFORMATETC         lpFormatetc,
      LPSTGMEDIUM         lpMedium
);
STDMETHODIMP OleDoc_DataObj_GetDataHere (
      LPDATAOBJECT        lpThis,
      LPFORMATETC         lpFormatetc,
      LPSTGMEDIUM         lpMedium
);
STDMETHODIMP OleDoc_DataObj_QueryGetData (
      LPDATAOBJECT        lpThis,
      LPFORMATETC         lpFormatetc
);
STDMETHODIMP OleDoc_DataObj_GetCanonicalFormatEtc(
      LPDATAOBJECT        lpThis,
      LPFORMATETC         lpformatetc,
      LPFORMATETC         lpformatetcOut
);
STDMETHODIMP OleDoc_DataObj_SetData (
      LPDATAOBJECT    lpThis,
      LPFORMATETC     lpFormatetc,
      LPSTGMEDIUM     lpMedium,
      BOOL            fRelease
);
STDMETHODIMP OleDoc_DataObj_EnumFormatEtc(
      LPDATAOBJECT            lpThis,
      DWORD                   dwDirection,
      LPENUMFORMATETC FAR*    lplpenumFormatEtc
);
STDMETHODIMP OleDoc_DataObj_DAdvise(
      LPDATAOBJECT        lpThis,
      FORMATETC FAR*      lpFormatetc,
      DWORD               advf,
      LPADVISESINK        lpAdvSink,
      DWORD FAR*          lpdwConnection
);
STDMETHODIMP OleDoc_DataObj_DUnadvise(LPDATAOBJECT lpThis,DWORD
dwConnection);
STDMETHODIMP OleDoc_DataObj_EnumDAdvise(
      LPDATAOBJECT        lpThis,
      LPENUMSTATDATA FAR* lplpenumAdvise
);


#ifdef USE_DRAGDROP

/* OleDoc::IDropSource methods (functions) */
STDMETHODIMP OleDoc_DropSource_QueryInterface(
   LPDROPSOURCE            lpThis,
   REFIID                  riid,
   LPVOID FAR*             lplpvObj
);
STDMETHODIMP_(ULONG) OleDoc_DropSource_AddRef( LPDROPSOURCE lpThis );
STDMETHODIMP_(ULONG) OleDoc_DropSource_Release ( LPDROPSOURCE lpThis);
STDMETHODIMP    OleDoc_DropSource_QueryContinueDrag (
   LPDROPSOURCE            lpThis,
   BOOL                    fEscapePressed,
   DWORD                   grfKeyState
);
STDMETHODIMP    OleDoc_DropSource_GiveFeedback (
   LPDROPSOURCE            lpThis,
   DWORD                   dwEffect
);

/* OleDoc::IDropTarget methods (functions) */
STDMETHODIMP OleDoc_DropTarget_QueryInterface(
      LPDROPTARGET        lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
);
STDMETHODIMP_(ULONG) OleDoc_DropTarget_AddRef(LPDROPTARGET lpThis);
STDMETHODIMP_(ULONG) OleDoc_DropTarget_Release ( LPDROPTARGET lpThis);
STDMETHODIMP    OleDoc_DropTarget_DragEnter (
   LPDROPTARGET            lpThis,
   LPDATAOBJECT            lpDataObj,
   DWORD                   grfKeyState,
   POINTL                  pointl,
   LPDWORD                 lpdwEffect
);
STDMETHODIMP    OleDoc_DropTarget_DragOver (
   LPDROPTARGET            lpThis,
   DWORD                   grfKeyState,
   POINTL                  pointl,
   LPDWORD                 lpdwEffect
);
STDMETHODIMP    OleDoc_DropTarget_DragLeave ( LPDROPTARGET lpThis);
STDMETHODIMP    OleDoc_DropTarget_Drop (
   LPDROPTARGET            lpThis,
   LPDATAOBJECT            lpDataObj,
   DWORD                   grfKeyState,
   POINTL                  pointl,
   LPDWORD                 lpdwEffect
);

#endif  // USE_DRAGDROP


/*************************************************************************
** class APPCLASSFACTORY
**  APPCLASSFACTORY implements the IClassFactory interface. it
**    instantiates document instances of the correct type depending on
**    how the application is compiled (either ServerDoc or ContainerDoc
**    instances). by implementing this
**    interface in a seperate interface from the App object itself, it
**    is easier to manage when the IClassFactory should be
**    registered/revoked. when the OleApp object is first initialized
**    in OleApp_InitInstance an instance of APPCLASSFACTORY is created
**    and registered (CoRegisterClassObject called). when the App
**    object gets destroyed (in OleApp_Destroy) this APPCLASSFACTORY is
**    revoked (CoRevokeClassObject called) and released. the simple
**    fact that the IClassFactory is registered does not on its own keep
**    the application alive.
*************************************************************************/

typedef struct tagAPPCLASSFACTORY {
   IClassFactoryVtbl FAR*  m_lpVtbl;
   UINT                    m_cRef;
#if defined( _DEBUG )
   LONG                    m_cSvrLock; // total count of LockServer locks
                              //  (for debugging purposes only)
#endif
 } APPCLASSFACTORY, FAR* LPAPPCLASSFACTORY;

/* PUBLIC FUNCTIONS */
LPCLASSFACTORY WINAPI AppClassFactory_Create(void);

/* interface IClassFactory implementation */
STDMETHODIMP AppClassFactory_QueryInterface(
      LPCLASSFACTORY lpThis, REFIID riid, LPVOID FAR* ppvObj);
STDMETHODIMP_(ULONG) AppClassFactory_AddRef(LPCLASSFACTORY lpThis);
STDMETHODIMP_(ULONG) AppClassFactory_Release(LPCLASSFACTORY lpThis);
STDMETHODIMP AppClassFactory_CreateInstance (
      LPCLASSFACTORY      lpThis,
      LPUNKNOWN           lpUnkOuter,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
);
STDMETHODIMP AppClassFactory_LockServer (
      LPCLASSFACTORY      lpThis,
      BOOL                fLock
);


/*************************************************************************
** class OLEAPP : OUTLINEAPP
**    OLEAPP is an extention to the base OUTLINEAPP object (structure)
**    that adds common OLE 2.0 functionality used by both the server
**    and container versions. This is an abstract class. You do not
**    instantiate an instance of OLEAPP directly but instead
**    instantiate one of its concrete subclasses: SERVERAPP or
**    CONTAINERAPP. There is one instance of an document application
**    object created per running application instance. This
**    object holds many fields that could otherwise be organized as
**    global variables. The OLEAPP class inherits all fields
**    from the OUTLINEAPP class. This inheritance is achieved by including a
**    member variable of type OUTLINEAPP as the first field in the OLEAPP
**    structure. Thus a pointer to a OLEAPP object can be cast to be
**    a pointer to a OUTLINEAPP object.
*************************************************************************/

typedef struct tagOLEAPP {
   OUTLINEAPP  m_OutlineApp;       // inherits all fields of OutlineApp
   ULONG       m_cRef;             // total ref count for app
   ULONG       m_cDoc;             // total count of open documents
   BOOL        m_fUserCtrl;        // does user control life-time of app?
   DWORD       m_dwRegClassFac;    // value returned by
CoRegisterClassObject
   LPCLASSFACTORY m_lpClassFactory;// ptr to allocated ClassFactory instance
#if defined( USE_MSGFILTER )
   LPMESSAGEFILTER m_lpMsgFilter;  // ptr to allocated MsgFilter instance
   MSGPENDINGPROC m_lpfnMsgPending;// ptr to msg pending callback function
#endif  // USE_MSGFILTER
   BOOL        m_fOleInitialized;  // was OleInitialize called
   UINT        m_cModalDlgActive;  // count of modal dialogs up; 0 = no dlg.
   UINT        m_cfEmbedSource;    // OLE 2.0 clipboard format
   UINT        m_cfEmbeddedObject; // OLE 2.0 clipboard format
   UINT        m_cfLinkSource;     // OLE 2.0 clipboard format
   UINT        m_cfObjectDescriptor; // OLE 2.0 clipboard format
   UINT        m_cfLinkSrcDescriptor; // OLE 2.0 clipboard format
   UINT        m_cfFileName;       // std Windows clipboard format
   FORMATETC   m_arrDocGetFmts[MAXNOFMTS];  // fmts offered by copy &
GetData
   UINT        m_nDocGetFmts;      // no of fmtetc's for GetData

   OLEUIPASTEENTRY m_arrPasteEntries[MAXNOFMTS];   // input for PasteSpl.
   int         m_nPasteEntries;                    // input for PasteSpl.
   UINT        m_arrLinkTypes[MAXNOLINKTYPES];     // input for PasteSpl.
   int         m_nLinkTypes;                       // input for PasteSpl.

#if defined( USE_DRAGDROP )
   int m_nDragDelay;       // time delay (in msec) before drag should start
   int m_nDragMinDist;     // min. distance (radius) before drag should
start
   int m_nScrollDelay;     // time delay (in msec) before scroll should
start
   int m_nScrollInset;     // Border inset distance to start drag scroll
   int m_nScrollInterval;  // scroll interval time (in msec)

#if defined( IF_SPECIAL_DD_CURSORS_NEEDED )
   // This would be used if the app wanted to have custom drag/drop cursors
   HCURSOR     m_hcursorDragNone;
   HCURSOR     m_hcursorDragCopy;
   HCURSOR     m_hcursorDragLink;
#endif  // IF_SPECIAL_DD_CURSORS_NEEDED
#endif  // USE_DRAGDROP


#if defined( OLE_CNTR )
   HPALETTE    m_hStdPal;        // standard color palette for OLE
                             //  it is a good idea for containers
                             //  to use this standard palette
                             //  even if they do not use colors
                             //  themselves. this will allow
                             //  embedded object to get a good
                             //  distribution of colors when they
                             //  are being drawn by the container.
                             //
#endif
```

```
   struct CAppUnknownImpl {
      IUnknownVtbl FAR*       lpVtbl;
      LPOLEAPP                lpOleApp;
      int                     cRef;   // interface specific ref count.
   } m_Unknown;

} OLEAPP;

/* ServerApp methods (functions) */
BOOL OleApp_InitInstance(LPOLEAPP lpOleApp, HINSTANCE hInst, int nCmdShow);
void OleApp_TerminateApplication(LPOLEAPP lpOleApp);
BOOL OleApp_ParseCmdLine(LPOLEAPP lpOleApp, LPSTR lpszCmdLine, int
nCmdShow);
void OleApp_Destroy(LPOLEAPP lpOleApp);
BOOL OleApp_CloseAllDocsAndExitCommand(
      LPOLEAPP            lpOleApp,
      BOOL                fForceEndSession
);
void OleApp_ShowWindow(LPOLEAPP lpOleApp, BOOL fGiveUserCtrl);
void OleApp_HideWindow(LPOLEAPP lpOleApp);
void OleApp_HideIfNoReasonToStayVisible(LPOLEAPP lpOleApp);
void OleApp_DocLockApp(LPOLEAPP lpOleApp);
void OleApp_DocUnlockApp(LPOLEAPP lpOleApp, LPOUTLINEDOC lpOutlineDoc);
HRESULT OleApp_Lock(LPOLEAPP lpOleApp, BOOL fLock, BOOL
fLastUnlockReleases);
ULONG OleApp_AddRef(LPOLEAPP lpOleApp);
ULONG OleApp_Release (LPOLEAPP lpOleApp);
HRESULT OleApp_QueryInterface (
      LPOLEAPP                lpOleApp,
      REFIID                  riid,
      LPVOID FAR*             lplpUnk
);
void OleApp_RejectInComingCalls(LPOLEAPP lpOleApp, BOOL fReject);
void OleApp_DisableBusyDialogs(
      LPOLEAPP        lpOleApp,
      BOOL FAR*       lpfPrevBusyEnable,
      BOOL FAR*       lpfPrevNREnable
);
void OleApp_EnableBusyDialogs(
      LPOLEAPP        lpOleApp,
      BOOL            fPrevBusyEnable,
      BOOL            fPrevNREnable
);
void OleApp_PreModalDialog(LPOLEAPP lpOleApp, LPOLEDOC lpActiveOleDoc);
void OleApp_PostModalDialog(LPOLEAPP lpOleApp, LPOLEDOC lpActiveOleDoc);
BOOL OleApp_InitVtbls (LPOLEAPP lpOleApp);
void OleApp_InitMenu(
      LPOLEAPP                lpOleApp,
      LPOLEDOC                lpOleDoc,
      HMENU                   hMenu
);
void OleApp_UpdateEditMenu(
      LPOLEAPP                lpOleApp,
      LPOUTLINEDOC            lpOutlineDoc,
      HMENU                   hMenuEdit
);
BOOL OleApp_RegisterClassFactory(LPOLEAPP lpOleApp);
void OleApp_RevokeClassFactory(LPOLEAPP lpOleApp);

#if defined( USE_MSGFILTER )
BOOL OleApp_RegisterMessageFilter(LPOLEAPP lpOleApp);
void OleApp_RevokeMessageFilter(LPOLEAPP lpOleApp);
BOOL FAR PASCAL EXPORT MessagePendingProc(MSG FAR *lpMsg);
#endif  // USE_MSGFILTER

void OleApp_FlushClipboard(LPOLEAPP lpOleApp);
void OleApp_NewCommand(LPOLEAPP lpOleApp);
void OleApp_OpenCommand(LPOLEAPP lpOleApp);

#if defined( OLE_CNTR )
LRESULT OleApp_QueryNewPalette(LPOLEAPP lpOleApp);
#endif // OLE_CNTR

LRESULT wSelectPalette(HWND hWnd, HPALETTE hPal, BOOL fBackground);


/* OleApp::IUnknown methods (functions) */
STDMETHODIMP OleApp_Unk_QueryInterface(
      LPUNKNOWN           lpThis,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
);
STDMETHODIMP_(ULONG) OleApp_Unk_AddRef(LPUNKNOWN lpThis);
STDMETHODIMP_(ULONG) OleApp_Unk_Release (LPUNKNOWN lpThis);


/* Function prototypes in debug.c */
void InstallMessageFilterCommand(void);
void RejectIncomingCommand(void);


#if defined( OLE_SERVER )
#include "svroutl.h"
#endif
#if defined( OLE_CNTR )
#include "cntroutl.h"
#endif

#endif // _OLEOUTL_H_
_
```

OUTLAPP.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    outlapp.c
**
**    This file contains OutlineApp functions.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"
#include "ansiapi.h"

#if defined( USE_STATUSBAR )
#include "status.h"
#endif

#if !defined( WIN32 )
#include <print.h>
#endif

OLEDBGDATA

extern LPOUTLINEAPP g_lpApp;
extern RECT g_rectNull;


// REVIEW: should use string resource for messages
OLECHAR ErrMsgClass[] = OLESTR("Can't register window classes!");
OLECHAR ErrMsgFrame[] = OLESTR("Can't create Frame Window!");
OLECHAR ErrMsgPrinting[] = OLESTR("Can't access printer!");


/* OutlineApp_InitApplication
** --------------------------
** Sets up the class data structures and does a one-time
**      initialization of the app by registering the window classes.
**      Returns TRUE if initialization is successful
**              FALSE otherwise
*/

BOOL OutlineApp_InitApplication(LPOUTLINEAPP lpOutlineApp, HINSTANCE hInst)
{
   WNDCLASS    wndclass;

   // REVIEW: should load msg strings from string resource

   /* Register the app frame class */
   wndclass.style = CS_HREDRAW | CS_VREDRAW | CS_BYTEALIGNWINDOW;
   wndclass.lpfnWndProc = AppWndProc;
   /* Extra storage for Class and Window objects */
   wndclass.cbClsExtra = 0;
   wndclass.cbWndExtra = sizeof(LPOUTLINEAPP); /* to store lpApp */
   wndclass.hInstance = hInst;
   wndclass.hIcon = LoadIcon(hInst, APPICON);
   wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
   /* Create brush for erasing background */
   wndclass.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
   wndclass.lpszMenuName = APPMENU;     /* Menu Name is App Name */
   wndclass.lpszClassName = APPWNDCLASS; /* Class Name is App Name */

   if(! RegisterClass(&wndclass)) {
      OutlineApp_ErrorMessage(lpOutlineApp, ErrMsgFrame);
      return FALSE;
   }

   /* Register the document window class */
   wndclass.style = CS_BYTEALIGNWINDOW;
   wndclass.lpfnWndProc = DocWndProc;
   wndclass.hIcon = NULL;
   wndclass.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
   wndclass.lpszMenuName = NULL;
   wndclass.lpszClassName = DOCWNDCLASS;
   wndclass.cbWndExtra = sizeof(LPOUTLINEDOC); /* to store lpDoc */
   if(! RegisterClass(&wndclass)) {
      OutlineApp_ErrorMessage(lpOutlineApp, ErrMsgClass);
      return FALSE;
   }

#if defined( USE_STATUSBAR )
   if (! RegisterStatusClass(hInst))
      return FALSE;
#endif

#if defined( USE_FRAMETOOLS )
   if (! FrameToolsRegisterClass(hInst)) {
      return FALSE;
   }
#endif

#if defined( INPLACE_SVR )
   // We should only register the hatch window class
   // in the UI Library once per application.
   RegisterHatchWindowClass(hInst);

#endif

   return TRUE;
}


/* OutlineApp_InitInstance
 * -----------------------
 *
 *  Performs a per-instance initialization of app.
 *  This method creates the frame window.
 *
 *  RETURNS    : TRUE  - If initialization was successful.
 *               FALSE - otherwise.
 */

BOOL OutlineApp_InitInstance(LPOUTLINEAPP lpOutlineApp, HINSTANCE hInst, int
nCmdShow)
{
   char  szAnsiString[256];
   lpOutlineApp->m_hInst = hInst;

   /* create application's Frame window */
   W2A (APPNAME, szAnsiString, 256);
   lpOutlineApp->m_hWndApp = CreateWindow(
         APPWNDCLASS,             /* Window class name */
         szAnsiString,            /* initial Window title */
         WS_OVERLAPPEDWINDOW|
         WS_CLIPCHILDREN,
         CW_USEDEFAULT, 0,        /* Use default X, Y            */
         CW_USEDEFAULT, 0,        /* Use default X, Y            */
         HWND_DESKTOP,            /* Parent window's handle      */
         NULL,                    /* Default to Class Menu       */
         hInst,                   /* Instance of window          */
         NULL                     /* Create struct for WM_CREATE */
   );

   if(! lpOutlineApp->m_hWndApp) {
      // REVIEW: should load string from string resource
      OutlineApp_ErrorMessage(lpOutlineApp, ErrMsgFrame);
      return FALSE;
   }

   SetWindowLong(lpOutlineApp->m_hWndApp, 0, (LONG) lpOutlineApp);

   /* defer creating the user's SDI document until we parse the cmd line. */
   lpOutlineApp->m_lpDoc = NULL;

   /* Initialize clipboard.
   */
   lpOutlineApp->m_lpClipboardDoc = NULL;
   W2A (OUTLINEDOCFORMAT, szAnsiString, 256);
   if(!(lpOutlineApp->m_cfOutline = RegisterClipboardFormat(szAnsiString)))
{
      // REVIEW: should load string from string resource
      OutlineApp_ErrorMessage(lpOutlineApp, OLESTR("Can't register clipboard
format!"));
      return FALSE;
   }

   /* init the standard font to be used for drawing/printing text
   *       request a Roman style True Type font of the desired size
   */
   lpOutlineApp->m_hStdFont = CreateFont(
         -DEFFONTSIZE,
         0,0,0,0,0,0,0,0,
         OUT_TT_PRECIS,      // use TrueType
         CLIP_DEFAULT_PRECIS,
         DEFAULT_QUALITY,
         VARIABLE_PITCH | FF_ROMAN,
         DEFFONTFACE
   );

   // Load special cursor for selection of Lines in ListBox.
   lpOutlineApp->m_hcursorSelCur = LoadCursor ( hInst, "SelCur" );

   /* init the Print Dialog structure */
   _fmemset((LPVOID)&lpOutlineApp->m_PrintDlg,0,sizeof(PRINTDLG));
   lpOutlineApp->m_PrintDlg.lStructSize = sizeof(PRINTDLG);
   lpOutlineApp->m_PrintDlg.hDevMode = NULL;
   lpOutlineApp->m_PrintDlg.hDevNames = NULL;
   lpOutlineApp->m_PrintDlg.Flags = PD_RETURNDC | PD_NOSELECTION |
PD_NOPAGENUMS |
               PD_HIDEPRINTTOFILE;
   lpOutlineApp->m_PrintDlg.nCopies = 1;
   lpOutlineApp->m_PrintDlg.hwndOwner = lpOutlineApp->m_hWndApp;

#if defined( USE_STATUSBAR )
   lpOutlineApp->m_hWndStatusBar = CreateStatusWindow(lpOutlineApp-
>m_hWndApp, hInst);
   if (! lpOutlineApp->m_hWndStatusBar)
      return FALSE;

   lpOutlineApp->m_hMenuApp = GetMenu(lpOutlineApp->m_hWndApp);

   /* setup status messages for the application menus */
   {
      HMENU hMenuFile = GetSubMenu(lpOutlineApp->m_hMenuApp, 0);
      HMENU hMenuEdit = GetSubMenu(lpOutlineApp->m_hMenuApp, 1);
      HMENU hMenuOutline = GetSubMenu(lpOutlineApp->m_hMenuApp, 2);
      HMENU hMenuLine = GetSubMenu(lpOutlineApp->m_hMenuApp, 3);
      HMENU hMenuName = GetSubMenu(lpOutlineApp->m_hMenuApp, 4);
      HMENU hMenuOptions = GetSubMenu(lpOutlineApp->m_hMenuApp, 5);
      HMENU hMenuDebug = GetSubMenu(lpOutlineApp->m_hMenuApp, 6);
      HMENU hMenuHelp = GetSubMenu(lpOutlineApp->m_hMenuApp, 7);
      HMENU hMenuSys = GetSystemMenu(lpOutlineApp->m_hWndApp, FALSE);

      AssignPopupMessage(hMenuFile, "Create, open, save, print outlines or
quit application");
      AssignPopupMessage(hMenuEdit, "Cut, copy, paste or clear selection");
      AssignPopupMessage(hMenuOutline, "Set zoom and margins");
      AssignPopupMessage(hMenuLine, "Create, edit, and indent lines");
      AssignPopupMessage(hMenuName, "Create, edit, delete and goto names");
      AssignPopupMessage(hMenuOptions, "Modify tools, row/col headings,
display options");
      AssignPopupMessage(hMenuDebug, "Set debug trace level and other debug
options");
      AssignPopupMessage(hMenuHelp, "Get help on using the application");
      AssignPopupMessage(hMenuSys,"Move, size or close application window");
   }
```

```
#endif

#if defined ( USE_FRAMETOOLS ) || defined ( INPLACE_CNTR )
   lpOutlineApp->m_FrameToolWidths = g_rectNull;
#endif  // USE_FRAMETOOLS || INPLACE_CNTR

#if defined( USE_FRAMETOOLS )
   if (! FrameTools_Init(&lpOutlineApp->m_frametools,
         lpOutlineApp->m_hWndApp, lpOutlineApp->m_hInst))
      return FALSE;
#endif

#if defined( OLE_VERSION )

   /* OLE2NOTE: perform initialization required for OLE */
   if (! OleApp_InitInstance((LPOLEAPP)lpOutlineApp, hInst, nCmdShow))
      return FALSE;
#else
   /* OLE2NOTE: Although no OLE call is made in the base outline,
   **    OLE memory allocator is used and thus CoInitialize() needs to
   **    be called.
   */
   {
      HRESULT hrErr;

      hrErr = CoInitialize(NULL);
      if (hrErr != NOERROR) {
         OutlineApp_ErrorMessage(lpOutlineApp,
               OLESTR("CoInitialize initialization failed!"));
         return FALSE;
      }
   }
#endif

   return TRUE;
}


/* OutlineApp_ParseCmdLine
 * -----------------------
 *
 * Parse the command line for any execution flags/arguments.
 */
BOOL OutlineApp_ParseCmdLine(LPOUTLINEAPP lpOutlineApp, LPSTR lpszCmdLine,
int nCmdShow)
{

#if defined( OLE_VERSION )
   // Call OLE version of this function instead
   return OleApp_ParseCmdLine((LPOLEAPP)lpOutlineApp,lpszCmdLine,nCmdShow);

#else

   BOOL fStatus = TRUE;
   char szFileName[256];   /* buffer for filename in command line */
   OLECHAR szTempW[256];   /* buffer for unicode/ansi strings*/

   szFileName[0] = '\0';
   ParseCmdLine(lpszCmdLine, NULL, (LPSTR)szFileName);

   if(*szFileName) {
      // allocate a new document
      lpOutlineApp->m_lpDoc = OutlineApp_CreateDoc(lpOutlineApp, FALSE);
      if (! lpOutlineApp->m_lpDoc) goto error;

      // open the specified file
      A2W(szFileName, szTempW, 256);
      if (! OutlineDoc_LoadFromFile(lpOutlineApp->m_lpDoc, szTempW))
         goto error;
   } else {
      // create a new document
      lpOutlineApp->m_lpDoc = OutlineApp_CreateDoc(lpOutlineApp, FALSE);
      if (! lpOutlineApp->m_lpDoc) goto error;

      // set the doc to an (Untitled) doc.
      if (! OutlineDoc_InitNewFile(lpOutlineApp->m_lpDoc))
         goto error;
   }

   // position and size the new doc window
   OutlineApp_ResizeWindows(lpOutlineApp);
   OutlineDoc_ShowWindow(lpOutlineApp->m_lpDoc);

   // show main app window
   ShowWindow(lpOutlineApp->m_hWndApp, nCmdShow);
   UpdateWindow(lpOutlineApp->m_hWndApp);

   return TRUE;

error:
   // REVIEW: should load string from string resource
   OutlineApp_ErrorMessage(lpOutlineApp, OLESTR("Could not create new
document"));

   if (lpOutlineApp->m_lpDoc) {
      OutlineDoc_Destroy(lpOutlineApp->m_lpDoc);
      lpOutlineApp->m_lpDoc = NULL;
   }

   return FALSE;

#endif
}


/* OutlineApp_InitMenu
 * -------------------
 *
 *      Enable or Disable menu items depending on the state of
 * the appliation
 */
void OutlineApp_InitMenu(LPOUTLINEAPP lpOutlineApp, LPOUTLINEDOC
lpOutlineDoc, HMENU hMenu)
{
   WORD status;
   static UINT     uCurrentZoom = (UINT)-1;
   static UINT     uCurrentMargin = (UINT)-1;
   static UINT     uBBState = (UINT)-1;
   static UINT     uFBState = (UINT)-1;

   if (!lpOutlineApp || !lpOutlineDoc || !hMenu)
      return;

   EnableMenuItem(lpOutlineApp->m_hMenuApp, IDM_E_UNDO, MF_GRAYED);

   status = (WORD)(OutlineDoc_GetLineCount(lpOutlineDoc) ? MF_ENABLED :
MF_GRAYED);

   EnableMenuItem(lpOutlineApp->m_hMenuApp, IDM_E_CUT ,status);
   EnableMenuItem(lpOutlineApp->m_hMenuApp, IDM_E_COPY ,status);
   EnableMenuItem(lpOutlineApp->m_hMenuApp, IDM_E_CLEAR ,status);
   EnableMenuItem(lpOutlineApp->m_hMenuApp, IDM_E_SELECTALL ,status);

   EnableMenuItem(lpOutlineApp->m_hMenuApp, IDM_L_EDITLINE ,status);
   EnableMenuItem(lpOutlineApp->m_hMenuApp, IDM_L_INDENTLINE ,status);
   EnableMenuItem(lpOutlineApp->m_hMenuApp, IDM_L_UNINDENTLINE ,status);
   EnableMenuItem(lpOutlineApp->m_hMenuApp, IDM_L_SETLINEHEIGHT ,status);

   EnableMenuItem(lpOutlineApp->m_hMenuApp, IDM_N_DEFINENAME ,status);

   status = (WORD)(OutlineDoc_GetNameCount(lpOutlineDoc) ? MF_ENABLED :
MF_GRAYED);

   EnableMenuItem(lpOutlineApp->m_hMenuApp, IDM_N_GOTONAME, status);

   if (uCurrentZoom != (UINT)-1)
      CheckMenuItem(lpOutlineApp->m_hMenuApp, uCurrentZoom, MF_UNCHECKED);
   uCurrentZoom = OutlineDoc_GetCurrentZoomMenuCheck(lpOutlineDoc);
   CheckMenuItem(lpOutlineApp->m_hMenuApp, uCurrentZoom, MF_CHECKED);

   if (uCurrentMargin != (UINT)-1)
      CheckMenuItem(lpOutlineApp->m_hMenuApp, uCurrentMargin, MF_UNCHECKED);
   uCurrentMargin = OutlineDoc_GetCurrentMarginMenuCheck(lpOutlineDoc);
   CheckMenuItem(lpOutlineApp->m_hMenuApp, uCurrentMargin, MF_CHECKED);

#if defined( USE_FRAMETOOLS )
   if (uBBState != (UINT)-1)
      CheckMenuItem(lpOutlineApp->m_hMenuApp, uBBState, MF_UNCHECKED);
   if (lpOutlineDoc->m_lpFrameTools) {
      switch (FrameTools_BB_GetState(lpOutlineDoc->m_lpFrameTools)) {
         case BARSTATE_TOP:
            uBBState = IDM_O_BB_TOP;
            break;
         case BARSTATE_BOTTOM:
            uBBState = IDM_O_BB_BOTTOM;
            break;
         case BARSTATE_POPUP:
            uBBState = IDM_O_BB_POPUP;
            break;
         case BARSTATE_HIDE:
            uBBState = IDM_O_BB_HIDE;
            break;
      }
      CheckMenuItem(lpOutlineApp->m_hMenuApp, uBBState, MF_CHECKED);
   }

   if (uFBState != (UINT)-1)
      CheckMenuItem(lpOutlineApp->m_hMenuApp, uFBState, MF_UNCHECKED);
   if (lpOutlineDoc->m_lpFrameTools) {
      switch (FrameTools_FB_GetState(lpOutlineDoc->m_lpFrameTools)) {
         case BARSTATE_TOP:
            uFBState = IDM_O_FB_TOP;
            break;
         case BARSTATE_BOTTOM:
            uFBState = IDM_O_FB_BOTTOM;
            break;
         case BARSTATE_POPUP:
            uFBState = IDM_O_FB_POPUP;
            break;
      }
      CheckMenuItem(lpOutlineApp->m_hMenuApp, uFBState, MF_CHECKED);
   }
#endif  // USE_FRAMETOOLS

#if defined( OLE_VERSION )
   /* OLE2NOTE: perform OLE specific menu initialization.
   **    the OLE versions use the OleGetClipboard mechanism for
   **    clipboard handling. thus, they determine if the Paste and
   **    PasteSpecial commands should be enabled in an OLE specific
   **    manner.
   **    (Container only) build the OLE object verb menu if necessary.
   */
   OleApp_InitMenu(
         (LPOLEAPP)lpOutlineApp,
         (LPOLEDOC)lpOutlineDoc,
         lpOutlineApp->m_hMenuApp
   );

   /* OLE2NOTE: To avoid the overhead of initializing the Edit menu,
   **    we do it only when it is popped up. Thus we just set a flag
   **    in the OleDoc saying that the Edit menu needs to be updated
   **    but we don't do it immediately
   */
   OleDoc_SetUpdateEditMenuFlag((LPOLEDOC)lpOutlineDoc, TRUE);

#else
   // Base Outline version uses standard Windows clipboard handling
   if(IsClipboardFormatAvailable(lpOutlineApp->m_cfOutline) ||
      IsClipboardFormatAvailable(CF_TEXT))
      status = MF_ENABLED;
   else
      status = MF_GRAYED;
   EnableMenuItem(lpOutlineApp->m_hMenuApp, IDM_E_PASTE, status);

#endif

#if defined( USE_FRAMETOOLS )
   if (! OutlineDoc_IsEditFocusInFormulaBar(lpOutlineDoc)) {
      EnableMenuItem(lpOutlineApp->m_hMenuApp, IDM_L_ADDLINE, MF_GRAYED);
      EnableMenuItem(lpOutlineApp->m_hMenuApp, IDM_L_EDITLINE, MF_GRAYED);
   }
   else
      EnableMenuItem(lpOutlineApp->m_hMenuApp, IDM_L_ADDLINE, MF_ENABLED);

#endif      // USE_FRAMETOOLS

}


/* OutlineApp_GetWindow
 * --------------------
 *
 *      Get the window handle of the application frame.
 */
HWND OutlineApp_GetWindow(LPOUTLINEAPP lpOutlineApp)
{
   if (!lpOutlineApp)
      return NULL;

   return lpOutlineApp->m_hWndApp;
}


/* OutlineApp_GetFrameWindow
** -------------------------
**    Gets the current frame window to use as a parent to any dialogs
**    this app uses.
**
**    OLE2NOTE: normally this is simply the main hWnd of the app. but,
**    if the app is currently supporting an in-place server document,
**    then the frame window of the top in-place container must be used.
*/
HWND OutlineApp_GetFrameWindow(LPOUTLINEAPP lpOutlineApp)
{
   HWND hWndApp = OutlineApp_GetWindow(lpOutlineApp);

#if defined( INPLACE_SVR )
   LPSERVERDOC lpServerDoc =
         (LPSERVERDOC)OutlineApp_GetActiveDoc(lpOutlineApp);
   if (lpServerDoc && lpServerDoc->m_fUIActive)
      return lpServerDoc->m_lpIPData->frameInfo.hwndFrame;
#endif

   return hWndApp;
}
```

```
/* OutlineApp_GetInstance
 * ----------------------
 *
 *      Get the process instance of the application.
 */
HINSTANCE OutlineApp_GetInstance(LPOUTLINEAPP lpOutlineApp)
{
   if (!lpOutlineApp)
      return NULL;

   return lpOutlineApp->m_hInst;
}


/* OutlineApp_CreateDoc
 * --------------------
 *
 * Allocate a new document of the appropriate type.
 *  OutlineApp  --> creates OutlineDoc type documents
 *
 *      Returns lpOutlineDoc for successful, NULL if error.
 */
LPOUTLINEDOC OutlineApp_CreateDoc(
      LPOUTLINEAPP    lpOutlineApp,
      BOOL            fDataTransferDoc
)
{
   LPOUTLINEDOC lpOutlineDoc;

   OLEDBG_BEGIN3("OutlineApp_CreateDoc\r\n")

#if defined( OLE_SERVER )
   lpOutlineDoc = (LPOUTLINEDOC)New((DWORD)sizeof(SERVERDOC));
   _fmemset(lpOutlineDoc, 0, sizeof(SERVERDOC));
#endif
#if defined( OLE_CNTR )
   lpOutlineDoc = (LPOUTLINEDOC)New((DWORD)sizeof(CONTAINERDOC));
   _fmemset(lpOutlineDoc, 0, sizeof(CONTAINERDOC));
#endif
#if !defined( OLE_VERSION )
   lpOutlineDoc = (LPOUTLINEDOC)New((DWORD)sizeof(OUTLINEDOC));
   _fmemset(lpOutlineDoc, 0, sizeof(OUTLINEDOC));
#endif

   OleDbgAssertSz(lpOutlineDoc != NULL, "Error allocating OutlineDoc");
   if (lpOutlineDoc == NULL)
      return NULL;

   // initialize new document
   if (! OutlineDoc_Init(lpOutlineDoc, fDataTransferDoc))
      goto error;

   OLEDBG_END3
   return lpOutlineDoc;

error:
   if (lpOutlineDoc)
      Delete(lpOutlineDoc);

   OLEDBG_END3
   return NULL;
}


/* OutlineApp_CreateName
 * ---------------------
 *
 * Allocate a new Name of the appropriate type.
 *  OutlineApp --> creates standard OutlineName type names.
 *  ServerApp  --> creates enhanced SeverName type names.
 *
 *      Returns lpOutlineName for successful, NULL if error.
 */
LPOUTLINENAME OutlineApp_CreateName(LPOUTLINEAPP lpOutlineApp)
{
   LPOUTLINENAME lpOutlineName;

#if defined( OLE_SERVER )
   lpOutlineName = (LPOUTLINENAME)New((DWORD)sizeof(SERVERNAME));
#else
   lpOutlineName = (LPOUTLINENAME)New((DWORD)sizeof(OUTLINENAME));
#endif

   OleDbgAssertSz(lpOutlineName != NULL, "Error allocating Name");
   if (lpOutlineName == NULL)
      return NULL;

#if defined( OLE_SERVER )
   _fmemset((LPVOID)lpOutlineName,0,sizeof(SERVERNAME));
#else
   _fmemset((LPVOID)lpOutlineName,0,sizeof(OUTLINENAME));
#endif

   return lpOutlineName;
}


/* OutlineApp_DocUnlockApp
** -----------------------
**    Forget all references to a closed document.
*/
void OutlineApp_DocUnlockApp(LPOUTLINEAPP lpOutlineApp, LPOUTLINEDOC
lpOutlineDoc)
{
   /* forget pointers to destroyed document */
   if (lpOutlineApp->m_lpDoc == lpOutlineDoc)
      lpOutlineApp->m_lpDoc = NULL;
   else if (lpOutlineApp->m_lpClipboardDoc == lpOutlineDoc)
```

```
      lpOutlineApp->m_lpClipboardDoc = NULL;

#if defined( OLE_VERSION )
   /* OLE2NOTE: when there are no open documents and the app is not
   **    under the control of the user then revoke our ClassFactory to
   **    enable the app to shut down.
   **
   **    NOTE: data transfer documents (non-user documents) do NOT
   **    hold the app alive. therefore they do not Lock the app.
   */
   if (! lpOutlineDoc->m_fDataTransferDoc)
      OleApp_DocUnlockApp((LPOLEAPP)lpOutlineApp, lpOutlineDoc);
#endif
}


/* OutlineApp_NewCommand
 * ---------------------
 *
 *  Start a new untitled document (File.New command).
 */
void OutlineApp_NewCommand(LPOUTLINEAPP lpOutlineApp)
{
#if defined( OLE_VERSION )
   // Call OLE version of this function instead
   OleApp_NewCommand((LPOLEAPP)lpOutlineApp);

#else

   LPOUTLINEDOC lpOutlineDoc = lpOutlineApp->m_lpDoc;

   if (! OutlineDoc_Close(lpOutlineDoc, OLECLOSE_PROMPTSAVE))
      return;

   OleDbgAssertSz(lpOutlineApp->m_lpDoc==NULL,"Closed doc NOT properly
destroyed");

   lpOutlineApp->m_lpDoc = OutlineApp_CreateDoc(lpOutlineApp, FALSE);
   if (! lpOutlineApp->m_lpDoc) goto error;

   // set the doc to an (Untitled) doc.
   if (! OutlineDoc_InitNewFile(lpOutlineApp->m_lpDoc))
      goto error;

   // position and size the new doc window
   OutlineApp_ResizeWindows(lpOutlineApp);
   OutlineDoc_ShowWindow(lpOutlineApp->m_lpDoc); // calls OleDoc_Lock

   return;

error:
   // REVIEW: should load string from string resource
   OutlineApp_ErrorMessage(lpOutlineApp, OLESTR("Could not create new
document"));
```

```
   if (lpOutlineApp->m_lpDoc) {
      OutlineDoc_Destroy(lpOutlineApp->m_lpDoc);
      lpOutlineApp->m_lpDoc = NULL;
   }

   return;

#endif
}


/* OutlineApp_OpenCommand
 * ----------------------
 *
 *  Load a document from file (File.Open command).
 */
void OutlineApp_OpenCommand(LPOUTLINEAPP lpOutlineApp)
{
#if defined( OLE_VERSION )
   // Call OLE version of this function instead
   OleApp_OpenCommand((LPOLEAPP)lpOutlineApp);

#else

   OPENFILENAME ofn;
   char szFilter[]=APPFILENAMEFILTER;
   char szFileName[256];
   OLECHAR szUniFileName[256];
   UINT i;
   DWORD dwSaveOption = OLECLOSE_PROMPTSAVE;
   BOOL fStatus = TRUE;

   if (! OutlineDoc_CheckSaveChanges(lpOutlineApp->m_lpDoc, &dwSaveOption))
      return;           // abort opening new doc

   for(i=0; szFilter[i]; i++)
      if(szFilter[i]=='|') szFilter[i]='\0';

   _fmemset((LPOPENFILENAME)&ofn,0,sizeof(OPENFILENAME));

   szFileName[0]='\0';

   ofn.lStructSize=sizeof(OPENFILENAME);
   ofn.hwndOwner=lpOutlineApp->m_hWndApp;
   ofn.lpstrFilter=(LPSTR)szFilter;
   ofn.lpstrFile=(LPSTR)szFileName;
   ofn.nMaxFile=256;
   ofn.Flags=OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;
   ofn.lpstrDefExt=DEFEXTENSION;

   if(! GetOpenFileName((LPOPENFILENAME)&ofn))
      return;         // user canceled file open dialog

   OutlineDoc_Close(lpOutlineApp->m_lpDoc, OLECLOSE_NOSAVE);
   OleDbgAssertSz(lpOutlineApp->m_lpDoc==NULL,"Closed doc NOT properly
destroyed");

   lpOutlineApp->m_lpDoc = OutlineApp_CreateDoc(lpOutlineApp, FALSE);
   if (! lpOutlineApp->m_lpDoc) goto error;

   A2W (szFileName, szUniFileName, 256);
   fStatus=OutlineDoc_LoadFromFile(lpOutlineApp->m_lpDoc,
(LPOLESTR)szUniFileName);

   if (! fStatus) {
      // loading the doc failed; create an untitled instead
      OutlineDoc_Destroy(lpOutlineApp->m_lpDoc);  // destroy unused doc
      lpOutlineApp->m_lpDoc = OutlineApp_CreateDoc(lpOutlineApp, FALSE);
      if (! lpOutlineApp->m_lpDoc) goto error;
      if (! OutlineDoc_InitNewFile(lpOutlineApp->m_lpDoc))
         goto error;
   }

   // position and size the new doc window
   OutlineApp_ResizeWindows(lpOutlineApp);
   OutlineDoc_ShowWindow(lpOutlineApp->m_lpDoc);

   return;

error:
   // REVIEW: should load string from string resource
   OutlineApp_ErrorMessage(lpOutlineApp, OLESTR("Could not create new
document"));

   if (lpOutlineApp->m_lpDoc) {
      OutlineDoc_Destroy(lpOutlineApp->m_lpDoc);
      lpOutlineApp->m_lpDoc = NULL;
   }

   return;

#endif
}


/* OutlineApp_PrintCommand
 * -----------------------
 *
 *      Print the document
 */
void OutlineApp_PrintCommand(LPOUTLINEAPP lpOutlineApp)
{
   LPOUTLINEDOC    lpOutlineDoc = lpOutlineApp->m_lpDoc;
   HDC             hDC=NULL;
   BOOL            fMustDeleteDC = FALSE;
   BOOL            fStatus;

#if defined( OLE_VERSION )
   OleApp_PreModalDialog(
         (LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineApp->m_lpDoc);
#endif

   fStatus = PrintDlg((LPPRINTDLG)&lpOutlineApp->m_PrintDlg);

#if defined( OLE_VERSION )
   OleApp_PostModalDialog(
         (LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineApp->m_lpDoc);
#endif

   if (!fStatus) {
      if (!CommDlgExtendedError()) {      // Cancel button pressed
         return;
      }
   }
   else {
      hDC = OutlineApp_GetPrinterDC(lpOutlineApp);
      if (hDC) {

#if defined( OLE_VERSION )
         /* OLE2NOTE: while we are printing we do NOT want to
         **    receive any OnDataChange notifications or other OLE
         **    interface calls which could disturb the printing of
         **    the document. we will temporarily reply
         **    SERVERCALL_RETRYLATER
         */
         OleApp_RejectInComingCalls((LPOLEAPP)lpOutlineApp, TRUE);
#endif

         OutlineDoc_Print(lpOutlineDoc, hDC);
         DeleteDC(hDC);

#if defined( OLE_VERSION )
         // re-enable LRPC calls
         OleApp_RejectInComingCalls((LPOLEAPP)lpOutlineApp, FALSE);
#endif

         return;                          // Printing completed
      }
   }

   // REVIEW: should load string from string resource
   OutlineApp_ErrorMessage(lpOutlineApp, ErrMsgPrinting);
}


/* OutlineApp_PrinterSetupCommand
 * ------------------------------
 *
 *      Setup a different printer for printing
 */
void OutlineApp_PrinterSetupCommand(LPOUTLINEAPP lpOutlineApp)
{
   DWORD FlagSave;
   FlagSave = lpOutlineApp->m_PrintDlg.Flags;
   lpOutlineApp->m_PrintDlg.Flags |= PD_PRINTSETUP;

#if defined( OLE_VERSION )
   OleApp_PreModalDialog(
         (LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineApp->m_lpDoc);
#endif

   PrintDlg((LPPRINTDLG)&lpOutlineApp->m_PrintDlg);

#if defined( OLE_VERSION )
   OleApp_PostModalDialog(
         (LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineApp->m_lpDoc);
#endif

   lpOutlineApp->m_PrintDlg.Flags = FlagSave;
}

/*
 *  FUNCTION   : OutlineApp_GetPrinterDC ()
 *
 *  PURPOSE    : Creates a printer display context for the printer
 *
 *  RETURNS    : HDC   - A handle to printer DC.
 */
HDC OutlineApp_GetPrinterDC(LPOUTLINEAPP lpApp)
{

   HDC         hDC;
   LPDEVMODE   lpDevMode = NULL;
   LPDEVNAMES  lpDevNames;
   LPSTR       lpszDriverName;
   LPSTR       lpszDeviceName;
   LPSTR       lpszPortName;

   if(lpApp->m_PrintDlg.hDC) {
      hDC = lpApp->m_PrintDlg.hDC;
   } else {
      if(! lpApp->m_PrintDlg.hDevNames)
         return(NULL);
      lpDevNames = (LPDEVNAMES)GlobalLock(lpApp->m_PrintDlg.hDevNames);
      lpszDriverName = (LPSTR)lpDevNames + lpDevNames->wDriverOffset;
      lpszDeviceName = (LPSTR)lpDevNames + lpDevNames->wDeviceOffset;
      lpszPortName   = (LPSTR)lpDevNames + lpDevNames->wOutputOffset;
      GlobalUnlock(lpApp->m_PrintDlg.hDevNames);

      if(lpApp->m_PrintDlg.hDevMode)
         lpDevMode = (LPDEVMODE)GlobalLock(lpApp->m_PrintDlg.hDevMode);
#if defined( WIN32 )
      hDC = CreateDC(
            lpszDriverName,
            lpszDeviceName,
            lpszPortName,
            (CONST DEVMODE FAR*)lpDevMode);
#else
```

```
      hDC = CreateDC(
            lpszDriverName,
            lpszDeviceName,
            lpszPortName,
            (LPSTR)lpDevMode);
#endif

      if(lpApp->m_PrintDlg.hDevMode && lpDevMode)
         GlobalUnlock(lpApp->m_PrintDlg.hDevMode);
   }

   return(hDC);
}


/* OutlineApp_SaveCommand
 * ----------------------
 *
 *      Save the document with same name. If no name exists, prompt the user
 *      for a name (via SaveAsCommand)
 *
 *  Parameters:
 *
 *  Returns:
 *      TRUE    if succesfully
 *      FALSE   if failed or aborted
 */
BOOL OutlineApp_SaveCommand(LPOUTLINEAPP lpOutlineApp)
{
   LPOUTLINEDOC lpOutlineDoc = OutlineApp_GetActiveDoc(lpOutlineApp);

   if(lpOutlineDoc->m_docInitType == DOCTYPE_NEW)  /* file with no name */
      return OutlineApp_SaveAsCommand(lpOutlineApp);


   if(OutlineDoc_IsModified(lpOutlineDoc)) {

#if defined( OLE_SERVER )

      if (lpOutlineDoc->m_docInitType == DOCTYPE_EMBEDDED) {
         LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOutlineDoc;
         HRESULT hrErr;

         /* OLE2NOTE: if the document is an embedded object, then
         **    the "File.Save" command is changed to "File.Update".
         **    in order to update our container, we must ask our
         **    container to save us.
         */
         OleDbgAssert(lpServerDoc->m_lpOleClientSite != NULL);
         OLEDBG_BEGIN2("IOleClientSite::SaveObject called\r\n")
         hrErr = lpServerDoc->m_lpOleClientSite->lpVtbl->SaveObject(
               lpServerDoc->m_lpOleClientSite
         );
         OLEDBG_END2
         if (hrErr != NOERROR) {
            OleDbgOutHResult("IOleClientSite::SaveObject returned",hrErr);
            return FALSE;
         }
      } else
         // document is file-base user document, save it to its file.

#endif      // OLE_SERVER

      (void)OutlineDoc_SaveToFile(
            lpOutlineDoc,
            NULL,
            lpOutlineDoc->m_cfSaveFormat,
            TRUE
      );
   }

   return TRUE;
}


/* OutlineApp_SaveAsCommand
 * ------------------------
 *
 *      Save the document as another name
 *
 *  Parameters:
 *
 *  Returns:
 *      TRUE    if saved successful
 *      FALSE   if failed or aborted
 */
BOOL OutlineApp_SaveAsCommand(LPOUTLINEAPP lpOutlineApp)
{
   LPOUTLINEDOC lpOutlineDoc = lpOutlineApp->m_lpDoc;
   OPENFILENAME ofn;
   char szFilter[]=APPFILENAMEFILTER;
   char szFileName[256]="";
   OLECHAR szUniFileName[256];
   int i;
   UINT uFormat;
   BOOL fNoError = TRUE;
   BOOL fRemember = TRUE;
   BOOL fStatus;

   for(i=0; szFilter[i]; i++)
      if(szFilter[i]=='|') szFilter[i]='\0';

   _fmemset((LPOPENFILENAME)&ofn,0,sizeof(OPENFILENAME));

   ofn.lStructSize=sizeof(OPENFILENAME);
   ofn.hwndOwner=lpOutlineDoc->m_hWndDoc;
   ofn.lpstrFilter=(LPSTR)szFilter;
   ofn.lpstrFile=(LPSTR)szFileName;
   ofn.nMaxFile=256;
   ofn.Flags=OFN_PATHMUSTEXIST | OFN_OVERWRITEPROMPT | OFN_HIDEREADONLY;
   ofn.lpstrDefExt=DEFEXTENSION;

#if defined( OLE_VERSION )
   OleApp_PreModalDialog(
         (LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineApp->m_lpDoc);
#endif

   fStatus = GetSaveFileName((LPOPENFILENAME)&ofn);

#if defined( OLE_VERSION )
   OleApp_PostModalDialog(
         (LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineApp->m_lpDoc);
#endif

   if (fStatus) {

#if defined( OLE_CNTR )
      // determine which file type the user selected.
      switch (ofn.nFilterIndex) {
         case 1:
            uFormat = ((LPCONTAINERAPP)lpOutlineApp)->m_cfCntrOutl;
            break;
         case 2:
            uFormat = lpOutlineApp->m_cfOutline;
            break;
         default:
            uFormat = ((LPCONTAINERAPP)lpOutlineApp)->m_cfCntrOutl;
            break;
      }
#else
      uFormat = lpOutlineApp->m_cfOutline;
#endif

#if defined( OLE_SERVER )
      /* OLE2NOTE: if the document is an embedded object, then the
      **    File.SaveAs command is changed to File.SaveCopyAs. with the
      **    Save Copy As operation, the document does NOT remember the
      **    saved file as the associated file for the document.
      */
      if (lpOutlineDoc->m_docInitType == DOCTYPE_EMBEDDED)
         fRemember = FALSE;
#endif
      A2W (szFileName, szUniFileName, 256);
      (void)OutlineDoc_SaveToFile(
            lpOutlineDoc,
            szUniFileName,
            uFormat,
            fRemember
      );

   }
   else
      fNoError = FALSE;
   return fNoError;

}


/* OutlineApp_AboutCommand
 * -----------------------
 *
 *      Show the About dialog box
 */
void OutlineApp_AboutCommand(LPOUTLINEAPP lpOutlineApp)
{
#if defined( OLE_VERSION )
   OleApp_PreModalDialog(
         (LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineApp->m_lpDoc);
#endif

   DialogBox(
         lpOutlineApp->m_hInst,
         (LPSTR)"About",
         OutlineApp_GetFrameWindow(lpOutlineApp),
         (DLGPROC)AboutDlgProc
   );

#if defined( OLE_VERSION )
   OleApp_PostModalDialog(
         (LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineApp->m_lpDoc);
#endif
}


/* OutlineApp_CloseAllDocsAndExitCommand
 * -------------------------------------
 *
 *  Close all active documents and exit the app.
 *  Because this is an SDI, there is only one document
 *  If the doc was modified, prompt the user if he wants to save it.
 *
 *  Returns:
 *      TRUE if the app is successfully closed
 *      FALSE if failed or aborted
 */
BOOL OutlineApp_CloseAllDocsAndExitCommand(
      LPOUTLINEAPP        lpOutlineApp,
      BOOL                fForceEndSession
)
{
   BOOL fResult;

   OLEDBG_BEGIN2("OutlineApp_CloseAllDocsAndExitCommand\r\n")

#if defined( OLE_VERSION )
   // Call OLE specific version of this function
   fResult = OleApp_CloseAllDocsAndExitCommand(
         (LPOLEAPP)lpOutlineApp, fForceEndSession);

#else

   /* Because this is an SDI app, there is only one document.
   ** Close the doc. if it is successfully closed and the app will
   ** not automatically exit, then also exit the app.
   ** if this were an MDI app, we would loop through and close all
   ** open MDI child documents.
   */
   if (OutlineDoc_Close(lpOutlineApp->m_lpDoc, OLECLOSE_PROMPTSAVE)) {

#if defined( _DEBUG )
      OleDbgAssertSz(
            lpOutlineApp->m_lpDoc==NULL,
            "Closed doc NOT properly destroyed"
      );
#endif

      OutlineApp_Destroy(lpOutlineApp);
      fResult = TRUE;

   } // else User Canceled shutdown
   else
      fResult = FALSE;

#endif

   OLEDBG_END2

   return fResult;
}


/* OutlineApp_Destroy
 * ------------------
 *
 *      Destroy all data structures used by the app and force the
 * app to shut down. This should be called after all documents have
 * been closed.
 */
void OutlineApp_Destroy(LPOUTLINEAPP lpOutlineApp)
{
   OLEDBG_BEGIN3("OutlineApp_Destroy\r\n");

#if defined( OLE_VERSION )
   /* OLE2NOTE: perform processing required for OLE */
   OleApp_Destroy((LPOLEAPP)lpOutlineApp);
#endif

   SetCursor(LoadCursor(NULL, MAKEINTRESOURCE(IDC_ARROW)));
   DestroyCursor(lpOutlineApp->m_hcursorSelCur);

#if defined( USE_FRAMETOOLS )
   FrameTools_Destroy(&lpOutlineApp->m_frametools);
```

```
#endif

   if (lpOutlineApp->m_hStdFont)
      DeleteObject(lpOutlineApp->m_hStdFont);

   if(lpOutlineApp->m_PrintDlg.hDevMode)
      GlobalFree(lpOutlineApp->m_PrintDlg.hDevMode);
   if(lpOutlineApp->m_PrintDlg.hDevNames)
      GlobalFree(lpOutlineApp->m_PrintDlg.hDevNames);

#if defined( USE_STATUSBAR )
   if(lpOutlineApp->m_hWndStatusBar) {
      DestroyStatusWindow(lpOutlineApp->m_hWndStatusBar);
      lpOutlineApp->m_hWndStatusBar = NULL;
   }
#endif

   OutlineApp_DestroyWindow(lpOutlineApp);
   OleDbgOut1("@@@@ APP DESTROYED\r\n");

   OLEDBG_END3
}


/* OutlineApp_DestroyWindow
 * ------------------------
 *
 *  Destroy all windows created by the App.
 */
void OutlineApp_DestroyWindow(LPOUTLINEAPP lpOutlineApp)
{
   HWND hWndApp = lpOutlineApp->m_hWndApp;

   if(hWndApp) {
      lpOutlineApp->m_hWndApp = NULL;
      lpOutlineApp->m_hWndAccelTarget = NULL;
      DestroyWindow(hWndApp);  /* Quit the app */
   }
}


/* OutlineApp_GetFrameRect
** -----------------------
**    Get the rectangle of the app frame window EXCLUDING space for the
**    status line.
**
**    OLE2NOTE: this is the rectangle that an in-place container can
**    offer to an in-place active object from which to get frame tool
**    space.
*/
void OutlineApp_GetFrameRect(LPOUTLINEAPP lpOutlineApp, LPRECT
lprcFrameRect)
{
   GetClientRect(lpOutlineApp->m_hWndApp, lprcFrameRect);
```

```
#if defined( USE_STATUSBAR )
   lprcFrameRect->bottom -= STATUS_HEIGHT;
#endif

}


/* OutlineApp_GetClientAreaRect
** ----------------------------
**    Get the rectangle of the app frame window EXCLUDING space for the
**    status line AND EXCLUDING space for any frame-level tools.
**
**    OLE2NOTE: this is the rectangle that an in-place container gives
**    to its in-place active object as the lpClipRect in
**    IOleInPlaceSite::GetWindowContext.
*/
void OutlineApp_GetClientAreaRect(
      LPOUTLINEAPP        lpOutlineApp,
      LPRECT              lprcClientAreaRect
)
{
   OutlineApp_GetFrameRect(lpOutlineApp, lprcClientAreaRect);

   /* if the app either uses frame-level tools itself or, as in-place
   **    container, is prepared to allow an in-place active object to
   **    have space for tools, then it must subtract away the space
   **    required for the tools.
   */
#if defined ( USE_FRAMETOOLS ) || defined ( INPLACE_CNTR )

   lprcClientAreaRect->top    += lpOutlineApp->m_FrameToolWidths.top;
   lprcClientAreaRect->left   += lpOutlineApp->m_FrameToolWidths.left;
   lprcClientAreaRect->right  -= lpOutlineApp->m_FrameToolWidths.right;
   lprcClientAreaRect->bottom -= lpOutlineApp->m_FrameToolWidths.bottom;
#endif  // USE_FRAMETOOLS || INPLACE_CNTR

}


/* OutlineApp_GetStatusLineRect
** ----------------------------
**    Get the rectangle required for the status line.
**
**    OLE2NOTE: the top frame-level in-place container displays its
**    status line even when an object is active in-place.
*/
void OutlineApp_GetStatusLineRect(
      LPOUTLINEAPP        lpOutlineApp,
      LPRECT              lprcStatusLineRect
)
{
   RECT rcFrameRect;
   GetClientRect(lpOutlineApp->m_hWndApp, (LPRECT)&rcFrameRect);
   lprcStatusLineRect->left    = rcFrameRect.left;
   lprcStatusLineRect->top     = rcFrameRect.bottom - STATUS_HEIGHT;
   lprcStatusLineRect->right   = rcFrameRect.right;
   lprcStatusLineRect->bottom  = rcFrameRect.bottom;
}


/* OutlineApp_ResizeWindows
 * ------------------------
 *
 * Changes the size and position of the SDI document and tool windows.
 * Normally called on a WM_SIZE message.
 *
 * Currently the app supports a status bar and a single SDI document window.
 * In the future it will have a formula bar and possibly multiple MDI
 * document windows.
 *
 * CUSTOMIZATION: Change positions of windows.
 */
void OutlineApp_ResizeWindows(LPOUTLINEAPP lpOutlineApp)
{
   LPOUTLINEDOC lpOutlineDoc = OutlineApp_GetActiveDoc(lpOutlineApp);
   RECT rcStatusLineRect;

   if (! lpOutlineApp)
      return;

#if defined( INPLACE_CNTR )
   if (lpOutlineDoc)
      ContainerDoc_FrameWindowResized((LPCONTAINERDOC)lpOutlineDoc);
#else
#if defined( USE_FRAMETOOLS )
   if (lpOutlineDoc)
      OutlineDoc_AddFrameLevelTools(lpOutlineDoc);
#else
   OutlineApp_ResizeClientArea(lpOutlineApp);
#endif  // ! USE_FRAMETOOLS
#endif  // ! INPLACE_CNTR

#if defined( USE_STATUSBAR )
   if (lpOutlineApp->m_hWndStatusBar) {
      OutlineApp_GetStatusLineRect(lpOutlineApp, (LPRECT)&rcStatusLineRect);
      MoveWindow(
            lpOutlineApp->m_hWndStatusBar,
            rcStatusLineRect.left,
            rcStatusLineRect.top,
            rcStatusLineRect.right - rcStatusLineRect.left,
            rcStatusLineRect.bottom - rcStatusLineRect.top,
            TRUE    /* fRepaint */
         );
   }
#endif  // USE_STATUSBAR
}


#if defined( USE_FRAMETOOLS ) || defined( INPLACE_CNTR )
```

```
void OutlineApp_SetBorderSpace(
      LPOUTLINEAPP        lpOutlineApp,
      LPBORDERWIDTHS      lpBorderWidths
)
{
   lpOutlineApp->m_FrameToolWidths = *lpBorderWidths;
   OutlineApp_ResizeClientArea(lpOutlineApp);
}
#endif  // USE_FRAMETOOLS || INPLACE_CNTR


void OutlineApp_ResizeClientArea(LPOUTLINEAPP lpOutlineApp)
{
   RECT rcClientAreaRect;

#if defined( MDI_VERSION )

   // Resize MDI Client Area Window here

#else

   if (lpOutlineApp->m_lpDoc) {
         OutlineApp_GetClientAreaRect(
               lpOutlineApp, (LPRECT)&rcClientAreaRect);
         OutlineDoc_Resize(lpOutlineApp->m_lpDoc,
               (LPRECT)&rcClientAreaRect);
   }

#endif

}


/* OutlineApp_GetActiveDoc
 * -----------------------
 *
 * Return the document in focus. For SDI, the same (only one) document is
 * always returned.
 */
LPOUTLINEDOC OutlineApp_GetActiveDoc(LPOUTLINEAPP lpOutlineApp)
{
   return lpOutlineApp->m_lpDoc;
}

/* OutlineApp_GetMenu
 * ------------------
 *
 * Return the menu handle of the app
 */
HMENU OutlineApp_GetMenu(LPOUTLINEAPP lpOutlineApp)
{
   if (!lpOutlineApp) {
      return NULL;
   }
   return lpOutlineApp->m_hMenuApp;
}


#if defined( USE_FRAMETOOLS )

/* OutlineApp_GetFrameTools
 * ---------------------
 *
 * Return the pointer to the toolbar object
 */
LPFRAMETOOLS OutlineApp_GetFrameTools(LPOUTLINEAPP lpOutlineApp)
{
   return (LPFRAMETOOLS)&lpOutlineApp->m_frametools;
}
#endif


/* OutlineApp_SetStatusText
 * ------------------------
 *
 * Show the given string in the status line
 */
void OutlineApp_SetStatusText(LPOUTLINEAPP lpOutlineApp, LPOLESTR
lpszMessage)
{
   char szAnsiMessage[256];

   W2A (lpszMessage, szAnsiMessage, 256);
   SetStatusText(lpOutlineApp->m_hWndStatusBar, szAnsiMessage);
}


/* OutlineApp_GetActiveFont
 * ------------------------
 *
 *      Return the font used by the application
 */
HFONT OutlineApp_GetActiveFont(LPOUTLINEAPP lpOutlineApp)
{
   return lpOutlineApp->m_hStdFont;
}


/* OutlineApp_GetAppName
 * ---------------------
 *
 *      Retrieve the application name
 */
void OutlineApp_GetAppName(LPOUTLINEAPP lpOutlineApp, LPOLESTR lpszAppName)
{
   OLESTRCPY (lpszAppName, APPNAME);
}
```

```
/* OutlineApp_GetAppVersionNo
 * --------------------------
 *
 *      Get the version number (major and minor) of the application
 */
void OutlineApp_GetAppVersionNo(LPOUTLINEAPP lpOutlineApp, int
narrAppVersionNo[])
{
   narrAppVersionNo[0] = APPMAJORVERSIONNO;
   narrAppVersionNo[1] = APPMINORVERSIONNO;
}


/* OutlineApp_VersionNoCheck
 * -------------------------
 *
 *      Check if the version stamp read from a file is compatible
 *      with the current instance of the application.
 *      returns TRUE if the file can be read, else FALSE.
 */
BOOL OutlineApp_VersionNoCheck(LPOUTLINEAPP lpOutlineApp, LPOLESTR
lpszFormatName, int narrAppVersionNo[])
{
#if defined( OLE_CNTR )

   /* ContainerApp accepts both CF_OUTLINE and CF_CONTAINEROUTLINE formats
*/
   if (OLESTRCMP(lpszFormatName, CONTAINERDOCFORMAT) != 0 &&
      OLESTRCMP(lpszFormatName, OUTLINEDOCFORMAT) != 0) {
      // REVIEW: should load string from string resource
      OutlineApp_ErrorMessage(
            lpOutlineApp,
            OLESTR("File is either corrupted or not of proper type.")
         );
      return FALSE;
   }

#else

   /* OutlineApp accepts CF_OUTLINE format only */
   if (OLESTRCMP(lpszFormatName, OUTLINEDOCFORMAT) != 0) {
      // REVIEW: should load string from string resource
      OutlineApp_ErrorMessage(
            lpOutlineApp,
            OLESTR("File is either corrupted or not of proper type.")
         );
      return FALSE;
   }
#endif

   if (narrAppVersionNo[0] < APPMAJORVERSIONNO) {
      // REVIEW: should load string from string resource
      OutlineApp_ErrorMessage(
            lpOutlineApp,
            OLESTR("File was created by an older version; it can not be
read.")
         );
      return FALSE;
   }

   return TRUE;
}


/* OutlineApp_ErrorMessage
 * -----------------------
 *
 *      Display an error message box
 */
void OutlineApp_ErrorMessage(LPOUTLINEAPP lpOutlineApp, LPOLESTR lpszErrMsg)
{
   HWND hWndFrame = OutlineApp_GetFrameWindow(lpOutlineApp);
   char   szAnsiErrMsg[256];

   // OLE2NOTE: only put up user message boxes if app is visible
   if (IsWindowVisible(hWndFrame)) {
#if defined( OLE_VERSION )
      OleApp_PreModalDialog(
            (LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineApp->m_lpDoc);
#endif
      W2A (lpszErrMsg, szAnsiErrMsg, 256);
      MessageBox(hWndFrame, szAnsiErrMsg, NULL, MB_ICONEXCLAMATION | MB_OK);

#if defined( OLE_VERSION )
      OleApp_PostModalDialog(
            (LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineApp->m_lpDoc);
#endif
   }
}


#if defined( USE_FRAMETOOLS )

/* OutlineApp_SetFormulaBarAccel
 * -----------------------------
 *
 *  Set accelerator table based on state of formula bar.
 */
void OutlineApp_SetFormulaBarAccel(
      LPOUTLINEAPP            lpOutlineApp,
      BOOL                    fEditFocus
)
{
   if (fEditFocus)
      lpOutlineApp->m_hAccel = lpOutlineApp->m_hAccelFocusEdit;
   else
      lpOutlineApp->m_hAccel = lpOutlineApp->m_hAccelApp;
}
```

```
#endif  // USE_FRAMETOOLS




/* OutlineApp_ForceRedraw
 * ----------------------
 *
 *      Force the Application window to repaint.
 */
void OutlineApp_ForceRedraw(LPOUTLINEAPP lpOutlineApp, BOOL fErase)
{
   if (!lpOutlineApp)
      return;

   InvalidateRect(lpOutlineApp->m_hWndApp, NULL, fErase);
}
```

OUTLDOC.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    outldoc.c
**
**    This file contains OutlineDoc functions.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"

#if !defined( OLE_VERSION )
#include <commdlg.h>
#endif


OLEDBGDATA

extern LPOUTLINEAPP g_lpApp;

#ifdef WIN32S
#define GALLOCFLG (GMEM_SHARE | GMEM_ZEROINIT | GMEM_MOVEABLE)
#else
#define GALLOCFLG (GMEM_SHARE | GMEM_ZEROINIT)
#endif

// REVIEW: should use string resource for messages
OLECHAR ErrMsgDocWnd[] = OLESTR("Can't create Document Window!");
OLECHAR ErrMsgFormatNotSupported[] = OLESTR("Clipboard format not
supported!");
OLECHAR MsgSaveFile[] = OLESTR("Save existing file ?");
OLECHAR ErrMsgSaving[] = OLESTR("Error in saving file!");
OLECHAR ErrMsgOpening[] = OLESTR("Error in opening file!");
OLECHAR ErrMsgFormat[] = OLESTR("Improper file format!");
OLECHAR ErrOutOfMemory[] = OLESTR("Error: out of memory!");
static OLECHAR ErrMsgPrint[] = OLESTR("Printing Error!");

static BOOL fCancelPrint;    // TRUE if the user has canceled the print job
static HWND hWndPDlg;       // Handle to the cancel print dialog


/* OutlineDoc_Init
 * ---------------
 *
 *  Initialize the fields of a new OutlineDoc object. The object is
initially
 *  not associated with a file or an (Untitled) document. This function sets
 *  the docInitType to DOCTYPE_UNKNOWN. After calling this function the
 *  caller should call:
```

```
 *      1. OutlineDoc_InitNewFile to set the OutlineDoc to (Untitled)
 *      2. OutlineDoc_LoadFromFile to associate the OutlineDoc with a file.
 *  This function creates a new window for the document.
 *
 *  NOTE: the window is initially created with a NIL size. it must be
 *        sized and positioned by the caller. also the document is initially
 *        created invisible. the caller must call OutlineDoc_ShowWindow
 *        after sizing it to make the document window visible.
 */
BOOL OutlineDoc_Init(LPOUTLINEDOC lpOutlineDoc, BOOL fDataTransferDoc)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;

#if defined( INPLACE_CNTR )
   lpOutlineDoc->m_hWndDoc = CreateWindow(
               DOCWNDCLASS,            // Window class name
               NULL,                   // Window's title

               /* OLE2NOTE: an in-place contanier MUST use
               **    WS_CLIPCHILDREN window style for the window
               **    that it uses as the parent for the server's
               **    in-place active window so that its
               **    painting does NOT interfere with the painting
               **    of the server's in-place active child window.
               */

               WS_CLIPCHILDREN | WS_CLIPSIBLINGS |
               WS_CHILDWINDOW,
               0, 0,
               0, 0,
               lpOutlineApp->m_hWndApp,// Parent window's handle
               (HMENU)1,               // child window id
               lpOutlineApp->m_hInst,  // Instance of window
               NULL);                  // Create struct for WM_CREATE

#else

   lpOutlineDoc->m_hWndDoc = CreateWindow(
               DOCWNDCLASS,            // Window class name
               NULL,                   // Window's title
               WS_CHILDWINDOW,
               0, 0,
               0, 0,
               lpOutlineApp->m_hWndApp,// Parent window's handle
               (HMENU)1,               // child window id
               lpOutlineApp->m_hInst,  // Instance of window
               NULL);                  // Create struct for WM_CREATE
#endif

   if(! lpOutlineDoc->m_hWndDoc) {
      OutlineApp_ErrorMessage(lpOutlineApp, ErrMsgDocWnd);
      return FALSE;
   }

   SetWindowLong(lpOutlineDoc->m_hWndDoc, 0, (LONG) lpOutlineDoc);
   if (! LineList_Init(&lpOutlineDoc->m_LineList, lpOutlineDoc))
      return FALSE;

   lpOutlineDoc->m_lpNameTable = OutlineDoc_CreateNameTable(lpOutlineDoc);
   if (! lpOutlineDoc->m_lpNameTable )
      return FALSE;

   lpOutlineDoc->m_docInitType = DOCTYPE_UNKNOWN;
   lpOutlineDoc->m_cfSaveFormat = lpOutlineApp->m_cfOutline;
   lpOutlineDoc->m_szFileName[0] = '\0';
   lpOutlineDoc->m_lpszDocTitle = lpOutlineDoc->m_szFileName;
   lpOutlineDoc->m_fDataTransferDoc = fDataTransferDoc;
   lpOutlineDoc->m_uCurrentZoom = IDM_V_ZOOM_100;
   lpOutlineDoc->m_scale.dwSxN  = (DWORD) 1;
   lpOutlineDoc->m_scale.dwSxD  = (DWORD) 1;
   lpOutlineDoc->m_scale.dwSyN  = (DWORD) 1;
   lpOutlineDoc->m_scale.dwSyD  = (DWORD) 1;
   lpOutlineDoc->m_uCurrentMargin = IDM_V_SETMARGIN_0;
   lpOutlineDoc->m_nLeftMargin  = 0;
   lpOutlineDoc->m_nRightMargin = 0;
   lpOutlineDoc->m_nDisableDraw = 0;
   OutlineDoc_SetModified(lpOutlineDoc, FALSE, FALSE, FALSE);

#if defined( USE_HEADING )
   if (! fDataTransferDoc) {
      if (!Heading_Create((LPHEADING)&lpOutlineDoc->m_heading,
            lpOutlineDoc->m_hWndDoc, lpOutlineApp->m_hInst)) {
         return FALSE;

      }
   }
#endif  // USE_HEADING

#if defined( USE_FRAMETOOLS )
   if (! fDataTransferDoc) {
      lpOutlineDoc->m_lpFrameTools = OutlineApp_GetFrameTools(lpOutlineApp);
      FrameTools_AssociateDoc(
            lpOutlineDoc->m_lpFrameTools,
            lpOutlineDoc
      );
   }
#endif  // USE_FRAMETOOLS

#if defined( OLE_VERSION )
   /* OLE2NOTE: perform initialization required for OLE */
   if (! OleDoc_Init((LPOLEDOC)lpOutlineDoc, fDataTransferDoc))
      return FALSE;
#endif  // OLE_VERSION

   return TRUE;
}


/* OutlineDoc_InitNewFile
```

```
 * ----------------------
 *
 *  Initialize the OutlineDoc object to be a new (Untitled) document.
 *  This function sets the docInitType to DOCTYPE_NEW.
 */
BOOL OutlineDoc_InitNewFile(LPOUTLINEDOC lpOutlineDoc)
{
#if defined( OLE_VERSION )
   // OLE2NOTE: call OLE version of this function instead
   return OleDoc_InitNewFile((LPOLEDOC)lpOutlineDoc);

#else

   OleDbgAssert(lpOutlineDoc->m_docInitType == DOCTYPE_UNKNOWN);

   // set file name to untitled
   // REVIEW: should load from string resource
   lstrcpyW(lpOutlineDoc->m_szFileName, UNTITLED);
   lpOutlineDoc->m_lpszDocTitle = lpOutlineDoc->m_szFileName;
   lpOutlineDoc->m_docInitType = DOCTYPE_NEW;

   if (! lpOutlineDoc->m_fDataTransferDoc)
      OutlineDoc_SetTitle(lpOutlineDoc, FALSE /*fMakeUpperCase*/);

   return TRUE;

#endif      // BASE OUTLINE VERSION
}


/* OutlineDoc_CreateNameTable
 * --------------------------
 *
 * Allocate a new NameTable of the appropriate type. Each document has
 * a NameTable and a LineList.
 *  OutlineDoc --> creates standard OutlineNameTable type name tables.
 *  ServerDoc  --> creates enhanced SeverNameTable type name tables.
 *
 *      Returns lpNameTable for successful, NULL if error.
 */
LPOUTLINENAMETABLE OutlineDoc_CreateNameTable(LPOUTLINEDOC lpOutlineDoc)
{
   LPOUTLINENAMETABLE lpOutlineNameTable;

   lpOutlineNameTable = (LPOUTLINENAMETABLE)New(
         (DWORD)sizeof(OUTLINENAMETABLE)
   );

   OleDbgAssertSz(lpOutlineNameTable != NULL,"Error allocating NameTable");
   if (lpOutlineNameTable == NULL)
      return NULL;

   // initialize new NameTable
   if (! OutlineNameTable_Init(lpOutlineNameTable, lpOutlineDoc) )
      goto error;
   return lpOutlineNameTable;

error:
   if (lpOutlineNameTable)
      Delete(lpOutlineNameTable);
   return NULL;
}


/* OutlineDoc_ClearCommand
 * -----------------------
 *
 *      Delete selection in list box by calling OutlineDoc_Delete
 */
void OutlineDoc_ClearCommand(LPOUTLINEDOC lpOutlineDoc)
{
   LPLINELIST lpLL = &lpOutlineDoc->m_LineList;
   int i;
   int nNumSel;
   LINERANGE lrSel;

   nNumSel=LineList_GetSel(lpLL, (LPLINERANGE)&lrSel);

   OutlineDoc_SetRedraw ( lpOutlineDoc, FALSE );
   for(i = 0; i < nNumSel; i++)
      OutlineDoc_DeleteLine(lpOutlineDoc, lrSel.m_nStartLine);

   OutlineDoc_SetRedraw ( lpOutlineDoc, TRUE );

   LineList_RecalcMaxLineWidthInHimetric(lpLL, 0);
}


/* OutlineDoc_CutCommand
 * ---------------------
 *
 * Cut selection to clipboard
 */
void OutlineDoc_CutCommand(LPOUTLINEDOC lpOutlineDoc)
{
   OutlineDoc_CopyCommand(lpOutlineDoc);
   OutlineDoc_ClearCommand(lpOutlineDoc);
}


/* OutlineDoc_CopyCommand
 * ----------------------
 *  Copy selection to clipboard.
 *  Post to the clipboard the formats that the app can render.
 *  the actual data is not rendered at this time. using the
 *  delayed rendering technique, Windows will send the clipboard
 *  owner window either a WM_RENDERALLFORMATS or a WM_RENDERFORMAT
 *  message when the actual data is requested.
 *
 *    OLE2NOTE: the normal delayed rendering technique where Windows
 *    sends the clipboard owner window either a WM_RENDERALLFORMATS or
 *    a WM_RENDERFORMAT message when the actual data is requested is
 *    NOT exposed to the app calling OleSetClipboard. OLE internally
 *    creates its own window as the clipboard owner and thus our app
 *    will NOT get these WM_RENDER messages.
 */
void OutlineDoc_CopyCommand(LPOUTLINEDOC lpSrcOutlineDoc)
{
#if defined( OLE_VERSION )
   // Call OLE version of this function instead
   OleDoc_CopyCommand((LPOLEDOC)lpSrcOutlineDoc);

#else
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPOUTLINEDOC lpClipboardDoc;

   OpenClipboard(lpSrcOutlineDoc->m_hWndDoc);
   EmptyClipboard();

   /* squirrel away a copy of the current selection to the ClipboardDoc */
   lpClipboardDoc = OutlineDoc_CreateDataTransferDoc(lpSrcOutlineDoc);

   if (! lpClipboardDoc)
      return;     // Error: could not create DataTransferDoc

   lpOutlineApp->m_lpClipboardDoc = (LPOUTLINEDOC)lpClipboardDoc;

   SetClipboardData(lpOutlineApp->m_cfOutline, NULL);
   SetClipboardData(CF_TEXT, NULL);

   CloseClipboard();

#endif  // ! OLE_VERSION
}


/* OutlineDoc_ClearAllLines
 * ------------------------
 *
 *      Delete all lines in the document.
 */
void OutlineDoc_ClearAllLines(LPOUTLINEDOC lpOutlineDoc)
{
   LPLINELIST lpLL = &lpOutlineDoc->m_LineList;
   int i;

   for(i = 0; i < lpLL->m_nNumLines; i++)
      OutlineDoc_DeleteLine(lpOutlineDoc, 0);

   LineList_RecalcMaxLineWidthInHimetric(lpLL, 0);
}


/* OutlineDoc_CreateDataTransferDoc
```

```
 * --------------------------------
 *
 *      Create a document to be use to transfer data (either via a
 *  drag/drop operation of the clipboard). Copy the selection of the
 *  source doc to the data transfer document. A data transfer document is
 *  the same as a document that is created by the user except that it is
 *  NOT made visible to the user. it is specially used to hold a copy of
 *  data that the user should not be able to change.
 *
 *  OLE2NOTE: in the OLE version the data transfer document is used
 *      specifically to provide an IDataObject* that renders the data
copied.
 */
LPOUTLINEDOC OutlineDoc_CreateDataTransferDoc(LPOUTLINEDOC lpSrcOutlineDoc)
{
#if defined( OLE_VERSION )
   // Call OLE version of this function instead
   return OleDoc_CreateDataTransferDoc((LPOLEDOC)lpSrcOutlineDoc);

#else
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPOUTLINEDOC lpDestOutlineDoc;
   LPLINELIST lpSrcLL = &lpSrcOutlineDoc->m_LineList;
   LINERANGE lrSel;
   int nCopied;

   lpDestOutlineDoc = OutlineApp_CreateDoc(lpOutlineApp, TRUE);
   if (! lpDestOutlineDoc) return NULL;

   // set the ClipboardDoc to an (Untitled) doc.
   if (! OutlineDoc_InitNewFile(lpDestOutlineDoc))
      goto error;

   LineList_GetSel(lpSrcLL, (LPLINERANGE)&lrSel);
   nCopied = LineList_CopySelToDoc(
         lpSrcLL,
         (LPLINERANGE)&lrSel,
         lpDestOutlineDoc
   );

   return lpDestOutlineDoc;

error:
   if (lpDestOutlineDoc)
      OutlineDoc_Destroy(lpDestOutlineDoc);

   return NULL;

#endif  // ! OLE_VERSION
}


/* OutlineDoc_PasteCommand
 * -----------------------
 *
 * Paste lines from clipboard
 */
void OutlineDoc_PasteCommand(LPOUTLINEDOC lpOutlineDoc)
{
#if defined( OLE_VERSION )
   // Call OLE version of this function instead
   OleDoc_PasteCommand((LPOLEDOC)lpOutlineDoc);

#else

   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPLINELIST lpLL = (LPLINELIST)&lpOutlineDoc->m_LineList;
   int nIndex;
   int nCount;
   HGLOBAL hData;
   LINERANGE lrSel;
   UINT uFormat;

   if (LineList_GetCount(lpLL) == 0)
      nIndex = -1;    // pasting to empty list
   else
      nIndex=LineList_GetFocusLineIndex(lpLL);

   OutlineDoc_SetRedraw ( lpOutlineDoc, FALSE );

   OpenClipboard(lpOutlineDoc->m_hWndDoc);

   uFormat = 0;
   while(uFormat = EnumClipboardFormats(uFormat)) {
      if(uFormat == lpOutlineApp->m_cfOutline) {
         hData = GetClipboardData(lpOutlineApp->m_cfOutline);
         nCount = OutlineDoc_PasteOutlineData(lpOutlineDoc, hData, nIndex);
         break;
      }
      if(uFormat == CF_TEXT) {
         hData = GetClipboardData(CF_TEXT);
         nCount = OutlineDoc_PasteTextData(lpOutlineDoc, hData, nIndex);
         break;
      }
   }

   lrSel.m_nStartLine = nIndex + nCount;
   lrSel.m_nEndLine = nIndex + 1;
   LineList_SetSel(lpLL, &lrSel);
   OutlineDoc_SetRedraw ( lpOutlineDoc, TRUE );

   CloseClipboard();

#endif      // ! OLE_VERSION
}


/* OutlineDoc_PasteOutlineData
 * ---------------------------
 *
 *      Put an array of Line Objects (stored in hOutline) into the document
 *
 * Return the number of items added
 */
int OutlineDoc_PasteOutlineData(LPOUTLINEDOC lpOutlineDoc, HGLOBAL hOutline,
int nStartIndex)
{
   int nCount;
   int i;
   LPTEXTLINE arrLine;

   nCount = (int) GlobalSize(hOutline) / sizeof(TEXTLINE);
   arrLine = (LPTEXTLINE)GlobalLock(hOutline);
   if (!arrLine)
      return 0;

   for(i = 0; i < nCount; i++)
      Line_CopyToDoc((LPLINE)&arrLine[i], lpOutlineDoc, nStartIndex+i);

   GlobalUnlock(hOutline);

   return nCount;
}


/* OutlineDoc_PasteTextData
 * ------------------------
 *
 *      Build Line Objects from the strings (separated by '\n') in hText
 * and put them into the document
 */
int OutlineDoc_PasteTextData(LPOUTLINEDOC lpOutlineDoc, HGLOBAL hText, int
nStartIndex)
{
   LPLINELIST  lpLL = (LPLINELIST)&lpOutlineDoc->m_LineList;
   HDC         hDC;
   LPSTR       lpszText;
   LPSTR       lpszEnd;
   LPTEXTLINE  lpLine;
   int         nLineCount;
   int         i;
   UINT        nTab;
   char        szBuf[MAXSTRLEN+1];

   lpszText=(LPSTR)GlobalLock(hText);
   if(!lpszText)
      return 0;

   lpszEnd = lpszText + lstrlen(lpszText);
   nLineCount=0;

   while(*lpszText && (lpszText<lpszEnd)) {

      // count the tab level
      nTab = 0;
      while((*lpszText == '\t') && (lpszText<lpszEnd)) {
         nTab++;
         lpszText++;
      }

      // collect the text string character by character
      for(i=0; (i<MAXSTRLEN) && (lpszText<lpszEnd); i++) {
         if ((! *lpszText) || (*lpszText == '\n'))
            break;
         szBuf[i] = *lpszText++;
      }
      szBuf[i] = 0;
      lpszText++;
      if ((i > 0) && (szBuf[i-1] == '\r'))
         szBuf[i-1] = 0;     // remove carriage return at the end

      hDC = LineList_GetDC(lpLL);
      lpLine = TextLine_Create(hDC, nTab, szBuf);
      LineList_ReleaseDC(lpLL, hDC);

      OutlineDoc_AddLine(
            lpOutlineDoc,
            (LPLINE)lpLine,
            nStartIndex + nLineCount
      );
      nLineCount++;

   }

   GlobalUnlock(hText);

   return nLineCount;
}


/* OutlineDoc_AddTextLineCommand
 * -----------------------------
 *
 *      Add a new text line following the current focus line.
 */
void OutlineDoc_AddTextLineCommand(LPOUTLINEDOC lpOutlineDoc)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPLINELIST lpLL = &lpOutlineDoc->m_LineList;
   HDC hDC;
   int nIndex = LineList_GetFocusLineIndex(lpLL);
   char szBuf[MAXSTRLEN+1];
   UINT nTab = 0;
   LPLINE lpLine;
   LPTEXTLINE lpTextLine;

   szBuf[0] = '\0';

#if defined( USE_FRAMETOOLS )
   FrameTools_FB_GetEditText(
         lpOutlineDoc->m_lpFrameTools, szBuf, MAXSTRLEN+1);
#else
   if (! InputTextDlg(lpOutlineDoc->m_hWndDoc, szBuf, "Add Line"))
      return;
#endif

   hDC = LineList_GetDC(lpLL);
   lpLine = LineList_GetLine(lpLL, nIndex);
   if (lpLine)
      nTab = Line_GetTabLevel(lpLine);

   lpTextLine=TextLine_Create(hDC, nTab, szBuf);
   LineList_ReleaseDC(lpLL, hDC);

   if (! lpTextLine) {
      OutlineApp_ErrorMessage(lpOutlineApp, ErrOutOfMemory);
      return;
   }
   OutlineDoc_AddLine(lpOutlineDoc, (LPLINE)lpTextLine, nIndex);
}


/* OutlineDoc_AddTopLineCommand
 * ----------------------------
 *
 *      Add a top (margin) line as the first line in the LineList.
 *      (do not change the current selection)
 */
void OutlineDoc_AddTopLineCommand(
      LPOUTLINEDOC        lpOutlineDoc,
      UINT                nHeightInHimetric
)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPLINELIST  lpLL = &lpOutlineDoc->m_LineList;
   HDC         hDC = LineList_GetDC(lpLL);
   LPTEXTLINE  lpTextLine = TextLine_Create(hDC, 0, NULL);
   LPLINE      lpLine = (LPLINE)lpTextLine;
   LINERANGE   lrSel;
   int         nNumSel;

   LineList_ReleaseDC(lpLL, hDC);

   if (! lpTextLine) {
      OutlineApp_ErrorMessage(lpOutlineApp, ErrOutOfMemory);
      return;
   }

   Line_SetHeightInHimetric(lpLine, nHeightInHimetric);

   nNumSel=LineList_GetSel(lpLL, (LPLINERANGE)&lrSel);
   if (nNumSel > 0) {
      // adjust current selection to keep equivalent selection
      lrSel.m_nStartLine += 1;
      lrSel.m_nEndLine += 1;
   }
   OutlineDoc_AddLine(lpOutlineDoc, lpLine, -1);
   if (nNumSel > 0)
      LineList_SetSel(lpLL, (LPLINERANGE)&lrSel);
}


#if defined( USE_FRAMETOOLS )


/* OutlineDoc_SetFormulaBarEditText
 * --------------------------------
 *
 *      Fill the edit control in the formula with the text string from a
 * TextLine in focus.
 */
void OutlineDoc_SetFormulaBarEditText(
      LPOUTLINEDOC            lpOutlineDoc,
      LPLINE                  lpLine
)
{
   LPLINELIST lpLL = &lpOutlineDoc->m_LineList;
   char cBuf[MAXSTRLEN+1];

   if (! lpOutlineDoc || ! lpOutlineDoc->m_lpFrameTools)
      return;

   if (Line_GetLineType(lpLine) != TEXTLINETYPE) {
      FrameTools_FB_SetEditText(lpOutlineDoc->m_lpFrameTools, NULL);
   } else {
      TextLine_GetTextData((LPTEXTLINE)lpLine, (LPSTR)cBuf);
      FrameTools_FB_SetEditText(lpOutlineDoc->m_lpFrameTools, (LPSTR)cBuf);
   }
}


/* OutlineDoc_SetFormulaBarEditFocus
 * ---------------------------------
 *
 *  Setup for formula bar to gain or loose edit focus.
 *  if gaining focus, setup up special accelerator table and scroll line
 *      into view.
 *  else restore normal accelerator table.
 */
void OutlineDoc_SetFormulaBarEditFocus(
      LPOUTLINEDOC            lpOutlineDoc,
      BOOL                    fEditFocus
)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPLINELIST lpLL;
   int nFocusIndex;

   if (! lpOutlineDoc || ! lpOutlineDoc->m_lpFrameTools)
      return;
   lpOutlineDoc->m_lpFrameTools->m_fInFormulaBar = fEditFocus;

   if (fEditFocus && lpOutlineDoc->m_lpFrameTools) {
      lpLL = OutlineDoc_GetLineList(lpOutlineDoc);

      nFocusIndex = LineList_GetFocusLineIndex(lpLL);
      LineList_ScrollLineIntoView(lpLL, nFocusIndex);
      FrameTools_FB_FocusEdit(lpOutlineDoc->m_lpFrameTools);
   }

   OutlineApp_SetFormulaBarAccel(lpOutlineApp, fEditFocus);
}


/* OutlineDoc_IsEditFocusInFormulaBar
** ----------------------------------
**    Returns TRUE if edit focus is currently in the formula bar
**    else FALSE if not.
*/
BOOL OutlineDoc_IsEditFocusInFormulaBar(LPOUTLINEDOC lpOutlineDoc)
{
   if (! lpOutlineDoc || ! lpOutlineDoc->m_lpFrameTools)
      return FALSE;

   return lpOutlineDoc->m_lpFrameTools->m_fInFormulaBar;
}


/* OutlineDoc_UpdateFrameToolButtons
** ---------------------------------
**    Update the Enable/Disable states of the buttons in the formula
**    bar and button bar.
*/
void OutlineDoc_UpdateFrameToolButtons(LPOUTLINEDOC lpOutlineDoc)
{
   if (! lpOutlineDoc || ! lpOutlineDoc->m_lpFrameTools)
      return;
   FrameTools_UpdateButtons(lpOutlineDoc->m_lpFrameTools, lpOutlineDoc);
}
#endif  // USE_FRAMETOOLS


/* OutlineDoc_EditLineCommand
 * --------------------------
 *
 *      Edit the current focus line.
 */
void OutlineDoc_EditLineCommand(LPOUTLINEDOC lpOutlineDoc)
{
   LPLINELIST lpLL = &lpOutlineDoc->m_LineList;
   HDC hDC = LineList_GetDC(lpLL);
   int nIndex = LineList_GetFocusLineIndex(lpLL);
   LPLINE lpLine = LineList_GetLine(lpLL, nIndex);
   int nOrgLineWidthInHimetric;
   int nNewLineWidthInHimetric;
   BOOL fSizeChanged;

   if (!lpLine)
      return;

   nOrgLineWidthInHimetric = Line_GetTotalWidthInHimetric(lpLine);
   if (Line_Edit(lpLine, lpOutlineDoc->m_hWndDoc, hDC)) {
      nNewLineWidthInHimetric = Line_GetTotalWidthInHimetric(lpLine);

      if (nNewLineWidthInHimetric > nOrgLineWidthInHimetric) {
         fSizeChanged = LineList_SetMaxLineWidthInHimetric(
               lpLL,
               nNewLineWidthInHimetric
            );
      } else {
         fSizeChanged = LineList_RecalcMaxLineWidthInHimetric(
               lpLL,
               nOrgLineWidthInHimetric
            );
      }

#if defined( OLE_SERVER )
      /* Update Name Table */
      ServerNameTable_EditLineUpdate(
            (LPSERVERNAMETABLE)lpOutlineDoc->m_lpNameTable,
            nIndex
      );
#endif

      OutlineDoc_SetModified(lpOutlineDoc, TRUE, TRUE, fSizeChanged);

      LineList_ForceLineRedraw(lpLL, nIndex, TRUE);
   }
   LineList_ReleaseDC(lpLL, hDC);
}


/* OutlineDoc_IndentCommand
 * ------------------------
 *
 *      Indent selection of lines
 */
void OutlineDoc_IndentCommand(LPOUTLINEDOC lpOutlineDoc)
{
   LPLINELIST  lpLL = &lpOutlineDoc->m_LineList;
   LPLINE      lpLine;
   HDC         hDC = LineList_GetDC(lpLL);
   int         i;
   int         nIndex;
   int         nNumSel;
   LINERANGE   lrSel;
   BOOL        fSizeChanged = FALSE;

   nNumSel=LineList_GetSel(lpLL, (LPLINERANGE)&lrSel);
   OutlineDoc_SetRedraw ( lpOutlineDoc, FALSE );

   for(i = 0; i < nNumSel; i++) {
      nIndex = lrSel.m_nStartLine + i;
      lpLine=LineList_GetLine(lpLL, nIndex);
      if (! lpLine)
         continue;

      Line_Indent(lpLine, hDC);
      if (LineList_SetMaxLineWidthInHimetric(lpLL,
         Line_GetTotalWidthInHimetric(lpLine))) {
         fSizeChanged = TRUE;
      }
      LineList_ForceLineRedraw(lpLL, nIndex, TRUE);

#if defined( OLE_SERVER )
      /* Update Name Table */
      ServerNameTable_EditLineUpdate(
            (LPSERVERNAMETABLE)lpOutlineDoc->m_lpNameTable,
            nIndex
      );
#endif

   }

   LineList_ReleaseDC(lpLL, hDC);

   OutlineDoc_SetModified(lpOutlineDoc, TRUE, TRUE, fSizeChanged);
   OutlineDoc_SetRedraw ( lpOutlineDoc, TRUE );
}


/* OutlineDoc_UnindentCommand
 * --------------------------
 *
 *      Unindent selection of lines
 */
void OutlineDoc_UnindentCommand(LPOUTLINEDOC lpOutlineDoc)
{
   LPLINELIST  lpLL = &lpOutlineDoc->m_LineList;
   LPLINE      lpLine;
   HDC         hDC = LineList_GetDC(lpLL);
   int         nOrgLineWidthInHimetric;
   int         nOrgMaxLineWidthInHimetric = 0;
   int         i;
   int         nIndex;
   int         nNumSel;
   LINERANGE   lrSel;
   BOOL        fSizeChanged;

   nNumSel=LineList_GetSel(lpLL, (LPLINERANGE)&lrSel);

   OutlineDoc_SetRedraw ( lpOutlineDoc, FALSE );
   for(i = 0; i < nNumSel; i++) {
      nIndex = lrSel.m_nStartLine + i;
      lpLine=LineList_GetLine(lpLL, nIndex);
      if (!lpLine)
         continue;

      nOrgLineWidthInHimetric = Line_GetTotalWidthInHimetric(lpLine);
      nOrgMaxLineWidthInHimetric =
            (nOrgLineWidthInHimetric > nOrgMaxLineWidthInHimetric ?
               nOrgLineWidthInHimetric : nOrgMaxLineWidthInHimetric);
      Line_Unindent(lpLine, hDC);
      LineList_ForceLineRedraw(lpLL, nIndex, TRUE);

#if defined( OLE_SERVER )
      /* Update Name Table */
      ServerNameTable_EditLineUpdate(
            (LPSERVERNAMETABLE)lpOutlineDoc->m_lpNameTable,
            nIndex
      );
#endif

   }

   LineList_ReleaseDC(lpLL, hDC);

   fSizeChanged = LineList_RecalcMaxLineWidthInHimetric(
         lpLL,
         nOrgMaxLineWidthInHimetric
      );

   OutlineDoc_SetModified(lpOutlineDoc, TRUE, TRUE, fSizeChanged);
   OutlineDoc_SetRedraw ( lpOutlineDoc, TRUE );
}


/* OutlineDoc_SetLineHeightCommand
 * -------------------------------
 *
 *      Set height of the selection of lines
 */
void OutlineDoc_SetLineHeightCommand(LPOUTLINEDOC lpOutlineDoc)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPLINELIST  lpLL;
   HDC         hDC;
   LPLINE      lpLine;
   int         nNewHeight;
   int         i;
   int         nIndex;
   int         nNumSel;
   LINERANGE   lrSel;

   if (!lpOutlineDoc)
      return;
   lpLL = &lpOutlineDoc->m_LineList;
   nNumSel=LineList_GetSel(lpLL, (LPLINERANGE)&lrSel);
   lpLine = LineList_GetLine(lpLL, lrSel.m_nStartLine);
   if (!lpLine)
      return;

   nNewHeight = Line_GetHeightInHimetric(lpLine);

#if defined( OLE_VERSION )
   OleApp_PreModalDialog((LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineDoc);
#endif

   DialogBoxParam(
         lpOutlineApp->m_hInst,
         (LPSTR)"SetLineHeight",
         lpOutlineDoc->m_hWndDoc,
         (DLGPROC)SetLineHeightDlgProc,
         (LPARAM)(LPINT)&nNewHeight
   );

#if defined( OLE_VERSION )
   OleApp_PostModalDialog((LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineDoc);
#endif

   if (nNewHeight == 0)
      return;     /* user hit cancel */

   hDC = LineList_GetDC(lpLL);

   for (i = 0; i < nNumSel; i++) {
      nIndex = lrSel.m_nStartLine + i;
      lpLine=LineList_GetLine(lpLL, nIndex);
      if (nNewHeight == -1) {
         switch (Line_GetLineType(lpLine)) {

            case TEXTLINETYPE:

               TextLine_CalcExtents((LPTEXTLINE)lpLine, hDC);
               break;

#if defined( OLE_CNTR )
            case CONTAINERLINETYPE:

               ContainerLine_SetHeightInHimetric(
                     (LPCONTAINERLINE)lpLine, -1);
               break;
#endif

         }
      }
      else
         Line_SetHeightInHimetric(lpLine, nNewHeight);


      LineList_SetLineHeight(lpLL, nIndex,
            Line_GetHeightInHimetric(lpLine));
   }

   LineList_ReleaseDC(lpLL, hDC);

   OutlineDoc_SetModified(lpOutlineDoc, TRUE, TRUE, TRUE);
   LineList_ForceRedraw(lpLL, TRUE);
}



/* OutlineDoc_SelectAllCommand
 * ---------------------------
 *
 *      Select all the lines in the document.
 */
void OutlineDoc_SelectAllCommand(LPOUTLINEDOC lpOutlineDoc)
{
   LPLINELIST lpLL = &lpOutlineDoc->m_LineList;
   LINERANGE lrSel;

   lrSel.m_nStartLine = 0;
   lrSel.m_nEndLine = LineList_GetCount(lpLL) - 1;
   LineList_SetSel(lpLL, &lrSel);
}


/* OutlineDoc_DefineNameCommand
 * ----------------------------
 *
 *      Define a name in the document
 */
void OutlineDoc_DefineNameCommand(LPOUTLINEDOC lpOutlineDoc)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;

#if defined( OLE_VERSION )
   OleApp_PreModalDialog((LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineDoc);
#endif

   DialogBoxParam(
         lpOutlineApp->m_hInst,
         (LPSTR)"DefineName",
         lpOutlineDoc->m_hWndDoc,
         (DLGPROC)DefineNameDlgProc,
         (LPARAM) lpOutlineDoc
   );

#if defined( OLE_VERSION )
   OleApp_PostModalDialog((LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineDoc);
#endif
}


/* OutlineDoc_GotoNameCommand
```

```
 * --------------------------
 *
 *      Goto a predefined name in the document
 */
void OutlineDoc_GotoNameCommand(LPOUTLINEDOC lpOutlineDoc)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;

#if defined( OLE_VERSION )
   OleApp_PreModalDialog((LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineDoc);
#endif

   DialogBoxParam(
         lpOutlineApp->m_hInst,
         (LPSTR)"GotoName",
         lpOutlineDoc->m_hWndDoc,
         (DLGPROC)GotoNameDlgProc,
         (LPARAM)lpOutlineDoc
   );

#if defined( OLE_VERSION )
   OleApp_PostModalDialog((LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineDoc);
#endif
}


/* OutlineDoc_ShowWindow
 * ---------------------
 *
 *      Show the window of the document to the user.
 */
void OutlineDoc_ShowWindow(LPOUTLINEDOC lpOutlineDoc)
{
#if defined( _DEBUG )
   OleDbgAssertSz(lpOutlineDoc->m_docInitType != DOCTYPE_UNKNOWN,
            "OutlineDoc_ShowWindow: can't show unitialized document\r\n");
#endif
   if (lpOutlineDoc->m_docInitType == DOCTYPE_UNKNOWN)
      return;

#if defined( OLE_VERSION )
   // Call OLE version of this function instead
   OleDoc_ShowWindow((LPOLEDOC)lpOutlineDoc);
#else
   ShowWindow(lpOutlineDoc->m_hWndDoc, SW_SHOWNORMAL);
   SetForegroundWindow(lpOutlineDoc->m_hWndDoc);
#endif
}


#if defined( USE_FRAMETOOLS )

void OutlineDoc_AddFrameLevelTools(LPOUTLINEDOC lpOutlineDoc)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
```

```
#if defined( INPLACE_CNTR )
   // Call OLE In-Place Container version of this function instead
   ContainerDoc_AddFrameLevelTools((LPCONTAINERDOC)lpOutlineDoc);

#else   // ! INPLACE_CNTR
   RECT rcFrameRect;
   BORDERWIDTHS frameToolWidths;

#if defined( INPLACE_SVR )
   LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOutlineDoc;
   LPOLEINPLACEFRAME lpTopIPFrame=ServerDoc_GetTopInPlaceFrame(lpServerDoc);

   // if in-place active, add our tools to our in-place container's frame.
   if (lpTopIPFrame) {
      ServerDoc_AddFrameLevelTools(lpServerDoc);
      return;
   }
#endif  // INPLACE_SVR

   OutlineApp_GetFrameRect(g_lpApp, (LPRECT)&rcFrameRect);
   FrameTools_GetRequiredBorderSpace(
         lpOutlineDoc->m_lpFrameTools,
         (LPBORDERWIDTHS)&frameToolWidths
   );
   OutlineApp_SetBorderSpace(g_lpApp, (LPBORDERWIDTHS)&frameToolWidths);
   FrameTools_AttachToFrame(
         lpOutlineDoc->m_lpFrameTools, OutlineApp_GetWindow(lpOutlineApp));
   FrameTools_Move(lpOutlineDoc->m_lpFrameTools, (LPRECT)&rcFrameRect);
#endif  // ! INPLACE_CNTR

}

#endif  // USE_FRAMETOOLS


/* OutlineDoc_GetWindow
 * --------------------
 *
 *      Get the window handle of the document.
 */
HWND OutlineDoc_GetWindow(LPOUTLINEDOC lpOutlineDoc)
{
   if(! lpOutlineDoc) return NULL;
   return lpOutlineDoc->m_hWndDoc;
}


/* OutlineDoc_AddLine
 * ------------------
 *
 *      Add one line to the Document's LineList
 */
void OutlineDoc_AddLine(LPOUTLINEDOC lpOutlineDoc, LPLINE lpLine, int
nIndex)
{
   LPLINELIST lpLL = &lpOutlineDoc->m_LineList;

   LineList_AddLine(lpLL, lpLine, nIndex);

   /* Update Name Table */
   OutlineNameTable_AddLineUpdate(lpOutlineDoc->m_lpNameTable, nIndex);

#if defined( INPLACE_CNTR )
   {
      LPCONTAINERDOC lpContainerDoc = (LPCONTAINERDOC)lpOutlineDoc;
      /* OLE2NOTE: after adding a line we need to
      **    update the PosRect of the In-Place active
      **    objects (if any) that follow the added line.
      **    NOTE: nIndex is index of line before new line.
      **          nIndex+1 is index of new line
      **          nIndex+2 is index of line after new line.
      */
      ContainerDoc_UpdateInPlaceObjectRects(lpContainerDoc, nIndex+2);
   }
#endif

   OutlineDoc_SetModified(lpOutlineDoc, TRUE, TRUE, TRUE);
}


/* OutlineDoc_DeleteLine
 * ---------------------
 *
 *
 *      Delete one line from the document's LineList
 */
void OutlineDoc_DeleteLine(LPOUTLINEDOC lpOutlineDoc, int nIndex)
{
   LPLINELIST lpLL = &lpOutlineDoc->m_LineList;

#if defined( OLE_CNTR )
   LPLINE lpLine = LineList_GetLine(lpLL, nIndex);
   LPSTORAGE lpStgDoc = NULL;
   OLECHAR szSaveStgName[CWCSTORAGENAME];
   BOOL fDeleteChildStg = FALSE;

   if (lpLine && (Line_GetLineType(lpLine) == CONTAINERLINETYPE) ) {

      /* OLE2NOTE: when a ContainerLine is being deleted by the user,
      **    it is important to delete the object's sub-storage
      **    otherwise it wastes space in the ContainerDoc's file.
      **    this function is called when lines are deleted by the
      **    Clear command and when lines are deleted by a DRAGMOVE
      **    operation.
      */
      LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)lpLine;

      // save name of child storage
//      LSTRCPYN(szSaveStgName, lpContainerLine-
>m_szStgName,sizeof(szSaveStgName));
```

```
      OLESTRCPY(szSaveStgName, lpContainerLine->m_szStgName);
      lpStgDoc = ((LPOLEDOC)lpContainerLine->m_lpDoc)->m_lpStg;
      fDeleteChildStg = TRUE;
   }
#endif  // OLE_CNTR

   LineList_DeleteLine(lpLL, nIndex);

#if defined( OLE_CNTR )
   if (fDeleteChildStg && lpStgDoc) {
      HRESULT hrErr;

      // delete the obsolete child storage. it is NOT fatal if this fails

      hrErr = lpStgDoc->lpVtbl->DestroyElement(lpStgDoc, szSaveStgName);

#if defined( _DEBUG )
      if (hrErr != NOERROR) {
         OleDbgOutHResult("IStorage::DestroyElement return", hrErr);
      }
#endif
   }
#endif  // OLE_CNTR

   /* Update Name Table */
   OutlineNameTable_DeleteLineUpdate(lpOutlineDoc->m_lpNameTable, nIndex);

#if defined( INPLACE_CNTR )
   {
      LPCONTAINERDOC lpContainerDoc = (LPCONTAINERDOC)lpOutlineDoc;
      /* OLE2NOTE: after deleting a line we need to
      **    update the PosRect of the In-Place active
      **    objects (if any).
      */
      ContainerDoc_UpdateInPlaceObjectRects(lpContainerDoc, nIndex);
   }
#endif

   OutlineDoc_SetModified(lpOutlineDoc, TRUE, TRUE, TRUE);

#if defined( OLE_VERSION )
   {
      LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
      LPOLEDOC    lpClipboardDoc = (LPOLEDOC)lpOutlineApp->m_lpClipboardDoc;

      /* OLE2NOTE: if the document that is the source of data on the
      **    clipborad has just had lines deleted, then the copied data
      **    is no longer considered a valid potential link source.
      **    disable the offering of CF_LINKSOURCE from the clipboard
      **    document. this avoids problems that arise when the
      **    editing operation changes or deletes the original data
      **    copied. we will not go to the trouble of determining if
      **    the deleted line actually is part of the link source.
      */
      if (lpClipboardDoc
         && lpClipboardDoc->m_fLinkSourceAvail
         && lpClipboardDoc->m_lpSrcDocOfCopy == (LPOLEDOC)lpOutlineDoc) {
         lpClipboardDoc->m_fLinkSourceAvail = FALSE;

         /* OLE2NOTE: since we are changing the list of formats on
         **    the clipboard (ie. removing CF_LINKSOURCE), we must
         **    call OleSetClipboard again. to be sure that the
         **    clipboard datatransfer document object does not get
         **    destroyed we will guard the call to OleSetClipboard
         **    within a pair of AddRef/Release.
         */
         OleDoc_AddRef((LPOLEDOC)lpClipboardDoc);    // guard obj life-time

         OLEDBG_BEGIN2("OleSetClipboard called\r\n")
         OleSetClipboard(
               (LPDATAOBJECT)&((LPOLEDOC)lpClipboardDoc)->m_DataObject);
         OLEDBG_END2

         OleDoc_Release((LPOLEDOC)lpClipboardDoc);    // rel. AddRef above
      }
   }
#endif  // OLE_VERSION
}


/* OutlineDoc_AddName
 * ------------------
 *
 *      Add a Name to the Document's NameTable
 */
void OutlineDoc_AddName(LPOUTLINEDOC lpOutlineDoc, LPOUTLINENAME
lpOutlineName)
{
   LPOUTLINENAMETABLE lpOutlineNameTable = lpOutlineDoc->m_lpNameTable;

   OutlineNameTable_AddName(lpOutlineNameTable, lpOutlineName);

   OutlineDoc_SetModified(lpOutlineDoc, TRUE, FALSE, FALSE);
}


/* OutlineDoc_DeleteName
 * ---------------------
 *
 *
 *      Delete Name from the document's NameTable
 */
void OutlineDoc_DeleteName(LPOUTLINEDOC lpOutlineDoc, int nIndex)
{
   LPOUTLINENAMETABLE lpOutlineNameTable = lpOutlineDoc->m_lpNameTable;

   OutlineNameTable_DeleteName(lpOutlineNameTable, nIndex);

   OutlineDoc_SetModified(lpOutlineDoc, TRUE, FALSE, FALSE);
}
```

```
/* OutlineDoc_Destroy
 * ------------------
 *
 *  Free all memory that had been allocated for a document.
 *      this destroys the LineList & NameTable of the document.
 */
void OutlineDoc_Destroy(LPOUTLINEDOC lpOutlineDoc)
{
   LPLINELIST lpLL = &lpOutlineDoc->m_LineList;
#if defined( OLE_VERSION )
   LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpOutlineDoc;

   if (lpOleDoc->m_fObjIsDestroying)
      return;     // doc destruction is in progress
#endif  // OLE_VERSION

   OLEDBG_BEGIN3("OutlineDoc_Destroy\r\n");

#if defined( OLE_VERSION )

   /* OLE2NOTE: in order to guarantee that the application does not
   **    prematurely exit before the destruction of the document is
   **    complete, we intially AddRef the App refcnt later Release it.
   **    This initial AddRef is artificial; it simply guarantees that
   **    the app object does not get destroyed until the end of this
   **    routine.
   */
   OleApp_AddRef(lpOleApp);

   /* OLE2NOTE: perform processing required for OLE */
   OleDoc_Destroy(lpOleDoc);
#endif

   LineList_Destroy(lpLL);
   OutlineNameTable_Destroy(lpOutlineDoc->m_lpNameTable);

#if defined( USE_HEADING )
   if (! lpOutlineDoc->m_fDataTransferDoc)
      Heading_Destroy((LPHEADING)&lpOutlineDoc->m_heading);
#endif

#if defined( USE_FRAMETOOLS )
   if (! lpOutlineDoc->m_fDataTransferDoc)
      FrameTools_AssociateDoc(lpOutlineDoc->m_lpFrameTools, NULL);
#endif  // USE_FRAMETOOLS

   DestroyWindow(lpOutlineDoc->m_hWndDoc);
   Delete(lpOutlineDoc);   // free memory for doc itself
   OleDbgOut1("@@@@ DOC DESTROYED\r\n");

#if defined( OLE_VERSION )
   OleApp_Release(lpOleApp);       // release artificial AddRef above
```

```
#endif

   OLEDBG_END3
}


/* OutlineDoc_ReSize
 * -----------------
 *
 *  Resize the document and its components
 *
 * Parameter:
 *      lpRect  the new size of the document. Use current size if NULL
 */
void OutlineDoc_Resize(LPOUTLINEDOC lpOutlineDoc, LPRECT lpRect)
{
   RECT            rect;
   LPLINELIST      lpLL;

#if defined( USE_HEADING )
   LPHEADING       lphead;
#endif  // USE_HEADING

   LPSCALEFACTOR   lpscale;
   HWND            hWndLL;

   if (!lpOutlineDoc)
      return;

   lpLL = (LPLINELIST)&lpOutlineDoc->m_LineList;
   lpscale = (LPSCALEFACTOR)&lpOutlineDoc->m_scale;
   hWndLL = LineList_GetWindow(lpLL);

   if (lpRect) {
      CopyRect((LPRECT)&rect, lpRect);
      MoveWindow(lpOutlineDoc->m_hWndDoc, rect.left, rect.top,
            rect.right-rect.left, rect.bottom-rect.top, TRUE);
   }

   GetClientRect(lpOutlineDoc->m_hWndDoc, (LPRECT)&rect);

#if defined( USE_HEADING )
   lphead = OutlineDoc_GetHeading(lpOutlineDoc);
   rect.left += Heading_RH_GetWidth(lphead, lpscale);
   rect.top += Heading_CH_GetHeight(lphead, lpscale);
#endif  // USE_HEADING

   if (lpLL) {
      MoveWindow(hWndLL, rect.left, rect.top,
            rect.right-rect.left, rect.bottom-rect.top, TRUE);
   }

#if defined( USE_HEADING )
   if (lphead)
      Heading_Move(lphead, lpOutlineDoc->m_hWndDoc, lpscale);
```

```
#endif  // USE_HEADING

#if defined( INPLACE_CNTR )
   ContainerDoc_UpdateInPlaceObjectRects((LPCONTAINERDOC)lpOutlineDoc, 0);
#endif
}


/* OutlineDoc_GetNameTable
 * -----------------------
 *
 *      Get nametable associated with the line list
 */
LPOUTLINENAMETABLE OutlineDoc_GetNameTable(LPOUTLINEDOC lpOutlineDoc)
{
   if (!lpOutlineDoc)
      return NULL;
   else
      return lpOutlineDoc->m_lpNameTable;
}


/* OutlineDoc_GetLineList
 * ----------------------
 *
 *      Get listlist associated with the OutlineDoc
 */
LPLINELIST OutlineDoc_GetLineList(LPOUTLINEDOC lpOutlineDoc)
{
   if (!lpOutlineDoc)
      return NULL;
   else
      return (LPLINELIST)&lpOutlineDoc->m_LineList;
}


/* OutlineDoc_GetNameCount
 * -----------------------
 *
 * Return number of names in table
 */
int OutlineDoc_GetNameCount(LPOUTLINEDOC lpOutlineDoc)
{
   return OutlineNameTable_GetCount(lpOutlineDoc->m_lpNameTable);
}


/* OutlineDoc_GetLineCount
 * -----------------------
 *
 * Return number of lines in the LineList
 */
int OutlineDoc_GetLineCount(LPOUTLINEDOC lpOutlineDoc)
{
   return LineList_GetCount(&lpOutlineDoc->m_LineList);
```

```
}


/* OutlineDoc_SetFileName
 * ----------------------
 *
 *      Set the filename of a document.
 *
 *  OLE2NOTE: If the ServerDoc has a valid filename then, the object is
 *  registered in the running object table (ROT). if the name of the doc
 *  changes (eg. via SaveAs) then the previous registration must be revoked
 *  and the document re-registered under the new name.
 */
BOOL OutlineDoc_SetFileName(LPOUTLINEDOC lpOutlineDoc, LPOLESTR
lpszNewFileName, LPSTORAGE lpNewStg)
{
   OleDbgAssertSz(lpszNewFileName != NULL,   "Can't reset doc to
Untitled!");
   if (lpszNewFileName == NULL)
      return FALSE;

//   AnsiLowerBuff(lpszNewFileName, (UINT)OLESTRLEN(lpszNewFileName));
   CharLowerBuffW(lpszNewFileName, OLESTRLEN(lpszNewFileName));

#if defined( OLE_CNTR )
   {
      LPCONTAINERDOC lpContainerDoc = (LPCONTAINERDOC)lpOutlineDoc;
      LPOLEDOC lpOleDoc = (LPOLEDOC)lpOutlineDoc;

      /* OLE2NOTE: the container version of the application keeps its
      **    storage open at all times. if the document's storage is not
      **    open, then open it.
      */

      if (lpNewStg) {

         /* CASE 1 -- document is being loaded from a file. lpNewStg is
         **    still open from the OutlineDoc_LoadFromFile function.
         */

         lpOutlineDoc->m_docInitType = DOCTYPE_FROMFILE;

      } else {

         /* CASE 2 -- document is being associated with a valid file
         **    that is not yet open. thus we must now open the file.
         */

         if (lpOutlineDoc->m_docInitType == DOCTYPE_FROMFILE &&
               OLESTRCMP(lpOutlineDoc->m_szFileName,lpszNewFileName)==0) {

            /* CASE 2a -- new filename is same as current file. if the
            **    stg is already open, then the lpStg is still valid.
            **    if it is not open, then open it.
            */
            if (! lpOleDoc->m_lpStg) {
               lpOleDoc->m_lpStg = OleStdOpenRootStorage(
                     lpszNewFileName,
                     STGM_READWRITE | STGM_SHARE_DENY_WRITE
               );
               if (! lpOleDoc->m_lpStg) return FALSE;
            }

         } else {

            /* CASE 2b -- new filename is NOT same as current file.
            **    a SaveAs operation is pending. open the new file and
            **    hold the storage pointer in m_lpNewStg. the
            **    subsequent call to Doc_SaveToFile will save the
            **    document into the new storage pointer and release the
            **    old storage pointer.
            */

            lpOutlineDoc->m_docInitType = DOCTYPE_FROMFILE;

            lpContainerDoc->m_lpNewStg = OleStdCreateRootStorage(
                  lpszNewFileName,
                  STGM_READWRITE | STGM_SHARE_DENY_WRITE | STGM_CREATE
            );
            if (! lpContainerDoc->m_lpNewStg) return FALSE;
         }
      }
   }
#endif      // OLE_CNTR

   if (lpOutlineDoc->m_docInitType != DOCTYPE_FROMFILE ||
      OLESTRCMP(lpOutlineDoc->m_szFileName, lpszNewFileName) != 0) {

      /* A new valid file name is being associated with the document */

      OLESTRCPY(lpOutlineDoc->m_szFileName, lpszNewFileName);
      lpOutlineDoc->m_docInitType = DOCTYPE_FROMFILE;

      // set lpszDocTitle to point to filename without path
      lpOutlineDoc->m_lpszDocTitle = lpOutlineDoc->m_szFileName +
         OLESTRLEN(lpOutlineDoc->m_szFileName) - 1;
      while (lpOutlineDoc->m_lpszDocTitle > lpOutlineDoc->m_szFileName
         && ! IS_FILENAME_DELIM(lpOutlineDoc->m_lpszDocTitle[-1])) {
         lpOutlineDoc->m_lpszDocTitle--;
      }

      OutlineDoc_SetTitle(lpOutlineDoc, TRUE /*fMakeUpperCase*/);

#if defined( OLE_VERSION )
      {
         /* OLE2NOTE: both containers and servers must properly
         **    register in the RunningObjectTable. if the document
         **    is performing a SaveAs operation, then it must
         **    re-register in the ROT with the new moniker. in
         **    addition any embedded object, pseudo objects, and/or
         **    linking clients must be informed that the document's
         **    moniker has changed.
         */

         LPOLEDOC lpOleDoc = (LPOLEDOC)lpOutlineDoc;

         if (lpOleDoc->m_lpFileMoniker) {
            OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpFileMoniker);
            lpOleDoc->m_lpFileMoniker = NULL;
         }

         CreateFileMoniker(lpszNewFileName,&lpOleDoc->m_lpFileMoniker);
         OleDoc_DocRenamedUpdate(lpOleDoc, lpOleDoc->m_lpFileMoniker);
      }
#endif      // OLE_VERSION

   }

   return TRUE;
}


/* OutlineDoc_SetTitle
 * -------------------
 *
 * Set window text to be current filename.
 * The following window hierarchy exits:
 *      hWndApp
 *          hWndDoc
 *              hWndListBox
 *  The frame window is the window which gets the title.
 */
void OutlineDoc_SetTitle(LPOUTLINEDOC lpOutlineDoc, BOOL fMakeUpperCase)
{
   HWND hWnd;
   LPSTR lpszText;
   char  szAnsiStr[256];

   if (!lpOutlineDoc->m_hWndDoc) return;
   if ((hWnd = GetParent(lpOutlineDoc->m_hWndDoc)) == NULL) return;

   lpszText = OleStdMalloc((UINT)(OLESTRLEN(APPNAME) + 4 +
                           OLESTRLEN(lpOutlineDoc->m_lpszDocTitle)));
   if (!lpszText) return;

   W2A(APPNAME, lpszText, OLESTRLEN(APPNAME)+1);
   lstrcat(lpszText," - ");
   W2A (lpOutlineDoc->m_lpszDocTitle, szAnsiStr, 256);
   lstrcat(lpszText, (LPSTR)szAnsiStr);

   if (fMakeUpperCase)
      AnsiUpperBuff(lpszText, (UINT)lstrlen(lpszText));

   SetWindowText(hWnd,lpszText);
   OleStdFree(lpszText);
```

```
}


/* OutlineDoc_Close
 * ----------------
 *
 * Close active document. If modified, prompt the user if
 * he wants to save.
 *
 *  Returns:
 *      FALSE -- user canceled the closing of the doc.
 *      TRUE -- the doc was successfully closed
 */
BOOL OutlineDoc_Close(LPOUTLINEDOC lpOutlineDoc, DWORD dwSaveOption)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;

#if defined( OLE_VERSION )
   /* OLE2NOTE: call OLE specific function instead */
   return OleDoc_Close((LPOLEDOC)lpOutlineDoc, dwSaveOption);

#else

   if (! lpOutlineDoc)
      return TRUE;            // active doc's are already destroyed

   if (! OutlineDoc_CheckSaveChanges(lpOutlineDoc, &dwSaveOption))
      return FALSE;           // abort closing the doc

   OutlineDoc_Destroy(lpOutlineDoc);

   OutlineApp_DocUnlockApp(lpOutlineApp, lpOutlineDoc);

   return TRUE;

#endif      // ! OLE_VERSION
}


/* OutlineDoc_CheckSaveChanges
 * ---------------------------
 *
 * Check if the document has been modified. if so, prompt the user if
 *      the changes should be saved. if yes save them.
 * Returns TRUE if the doc is safe to close (user answered Yes or No)
 *         FALSE if the user canceled the save changes option.
 */
BOOL OutlineDoc_CheckSaveChanges(
      LPOUTLINEDOC        lpOutlineDoc,
      LPDWORD             lpdwSaveOption
)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   int nResponse;
   char szAnsiString[256];
   char szAnsiString2[256];

   if (*lpdwSaveOption == OLECLOSE_NOSAVE)
      return TRUE;

   if(! OutlineDoc_IsModified(lpOutlineDoc))
      return TRUE;    // saving is not necessary

   /* OLE2NOTE: our document is dirty so it needs to be saved. if
   **    OLECLOSE_PROMPTSAVE the user should be prompted to see if the
   **    document should be saved. is specified but the document is NOT
   **    visible to the user, then the user can NOT be prompted. in
   **    the situation the document should be saved without prompting.
   **    if OLECLOSE_SAVEIFDIRTY is specified then, the document
   **    should also be saved without prompting.
   */
   if (*lpdwSaveOption == OLECLOSE_PROMPTSAVE &&
         IsWindowVisible(lpOutlineDoc->m_hWndDoc)) {

      // prompt the user to see if changes should be saved.
#if defined( OLE_VERSION )
      OleApp_PreModalDialog(
            (LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineApp->m_lpDoc);
#endif
      W2A (APPNAME, szAnsiString, 256);
      W2A (MsgSaveFile, szAnsiString2, 256);
      nResponse = MessageBox(
            lpOutlineApp->m_hWndApp,
            szAnsiString2,
            szAnsiString,
            MB_ICONQUESTION | MB_YESNOCANCEL
      );
#if defined( OLE_VERSION )
      OleApp_PostModalDialog(
            (LPOLEAPP)lpOutlineApp, (LPOLEDOC)lpOutlineApp->m_lpDoc);
#endif
      if(nResponse==IDCANCEL)
         return FALSE;   // close is canceled
      if(nResponse==IDNO) {
         // Reset the save option to NOSAVE per user choice
         *lpdwSaveOption = OLECLOSE_NOSAVE;
         return TRUE;    // don't save, but is ok to close
      }
   } else if (*lpdwSaveOption != OLECLOSE_SAVEIFDIRTY) {
      OleDbgAssertSz(FALSE, "Invalid dwSaveOption\r\n");
      *lpdwSaveOption = OLECLOSE_NOSAVE;
      return TRUE;        // unknown *lpdwSaveOption; close w/o saving
   }

#if defined( OLE_SERVER )
   if (lpOutlineDoc->m_docInitType == DOCTYPE_EMBEDDED) {
      LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOutlineDoc;
      HRESULT hrErr;

      /* OLE2NOTE: Update the container before closing without prompting
```

```
        **    the user. To update the container, we must ask our container
        **    to save us.
        */
        OleDbgAssert(lpServerDoc->m_lpOleClientSite != NULL);
        OLEDBG_BEGIN2("IOleClientSite::SaveObject called\r\n")
        hrErr = lpServerDoc->m_lpOleClientSite->lpVtbl->SaveObject(
              lpServerDoc->m_lpOleClientSite
        );
        OLEDBG_END2

        if (hrErr != NOERROR) {
           OleDbgOutHResult("IOleClientSite::SaveObject returned", hrErr);
           return FALSE;
        }

        return TRUE;    // doc is safe to be closed

     } else
#endif      // OLE_SERVER
     {
        return OutlineApp_SaveCommand(lpOutlineApp);
     }
}


/* OutlineDoc_IsModified
 * ---------------------
 *
 * Return modify flag of OUTLINEDOC
 */
BOOL OutlineDoc_IsModified(LPOUTLINEDOC lpOutlineDoc)
{
   if (lpOutlineDoc->m_fModified)
      return lpOutlineDoc->m_fModified;

#if defined( OLE_CNTR )
   {
      /* OLE2NOTE: if there are OLE objects, then we must ask if any of
      **    them are dirty. if so we must consider our document
      **    as modified.
      */
      LPCONTAINERDOC lpContainerDoc = (LPCONTAINERDOC)lpOutlineDoc;
      LPLINELIST  lpLL;
      int         nLines;
      int         nIndex;
      LPLINE      lpLine;
      HRESULT     hrErr;

      lpLL = (LPLINELIST)&((LPOUTLINEDOC)lpContainerDoc)->m_LineList;
      nLines = LineList_GetCount(lpLL);

      for (nIndex = 0; nIndex < nLines; nIndex++) {
         lpLine = LineList_GetLine(lpLL, nIndex);
         if (!lpLine)
            break;
         if (Line_GetLineType(lpLine) == CONTAINERLINETYPE) {
            LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)lpLine;
            if (lpContainerLine->m_lpPersistStg) {
               hrErr = lpContainerLine->m_lpPersistStg->lpVtbl->IsDirty(
                     lpContainerLine->m_lpPersistStg);
               /* OLE2NOTE: we will only accept an explicit "no i
               **    am NOT dirty statement" (ie. S_FALSE) as an
               **    indication that the object is clean. eg. if
               **    the object returns E_NOTIMPL we must
               **    interpret it as the object IS dirty.
               */
               if (GetScode(hrErr) != S_FALSE)
                  return TRUE;
            }
         }
      }
   }
#endif
   return FALSE;
}


/* OutlineDoc_SetModified
 * ----------------------
 *
 *  Set the modified flag of the document
 *
 */
void OutlineDoc_SetModified(LPOUTLINEDOC lpOutlineDoc, BOOL fModified, BOOL
fDataChanged, BOOL fSizeChanged)
{
   lpOutlineDoc->m_fModified = fModified;

#if defined( OLE_SERVER )
   if (! lpOutlineDoc->m_fDataTransferDoc) {
      LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOutlineDoc;

      /* OLE2NOTE: if the document has changed, then broadcast the change
      **    to all clients who have set up Advise connections. notify
      **    them that our data (and possibly also our extents) have
      **    changed.
      */
      if (fDataChanged) {
         lpServerDoc->m_fDataChanged     = TRUE;
         lpServerDoc->m_fSizeChanged     = fSizeChanged;
         lpServerDoc->m_fSendDataOnStop  = TRUE;

         ServerDoc_SendAdvise(
               lpServerDoc,
               OLE_ONDATACHANGE,
               NULL,   /* lpmkDoc -- not relevant here */
               0       /* advf -- no flags necessary */
         );
      }
   }
```

```
#endif  // OLE_SERVER
}


/* OutlineDoc_SetRedraw
 * --------------------
 *
 *  Enable/Disable the redraw of the document on screen.
 *  The calls to SetRedraw counted so that nested calls can be handled
 *  properly. calls to SetRedraw must be balanced.
 *
 *  fEnbaleDraw = TRUE      - enable redraw
 *                FALSE     - disable redraw
 */
void OutlineDoc_SetRedraw(LPOUTLINEDOC lpOutlineDoc, BOOL fEnableDraw)
{
   static HCURSOR hPrevCursor = NULL;

   if (fEnableDraw) {
      if (lpOutlineDoc->m_nDisableDraw == 0)
         return;     // already enabled; no state transition

      if (--lpOutlineDoc->m_nDisableDraw > 0)
         return;     // drawing should still be disabled
   } else {
      if (lpOutlineDoc->m_nDisableDraw++ > 0)
         return;     // already disabled; no state transition
   }

   if (lpOutlineDoc->m_nDisableDraw > 0) {
      // this may take a while, put up hourglass cursor
      hPrevCursor = SetCursor(LoadCursor(NULL, IDC_WAIT));
   } else {
      if (hPrevCursor) {
         SetCursor(hPrevCursor);     // restore original cursor
         hPrevCursor = NULL;
      }
   }

#if defined( OLE_SERVER )
   /* OLE2NOTE: for the Server version, while Redraw is disabled
   **    postpone sending advise notifications until Redraw is re-enabled.
   */
   {
      LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOutlineDoc;
      LPSERVERNAMETABLE lpServerNameTable =
            (LPSERVERNAMETABLE)lpOutlineDoc->m_lpNameTable;

      if (lpOutlineDoc->m_nDisableDraw == 0) {
         /* drawing is being Enabled. if changes occurred while drawing
         **    was disabled, then notify clients now.
         */
         if (lpServerDoc->m_fDataChanged)
            ServerDoc_SendAdvise(
                  lpServerDoc,
                  OLE_ONDATACHANGE,
                  NULL,   /* lpmkDoc -- not relevant here */
                  0       /* advf -- no flags necessary */
            );

         /* OLE2NOTE: send pending change notifications for pseudo objs. */
         ServerNameTable_SendPendingAdvises(lpServerNameTable);

      }
   }
#endif      // OLE_SERVER

#if defined( OLE_CNTR )
   /* OLE2NOTE: for the Container version, while Redraw is disabled
   **    postpone updating the extents of OLE objects until Redraw is
   **    re-enabled.
   */
   {
      LPCONTAINERDOC lpContainerDoc = (LPCONTAINERDOC)lpOutlineDoc;

      /* Update the extents of any OLE object that is marked that
      **    its size may  have changed. when an
      **    IAdviseSink::OnViewChange notification is received,
      **    the corresponding ContainerLine is marked
      **    (m_fDoGetExtent==TRUE) and a message
      **    (WM_U_UPDATEOBJECTEXTENT) is posted to the document
      **    indicating that there are dirty objects.
      */
      if (lpOutlineDoc->m_nDisableDraw == 0)
         ContainerDoc_UpdateExtentOfAllOleObjects(lpContainerDoc);
   }
#endif      // OLE_CNTR

   // enable/disable redraw of the LineList listbox
   LineList_SetRedraw(&lpOutlineDoc->m_LineList, fEnableDraw);
}


/* OutlineDoc_SetSel
 * -----------------
 *
 *      Set the selection in the documents's LineList
 */
void OutlineDoc_SetSel(LPOUTLINEDOC lpOutlineDoc, LPLINERANGE lplrSel)
{
   LineList_SetSel(&lpOutlineDoc->m_LineList, lplrSel);
}


/* OutlineDoc_GetSel
 * -----------------
 *
 *      Get the selection in the documents's LineList.
 *
 *      Returns the count of items selected
 */
int OutlineDoc_GetSel(LPOUTLINEDOC lpOutlineDoc, LPLINERANGE lplrSel)
{
   return LineList_GetSel(&lpOutlineDoc->m_LineList, lplrSel);
}


/* OutlineDoc_ForceRedraw
 * ----------------------
 *
 *      Force the document window to repaint.
 */
void OutlineDoc_ForceRedraw(LPOUTLINEDOC lpOutlineDoc, BOOL fErase)
{
   if (!lpOutlineDoc)
      return;

   LineList_ForceRedraw(&lpOutlineDoc->m_LineList, fErase);
   Heading_CH_ForceRedraw(&lpOutlineDoc->m_heading, fErase);
   Heading_RH_ForceRedraw(&lpOutlineDoc->m_heading, fErase);
}


/* OutlineDoc_RenderFormat
 * -----------------------
 *
 *      Render a clipboard format supported by ClipboardDoc
 */
void OutlineDoc_RenderFormat(LPOUTLINEDOC lpOutlineDoc, UINT uFormat)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   HGLOBAL      hData = NULL;

   if (uFormat == lpOutlineApp->m_cfOutline)
      hData = OutlineDoc_GetOutlineData(lpOutlineDoc, NULL);

   else if (uFormat == CF_TEXT)
      hData = OutlineDoc_GetTextData(lpOutlineDoc, NULL);

   else {
      OutlineApp_ErrorMessage(lpOutlineApp, ErrMsgFormatNotSupported);
      return;
   }

   SetClipboardData(uFormat, hData);
}


/* OutlineDoc_RenderAllFormats
 * ---------------------------
 *
 *      Render all formats supported by ClipboardDoc
 */
void OutlineDoc_RenderAllFormats(LPOUTLINEDOC lpOutlineDoc)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   HGLOBAL      hData = NULL;

   OpenClipboard(lpOutlineDoc->m_hWndDoc);

   hData = OutlineDoc_GetOutlineData(lpOutlineDoc, NULL);
   SetClipboardData(lpOutlineApp->m_cfOutline, hData);

   hData = OutlineDoc_GetTextData(lpOutlineDoc, NULL);
   SetClipboardData(CF_TEXT, hData);

   CloseClipboard();
}



/* OutlineDoc_GetOutlineData
 * -------------------------
 *
 * Return a handle to an array of TextLine objects for the desired line
 *      range.
 *  NOTE: if lplrSel == NULL, then all lines are returned
 *
 */
HGLOBAL OutlineDoc_GetOutlineData(LPOUTLINEDOC lpOutlineDoc, LPLINERANGE
lplrSel)
{
   HGLOBAL     hOutline  = NULL;
   LPLINELIST  lpLL=(LPLINELIST)&lpOutlineDoc->m_LineList;
   LPLINE      lpLine;
   LPTEXTLINE  arrLine;
   int     i;
   int     nStart = (lplrSel ? lplrSel->m_nStartLine : 0);
   int     nEnd =(lplrSel ? lplrSel->m_nEndLine :
LineList_GetCount(lpLL)-1);
   int     nLines = nEnd - nStart + 1;
   int     nCopied = 0;

   hOutline=GlobalAlloc(GALLOCFLG, sizeof(TEXTLINE)*nLines);

   if (! hOutline) return NULL;

   arrLine=(LPTEXTLINE)GlobalLock(hOutline);

   for (i = nStart; i <= nEnd; i++) {
      lpLine=LineList_GetLine(lpLL, i);
      if (lpLine && Line_GetOutlineData(lpLine, &arrLine[nCopied]))
         nCopied++;
   }

   GlobalUnlock(hOutline);

   return hOutline;
}
```

```
/* OutlineDoc_GetTextData
 * ----------------------
 *
 * Return a handle to an object's data in text form for the desired line
 *      range.
 *  NOTE: if lplrSel == NULL, then all lines are returned
 *
 */
HGLOBAL OutlineDoc_GetTextData(LPOUTLINEDOC lpOutlineDoc, LPLINERANGE
lplrSel)
{
   LPLINELIST  lpLL=(LPLINELIST)&lpOutlineDoc->m_LineList;
   LPLINE  lpLine;
   HGLOBAL hText = NULL;
   LPOLESTR   lpszText = NULL;
   DWORD   dwMemSize=0;
   int     i,j;
   int     nStart = (lplrSel ? lplrSel->m_nStartLine : 0);
   int     nEnd =(lplrSel ? lplrSel->m_nEndLine :
LineList_GetCount(lpLL)-1);
   int     nTabLevel;

   // calculate memory size required
   for(i = nStart; i <= nEnd; i++) {
      lpLine=LineList_GetLine(lpLL, i);
      if (! lpLine)
         continue;

      dwMemSize += Line_GetTabLevel(lpLine);
      dwMemSize += Line_GetTextLen(lpLine);

      dwMemSize += 2; // add 1 for '\r\n' at the end of each line
   }
   dwMemSize++;         // add 1 for '\0' at the end of string

   if(!(hText = GlobalAlloc(GALLOCFLG, dwMemSize)))
      return NULL;

   if(!(lpszText = (LPOLESTR)GlobalLock(hText)))
      return NULL;

   // put line text to memory
   for(i = nStart; i <= nEnd; i++) {
      lpLine=LineList_GetLine(lpLL, i);
      if (! lpLine)
         continue;

      nTabLevel=Line_GetTabLevel(lpLine);
      for(j = 0; j < nTabLevel; j++)
         *lpszText++='\t';

      Line_GetTextData(lpLine, lpszText);
      while(*lpszText)
```

```
            lpszText++;     // advance to end of string

         *lpszText++ = '\r';
         *lpszText++ = '\n';
      }

      GlobalUnlock (hText);

      return hText;
   }


   /* OutlineDoc_SaveToFile
    * ---------------------
    *
    *      Save the document to a file with the same name as stored in the
    * document
    */
   BOOL OutlineDoc_SaveToFile(LPOUTLINEDOC lpOutlineDoc, LPCOLESTR
   lpszFileName, UINT uFormat, BOOL fRemember)
   {
   #if defined( OLE_CNTR )
      // Call OLE container specific function instead
      return ContainerDoc_SaveToFile(
            (LPCONTAINERDOC)lpOutlineDoc,
            lpszFileName,
            uFormat,
            fRemember
      );

   #else

      LPSTORAGE lpDestStg = NULL;
      HRESULT hrErr;
      BOOL fStatus;
      LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;

      if (fRemember) {
         if (lpszFileName) {
            fStatus = OutlineDoc_SetFileName(
                  lpOutlineDoc,
                  (LPOLESTR)lpszFileName,
                  NULL
            );
            if (! fStatus) goto error;
         } else
            lpszFileName = lpOutlineDoc->m_szFileName; // use cur. file name
      } else if (! lpszFileName) {
         goto error;
      }

      hrErr = StgCreateDocfile(
            lpszFileName,
            STGM_READWRITE|STGM_DIRECT|STGM_SHARE_EXCLUSIVE|STGM_CREATE,
            0,
            &lpDestStg
      );

      OleDbgAssertSz(hrErr == NOERROR, "Could not create Docfile");
      if (hrErr != NOERROR)
         goto error;

   #if defined( OLE_SERVER )

      /*  OLE2NOTE: we must be sure to write our class ID into our
      **    storage. this information is used by OLE to determine the
      **    class of the data stored in our storage. Even for top
      **    "file-level" objects this information should be written to
      **    the file.
      */
      if(WriteClassStg(lpDestStg, &CLSID_APP) != NOERROR)
         goto error;
   #endif

      fStatus = OutlineDoc_SaveSelToStg(
            lpOutlineDoc,
            NULL,
            uFormat,
            lpDestStg,
            FALSE,      /* fSameAsLoad */
            fRemember
      );
      if (! fStatus) goto error;

      OleStdRelease((LPUNKNOWN)lpDestStg);

      if (fRemember)
         OutlineDoc_SetModified(lpOutlineDoc, FALSE, FALSE, FALSE);

   #if defined( OLE_SERVER )

      /* OLE2NOTE: (SERVER-ONLY) inform any linking clients that the
      **    document has been saved. in addition, any currently active
      **    pseudo objects should also inform their clients.
      */
      ServerDoc_SendAdvise (
            (LPSERVERDOC)lpOutlineDoc,
            OLE_ONSAVE,
            NULL,   /* lpmkDoc -- not relevant here */
            0       /* advf -- not relevant here */
      );

   #endif

      return TRUE;

   error:
      if (lpDestStg)
         OleStdRelease((LPUNKNOWN)lpDestStg);
      OutlineApp_ErrorMessage(lpOutlineApp, ErrMsgSaving);
      return FALSE;

   #endif  // ! OLE_CNTR
   }


   /* OutlineDoc_LoadFromFile
    * -----------------------
    *
    *      Load a document from a file
    */
   BOOL OutlineDoc_LoadFromFile(LPOUTLINEDOC lpOutlineDoc, LPOLESTR
   lpszFileName)
   {
      LPOUTLINEAPP    lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
      LPLINELIST      lpLL = &lpOutlineDoc->m_LineList;
      HRESULT         hrErr;
      SCODE           sc;
      LPSTORAGE       lpSrcStg;
      BOOL            fStatus;

      hrErr = StgOpenStorage(lpszFileName,
            NULL,
   #if defined( OLE_CNTR )
            STGM_READWRITE | STGM_TRANSACTED | STGM_SHARE_DENY_WRITE,
   #else
            STGM_READ | STGM_SHARE_DENY_WRITE,
   #endif
            NULL,
            0,
            &lpSrcStg
      );

      if ((sc = GetScode(hrErr)) == STG_E_FILENOTFOUND) {
         OutlineApp_ErrorMessage(lpOutlineApp, OLESTR("File not found"));
         return FALSE;
      } else if (sc == STG_E_FILEALREADYEXISTS) {
         OutlineApp_ErrorMessage(lpOutlineApp, ErrMsgFormat);
         return FALSE;
      } else if (sc != S_OK) {
         OleDbgOutScode("StgOpenStorage returned", sc);
         OutlineApp_ErrorMessage(
               lpOutlineApp,
               OLESTR("File already in use--could not be opened")
         );
         return FALSE;
      }

      if(! OutlineDoc_LoadFromStg(lpOutlineDoc, lpSrcStg)) goto error;

      fStatus = OutlineDoc_SetFileName(lpOutlineDoc, lpszFileName, lpSrcStg);
      if (! fStatus) goto error;

      OleStdRelease((LPUNKNOWN)lpSrcStg);
      return TRUE;

   error:
      OleStdRelease((LPUNKNOWN)lpSrcStg);
      OutlineApp_ErrorMessage(lpOutlineApp, ErrMsgOpening);
      return FALSE;
   }



   /* OutlineDoc_LoadFromStg
    * ----------------------
    *
    *  Load entire document from an open IStorage pointer (lpSrcStg)
    *      Return TRUE if ok, FALSE if error.
    */
   BOOL OutlineDoc_LoadFromStg(LPOUTLINEDOC lpOutlineDoc, LPSTORAGE lpSrcStg)
   {
      LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
      HRESULT hrErr;
      BOOL fStatus;
      ULONG nRead;
      LINERANGE lrSel = { 0, 0 };
      LPSTREAM lpLLStm;
      OUTLINEDOCHEADER docRecord;
      OUTLINEDOCHEADER_ONDISK docRecordOnDisk;

      hrErr = lpSrcStg->lpVtbl->OpenStream(
            lpSrcStg,
            OLESTR("LineList"),
            NULL,
            STGM_READ | STGM_SHARE_EXCLUSIVE,
            0,
            &lpLLStm
      );

      if (hrErr != NOERROR) {
         OleDbgOutHResult("Open LineList Stream returned", hrErr);
         goto error;
      }

      /* read OutlineDoc header record */
      hrErr = lpLLStm->lpVtbl->Read(
            lpLLStm,
            (LPVOID)&docRecordOnDisk,
            sizeof(docRecordOnDisk),
            &nRead
      );

      if (hrErr != NOERROR) {
         OleDbgOutHResult("Read OutlineDoc header returned", hrErr);
         goto error;
       }
       //  Transform docRecordOnDisk into docRecord
       //  Compilers should handle aligment correctly
       OLESTRCPY(docRecord.m_szFormatName, docRecordOnDisk.m_szFormatName);
       docRecord.m_narrAppVersionNo[0] = (int)
   docRecordOnDisk.m_narrAppVersionNo[0];
       docRecord.m_narrAppVersionNo[1] = (int)
   docRecordOnDisk.m_narrAppVersionNo[1];
       docRecord.m_fShowHeading = (BOOL) docRecordOnDisk.m_fShowHeading;
       docRecord.m_reserved1 = docRecordOnDisk.m_reserved1;
       docRecord.m_reserved2 = docRecordOnDisk.m_reserved2;
       docRecord.m_reserved3 = docRecordOnDisk.m_reserved3;
       docRecord.m_reserved4 = docRecordOnDisk.m_reserved4;

      fStatus = OutlineApp_VersionNoCheck(
            lpOutlineApp,
            docRecord.m_szFormatName,
            docRecord.m_narrAppVersionNo
      );

      /* storage is an incompatible version; file can not be read */
      if (! fStatus)
         goto error;

      lpOutlineDoc->m_heading.m_fShow = docRecord.m_fShowHeading;

   #if defined( OLE_SERVER )
      {
         // Load ServerDoc specific data
         LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOutlineDoc;
   #if defined( SVR_TREATAS )
         LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
         CLSID       clsid;
         CLIPFORMAT  cfFmt;
         LPOLESTR       lpszType;
   #endif  // SVR_TREATAS

         lpServerDoc->m_nNextRangeNo = (ULONG)docRecord.m_reserved1;

   #if defined( SVR_TREATAS )
         /* OLE2NOTE: if the Server is capable of supporting "TreatAs"
         **    (aka. ActivateAs), it must read the class that is written
         **    into the storage. if this class is NOT the app's own
         **    class ID, then this is a TreatAs operation. the server
         **    then must faithfully pretend to be the class that is
         **    written into the storage. it must also faithfully write
         **    the data back to the storage in the SAME format as is
         **    written in the storage.
         **
         **    SVROUTL and ISVROTL can emulate each other. they have the
         **    simplification that they both read/write the identical
         **    format. thus for these apps no actual conversion of the
         **    native bits is actually required.
         */
         lpServerDoc->m_clsidTreatAs = CLSID_NULL;
         if (OleStdGetTreatAsFmtUserType(&CLSID_APP, lpSrcStg, &clsid,
                        (CLIPFORMAT FAR*)&cfFmt, (LPOLESTR FAR*)&lpszType)) {

            if (cfFmt == lpOutlineApp->m_cfOutline) {
               // We should perform TreatAs operation
               if (lpServerDoc->m_lpszTreatAsType)
                  OleStdFreeString(lpServerDoc->m_lpszTreatAsType, NULL);

               lpServerDoc->m_clsidTreatAs = clsid;
               ((LPOUTLINEDOC)lpServerDoc)->m_cfSaveFormat = cfFmt;
               lpServerDoc->m_lpszTreatAsType = lpszType;

               OleDbgOut3("OutlineDoc_LoadFromStg: TreateAs ==> '");
   //          OleDbgOutNoPrefix3(lpServerDoc->m_lpszTreatAsType);
               OleDbgOutNoPrefix3("'\r\n");
            } else {
               // ERROR: we ONLY support TreatAs for CF_OUTLINE format
               OleDbgOut("SvrDoc_PStg_InitNew: INVALID TreatAs Format\r\n");
               OleStdFreeString(lpszType, NULL);
            }
         }
   #endif  // SVR_TREATAS
      }
   #endif  // OLE_SVR
   #if defined( OLE_CNTR )
      {
         // Load ContainerDoc specific data
         LPCONTAINERDOC lpContainerDoc = (LPCONTAINERDOC)lpOutlineDoc;

         lpContainerDoc->m_nNextObjNo = (ULONG)docRecord.m_reserved2;
      }
   #endif  // OLE_CNTR

      OutlineDoc_SetRedraw ( lpOutlineDoc, FALSE );

      if(! LineList_LoadFromStg(&lpOutlineDoc->m_LineList, lpSrcStg, lpLLStm))
         goto error;
      if(! OutlineNameTable_LoadFromStg(lpOutlineDoc->m_lpNameTable, lpSrcStg))
         goto error;

      OutlineDoc_SetModified(lpOutlineDoc, FALSE, FALSE, FALSE);
      OutlineDoc_SetSel(lpOutlineDoc, &lrSel);

      OutlineDoc_SetRedraw ( lpOutlineDoc, TRUE );

      OleStdRelease((LPUNKNOWN)lpLLStm);

   #if defined( OLE_CNTR )
      {
         LPOLEDOC lpOleDoc = (LPOLEDOC)lpOutlineDoc;

         /* A ContainerDoc keeps its storage open at all times. it is necessary
         *   to AddRef the lpSrcStg in order to hang on to it.
         */
         if (lpOleDoc->m_lpStg) {
            OleStdVerifyRelease((LPUNKNOWN)lpOleDoc->m_lpStg,
                  OLESTR("Doc Storage not released properly"));
         }
         lpSrcStg->lpVtbl->AddRef(lpSrcStg);
         lpOleDoc->m_lpStg = lpSrcStg;
      }
   #endif      // OLE_CNTR

      return TRUE;

   error:
      OutlineDoc_SetRedraw ( lpOutlineDoc, TRUE );
      if (lpLLStm)
         OleStdRelease((LPUNKNOWN)lpLLStm);
      return FALSE;
   }


   /* OutlineDoc_SaveSelToStg
    * -----------------------
    *
    *      Save the specified selection of document into an IStorage*. All
   lines
    * within the selection along with any names completely contained within the
    * selection will be written
    *
    *      Return TRUE if ok, FALSE if error
    */
   BOOL OutlineDoc_SaveSelToStg(
         LPOUTLINEDOC        lpOutlineDoc,
         LPLINERANGE         lplrSel,
         UINT                uFormat,
         LPSTORAGE           lpDestStg,
         BOOL                fSameAsLoad,
         BOOL                fRemember
   )
   {
      LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
      HRESULT hrErr = NOERROR;
      LPSTREAM lpLLStm = NULL;
      LPSTREAM lpNTStm = NULL;
      ULONG nWritten;
      BOOL fStatus;
      OUTLINEDOCHEADER docRecord;
      OUTLINEDOCHEADER_ONDISK docRecordOnDisk;
      HCURSOR  hPrevCursor;
      char szAnsiString[256];

   #if defined( OLE_VERSION )
      LPOLESTR lpszUserType;
      LPOLEDOC lpOleDoc = (LPOLEDOC)lpOutlineDoc;

      /*  OLE2NOTE: we must be sure to write the information required for
      **    OLE into our docfile. this includes user type
      **    name, data format, etc. Even for top "file-level" objects
      **    this information should be written to the file. Both
      **    containters and servers should write this information.
      */

   #if defined( OLE_SERVER ) && defined( SVR_TREATAS )
      LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOutlineDoc;

      /* OLE2NOTE: if the Server is emulating another class (ie.
      **    "TreatAs" aka. ActivateAs), it must write the same user type
      **    name and format that was was originally written into the
      **    storage rather than its own user type name.
      **
      **    SVROUTL and ISVROTL can emulate each other. they have the
      **    simplification that they both read/write the identical
      **    format. thus for these apps no actual conversion of the
      **    native bits is actually required.
      */
      if (! IsEqualCLSID(&lpServerDoc->m_clsidTreatAs, &CLSID_NULL))
         lpszUserType = lpServerDoc->m_lpszTreatAsType;
      else
   #endif  // OLE_SERVER && SVR_TREATAS

         lpszUserType = (LPOLESTR)FULLUSERTYPENAME;

      hrErr = WriteFmtUserTypeStg(lpDestStg, (CLIPFORMAT)uFormat,
   lpszUserType);
      if(hrErr != NOERROR) goto error;

      if (fSameAsLoad) {
         /* OLE2NOTE: we are saving into to same storage that we were
         **    passed an load time. we deliberatly opened the streams we
         **    need (lpLLStm and lpNTStm) at load time, so that we can
         **    robustly save at save time in a low-memory situation.
         **    this is particulary important the embedded objects do NOT
         **    consume additional memory when
         **    IPersistStorage::Save(fSameAsLoad==TRUE) is called.
         */
         LARGE_INTEGER libZero;
         ULARGE_INTEGER ulibZero;
         LISet32( libZero, 0 );
         LISet32( ulibZero, 0 );
         lpLLStm = lpOleDoc->m_lpLLStm;

         /*  because this is the fSameAsLoad==TRUE case, we will save
         **    into the streams that we hold open. we will AddRef the
         **    stream here so that the release below will NOT close the
         **    stream.
         */
         lpLLStm->lpVtbl->AddRef(lpLLStm);

         // truncate the current stream and seek to beginning
         lpLLStm->lpVtbl->SetSize(lpLLStm, ulibZero);
         lpLLStm->lpVtbl->Seek(
               lpLLStm, libZero, STREAM_SEEK_SET, NULL);

         lpNTStm = lpOleDoc->m_lpNTStm;
         lpNTStm->lpVtbl->AddRef(lpNTStm);   // (see comment above)

         // truncate the current stream and seek to beginning
         lpNTStm->lpVtbl->SetSize(lpNTStm, ulibZero);
         lpNTStm->lpVtbl->Seek(
               lpNTStm, libZero, STREAM_SEEK_SET, NULL);
      } else
   #endif  // OLE_VERSION
      {
         hrErr = lpDestStg->lpVtbl->CreateStream(
               lpDestStg,
               OLESTR("LineList"),
               STGM_WRITE | STGM_SHARE_EXCLUSIVE | STGM_CREATE,
               0,
               0,
               &lpLLStm
         );
         if (hrErr != NOERROR) {
            OleDbgAssertSz(hrErr==NOERROR,"Could not create LineList stream");
            OleDbgOutHResult("LineList CreateStream returned", hrErr);
            goto error;
         }

         hrErr = lpDestStg->lpVtbl->CreateStream(
               lpDestStg,
               OLESTR("NameTable"),
               STGM_WRITE | STGM_SHARE_EXCLUSIVE | STGM_CREATE,
               0,
               0,
               &lpNTStm
         );
         if (hrErr != NOERROR) {
            OleDbgAssertSz(hrErr==NOERROR,"Could not create NameTable stream");
            OleDbgOutHResult("NameTable CreateStream returned", hrErr);
            goto error;
         }
      }

      // this may take a while, put up hourglass cursor
      hPrevCursor = SetCursor(LoadCursor(NULL, IDC_WAIT));

      _fmemset((LPOUTLINEDOCHEADER)&docRecord,0,sizeof(OUTLINEDOCHEADER));
      GetClipboardFormatName(
            uFormat,
            szAnsiString,
            256
      );
      A2W(szAnsiString, docRecord.m_szFormatName, lstrlen(szAnsiString)+1);
      OutlineApp_GetAppVersionNo(lpOutlineApp, docRecord.m_narrAppVersionNo);

      docRecord.m_fShowHeading = lpOutlineDoc->m_heading.m_fShow;

   #if defined( OLE_SERVER )
      {
         // Store ServerDoc specific data
         LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOutlineDoc;

         docRecord.m_reserved1 = (DWORD)lpServerDoc->m_nNextRangeNo;
      }
   #endif
   #if defined( OLE_CNTR )
      {
         // Store ContainerDoc specific data
         LPCONTAINERDOC lpContainerDoc = (LPCONTAINERDOC)lpOutlineDoc;

         docRecord.m_reserved2 = (DWORD)lpContainerDoc->m_nNextObjNo;
      }
   #endif

      /* write OutlineDoc header record */

       //  Transform docRecord into docRecordOnDisk
       //  Compilers should handle aligment correctly
       OLESTRCPY(docRecordOnDisk.m_szFormatName, docRecord.m_szFormatName);
       docRecordOnDisk.m_narrAppVersionNo[0] = (short)
   docRecord.m_narrAppVersionNo[0];
       docRecordOnDisk.m_narrAppVersionNo[1] = (short)
   docRecord.m_narrAppVersionNo[1];
       docRecordOnDisk.m_fShowHeading = (USHORT) docRecord.m_fShowHeading;
       docRecordOnDisk.m_reserved1 = docRecord.m_reserved1;
       docRecordOnDisk.m_reserved2 = docRecord.m_reserved2;
       docRecordOnDisk.m_reserved3 = docRecord.m_reserved3;
       docRecordOnDisk.m_reserved4 = docRecord.m_reserved4;
      hrErr = lpLLStm->lpVtbl->Write(
            lpLLStm,
            (LPVOID)&docRecordOnDisk,
            sizeof(docRecordOnDisk),
            &nWritten
         );

      if (hrErr != NOERROR) {
         OleDbgOutHResult("Write OutlineDoc header returned", hrErr);
         goto error;
       }

      // Save LineList
      /* OLE2NOTE: A ContainerDoc keeps its storage open at all times. It is
      **    necessary to pass the current open storage (lpOleDoc->m_lpStg)
      **    to the LineList_SaveSelToStg method so that currently written data
      **    for any embeddings is also saved to the new destination
      **    storage. The data required by a contained object is both the
      **    ContainerLine information and the associated sub-storage that is
      **    written directly by the embedded object.
      */
      fStatus = LineList_SaveSelToStg(
         &lpOutlineDoc->m_LineList,
            lplrSel,
            uFormat,
   #if defined( OLE_CNTR )
            lpOleDoc->m_lpStg,
   #else
            NULL,
   #endif
            lpDestStg,
            lpLLStm,
            fRemember
      );
      if (! fStatus) goto error;

      // Save associated NameTable
      fStatus = OutlineNameTable_SaveSelToStg(
            lpOutlineDoc->m_lpNameTable,
            lplrSel,
            uFormat,
            lpNTStm
      );

      if (! fStatus) goto error;

      OleStdRelease((LPUNKNOWN)lpLLStm);
      lpOutlineDoc->m_cfSaveFormat = uFormat;  // remember format used to save

      SetCursor(hPrevCursor);     // restore original cursor
      return TRUE;

   error:
      if (lpLLStm)
         OleStdRelease((LPUNKNOWN)lpLLStm);

      SetCursor(hPrevCursor);     // restore original cursor
      return FALSE;
   }


   /* OutlineDoc_Print
    * ----------------
    *  Prints the contents of the list box in HIMETRIC mapping mode. Origin
    *  remains to be the upper left corner and the print proceeds down the
    *  page using a negative y-cordinate.
    *
    */
   void OutlineDoc_Print(LPOUTLINEDOC lpOutlineDoc, HDC hDC)
   {
      LPLINELIST lpLL = &lpOutlineDoc->m_LineList;
      LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
      WORD    nIndex;
      WORD    nTotal;
      int     dy;
      BOOL    fError = FALSE;
      LPLINE  lpLine;
      RECT    rcLine;
      RECT    rcPix;
      RECT    rcHim;
      RECT    rcWindowOld;
      RECT    rcViewportOld;
```

```
HFONT   hOldFont;
DOCINFO di;         /* Document information for StartDoc function */
char    szAnsiStr[256];

/* Get dimension of page */
rcPix.left = 0;
rcPix.top = 0;
rcPix.right = GetDeviceCaps(hDC, HORZRES);
rcPix.bottom = GetDeviceCaps(hDC, VERTRES);

SetDCToDrawInHimetricRect(hDC, (LPRECT)&rcPix, (LPRECT)&rcHim,
      (LPRECT)&rcWindowOld, (LPRECT)&rcViewportOld);

// Set the default font size, and font face name
hOldFont = SelectObject(hDC, lpOutlineApp->m_hStdFont);

/* Get the lines in document */
nIndex     = 0;
nTotal  = LineList_GetCount(lpLL);

/* Create the Cancel dialog */
// REVIEW: should load dialog title from string resource file
hWndPDlg = CreateDialog (
      lpOutlineApp->m_hInst,
      "Print",
      lpOutlineApp->m_hWndApp,
      (DLGPROC)PrintDlgProc
);

if(!hWndPDlg)
   goto getout;

/* Allow the app. to inform GDI of the abort function to call */
if(SetAbortProc(hDC, (ABORTPROC)AbortProc) < 0) {
   fError = TRUE;
   goto getout3;
}

/* Disable the main application window */
EnableWindow (lpOutlineApp->m_hWndApp, FALSE);

// initialize the rectangle for the first line
rcLine.left = rcHim.left;
rcLine.bottom = rcHim.top;

/* Initialize the document */
fCancelPrint = FALSE;

di.cbSize = sizeof(DOCINFO);
W2A (lpOutlineDoc->m_lpszDocTitle, szAnsiStr, 256);
di.lpszDocName = szAnsiStr;
di.lpszOutput = NULL;

if(StartDoc(hDC, (DOCINFO FAR*)&di) <= 0) {
   fError = TRUE;
```

```
         OleDbgOut2("StartDoc error\n");
         goto getout5;
      }

      if(StartPage(hDC) <= 0) {       // start first page
         fError = TRUE;
         OleDbgOut2("StartPage error\n");
         goto getout2;
      }

      /* While more lines print out the text */
      while(nIndex < nTotal) {
         lpLine = LineList_GetLine(lpLL, nIndex);
         if (! lpLine)
            continue;

         dy = Line_GetHeightInHimetric(lpLine);

         /* Reached end of page. Tell the device driver to eject a page */
         if(rcLine.bottom - dy < rcHim.bottom) {
            if (EndPage(hDC) < 0) {
               fError=TRUE;
               OleDbgOut2("EndPage error\n");
               goto getout2;
            }

            // NOTE: Reset the Mapping mode of DC
            SetDCToDrawInHimetricRect(hDC, (LPRECT)&rcPix, (LPRECT)&rcHim,
                  (LPRECT)&rcWindowOld, (LPRECT)&rcViewportOld);

            // Set the default font size, and font face name
            SelectObject(hDC, lpOutlineApp->m_hStdFont);

            if (StartPage(hDC) <= 0) {
               fError=TRUE;
               OleDbgOut2("StartPage error\n");
               goto getout2;
            }

            rcLine.bottom = rcHim.top;
         }

         rcLine.top = rcLine.bottom;
         rcLine.bottom -= dy;
         rcLine.right = rcLine.left + Line_GetWidthInHimetric(lpLine);

         /* Print the line */
         Line_Draw(lpLine, hDC, &rcLine, NULL, FALSE /*fHighlight*/);

         OleDbgOut2("a line is drawn\n");

         /* Test and see if the Abort flag has been set. If yes, exit. */
         if (fCancelPrint)
            goto getout2;
         /* Move down the page */
         nIndex++;
      }

      {
         int nCode;

         /* Eject the last page. */
         if((nCode = EndPage(hDC)) < 0) {
   #if defined( _DEBUG )
            char szBuf[255];
            wsprintf(szBuf, "EndPage error code is %d\n", nCode);
            OleDbgOut2(szBuf);
   #endif
            fError=TRUE;
            goto getout2;
         }
      }


      /* Complete the document. */
      if(EndDoc(hDC) < 0) {
         fError=TRUE;
         OleDbgOut2("EndDoc error\n");

   getout2:
         /* Ran into a problem before NEWFRAME? Abort the document */
         AbortDoc(hDC);
      }

   getout5:
      /* Re-enable main app. window */
      EnableWindow (lpOutlineApp->m_hWndApp, TRUE);

   getout3:
      /* Close the cancel dialog */
      DestroyWindow (hWndPDlg);

   getout:

      /* Error? make sure the user knows... */
      if(fError || CommDlgExtendedError())
         OutlineApp_ErrorMessage(lpOutlineApp, ErrMsgPrint);

      SelectObject(hDC, hOldFont);
   }





   /* OutlineDoc_DialogHelp
    * ---------------------
    *
    *  Show help message for ole2ui dialogs.
    *
    * Parameters:
    *
    *   hDlg      HWND to the dialog the help message came from - use
    *             this in the call to WinHelp/MessageBox so that
    *             activation/focus goes back to the dialog, and not the
    *             main window.
    *
    *   wParam    ID of the dialog (so we know what type of dialog it is).
    */
   void OutlineDoc_DialogHelp(HWND hDlg, WPARAM wDlgID)
   {

      char szMessageBoxText[64];

      if (!IsWindow(hDlg))  // don't do anything if we've got a bogus hDlg.
       return;

      lstrcpy(szMessageBoxText, "Help Message for ");

      switch (wDlgID)
      {

      case IDD_CONVERT:
         lstrcat(szMessageBoxText, "Convert");
         break;

      case IDD_CHANGEICON:
         lstrcat(szMessageBoxText, "Change Icon");
         break;

      case IDD_INSERTOBJECT:
         lstrcat(szMessageBoxText, "Insert Object");
         break;

      case IDD_PASTESPECIAL:
         lstrcat(szMessageBoxText, "Paste Special");
         break;

      case IDD_EDITLINKS:
         lstrcat(szMessageBoxText, "Edit Links");
         break;

      case IDD_CHANGESOURCE:
         lstrcat(szMessageBoxText, "Change Source");
         break;

      case IDD_INSERTFILEBROWSE:
         lstrcat(szMessageBoxText, "Insert From File Browse");
         break;

      case IDD_CHANGEICONBROWSE:
         lstrcat(szMessageBoxText, "Change Icon Browse");
         break;
      default:
         lstrcat(szMessageBoxText, "Unknown");
         break;
      }

      lstrcat(szMessageBoxText, " Dialog.");

      // You'd probably really a call to WinHelp here.
      MessageBox(hDlg, szMessageBoxText, "Help", MB_OK);

      return;
   }


   /* OutlineDoc_SetCurrentZoomCommand
    * --------------------------------
    *
    *  Set current zoom level to be checked in the menu.
    *  Set the corresponding scalefactor for the document.
    */
   void OutlineDoc_SetCurrentZoomCommand(
         LPOUTLINEDOC        lpOutlineDoc,
         UINT                uCurrentZoom
   )
   {
      SCALEFACTOR scale;

      if (!lpOutlineDoc)
         return;

      lpOutlineDoc->m_uCurrentZoom = uCurrentZoom;

      switch (uCurrentZoom) {

   #if !defined( OLE_CNTR )
            case IDM_V_ZOOM_400:
               scale.dwSxN = (DWORD) 4;
               scale.dwSxD = (DWORD) 1;
               scale.dwSyN = (DWORD) 4;
               scale.dwSyD = (DWORD) 1;
               break;

            case IDM_V_ZOOM_300:
               scale.dwSxN = (DWORD) 3;
               scale.dwSxD = (DWORD) 1;
               scale.dwSyN = (DWORD) 3;
               scale.dwSyD = (DWORD) 1;
               break;

            case IDM_V_ZOOM_200:
               scale.dwSxN = (DWORD) 2;
               scale.dwSxD = (DWORD) 1;
               scale.dwSyN = (DWORD) 2;
               scale.dwSyD = (DWORD) 1;
               break;
```

```
#endif      // !OLE_CNTR

         case IDM_V_ZOOM_100:
            scale.dwSxN = (DWORD) 1;
            scale.dwSxD = (DWORD) 1;
            scale.dwSyN = (DWORD) 1;
            scale.dwSyD = (DWORD) 1;
            break;

         case IDM_V_ZOOM_75:
            scale.dwSxN = (DWORD) 3;
            scale.dwSxD = (DWORD) 4;
            scale.dwSyN = (DWORD) 3;
            scale.dwSyD = (DWORD) 4;
            break;

         case IDM_V_ZOOM_50:
            scale.dwSxN = (DWORD) 1;
            scale.dwSxD = (DWORD) 2;
            scale.dwSyN = (DWORD) 1;
            scale.dwSyD = (DWORD) 2;
            break;

         case IDM_V_ZOOM_25:
            scale.dwSxN = (DWORD) 1;
            scale.dwSxD = (DWORD) 4;
            scale.dwSyN = (DWORD) 1;
            scale.dwSyD = (DWORD) 4;
            break;
   }

   OutlineDoc_SetScaleFactor(lpOutlineDoc, (LPSCALEFACTOR)&scale, NULL);
}


/* OutlineDoc_GetCurrentZoomMenuCheck
 * ----------------------------------
 *
 *  Get current zoom level to be checked in the menu.
 */
UINT OutlineDoc_GetCurrentZoomMenuCheck(LPOUTLINEDOC lpOutlineDoc)
{
   return lpOutlineDoc->m_uCurrentZoom;
}


/* OutlineDoc_SetScaleFactor
 * -------------------------
 *
 *  Set the scale factor of the document which will affect the
 *      size of the document on the screen
 *
 * Parameters:
 *
 *   scale      structure containing x and y scales
 */
void OutlineDoc_SetScaleFactor(
      LPOUTLINEDOC        lpOutlineDoc,
      LPSCALEFACTOR       lpscale,
      LPRECT              lprcDoc
)
{
   LPLINELIST      lpLL = OutlineDoc_GetLineList(lpOutlineDoc);
   HWND            hWndLL = LineList_GetWindow(lpLL);

   if (!lpOutlineDoc || !lpscale)
      return;

   InvalidateRect(hWndLL, NULL, TRUE);

   lpOutlineDoc->m_scale = *lpscale;
   LineList_ReScale((LPLINELIST)&lpOutlineDoc->m_LineList, lpscale);

#if defined( USE_HEADING )
   Heading_ReScale((LPHEADING)&lpOutlineDoc->m_heading, lpscale);
#endif

   OutlineDoc_Resize(lpOutlineDoc, lprcDoc);
}


/* OutlineDoc_GetScaleFactor
 * -------------------------
 *
 *  Retrieve the scale factor of the document
 *
 * Parameters:
 *
 */
LPSCALEFACTOR OutlineDoc_GetScaleFactor(LPOUTLINEDOC lpOutlineDoc)
{
   if (!lpOutlineDoc)
      return NULL;

   return (LPSCALEFACTOR)&lpOutlineDoc->m_scale;
}


/* OutlineDoc_SetCurrentMarginCommand
 * ----------------------------------
 *
 *  Set current Margin level to be checked in the menu.
 */
void OutlineDoc_SetCurrentMarginCommand(
      LPOUTLINEDOC        lpOutlineDoc,
      UINT                uCurrentMargin
)
{
   if (!lpOutlineDoc)
      return;
   lpOutlineDoc->m_uCurrentMargin = uCurrentMargin;

   switch (uCurrentMargin) {
      case IDM_V_SETMARGIN_0:
         OutlineDoc_SetMargin(lpOutlineDoc, 0, 0);
         break;

      case IDM_V_SETMARGIN_1:
         OutlineDoc_SetMargin(lpOutlineDoc, 1000, 1000);
         break;

      case IDM_V_SETMARGIN_2:
         OutlineDoc_SetMargin(lpOutlineDoc, 2000, 2000);
         break;

      case IDM_V_SETMARGIN_3:
         OutlineDoc_SetMargin(lpOutlineDoc, 3000, 3000);
         break;

      case IDM_V_SETMARGIN_4:
         OutlineDoc_SetMargin(lpOutlineDoc, 4000, 4000);
         break;
   }
}


/* OutlineDoc_GetCurrentMarginMenuCheck
 * ------------------------------------
 *
 *  Get current Margin level to be checked in the menu.
 */
UINT OutlineDoc_GetCurrentMarginMenuCheck(LPOUTLINEDOC lpOutlineDoc)
{
   return lpOutlineDoc->m_uCurrentMargin;
}


/* OutlineDoc_SetMargin
 * --------------------
 *
 *  Set the left and right margin of the document
 *
 * Parameters:
 *      nLeftMargin  - left margin in Himetric values
 *      nRightMargin - right margin in Himetric values
 */
void OutlineDoc_SetMargin(LPOUTLINEDOC lpOutlineDoc, int nLeftMargin, int
nRightMargin)
{
   LPLINELIST lpLL;
   int        nMaxWidthInHim;

   if (!lpOutlineDoc)
      return;
   lpOutlineDoc->m_nLeftMargin = nLeftMargin;
   lpOutlineDoc->m_nRightMargin = nRightMargin;
   lpLL = OutlineDoc_GetLineList(lpOutlineDoc);

   // Force recalculation of Horizontal extent
   nMaxWidthInHim = LineList_GetMaxLineWidthInHimetric(lpLL);
   LineList_SetMaxLineWidthInHimetric(lpLL, -nMaxWidthInHim);

#if defined( INPLACE_CNTR )
   ContainerDoc_UpdateInPlaceObjectRects((LPCONTAINERDOC)lpOutlineDoc, 0);
#endif

   OutlineDoc_ForceRedraw(lpOutlineDoc, TRUE);
}


/* OutlineDoc_GetMargin
 * --------------------
 *
 *  Get the left and right margin of the document
 *
 *  Parameters:
 *      nLeftMargin  - left margin in Himetric values
 *      nRightMargin - right margin in Himetric values
 *
 *  Returns:
 *      low order word  - left margin
 *      high order word - right margin
 */
LONG OutlineDoc_GetMargin(LPOUTLINEDOC lpOutlineDoc)
{
   if (!lpOutlineDoc)
      return 0;

   return MAKELONG(lpOutlineDoc->m_nLeftMargin, lpOutlineDoc-
>m_nRightMargin);
}

#if defined( USE_HEADING )

/* OutlineDoc_GetHeading
 * ---------------------
 *
 *      Get Heading Object in OutlineDoc
 */
LPHEADING OutlineDoc_GetHeading(LPOUTLINEDOC lpOutlineDoc)
{
   if (!lpOutlineDoc || lpOutlineDoc->m_fDataTransferDoc)
      return NULL;
   else
      return (LPHEADING)&lpOutlineDoc->m_heading;
}
```

```
/* OutlineDoc_ShowHeading
 * ----------------------
 *
 *  Show/Hide document row/column headings.
 */
void OutlineDoc_ShowHeading(LPOUTLINEDOC lpOutlineDoc, BOOL fShow)
{
   LPHEADING   lphead = OutlineDoc_GetHeading(lpOutlineDoc);
#if defined( INPLACE_SVR )
   LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpOutlineDoc;
#endif

   if (! lphead)
      return;

   Heading_Show(lphead, fShow);

#if defined( INPLACE_SVR )
   if (lpServerDoc->m_fUIActive) {
      LPINPLACEDATA lpIPData = lpServerDoc->m_lpIPData;

      /* OLE2NOTE: our extents have NOT changed; only our the size of
      **    our object-frame adornments is changing. we can use the
      **    current PosRect and ClipRect and simply resize our
      **    windows WITHOUT informing our in-place container.
      */
      ServerDoc_ResizeInPlaceWindow(
            lpServerDoc,
            (LPRECT)&(lpIPData->rcPosRect),
            (LPRECT)&(lpIPData->rcClipRect)
      );
   } else
#else   // !INPLACE_SVR

   OutlineDoc_Resize(lpOutlineDoc, NULL);

#if defined( INPLACE_CNTR )
   ContainerDoc_UpdateInPlaceObjectRects((LPCONTAINERDOC)lpOutlineDoc, 0);
#endif  // INPLACE_CNTR

#endif  // INPLACE_SVR

   OutlineDoc_ForceRedraw(lpOutlineDoc, TRUE);
}

#endif  // USE_HEADING


/* AbortProc
 * ---------
 *  AborProc is called by GDI print code to check for user abort.
 */
BOOL FAR PASCAL EXPORT AbortProc (HDC hdc, WORD reserved)
{
   MSG msg;
   /* Allow other apps to run, or get abort messages */
   while(! fCancelPrint && PeekMessage (&msg, NULL, 0, 0, PM_REMOVE)) {
      if(!hWndPDlg || !IsDialogMessage (hWndPDlg, &msg)) {
         TranslateMessage (&msg);
         DispatchMessage  (&msg);
      }
   }
   return !fCancelPrint;
}


/* PrintDlgProc
 * ------------
 *  Dialog function for the print cancel dialog box.
 *
 *  RETURNS    : TRUE  - OK to abort/ not OK to abort
 *               FALSE - otherwise.
 */
BOOL FAR PASCAL EXPORT PrintDlgProc(
      HWND hwnd,
      WORD msg,
      WORD wParam,
      LONG lParam
)
{
   switch (msg) {
      case WM_COMMAND:
      /* abort printing if the only button gets hit */
         fCancelPrint = TRUE;
         return TRUE;
   }

   return FALSE;
}
```

OUTLINE.DEF   (OUTLINE Sample)

```
;;
;;
;;    OLE 2.0 Sample Code
;;
;;    outline.def
;;
;;    Definition file for outline.exe
;;
;;    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
;;
;;

NAME            Outline
DESCRIPTION     'Microsoft OLE 2.0 Sample code--base version'
EXETYPE         WINDOWS
STUB            'WINSTUB.EXE'
CODE            PRELOAD MOVEABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        8192

;; NOTE: Do not add exports to this file.  Use __export
;; in your function prototype and definition instead.
```

OUTLINE.H   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    outline.h
**
**    This file contains file contains data structure defintions,
**    function prototypes, constants, etc. used by the Outline series
**    of sample applications:
**          Outline  -- base version of the app (without OLE functionality)
**          SvrOutl  -- OLE 2.0 Server sample app
**          CntrOutl -- OLE 2.0 Containter (Container) sample app
**          ISvrOtl  -- OLE 2.0 Server sample app
**          CntrOutl -- OLE 2.0 Containter (Container) sample app
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
**    For structures which we read from and write to disk we define shadow
**    structures (with the _ONDISK suffix) that allow us to maintain
**    16-bit Windows and Macintosh compatibility.
**
*************************************************************************/

#if !defined( _OUTLINE_H_ )
#define _OUTLINE_H_

#if !defined( RC_INVOKED )
#pragma message ("INCLUDING OUTLINE.H from " __FILE__)
#endif  /* RC_INVOKED */

// use strict ANSI standard (for DVOBJ.H)
#define NONAMELESSUNION

// use system defined bitmap, this line must go before windows.h
#define OEMRESOURCE

#ifdef WIN32
#define EXPORT

#define _fstrchr strchr

#else
#define EXPORT _export
#endif

#define SDI_VERSION         1   // ONLY SDI version is currently supported

#if defined( OLE_SERVER ) || defined( OLE_CNTR )
#define OLE_VERSION         1
#define USE_DRAGDROP        1   // enable drag/drop code in OLE versions
#define USE_MSGFILTER       1   // enable IMessageFilter implementation
#endif
```

```
#define USE_HEADING          1   // enable the row/col headings
#define USE_STATUSBAR        1   // enable status bar window
#define USE_FRAMETOOLS       1   // enable the toolbar
#ifndef WIN32   //BUGBUG32
#define USE_CTL3D            1   // enable 3D looking dialogs
#endif

#define STRICT  1
#include <windows.h>
#include <string.h>
#include <commdlg.h>
#include <ole2.h>
#include <ole2ui.h>
#include "outlrc.h"


#define APPMAJORVERSIONNO    3   // major no. incremented for major releases
                        //  (eg. when an incompatible change is made
                        //  to the storage format)
#define APPMINORVERSIONNO    5   // minor no. incremented for minor releases


/* Definition of SCALEFACTOR */
typedef struct tagSCALEFACTOR {
   ULONG       dwSxN;      // numerator in x direction
   ULONG       dwSxD;      // denominator in x direction
   ULONG       dwSyN;      // numerator in y direction
   ULONG       dwSyD;      // denominator in y direction
} SCALEFACTOR, FAR* LPSCALEFACTOR;


#if defined( USE_FRAMETOOLS )
#include "frametls.h"
#endif

#if defined( USE_HEADING )
#include "heading.h"
#endif

/* max line height (in pixels) allowed in a listbox */
#define LISTBOX_HEIGHT_LIMIT    255


#define MAXSTRLEN   80      // max string len in bytes
#define MAXNAMESIZE 30      // max length of names
#define MAXFORMATSIZE   10  // max length of DEFDOCFORMAT (actually size is
5)
#define TABWIDTH        2000 // 2000 in Himetric units, i.e. 2cm
#define DEFFONTPTSIZE   12
#define DEFFONTSIZE     ((DEFFONTPTSIZE*HIMETRIC_PER_INCH)/PTS_PER_INCH)
#define DEFFONTFACE     "Times New Roman"

#ifdef WIN32S
```

```
#define OUTLINEDOCFORMAT    OLESTR("Outline32")     // CF_Outline format
name
#else
#define OUTLINEDOCFORMAT    OLESTR("Outline")       // CF_Outline format
name
#endif

#define IS_FILENAME_DELIM(c)    ( (c) == '\\' || (c) == '/' || (c) == ':' )
// REVIEW: some of these strings should be loaded from a resource file
#define UNTITLED    OLESTR("Outline")   // title used for untitled document
#define HITTESTDELTA    5

/* Macro to get a random integer within a specified range */
#define getrandom( min, max ) ((rand() % (int)(((max)+1) - (min))) + (min))


// REVIEW: should load strings from string resource file
#ifdef TEST32

#define APPFILENAMEFILTER   "Outline Files (*.OLT)|*.olt|All files (*.*)|
*.*|"
#define DEFEXTENSION    "olt"           // Default file extension

#else

#define APPFILENAMEFILTER   "Outline Files (*.OLN)|*.oln|All files (*.*)|
*.*|"
#define DEFEXTENSION    "oln"           // Default file extension

#endif //TEST32

/* forward type references */
typedef struct tagOUTLINEDOC FAR* LPOUTLINEDOC;
typedef struct tagTEXTLINE FAR* LPTEXTLINE;


typedef enum tagLINETYPE {
   UNKNOWNLINETYPE,
   TEXTLINETYPE,
   CONTAINERLINETYPE
} LINETYPE;


/*************************************************************************
** class LINE
**    The class LINE is an abstract base class. Instances of class LINE
**    are NOT created; only instances of the concrete subclasses of
**    LINE can be created. In the base app version and the OLE 2.0
**    server-only version only TEXTLINE objects can be created. In the
**    OLE 2.0 client app version either TEXTLINE objects or CONTAINERLINE
**    objects can be created. The LINE class has all fields and methods
**    that are common independent of which subclass of LINE is used.
**    Each LINE object that is created in added to the LINELIST of the
**    OUTLINEDOC document.
*************************************************************************/
```

```
typedef struct tagLINE {
   LINETYPE    m_lineType;
   UINT        m_nTabLevel;
   UINT        m_nTabWidthInHimetric;
   UINT        m_nWidthInHimetric;
   UINT        m_nHeightInHimetric;
   BOOL        m_fSelected;        // does line have selection feedback

#if defined( USE_DRAGDROP )
   BOOL        m_fDragOverLine;    // does line have drop target feedback
#endif
} LINE, FAR* LPLINE;

/* Line methods (functions) */
void Line_Init(LPLINE lpLine, int nTab, HDC hDC);
void Line_Delete(LPLINE lpLine);
BOOL Line_CopyToDoc(LPLINE lpSrcLine, LPOUTLINEDOC lpDestDoc, int nIndex);
BOOL Line_Edit(LPLINE lpLine, HWND hWndDoc, HDC hDC);
void Line_Draw(
      LPLINE      lpLine,
      HDC         hDC,
      LPRECT      lpRect,
      LPRECT      lpRectWBounds,
      BOOL        fHighlight
);
void Line_DrawToScreen(
      LPLINE      lpLine,
      HDC         hDC,
      LPRECT      lprcPix,
      UINT        itemAction,
      UINT        itemState,
      LPRECT      lprcDevice
);
void Line_DrawSelHilight(LPLINE lpLine, HDC hDC, LPRECT lpRect, UINT
itemAction, UINT itemState);
void Line_DrawFocusRect(LPLINE lpLine, HDC hDC, LPRECT lpRect, UINT
itemAction, UINT itemState);
void Line_Unindent(LPLINE lpLine, HDC hDC);
void Line_Indent(LPLINE lpLine, HDC hDC);
LINETYPE Line_GetLineType(LPLINE lpLine);
UINT Line_GetTotalWidthInHimetric(LPLINE lpLine);
void Line_SetWidthInHimetric(LPLINE lpLine, int nWidth);
UINT Line_GetWidthInHimetric(LPLINE lpLine);
UINT Line_GetHeightInHimetric(LPLINE lpLine);
void Line_SetHeightInHimetric(LPLINE lpLine, int nHeight);
UINT Line_GetTabLevel(LPLINE lpLine);
int Line_GetTextLen(LPLINE lpLine);
void Line_GetTextData(LPLINE lpLine, LPOLESTR lpszBuf);
BOOL Line_GetOutlineData(LPLINE lpLine, LPTEXTLINE lpBuf);
int Line_CalcTabWidthInHimetric(LPLINE lpLine, HDC hDC);
BOOL Line_SaveToStg(LPLINE lpLine, UINT uFormat, LPSTORAGE lpSrcStg,
LPSTORAGE lpDestStg, LPSTREAM lpLLStm, BOOL fRemember);
LPLINE Line_LoadFromStg(LPSTORAGE lpSrcStg, LPSTREAM lpLLStm, LPOUTLINEDOC
lpDestDoc);
```

```
void Line_DrawDragFeedback(LPLINE lpLine, HDC hDC, LPRECT lpRect, UINT
itemState );
BOOL Line_IsSelected(LPLINE lpLine);


/*************************************************************************
** class TEXTLINE : LINE
**    The class TEXTLINE is a concrete subclass of the abstract base
**    class LINE. The TEXTLINE class holds a string that can be edited
**    by the user. In the base app version and the OLE 2.0
**    server-only version only TEXTLINE objects can be created. In the
**    OLE 2.0 client app version either TEXTLINE objects or CONTAINERLINE
**    objects can be created. The TEXTLINE class inherits all fields
**    from the LINE class. This inheritance is achieved by including a
**    member variable of type LINE as the first field in the TEXTLINE
**    structure. Thus a pointer to a TEXTLINE object can be cast to be
**    a pointer to a LINE object.
**    Each TEXTLINE object that is created in added to the LINELIST of
**    the associated OUTLINEDOC document.
*************************************************************************/

typedef struct tagTEXTLINE {
   LINE m_Line;        // TextLine inherits all fields of Line

   UINT m_nLength;
   char m_szText[MAXSTRLEN+1];
} TEXTLINE;

LPTEXTLINE TextLine_Create(HDC hDC, UINT nTab, LPSTR szText);
void TextLine_Init(LPTEXTLINE lpTextLine, int nTab, HDC hDC);
void TextLine_CalcExtents(LPTEXTLINE lpLine, HDC hDC);
void TextLine_SetHeightInHimetric(LPTEXTLINE lpTextLine, int nHeight);
void TextLine_Delete(LPTEXTLINE lpLine);
BOOL TextLine_Edit(LPTEXTLINE lpLine, HWND hWndDoc, HDC hDC);
void TextLine_Draw(
      LPTEXTLINE  lpTextLine,
      HDC         hDC,
      LPRECT      lpRect,
      LPRECT      lpRectWBounds,
      BOOL        fHighlight
);
void TextLine_DrawSelHilight(LPTEXTLINE lpTextLine, HDC hDC, LPRECT lpRect,
UINT itemAction, UINT itemState);
BOOL TextLine_Copy(LPTEXTLINE lpSrcLine, LPTEXTLINE lpDestLine);
BOOL TextLine_CopyToDoc(LPTEXTLINE lpSrcLine, LPOUTLINEDOC lpDestDoc, int
nIndex);
int TextLine_GetTextLen(LPTEXTLINE lpTextLine);
void TextLine_GetTextData(LPTEXTLINE lpTextLine, LPSTR lpszBuf);
BOOL TextLine_GetOutlineData(LPTEXTLINE lpTextLine, LPTEXTLINE lpBuf);
BOOL TextLine_SaveToStm(LPTEXTLINE lpLine, LPSTREAM lpLLStm);
LPLINE TextLine_LoadFromStg(LPSTORAGE lpSrcStg, LPSTREAM lpLLStm,
LPOUTLINEDOC lpDestDoc);
```

```
/**************************************************************************
** class LINERANGE
**    The class LINERANGE is a supporting object used to describe a
**    particular range in an OUTLINEDOC. A range is defined by a starting
**    line index and an ending line index.
**************************************************************************/

typedef struct tagLINERANGE {
   signed short    m_nStartLine;
   signed short    m_nEndLine;
} LINERANGE, FAR* LPLINERANGE;


/**************************************************************************
** class OUTLINENAME
**    The class OUTLINENAME stores a particular named selection in the
**    OUTLINEDOC document. The NAMETABLE class holds all of the names
**    defined in a particular OUTLINEDOC document. Each OUTLINENAME
**    object has a string as its key and a starting line index and an
**    ending line index for the named range.
**************************************************************************/

typedef struct tagOUTLINENAME {
   OLECHAR         m_szName[MAXNAMESIZE+1];
   signed short    m_nStartLine;  // must be signed for table update
   signed short    m_nEndLine;    // functions to work
} OUTLINENAME, FAR* LPOUTLINENAME;

void OutlineName_SetName(LPOUTLINENAME lpOutlineName, LPOLESTR lpszName);
void OutlineName_SetSel(LPOUTLINENAME lpOutlineName, LPLINERANGE lplrSel,
BOOL fRangeModified);
void OutlineName_GetSel(LPOUTLINENAME lpOutlineName, LPLINERANGE lplrSel);
BOOL OutlineName_SaveToStg(LPOUTLINENAME lpOutlineName, LPLINERANGE lplrSel,
UINT uFormat, LPSTREAM lpNTStm, BOOL FAR* lpfNameSaved);

BOOL OutlineName_SaveToStg(LPOUTLINENAME lpOutlineName, LPLINERANGE lplrSel,
UINT uFormat, LPSTREAM lpNTStm, BOOL FAR* lpfNameSaved);
BOOL OutlineName_LoadFromStg(LPOUTLINENAME lpOutlineName, LPSTREAM lpNTStm);


/**************************************************************************
** class OUTLINENAMETABLE
**    OUTLINENAMETABLE manages the table of named selections in the
**    OUTLINEDOC document. Each OUTLINENAMETABLE entry has a string as its
key
**    and a starting line index and an ending line index for the
**    named range. There is always one instance of OUTLINENAMETABLE for each
**    OUTLINEDOC created.
**************************************************************************/

typedef struct tagOUTLINENAMETABLE {
   HWND        m_hWndListBox;
   int         m_nCount;
} OUTLINENAMETABLE, FAR* LPOUTLINENAMETABLE;
```

```
/* OutlineNameTable methods (functions) */
BOOL OutlineNameTable_Init(LPOUTLINENAMETABLE lpOutlineNameTable,
LPOUTLINEDOC lpOutlineDoc);
void OutlineNameTable_Destroy(LPOUTLINENAMETABLE lpOutlineNameTable);
void OutlineNameTable_ClearAll(LPOUTLINENAMETABLE lpOutlineNameTable);
LPOUTLINENAME OutlineNameTable_CreateName(LPOUTLINENAMETABLE
lpOutlineNameTable);
void OutlineNameTable_AddName(LPOUTLINENAMETABLE lpOutlineNameTable,
LPOUTLINENAME lpOutlineName);
void OutlineNameTable_DeleteName(LPOUTLINENAMETABLE lpOutlineNameTable, int
nIndex);
int OutlineNameTable_GetNameIndex(LPOUTLINENAMETABLE lpOutlineNameTable,
LPOUTLINENAME lpOutlineName);
LPOUTLINENAME OutlineNameTable_GetName(LPOUTLINENAMETABLE
lpOutlineNameTable, int nIndex);
LPOUTLINENAME OutlineNameTable_FindName(LPOUTLINENAMETABLE
lpOutlineNameTable, LPOLESTR lpszName);
LPOUTLINENAME OutlineNameTable_FindNamedRange(LPOUTLINENAMETABLE
lpOutlineNameTable, LPLINERANGE lplrSel);
int OutlineNameTable_GetCount(LPOUTLINENAMETABLE lpOutlineNameTable);
void OutlineNameTable_AddLineUpdate(LPOUTLINENAMETABLE lpOutlineNameTable,
int nAddIndex);
void OutlineNameTable_DeleteLineUpdate(LPOUTLINENAMETABLE
lpOutlineNameTable, int nDeleteIndex);
BOOL OutlineNameTable_LoadFromStg(LPOUTLINENAMETABLE lpOutlineNameTable,
LPSTORAGE lpSrcStg);
BOOL OutlineNameTable_SaveSelToStg(
      LPOUTLINENAMETABLE      lpOutlineNameTable,
      LPLINERANGE             lplrSel,
      UINT                    uFormat,
      LPSTREAM                lpNTStm
);


/*************************************************************************
** class LINELIST
**    The class LINELIST manages the list of Line objects in the
**    OUTLINEDOC document. This class uses a Window's Owner-draw ListBox
**    to hold the list of LINE objects. There is always one instance of
**    LINELIST for each OUTLINEDOC created.
*************************************************************************/

typedef struct tagLINELIST {
   HWND            m_hWndListBox;  // hWnd of OwnerDraw listbox
   int             m_nNumLines;        // number of lines in LineList
   int             m_nMaxLineWidthInHimetric;  // max width of listbox
   LPOUTLINEDOC    m_lpDoc;        // ptr to associated OutlineDoc
   LINERANGE       m_lrSaveSel;    // selection saved on WM_KILLFOCUS

#if defined( USE_DRAGDROP )
   int             m_iDragOverLine;    // line index w/ drop target feedback
#endif
} LINELIST, FAR* LPLINELIST;

/* LineList methods (functions) */
```

```
BOOL LineList_Init(LPLINELIST lpLL, LPOUTLINEDOC lpOutlineDoc);
void LineList_Destroy(LPLINELIST lpLL);
void LineList_AddLine(LPLINELIST lpLL, LPLINE lpLine, int nIndex);
void LineList_DeleteLine(LPLINELIST lpLL, int nIndex);
void LineList_ReplaceLine(LPLINELIST lpLL, LPLINE lpLine, int nIndex);
int LineList_GetLineIndex(LPLINELIST lpLL, LPLINE lpLine);
LPLINE LineList_GetLine(LPLINELIST lpLL, int nIndex);
void LineList_SetFocusLine ( LPLINELIST lpLL, WORD wIndex );
BOOL LineList_GetLineRect(LPLINELIST lpLL, int nIndex, LPRECT lpRect);
int LineList_GetFocusLineIndex(LPLINELIST lpLL);
int LineList_GetCount(LPLINELIST lpLL);
BOOL LineList_SetMaxLineWidthInHimetric(
      LPLINELIST lpLL,
      int nWidthInHimetric
);
void LineList_ScrollLineIntoView(LPLINELIST lpLL, int nIndex);
int LineList_GetMaxLineWidthInHimetric(LPLINELIST lpLL);
BOOL LineList_RecalcMaxLineWidthInHimetric(
      LPLINELIST          lpLL,
      int                 nWidthInHimetric
);
void LineList_CalcSelExtentInHimetric(
      LPLINELIST          lpLL,
      LPLINERANGE         lplrSel,
      LPSIZEL             lpsizel
);
HWND LineList_GetWindow(LPLINELIST lpLL);
HDC LineList_GetDC(LPLINELIST lpLL);
void LineList_ReleaseDC(LPLINELIST lpLL, HDC hDC);
void LineList_SetLineHeight(LPLINELIST lpLL,int nIndex,int
nHeightInHimetric);
void LineList_ReScale(LPLINELIST lpLL, LPSCALEFACTOR lpscale);
void LineList_SetSel(LPLINELIST lpLL, LPLINERANGE lplrSel);
int LineList_GetSel(LPLINELIST lpLL, LPLINERANGE lplrSel);
void LineList_RemoveSel(LPLINELIST lpLL);
void LineList_RestoreSel(LPLINELIST lpLL);
void LineList_SetRedraw(LPLINELIST lpLL, BOOL fEnableDraw);
void LineList_ForceRedraw(LPLINELIST lpLL, BOOL fErase);
void LineList_ForceLineRedraw(LPLINELIST lpLL, int nIndex, BOOL fErase);
int LineList_CopySelToDoc(
      LPLINELIST              lpSrcLL,
      LPLINERANGE             lplrSel,
      LPOUTLINEDOC            lpDestDoc
);
BOOL LineList_SaveSelToStg(
      LPLINELIST              lpLL,
      LPLINERANGE             lplrSel,
      UINT                    uFormat,
      LPSTORAGE               lpSrcStg,
      LPSTORAGE               lpDestStg,
      LPSTREAM                lpLLStm,
      BOOL                    fRemember
);
BOOL LineList_LoadFromStg(
      LPLINELIST              lpLL,
      LPSTORAGE               lpSrcStg,
      LPSTREAM                lpLLStm
);

#if defined( USE_DRAGDROP )
void LineList_SetFocusLineFromPointl( LPLINELIST lpLL, POINTL pointl );
void LineList_SetDragOverLineFromPointl ( LPLINELIST lpLL, POINTL pointl );
void LineList_Scroll(LPLINELIST lpLL, DWORD dwScrollDir);
int LineList_GetLineIndexFromPointl(LPLINELIST lpLL, POINTL pointl);
void LineList_RestoreDragFeedback(LPLINELIST lpLL);
#endif

LRESULT FAR PASCAL LineListWndProc(
   HWND   hWnd,
   UINT   Message,
   WPARAM wParam,
   LPARAM lParam
);


// Document initialization type
#define DOCTYPE_UNKNOWN     0   // new doc created but not yet initialized
#define DOCTYPE_NEW         1   // init from scratch (new doc)
#define DOCTYPE_FROMFILE    2   // init from a file (open doc)



/*************************************************************************
** class OUTLINEDOC
**    There is one instance of the OutlineDoc class created per
**    document open in the app. The SDI version of the app supports one
**    OUTLINEDOC at a time. The MDI version of the app can manage
**    multiple documents at one time.
*************************************************************************/

/* Definition of OUTLINEDOC */
typedef struct tagOUTLINEDOC {
   LINELIST    m_LineList;         // list of lines in the doc
   LPOUTLINENAMETABLE m_lpNameTable;   // table of names in the doc
   HWND        m_hWndDoc;          // client area window for the Doc
   int         m_docInitType;      // is doc new or loaded from a file?
   BOOL        m_fDataTransferDoc; // is doc created for copy | drag/drop
   CLIPFORMAT  m_cfSaveFormat;      // format used to save the doc
   OLECHAR     m_szFileName[256];  // associated file; "(Untitled)" if none
   LPOLESTR    m_lpszDocTitle;     // name of doc to appear in window title
   BOOL        m_fModified;        // is the doc dirty (needs to be saved)?
   UINT        m_nDisableDraw;     // enable/disable updating the display
   SCALEFACTOR m_scale;            // current scale factor of the doc
   int         m_nLeftMargin;      // left margin in Himetric
   int         m_nRightMargin;     // right margin in Himetric
   UINT        m_uCurrentZoom;     // cur. zoom (used for menu checking)
   UINT        m_uCurrentMargin;   // cur. margin (used for menu checking)
#if defined( USE_HEADING )
   HEADING     m_heading;
#endif
```

```
#if defined( USE_FRAMETOOLS )
   LPFRAMETOOLS m_lpFrameTools;    // ptr to frame tools used by this doc
#endif

} OUTLINEDOC;

/* OutlineDoc methods (functions) */

BOOL OutlineDoc_Init(LPOUTLINEDOC lpOutlineDoc, BOOL fDataTransferDoc);
BOOL OutlineDoc_InitNewFile(LPOUTLINEDOC lpOutlineDoc);
LPOUTLINENAMETABLE OutlineDoc_CreateNameTable(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_Destroy(LPOUTLINEDOC lpOutlineDoc);
BOOL OutlineDoc_Close(LPOUTLINEDOC lpOutlineDoc, DWORD dwSaveOption);
void OutlineDoc_ShowWindow(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_FrameWindowResized(
      LPOUTLINEDOC        lpOutlineDoc,
      LPRECT              lprcFrameRect,
      LPBORDERWIDTHS      lpFrameToolWidths
);

void OutlineDoc_ClearCommand(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_CutCommand(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_CopyCommand(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_ClearAllLines(LPOUTLINEDOC lpOutlineDoc);
LPOUTLINEDOC OutlineDoc_CreateDataTransferDoc(LPOUTLINEDOC lpSrcOutlineDoc);
void OutlineDoc_PasteCommand(LPOUTLINEDOC lpOutlineDoc);
int OutlineDoc_PasteOutlineData(LPOUTLINEDOC lpOutlineDoc, HGLOBAL hOutline,
int nStartIndex);
int OutlineDoc_PasteTextData(LPOUTLINEDOC lpOutlineDoc, HGLOBAL hText, int
nStartIndex);
void OutlineDoc_AddTextLineCommand(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_AddTopLineCommand(
      LPOUTLINEDOC        lpOutlineDoc,
      UINT                nHeightInHimetric
);
void OutlineDoc_EditLineCommand(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_IndentCommand(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_UnindentCommand(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_SetLineHeightCommand(LPOUTLINEDOC lpDoc);
void OutlineDoc_SelectAllCommand(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_DefineNameCommand(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_GotoNameCommand(LPOUTLINEDOC lpOutlineDoc);

void OutlineDoc_Print(LPOUTLINEDOC lpOutlineDoc, HDC hDC);
BOOL OutlineDoc_SaveToFile(LPOUTLINEDOC lpOutlineDoc, LPCOLESTR
lpszFileName, UINT uFormat, BOOL fRemember);
void OutlineDoc_AddLine(LPOUTLINEDOC lpOutlineDoc, LPLINE lpLine, int
nIndex);
void OutlineDoc_DeleteLine(LPOUTLINEDOC lpOutlineDoc, int nIndex);
void OutlineDoc_AddName(LPOUTLINEDOC lpOutlineDoc, LPOUTLINENAME
lpOutlineName);
void OutlineDoc_DeleteName(LPOUTLINEDOC lpOutlineDoc, int nIndex);
void OutlineDoc_Resize(LPOUTLINEDOC lpDoc, LPRECT lpRect);
LPOUTLINENAMETABLE OutlineDoc_GetNameTable(LPOUTLINEDOC lpOutlineDoc);
```

```
LPLINELIST OutlineDoc_GetLineList(LPOUTLINEDOC lpOutlineDoc);
int OutlineDoc_GetNameCount(LPOUTLINEDOC lpOutlineDoc);
int OutlineDoc_GetLineCount(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_SetTitle(LPOUTLINEDOC lpOutlineDoc, BOOL fMakeUpperCase);
BOOL OutlineDoc_CheckSaveChanges(
      LPOUTLINEDOC        lpOutlineDoc,
      LPDWORD             lpdwSaveOption
);
BOOL OutlineDoc_IsModified(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_SetModified(LPOUTLINEDOC lpOutlineDoc, BOOL fModified, BOOL
fDataChanged, BOOL fSizeChanged);
void OutlineDoc_SetRedraw(LPOUTLINEDOC lpOutlineDoc, BOOL fEnableDraw);
BOOL OutlineDoc_LoadFromFile(LPOUTLINEDOC lpOutlineDoc, LPOLESTR
szFileName);
BOOL OutlineDoc_SaveSelToStg(
      LPOUTLINEDOC        lpOutlineDoc,
      LPLINERANGE         lplrSel,
      UINT                uFormat,
      LPSTORAGE           lpDestStg,
      BOOL                fSameAsLoad,
      BOOL                fRemember
);
BOOL OutlineDoc_LoadFromStg(LPOUTLINEDOC lpOutlineDoc, LPSTORAGE lpSrcStg);
BOOL OutlineDoc_SetFileName(LPOUTLINEDOC lpOutlineDoc, LPOLESTR
lpszFileName, LPSTORAGE lpNewStg);
HWND OutlineDoc_GetWindow(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_SetSel(LPOUTLINEDOC lpOutlineDoc, LPLINERANGE lplrSel);
int OutlineDoc_GetSel(LPOUTLINEDOC lpOutlineDoc, LPLINERANGE lplrSel);
void OutlineDoc_ForceRedraw(LPOUTLINEDOC lpOutlineDoc, BOOL fErase);
void OutlineDoc_RenderFormat(LPOUTLINEDOC lpOutlineDoc, UINT uFormat);
void OutlineDoc_RenderAllFormats(LPOUTLINEDOC lpOutlineDoc);
HGLOBAL OutlineDoc_GetOutlineData(LPOUTLINEDOC lpOutlineDoc, LPLINERANGE
lplrSel);
HGLOBAL OutlineDoc_GetTextData(LPOUTLINEDOC lpOutlineDoc, LPLINERANGE
lplrSel);
void OutlineDoc_DialogHelp(HWND hDlg, WPARAM wDlgID);
void OutlineDoc_SetCurrentZoomCommand(
      LPOUTLINEDOC        lpOutlineDoc,
      UINT                uCurrentZoom
);
UINT OutlineDoc_GetCurrentZoomMenuCheck(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_SetScaleFactor(
      LPOUTLINEDOC        lpOutlineDoc,
      LPSCALEFACTOR       lpscale,
      LPRECT              lprcDoc
);
LPSCALEFACTOR OutlineDoc_GetScaleFactor(LPOUTLINEDOC lpDoc);
void OutlineDoc_SetCurrentMarginCommand(
      LPOUTLINEDOC        lpOutlineDoc,
      UINT                uCurrentMargin
);
UINT OutlineDoc_GetCurrentMarginMenuCheck(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_SetMargin(LPOUTLINEDOC lpDoc, int nLeftMargin, int
nRightMargin);
LONG OutlineDoc_GetMargin(LPOUTLINEDOC lpDoc);
```

```
#if defined( USE_FRAMETOOLS )
void OutlineDoc_AddFrameLevelTools(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_SetFormulaBarEditText(
      LPOUTLINEDOC            lpOutlineDoc,
      LPLINE                  lpLine
);
void OutlineDoc_SetFormulaBarEditFocus(
      LPOUTLINEDOC            lpOutlineDoc,
      BOOL                    fEditFocus
);
BOOL OutlineDoc_IsEditFocusInFormulaBar(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_UpdateFrameToolButtons(LPOUTLINEDOC lpOutlineDoc);
#endif  // USE_FRAMETOOLS

#if defined( USE_HEADING )
LPHEADING OutlineDoc_GetHeading(LPOUTLINEDOC lpOutlineDoc);
void OutlineDoc_ShowHeading(LPOUTLINEDOC lpOutlineDoc, BOOL fShow);
#endif  // USE_HEADING

/***************************************************************************
** class OUTLINEAPP
**    There is one instance of the OUTLINEAPP class created per running
**    application instance. This object holds many fields that could
**    otherwise be organized as global variables.
***************************************************************************/

/* Definition of OUTLINEAPP */
typedef struct tagOUTLINEAPP {
   HWND            m_hWndApp;        // top-level frame window for the App
   HMENU           m_hMenuApp;       // handle to frame level menu for App
   HACCEL          m_hAccelApp;
   HACCEL          m_hAccelFocusEdit;// Accelerator when Edit in Focus
   LPOUTLINEDOC    m_lpDoc;          // main SDI document visible to user
   LPOUTLINEDOC    m_lpClipboardDoc; // hidden doc for snapshot of copied
sel
   HWND            m_hWndStatusBar;  // window for the status bar
   HCURSOR         m_hcursorSelCur;  // cursor used to select lines
   HINSTANCE       m_hInst;
   PRINTDLG        m_PrintDlg;
   HFONT           m_hStdFont;       // font used for TextLines
   UINT            m_cfOutline;      // clipboard format for Outline data
   HACCEL          m_hAccel;
   HWND            m_hWndAccelTarget;
   FARPROC         m_ListBoxWndProc; // orig listbox WndProc for subclassing

#if defined ( USE_FRAMETOOLS ) || defined ( INPLACE_CNTR )
   BORDERWIDTHS    m_FrameToolWidths;  // space required by frame-level
tools
#endif  // USE_FRAMETOOLS || INPLACE_CNTR

#if defined( USE_FRAMETOOLS )
   FRAMETOOLS      m_frametools;     // frame tools (button & formula bars)
#endif  // USE_FRAMETOOLS
```

```
} OUTLINEAPP, FAR* LPOUTLINEAPP;

/* OutlineApp methods (functions) */
BOOL OutlineApp_InitApplication(LPOUTLINEAPP lpOutlineApp, HINSTANCE hInst);
BOOL OutlineApp_InitInstance(LPOUTLINEAPP lpOutlineApp, HINSTANCE hInst, int
nCmdShow);
BOOL OutlineApp_ParseCmdLine(LPOUTLINEAPP lpOutlineApp, LPSTR lpszCmdLine,
int nCmdShow);
void OutlineApp_Destroy(LPOUTLINEAPP lpOutlineApp);
LPOUTLINEDOC OutlineApp_CreateDoc(
      LPOUTLINEAPP    lpOutlineApp,
      BOOL            fDataTransferDoc
);
HWND OutlineApp_GetWindow(LPOUTLINEAPP lpOutlineApp);
HWND OutlineApp_GetFrameWindow(LPOUTLINEAPP lpOutlineApp);
HINSTANCE OutlineApp_GetInstance(LPOUTLINEAPP lpOutlineApp);
LPOUTLINENAME OutlineApp_CreateName(LPOUTLINEAPP lpOutlineApp);
void OutlineApp_DocUnlockApp(LPOUTLINEAPP lpOutlineApp, LPOUTLINEDOC
lpOutlineDoc);
void OutlineApp_InitMenu(LPOUTLINEAPP lpOutlineApp, LPOUTLINEDOC lpDoc,
HMENU hMenu);
void OutlineApp_GetFrameRect(LPOUTLINEAPP lpOutlineApp, LPRECT
lprcFrameRect);
void OutlineApp_GetClientAreaRect(
      LPOUTLINEAPP        lpOutlineApp,
      LPRECT              lprcClientAreaRect
);
void OutlineApp_GetStatusLineRect(
      LPOUTLINEAPP        lpOutlineApp,
      LPRECT              lprcStatusLineRect
);
void OutlineApp_ResizeWindows(LPOUTLINEAPP lpOutlineApp);
void OutlineApp_ResizeClientArea(LPOUTLINEAPP lpOutlineApp);
void OutlineApp_AboutCommand(LPOUTLINEAPP lpOutlineApp);
void OutlineApp_NewCommand(LPOUTLINEAPP lpOutlineApp);
void OutlineApp_OpenCommand(LPOUTLINEAPP lpOutlineApp);
void OutlineApp_PrintCommand(LPOUTLINEAPP lpOutlineApp);
BOOL OutlineApp_SaveCommand(LPOUTLINEAPP lpOutlineApp);
BOOL OutlineApp_SaveAsCommand(LPOUTLINEAPP lpOutlineApp);
BOOL OutlineApp_CloseAllDocsAndExitCommand(
      LPOUTLINEAPP        lpOutlineApp,
      BOOL                fForceEndSession
);
void OutlineApp_DestroyWindow(LPOUTLINEAPP lpOutlineApp);

#if defined( USE_FRAMETOOLS )
void OutlineApp_SetBorderSpace(
      LPOUTLINEAPP        lpOutlineApp,
      LPBORDERWIDTHS      lpBorderWidths
);
LPFRAMETOOLS OutlineApp_GetFrameTools(LPOUTLINEAPP lpOutlineApp);
void OutlineApp_SetFormulaBarAccel(
      LPOUTLINEAPP            lpOutlineApp,
      BOOL                    fEditFocus
```

```
);
#endif  // USE_FRAMETOOLS

void OutlineApp_SetStatusText(LPOUTLINEAPP lpOutlineApp, LPOLESTR
lpszMessage);
LPOUTLINEDOC OutlineApp_GetActiveDoc(LPOUTLINEAPP lpOutlineApp);
HMENU OutlineApp_GetMenu(LPOUTLINEAPP lpOutlineApp);
HFONT OutlineApp_GetActiveFont(LPOUTLINEAPP lpOutlineApp);
HDC OutlineApp_GetPrinterDC(LPOUTLINEAPP lpApp);
void OutlineApp_PrinterSetupCommand(LPOUTLINEAPP lpOutlineApp);
void OutlineApp_ErrorMessage(LPOUTLINEAPP lpOutlineApp, LPOLESTR lpszMsg);
void OutlineApp_GetAppVersionNo(LPOUTLINEAPP lpOutlineApp, int
narrAppVersionNo[]);
void OutlineApp_GetAppName(LPOUTLINEAPP lpOutlineApp, LPOLESTR lpszAppName);
BOOL OutlineApp_VersionNoCheck(LPOUTLINEAPP lpOutlineApp, LPOLESTR
lpszAppName, int narrAppVersionNo[]);
void OutlineApp_SetEditText(LPOUTLINEAPP lpApp);
void OutlineApp_SetFocusEdit(LPOUTLINEAPP lpApp, BOOL bFocusEdit);
BOOL OutlineApp_GetFocusEdit(LPOUTLINEAPP lpApp);
void OutlineApp_ForceRedraw(LPOUTLINEAPP lpOutlineApp, BOOL fErase);

/* struct definition for persistant data storage of OutlineDoc data */

#pragma pack(push, 2)
typedef struct tagOUTLINEDOCHEADER_ONDISK {
        OLECHAR     m_szFormatName[32];
        short       m_narrAppVersionNo[2];
        USHORT      m_fShowHeading;
        DWORD       m_reserved1;            // space reserved for future use
        DWORD       m_reserved2;            // space reserved for future use
        DWORD       m_reserved3;            // space reserved for future use
        DWORD       m_reserved4;            // space reserved for future use
} OUTLINEDOCHEADER_ONDISK, FAR* LPOUTLINEDOCHEADER_ONDISK;
#pragma pack(pop)

typedef struct tagOUTLINEDOCHEADER {
   OLECHAR     m_szFormatName[32];
   int         m_narrAppVersionNo[2];
   BOOL        m_fShowHeading;
   DWORD       m_reserved1;            // space reserved for future use
   DWORD       m_reserved2;            // space reserved for future use
   DWORD       m_reserved3;            // space reserved for future use
   DWORD       m_reserved4;            // space reserved for future use
} OUTLINEDOCHEADER, FAR* LPOUTLINEDOCHEADER;

#pragma pack(push,2)
typedef struct tagLINELISTHEADER_ONDISK {
        USHORT      m_nNumLines;
        DWORD       m_reserved1;            // space reserved for future use
        DWORD       m_reserved2;            // space reserved for future use
} LINELISTHEADER_ONDISK, FAR* LPLINELISTHEADER_ONDISK;
#pragma pack(pop)


typedef struct tagLINELISTHEADER {
   int         m_nNumLines;
   DWORD       m_reserved1;            // space reserved for future use
   DWORD       m_reserved2;            // space reserved for future use
} LINELISTHEADER, FAR* LPLINELISTHEADER;

#pragma pack(push,2)
typedef struct tagLINERECORD_ONDISK {
        USHORT      m_lineType;
        USHORT      m_nTabLevel;
        USHORT      m_nTabWidthInHimetric;
        USHORT      m_nWidthInHimetric;
        USHORT      m_nHeightInHimetric;
        DWORD       m_reserved;         // space reserved for future use
} LINERECORD_ONDISK, FAR* LPLINERECORD_ONDISK;
#pragma pack(pop)

typedef struct tagLINERECORD {
   LINETYPE    m_lineType;
   UINT        m_nTabLevel;
   UINT        m_nTabWidthInHimetric;
   UINT        m_nWidthInHimetric;
   UINT        m_nHeightInHimetric;
   DWORD       m_reserved;         // space reserved for future use
} LINERECORD, FAR* LPLINERECORD;


/* Function prototypes in main.c */
int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                  LPSTR lpszCmdLine, int nCmdShow);
BOOL MyTranslateAccelerator(LPMSG lpmsg);
int GetAccelItemCount(HACCEL hAccel);

LRESULT CALLBACK EXPORT AppWndProc(HWND hWnd, UINT Message, WPARAM wParam,
                  LPARAM lParam);
LRESULT CALLBACK EXPORT DocWndProc(HWND hWnd, UINT Message, WPARAM wParam,
                  LPARAM lParam);

/* Function prototypes in outldlgs.c */
BOOL InputTextDlg(HWND hWnd, LPSTR lpszText, LPSTR lpszDlgTitle);
BOOL CALLBACK EXPORT AddEditDlgProc(HWND, UINT, WPARAM, LPARAM);
BOOL CALLBACK EXPORT SetLineHeightDlgProc(HWND hDlg, UINT Message, WPARAM
wParam, LPARAM lParam);
BOOL CALLBACK EXPORT DefineNameDlgProc(HWND, UINT, WPARAM, LPARAM);
BOOL CALLBACK EXPORT GotoNameDlgProc(HWND, UINT, WPARAM, LPARAM);
void NameDlg_LoadComboBox(LPOUTLINENAMETABLE lpOutlineNameTable,HWND
hCombo);
void NameDlg_LoadListBox(LPOUTLINENAMETABLE lpOutlineNameTable,HWND
hListBox);
void NameDlg_AddName(HWND hCombo, LPOUTLINEDOC lpOutlineDoc, LPOLESTR
lpszName, LPLINERANGE lplrSel);
void NameDlg_UpdateName(HWND hCombo, LPOUTLINEDOC lpOutlineDoc, int nIndex,
LPOLESTR lpszName, LPLINERANGE lplrSel);
void NameDlg_DeleteName(HWND hCombo, LPOUTLINEDOC lpOutlineDoc, UINT
nIndex);
```

```
BOOL CALLBACK EXPORT AboutDlgProc(HWND hDlg, UINT Message, WPARAM wParam,
LPARAM lParam);

/* Function prototypes in outldata.c */
LPVOID New(DWORD lSize);
void Delete(LPVOID p);

/* Function prototypes in outlprnt.c */
BOOL CALLBACK EXPORT AbortProc (HDC hdc, WORD reserved);
BOOL CALLBACK EXPORT PrintDlgProc(HWND hwnd, WORD msg, WORD wParam, LONG
lParam);

/* Function prototypes in debug.c */
void SetDebugLevelCommand(void);
void TraceDebug(HWND, int);

/*************************************************************************
** DEBUG ASSERTION ROUTINES
*************************************************************************/

#ifdef DBG
#include <assert.h>
#define FnAssert(lpstrExpr, lpstrMsg, lpstrFileName, iLine)     \
        (_assert(lpstrMsg ? lpstrMsg : lpstrExpr,               \
                 lpstrFileName,                                 \
                 iLine), NOERROR)
#endif //DBG

#if defined( OLE_VERSION )
#include "oleoutl.h"

#endif  // OLE_VERSION


#endif // _OUTLINE_H_
```

OUTLINE.MK   (OUTLINE Sample)

```
#
# Makefile : Builds the OLE 2.0 Outline series sample apps
#

!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

all: $(APP).exe

OLE_FLAGS = -DOLE2FINAL -DWIN32S

!ifndef NO_DEBUG
OLE_FLAGS = $(OLE_FLAGS) -DDBG -D_DEBUG -DOLE2SHIP
!endif

OLELIBS   = ole32.lib uuid.lib oleaut32.lib ole2ui.lib gizmobar.lib
bttncur.lib

PCHFLAGS  = -Yuoutline.h -Fp$(APP).pch

COMMONINCL =    outline.h frametls.h outlrc.h status.h cntroutl.h cntrrc.h
svroutl.h oleoutl.h

!if "$(USE_MSGFILTER)"=="1"
OLE_FLAGS  = $(OLE_FLAGS) /DUSE_MSGFILTER
!endif

PRECOMPOBJ = $(APP).obj


##########################################################################
#
# main obj lists; add new obj files here
#

!if "$(APP)" == "cntroutl"
OLE_FLAGS = $(OLE_FLAGS) /DOLE_CNTR
APP_OBJS = main.obj memmgr.obj status.obj frametls.obj \
       dialogs.obj debug.obj \
       outlapp.obj outldoc.obj heading.obj \
       outllist.obj outlline.obj outltxtl.obj \
       outlntbl.obj outlname.obj \
       oleapp.obj oledoc.obj classfac.obj debug2.obj \
       dragdrop.obj clipbrd.obj linking.obj \
       cntrbase.obj cntrline.obj
!else

!if "$(APP)" == "svroutl"
OLE_FLAGS = $(OLE_FLAGS) /DOLE_SERVER
APP_OBJS = main.obj memmgr.obj status.obj frametls.obj \
       dialogs.obj debug.obj \
       outlapp.obj outldoc.obj heading.obj \
       outllist.obj outlline.obj outltxtl.obj \
       outlntbl.obj outlname.obj \
       oleapp.obj oledoc.obj classfac.obj debug2.obj \
       dragdrop.obj clipbrd.obj linking.obj \
       svrbase.obj svrpsobj.obj
!else

!if "$(APP)" == "icntrotl"
OLE_FLAGS = $(OLE_FLAGS) /DOLE_CNTR /DINPLACE_CNTR
APP_OBJS = main.obj memmgr.obj status.obj frametls.obj \
       dialogs.obj debug.obj \
       outlapp.obj outldoc.obj heading.obj \
       outllist.obj outlline.obj outltxtl.obj \
       outlntbl.obj outlname.obj \
       oleapp.obj oledoc.obj classfac.obj debug2.obj \
       dragdrop.obj clipbrd.obj linking.obj \
       cntrbase.obj cntrline.obj cntrinpl.obj
!else

!if "$(APP)" == "isvrotl"
OLE_FLAGS = $(OLE_FLAGS) /DOLE_SERVER /DINPLACE_SVR
APP_OBJS = main.obj memmgr.obj status.obj frametls.obj \
       dialogs.obj debug.obj \
       outlapp.obj outldoc.obj heading.obj \
       outllist.obj outlline.obj outltxtl.obj \
       outlntbl.obj outlname.obj \
       oleapp.obj oledoc.obj classfac.obj debug2.obj \
       dragdrop.obj clipbrd.obj linking.obj \
       svrbase.obj svrpsobj.obj svrinpl.obj
!else
APP_OBJS = main.obj memmgr.obj status.obj frametls.obj \
       dialogs.obj debug.obj \
       outlapp.obj outldoc.obj heading.obj \
       outllist.obj outlline.obj outltxtl.obj debug2.obj \
       outlntbl.obj outlname.obj

!endif
!endif
!endif
!endif


##########################################################################
#
# create precomiled header
#
$(APP).pch : $(APP).c $(COMMONINCL)
    @echo Precompiling outline.h ...
    $(cc) $(cflags) $(cvars) $(cdebug) $(OLE_FLAGS) -Ycoutline.h -Fp$
(APP).pch -Fo$(APP) $(APP).c

##########################################################################
#
# link/res commands
```

```
$(APP).exe: $(APP).pch $(APP_OBJS) $(APP).def $(APP).res ..\lib\gizmobar.lib
..\lib\bttncur.lib
    $(link) $(linkdebug) $(guilflags) $(PRECOMPOBJ) $(APP_OBJS) $(APP).res
-out:$@ -map:$*.map $(OLELIBS) $(guilibsdll) advapi32.lib shell32.lib
    if not exist ..\bin mkdir ..\bin
    copy $(APP).exe ..\bin

$(APP).res: $(APP).rc outlrc.h cntrrc.h dialogs.dlg debug.rc
    rc -r -DWIN32 $(RCFLAGS) -fo$@ $(APP).rc


###########################################################################
#
# build rules for src directory
#

.c.obj:
    $(cc) $(cflags) $(cvars) $(cdebug) $(OLE_FLAGS) $(PCHFLAGS) $*.c


###########################################################################
#
# clean (erase) generated files

clean:
    -del *.obj
    -del *.res
    -del *.exe
    -del *.map
    -del *.pch


##########################################################
# Dependencies
##########################################################

main.obj : $(COMMONINCL)
    $(cc) $(cflags) $(cvars) $(cdebug) $(OLE_FLAGS) main.c

outlapp.obj : $(COMMONINCL)

outldoc.obj : $(COMMONINCL)

outllist.obj : $(COMMONINCL)

outlline.obj : $(COMMONINCL)

outltxtl.obj : $(COMMONINCL)

outlntbl.obj : $(COMMONINCL)

outlname.obj : $(COMMONINCL)

classfac.obj : $(COMMONINCL)
```

```
oleapp.obj : $(COMMONINCL)

oledoc.obj : $(COMMONINCL)

dragdrop.obj : $(COMMONINCL)

clipbrd.obj : $(COMMONINCL)

linking.obj : $(COMMONINCL)

cntrbase.obj : $(COMMONINCL)

cntrline.obj : $(COMMONINCL)

cntrinpl.obj : $(COMMONINCL)

svrpsobj.obj : $(COMMONINCL)

svrinpl.obj : $(COMMONINCL)

svrbase.obj : $(COMMONINCL)

status.obj : $(COMMONINCL) message.h status.h

memmgr.obj :
    $(cc) $(cflags) $(cvars) $(cdebug) $(OLE_FLAGS)  memmgr.c

frametls.obj : $(COMMONINCL)

heading.obj : $(COMMONINCL)

dialogs.obj : $(COMMONINCL)

debug.obj : $(COMMONINCL)

debug2.obj : $(COMMONINCL)
```

OUTLINE.RC (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Sample Code
**
**    outline.rc
**
**    Resource file for outline.exe
**
**     (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "windows.h"
#include "outlrc.h"

SelCur    CURSOR selcross.cur

OutlineMenu MENU
  BEGIN
    POPUP  "&File"
      BEGIN
        MENUITEM "&New", IDM_F_NEW
        MENUITEM "&Open...\t  Ctrl+F12", IDM_F_OPEN
        MENUITEM "&Save\t  Shift+F12", IDM_F_SAVE
        MENUITEM "Save &As...\t  F12", IDM_F_SAVEAS
        MENUITEM SEPARATOR
        MENUITEM "&Print...\t  Ctrl+Shift+F12", IDM_F_PRINT
        MENUITEM "Printer Se&tup...", IDM_F_PRINTERSETUP
        MENUITEM SEPARATOR
        MENUITEM "E&xit\t  Alt+F4", IDM_F_EXIT
      END
    POPUP  "&Edit"
      BEGIN
        MENUITEM "&Undo", IDM_E_UNDO
        MENUITEM SEPARATOR
        MENUITEM "Cu&t\t  Ctrl+X", IDM_E_CUT
        MENUITEM "&Copy\t  Ctrl+C", IDM_E_COPY
        MENUITEM "&Paste\t  Ctrl+V", IDM_E_PASTE
        MENUITEM "Cl&ear\t  Del", IDM_E_CLEAR
        MENUITEM SEPARATOR
        MENUITEM "Select A&ll\t  Ctrl+A", IDM_E_SELECTALL
      END
     POPUP  "O&utline"
       BEGIN
        POPUP  "&Zoom"
          BEGIN
            MENUITEM "&400%", IDM_V_ZOOM_400
            MENUITEM "&300%", IDM_V_ZOOM_300
            MENUITEM "&200%", IDM_V_ZOOM_200
            MENUITEM "&100%", IDM_V_ZOOM_100
            MENUITEM "&75%", IDM_V_ZOOM_75
            MENUITEM "&50%", IDM_V_ZOOM_50
            MENUITEM "&25%", IDM_V_ZOOM_25
          END
        POPUP  "&Left and Right margins"
          BEGIN
            MENUITEM "&nil", IDM_V_SETMARGIN_0
            MENUITEM "&1 cm", IDM_V_SETMARGIN_1
            MENUITEM "&2 cm", IDM_V_SETMARGIN_2
            MENUITEM "&3 cm", IDM_V_SETMARGIN_3
            MENUITEM "&4 cm", IDM_V_SETMARGIN_4
          END
        POPUP "Add &Top Line"
             BEGIN
               MENUITEM "&1 cm", IDM_V_ADDTOP_1
               MENUITEM "&2 cm", IDM_V_ADDTOP_2
               MENUITEM "&3 cm", IDM_V_ADDTOP_3
               MENUITEM "&4 cm", IDM_V_ADDTOP_4
             END
       END
    POPUP  "&Line"
      BEGIN
        MENUITEM "&Add Line\t  Enter", IDM_L_ADDLINE
        MENUITEM "E&dit Line\t  Alt+Enter", IDM_L_EDITLINE
        MENUITEM SEPARATOR
        MENUITEM "&Indent Line\t  Tab", IDM_L_INDENTLINE
        MENUITEM "U&nindent Line\t  Shift+Tab", IDM_L_UNINDENTLINE
           MENUITEM SEPARATOR
           MENUITEM "&Set Line Height...", IDM_L_SETLINEHEIGHT
      END
    POPUP  "&Name"
      BEGIN
        MENUITEM "&Define Name...", IDM_N_DEFINENAME
        MENUITEM "&Goto Name...", IDM_N_GOTONAME
      END
    POPUP  "&Options"
      BEGIN
        POPUP  "&Button Bar Display"
          BEGIN
            MENUITEM "At &Top", IDM_O_BB_TOP
            MENUITEM "At &Bottom", IDM_O_BB_BOTTOM
            MENUITEM "&Popup", IDM_O_BB_POPUP
            MENUITEM "&Hide", IDM_O_BB_HIDE
          END
        POPUP  "&Formula Bar Display"
          BEGIN
            MENUITEM "At &Top", IDM_O_FB_TOP
            MENUITEM "At &Bottom", IDM_O_FB_BOTTOM
            MENUITEM "&Popup", IDM_O_FB_POPUP
          END
        POPUP  "&Row and Column Heading"
          BEGIN
            MENUITEM "&Show", IDM_O_HEAD_SHOW
            MENUITEM "&Hide", IDM_O_HEAD_HIDE
          END
      END
    POPUP  "&Help"
```

```
      BEGIN
        MENUITEM "&About...", IDM_H_ABOUT
      END
  END

OutlineAccel ACCELERATORS
  BEGIN
    VK_F12, IDM_F_OPEN, VIRTKEY, CONTROL
    VK_F12, IDM_F_SAVE, VIRTKEY, SHIFT
    VK_F12, IDM_F_SAVEAS, VIRTKEY
    VK_F12, IDM_F_PRINT, VIRTKEY, CONTROL, SHIFT

    "x", IDM_E_CUT, VIRTKEY, CONTROL
    "c", IDM_E_COPY, VIRTKEY, CONTROL
    "v", IDM_E_PASTE, VIRTKEY, CONTROL
    VK_DELETE, IDM_E_CLEAR, VIRTKEY
    VK_RETURN, IDM_L_ADDLINE, VIRTKEY
    VK_RETURN, IDM_L_EDITLINE, VIRTKEY, ALT
    VK_TAB, IDM_L_INDENTLINE, VIRTKEY
    VK_TAB, IDM_L_UNINDENTLINE, VIRTKEY, SHIFT
    "a", IDM_E_SELECTALL, VIRTKEY, CONTROL

    ; old conventions for editing
    VK_INSERT, IDM_E_COPY, VIRTKEY, CONTROL
    VK_DELETE, IDM_E_CUT, VIRTKEY, SHIFT
    VK_INSERT, IDM_E_PASTE, VIRTKEY, SHIFT

    VK_F2, IDM_F2, VIRTKEY
  END


; Same as OutlineAccel but without Delete
; used when edit control of Formula Bar in focus
;
OutlineAccelFocusEdit ACCELERATORS
  BEGIN
    VK_F12, IDM_F_OPEN, VIRTKEY, CONTROL
    VK_F12, IDM_F_SAVE, VIRTKEY, SHIFT
    VK_F12, IDM_F_SAVEAS, VIRTKEY
    VK_F12, IDM_F_PRINT, VIRTKEY, CONTROL, SHIFT

    "x", IDM_E_CUT, VIRTKEY, CONTROL
    "c", IDM_E_COPY, VIRTKEY, CONTROL
    "v", IDM_E_PASTE, VIRTKEY, CONTROL
    VK_RETURN, IDM_L_ADDLINE, VIRTKEY
    VK_RETURN, IDM_L_EDITLINE, VIRTKEY, ALT
    VK_TAB, IDM_L_INDENTLINE, VIRTKEY
    VK_TAB, IDM_L_UNINDENTLINE, VIRTKEY, SHIFT
     "a", IDM_E_SELECTALL, VIRTKEY, CONTROL

     VK_ESCAPE, IDM_FB_CANCEL, VIRTKEY

    ; old conventions for editing
    VK_INSERT, IDM_E_COPY, VIRTKEY, CONTROL
    VK_DELETE, IDM_E_CUT, VIRTKEY, SHIFT
    VK_INSERT, IDM_E_PASTE, VIRTKEY, SHIFT
  END

OutlineIcon ICON outline.ico

Image72     BITMAP      image72.bmp
Image96     BITMAP      image96.bmp
Image120    BITMAP      image120.bmp
LogoBitmap  BITMAP      ole2.bmp

#include "DIALOGS.DLG"
```

OUTLINE.REG   (OUTLINE Sample)

```
REGEDIT
(the above line used as a quick check that we are indeed a registration
script)
/*************************************************************************
** REGISTRATION INFORMATION FOR ALL OUTLINE SERIES APPLICATIONS
*************************************************************************/

ALL LINES THAT DON'T START WITH 'HKEY_CLASSES_ROOT' ARE COMMENTS.

THIS FILE CONSISTS OF A LIST OF <key> <value> PAIRS. THE key AND value
SHOULD
BE SEPERATED BY A " = " mark (note spaces).
*


--- VERSIONLESS PROGID ROOT-KEY INFORMATION
    ---------------------------------------

--- ISVROTL is used as the current version of an OLEOutline server
HKEY_CLASSES_ROOT\OLEOutline = Ole 2.0 In-Place Server Outline
HKEY_CLASSES_ROOT\OLEOutline\CLSID = {00000402-0000-0000-C000-000000000046}
HKEY_CLASSES_ROOT\OLEOutline\CurVer = OLE2ISvrOtl
HKEY_CLASSES_ROOT\OLEOutline\CurVer\Insertable


/*************************************************************************
** REGISTRATION ENTRY FOR SVROUTL.EXE
*************************************************************************/

--- PROGID ROOT-KEY INFORMATION
    ---------------------------

HKEY_CLASSES_ROOT\OLE2SvrOutl = Ole 2.0 Server Sample Outline
HKEY_CLASSES_ROOT\OLE2SvrOutl\CLSID = {00000400-0000-0000-C000-000000000046}

--- SVROUTL is marked as insertable so it appears in the InsertObject dialog
HKEY_CLASSES_ROOT\OLE2SvrOutl\Insertable


--- OLE 1.0 COMPATIBILITY INFORMATION
    ---------------------------------

HKEY_CLASSES_ROOT\OLE2SvrOutl\protocol\StdFileEditing\verb\0 = &Edit
HKEY_CLASSES_ROOT\OLE2SvrOutl\protocol\StdFileEditing\server = svroutl.exe


--- WINDOWS 3.1 SHELL INFORMATION
    -----------------------------

HKEY_CLASSES_ROOT\OLE2SvrOutl\Shell\Print\Command = svroutl.exe %1
HKEY_CLASSES_ROOT\OLE2SvrOutl\Shell\Open\Command = svroutl.exe %1
```

```
--- OLE 2.0 CLSID ENTRY INFORMATION
    ------------------------------

HKEY_CLASSES_ROOT\CLSID\{00000400-0000-0000-C000-000000000046} = Ole 2.0
Server Sample Outline
HKEY_CLASSES_ROOT\CLSID\{00000400-0000-0000-C000-000000000046}\ProgID =
OLE2SvrOutl


--- OLE 2.0 OBJECT HANDLER/EXE INFORMATION
    --------------------------------------

HKEY_CLASSES_ROOT\CLSID\{00000400-0000-0000-C000-
000000000046}\InprocHandler32 = ole32.dll

HKEY_CLASSES_ROOT\CLSID\{00000400-0000-0000-C000-000000000046}\LocalServer32
= svroutl.exe


--- VERB MENU SUPPORT
    -----------------

--- Verb 0: "Edit", MF_UNCHECKED | MF_ENABLED, OLEVERBATTRIB_ONCONTAINERMENU
HKEY_CLASSES_ROOT\CLSID\{00000400-0000-0000-C000-000000000046}\Verb\0 =
&Edit,0,2

--- This class should appear in Insert New Object list
HKEY_CLASSES_ROOT\CLSID\{00000400-0000-0000-C000-000000000046}\Insertable


--- USER TYPE NAMES
    ---------------

--- ShortName (NOTE: max 15 chars) = Server Outline
HKEY_CLASSES_ROOT\CLSID\{00000400-0000-0000-C000-000000000046}\AuxUserType\2
= Outline

--- AppName = Ole 2.0 Outline Server
HKEY_CLASSES_ROOT\CLSID\{00000400-0000-0000-C000-000000000046}\AuxUserType\3
= Ole 2.0 Outline Server


--- ICON DEFINITION
    ---------------

HKEY_CLASSES_ROOT\CLSID\{00000400-0000-0000-C000-000000000046}\DefaultIcon =
svroutl.exe,0


--- DATA FORMATS SUPPORTED
    ----------------------

--- Default File Format = CF_Outline
```

```
HKEY_CLASSES_ROOT\CLSID\{00000400-0000-0000-C000-
000000000046}\DataFormats\DefaultFile = Outline

--- Format 0 = CF_OUTLINE, DVASPECT_CONTENT, TYMED_HGLOBAL, DATADIR_BOTH
HKEY_CLASSES_ROOT\CLSID\{00000400-0000-0000-C000-
000000000046}\DataFormats\GetSet\0 = Outline,1,1,3

--- Format 1 = CF_TEXT, DVASPECT_CONTENT, TYMED_HGLOBAL, DATADIR_BOTH
HKEY_CLASSES_ROOT\CLSID\{00000400-0000-0000-C000-
000000000046}\DataFormats\GetSet\1 = 1,1,1,3

--- Format 2 = CF_METAFILEPICT, DVASPECT_CONTENT, TYMED_MFPICT, DATADIR_GET
HKEY_CLASSES_ROOT\CLSID\{00000400-0000-0000-C000-
000000000046}\DataFormats\GetSet\2 = 3,1,32,1

--- Format 3 = CF_METAFILEPICT, DVASPECT_ICON, TYMED_MFPICT, DATADIR_GET
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\DataFormats\GetSet\3 = 3,4,32,1

--- MISC STATUS SUPPORTED
    ---------------------


/* DVASPECT_CONTENT = OLEMISC_RENDERINGISDEVICEINDEPENDENT */
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\MiscStatus =
512


--- CONVERSION FORMATS SUPPORTED
    ----------------------------

--- Readable Main formats: CF_OUTLINE, CF_TEXT
HKEY_CLASSES_ROOT\CLSID\{00000400-0000-0000-C000-
000000000046}\Conversion\Readable\Main = Outline
HKEY_CLASSES_ROOT\CLSID\{00000400-0000-0000-C000-
000000000046}\Conversion\Readwritable\Main = Outline



/***************************************************************************
** REGISTRATION ENTRY FOR CNTROUTL.EXE
***************************************************************************/


ENTRIES FOR Ole 2.0 Container Sample Outline


--- PROGID ROOT-KEY INFORMATION
    ---------------------------
    NOTE: CNTROUTL must have a ProgID assigned for the Windows 3.1 Shell
    file associations and Packager to function correctly.

HKEY_CLASSES_ROOT\OLE2CntrOutl = Ole 2.0 Container Sample Outline
HKEY_CLASSES_ROOT\OLE2CntrOutl\Clsid = {00000401-0000-0000-C000-
000000000046}
```

```
--- WINDOWS 3.1 SHELL INFORMATION
    -----------------------------

HKEY_CLASSES_ROOT\OLE2CntrOutl\Shell\Print\Command = cntroutl.exe %1
HKEY_CLASSES_ROOT\OLE2CntrOutl\Shell\Open\Command = cntroutl.exe %1


--- OLE 2.0 CLSID ENTRY INFORMATION
    -------------------------------

HKEY_CLASSES_ROOT\CLSID\{00000401-0000-0000-C000-000000000046} = Ole 2.0
Container Sample Outline
HKEY_CLASSES_ROOT\CLSID\{00000401-0000-0000-C000-000000000046}\ProgID =
OLE2CntrOutl


--- OLE 2.0 OBJECT HANDLER/EXE INFORMATION
    --------------------------------------

HKEY_CLASSES_ROOT\CLSID\{00000401-0000-0000-C000-
000000000046}\InprocHandler32 = ole32.dll
HKEY_CLASSES_ROOT\CLSID\{00000401-0000-0000-C000-000000000046}\LocalServer32
= cntroutl.exe


/*************************************************************************
** REGISTRATION ENTRY FOR ISVROTL.EXE
*************************************************************************/


--- PROGID ROOT-KEY INFORMATION
    ---------------------------

HKEY_CLASSES_ROOT\OLE2ISvrOtl = Ole 2.0 In-Place Server Outline
HKEY_CLASSES_ROOT\OLE2ISvrOtl\CLSID = {00000402-0000-0000-C000-000000000046}

--- ISVROTL is marked as insertable so it appears in the InsertObject dialog
HKEY_CLASSES_ROOT\OLE2ISvrOtl\Insertable


--- OLE 1.0 COMPATIBILITY INFORMATION
    ---------------------------------

HKEY_CLASSES_ROOT\OLE2ISvrOtl\protocol\StdFileEditing\verb\1 = &Open
HKEY_CLASSES_ROOT\OLE2ISvrOtl\protocol\StdFileEditing\verb\0 = &Edit
HKEY_CLASSES_ROOT\OLE2ISvrOtl\protocol\StdFileEditing\server = isvrotl.exe


--- WINDOWS 3.1 SHELL INFORMATION
    -----------------------------

HKEY_CLASSES_ROOT\OLE2ISvrOtl\Shell\Print\Command = isvrotl.exe %1
HKEY_CLASSES_ROOT\OLE2ISvrOtl\Shell\Open\Command = isvrotl.exe %1
```

```
--- File extension must have ProgID as its value
HKEY_CLASSES_ROOT\.oln = OLE2ISvrOtl


--- OLE 2.0 CLSID ENTRY INFORMATION
    ------------------------------

HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046} = Ole 2.0 In-
Place Server Outline
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\ProgID =
OLE2ISvrOtl


--- OLE 2.0 OBJECT HANDLER/EXE INFORMATION
    --------------------------------------

HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\ProgID =
OLE2ISvrOtl
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\InprocHandler32 = ole32.dll

HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\LocalServer32
= isvrotl.exe


--- VERB MENU SUPPORT
    -----------------

--- Verb 1: "Open", MF_UNCHECKED | MF_ENABLED, OLEVERBATTRIB_ONCONTAINERMENU
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\Verb\1 =
&Open,0,2

--- Verb 0: "Edit", MF_UNCHECKED | MF_ENABLED, OLEVERBATTRIB_ONCONTAINERMENU
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\Verb\0 =
&Edit,0,2

--- This class should appear in Insert New Object list
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\Insertable


--- USER TYPE NAMES
    ---------------

--- ShortName (NOTE: recommended max 15 chars) = In-Place Outline
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\AuxUserType\2
= Outline

--- AppName = Ole 2.0 In-Place Outline Server
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\AuxUserType\3
= Ole 2.0 In-Place Outline Server


--- ICON DEFINITION
    ---------------
```

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\DefaultIcon =
isvrotl.exe,0


--- DATA FORMATS SUPPORTED
    ----------------------

--- Default File Format = CF_OUTLINE
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\DataFormats\DefaultFile = Outline

--- Format 0 = CF_OUTLINE, DVASPECT_CONTENT, TYMED_HGLOBAL, DATADIR_BOTH
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\DataFormats\GetSet\0 = Outline,1,1,3

--- Format 1 = CF_TEXT, DVASPECT_CONTENT, TYMED_HGLOBAL, DATADIR_BOTH
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\DataFormats\GetSet\1 = 1,1,1,3

--- Format 2 = CF_METAFILEPICT, DVASPECT_CONTENT, TYMED_MFPICT, DATADIR_GET
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\DataFormats\GetSet\2 = 3,1,32,1

--- Format 3 = CF_METAFILEPICT, DVASPECT_ICON, TYMED_MFPICT, DATADIR_GET
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\DataFormats\GetSet\3 = 3,4,32,1


--- MISC STATUS SUPPORTED
    ---------------------

/* DVASPECT_CONTENT = OLEMISC_RENDERINGISDEVICEINDEPENDENT */
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\MiscStatus =
512

/* DVASPECT_CONTENT =
**    OLEMISC_INSIDEOUT | OLEMISC_ACTIVATEWHENVISIBLE |
**    OLEMISC_RENDERINGISDEVICEINDEPENDENT
*/
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\MiscStatus\1
= 896


--- CONVERSION FORMATS SUPPORTED
    ----------------------------

--- Readable Main formats: CF_OUTLINE, CF_TEXT
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\Conversion\Readable\Main = Outline
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\Conversion\Readwritable\Main = Outline


/**************************************************************************
```

```
** REGISTRATION ENTRY FOR ICNTROTL.EXE
*************************************************************************/


ENTRIES FOR Ole 2.0 In-Place Container Outline


--- PROGID ROOT-KEY INFORMATION
    ---------------------------
    NOTE: ICNTROTL must have a ProgID assigned for the Windows 3.1 Shell
    file associations and Packager to function correctly.

HKEY_CLASSES_ROOT\OLE2ICtrOtl = Ole 2.0 In-Place Container Outline
HKEY_CLASSES_ROOT\OLE2ICtrOtl\Clsid = {00000403-0000-0000-C000-000000000046}


--- WINDOWS 3.1 SHELL INFORMATION
    -----------------------------

HKEY_CLASSES_ROOT\OLE2ICtrOtl\Shell\Print\Command = icntrotl.exe %1
HKEY_CLASSES_ROOT\OLE2ICtrOtl\Shell\Open\Command = icntrotl.exe %1

--- File extension must have ProgID as its value
HKEY_CLASSES_ROOT\.olc = OLE2ICtrOtl


--- OLE 2.0 CLSID ENTRY INFORMATION
    -------------------------------

HKEY_CLASSES_ROOT\CLSID\{00000403-0000-0000-C000-000000000046} = Ole 2.0 In-
Place Container Outline
HKEY_CLASSES_ROOT\CLSID\{00000403-0000-0000-C000-000000000046}\ProgID =
OLE2ICtrOtl


--- OLE 2.0 OBJECT HANDLER/EXE INFORMATION
    --------------------------------------

HKEY_CLASSES_ROOT\CLSID\{00000403-0000-0000-C000-
000000000046}\InprocHandler32 = ole32.dll
HKEY_CLASSES_ROOT\CLSID\{00000403-0000-0000-C000-000000000046}\LocalServer32
= icntrotl.exe
_
```

OUTLINE.C (OUTLINE Sample)

```
/**************************************************************************
**
**    OLE 2.0 Sample Code
**
**    precomp.c
**
**    This file is used to precompile the OUTLINE.H header file
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
**************************************************************************/

#include "outline.h"
```

OUTLLINE.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    outlline.c
**
**    This file contains Line functions.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/


#include "outline.h"


OLEDBGDATA

extern LPOUTLINEAPP g_lpApp;


/* Line_Init
 * ---------
 *
 *      Init the calculated data of a line object
 */
void Line_Init(LPLINE lpLine, int nTab, HDC hDC)
{
   lpLine->m_lineType               = UNKNOWNLINETYPE;
   lpLine->m_nTabLevel              = nTab;
   lpLine->m_nTabWidthInHimetric    =
Line_CalcTabWidthInHimetric(lpLine,hDC);
   lpLine->m_nWidthInHimetric       = 0;
   lpLine->m_nHeightInHimetric      = 0;
   lpLine->m_fSelected              = FALSE;

#if defined( USE_DRAGDROP )
   lpLine->m_fDragOverLine          = FALSE;
#endif
}


/* Line_Edit
 * ---------
 *
 *      Edit the line object.
 *
 *      Returns TRUE if line was changed
 *              FALSE if the line was NOT changed
 */
BOOL Line_Edit(LPLINE lpLine, HWND hWndDoc, HDC hDC)
{
   switch (lpLine->m_lineType) {
      case TEXTLINETYPE:
         return TextLine_Edit((LPTEXTLINE)lpLine, hWndDoc, hDC);

#if defined( OLE_CNTR )
      case CONTAINERLINETYPE:
         ContainerLine_Edit((LPCONTAINERLINE)lpLine, hWndDoc, hDC);
         break;
#endif

      default:
         return FALSE;       // unknown line type
   }
}


/* Line_GetLineType
 * ----------------
 *
 * Return type of the line
 */
LINETYPE Line_GetLineType(LPLINE lpLine)
{
   if (! lpLine) return 0;

   return lpLine->m_lineType;
}


/* Line_GetTextLen
 * ---------------
 *
 * Return length of string representation of the Line
 *  (not considering the tab level).
 */
int Line_GetTextLen(LPLINE lpLine)
{
   switch (lpLine->m_lineType) {
      case TEXTLINETYPE:
         return TextLine_GetTextLen((LPTEXTLINE)lpLine);

#if defined( OLE_CNTR )
      case CONTAINERLINETYPE:
         return ContainerLine_GetTextLen((LPCONTAINERLINE)lpLine);
#endif

      default:
         return 0;       // unknown line type
   }
}


/* Line_GetTextData
 * ----------------
 *
 * Return the string representation of the Line.
 *  (not considering the tab level).
 */
void Line_GetTextData(LPLINE lpLine, LPOLESTR lpszBuf)
{
   char szAnsiStr[256];

   switch (lpLine->m_lineType) {
     case TEXTLINETYPE:
         W2A (lpszBuf, szAnsiStr, 256);
         TextLine_GetTextData((LPTEXTLINE)lpLine, szAnsiStr);
         break;

#if defined( OLE_CNTR )
      case CONTAINERLINETYPE:
         ContainerLine_GetTextData((LPCONTAINERLINE)lpLine, lpszBuf);
         break;
#endif

      default:
         *lpszBuf = '\0';
         return;     // unknown line type
   }
}


/* Line_GetOutlineData
 * -------------------
 *
 * Return the CF_OUTLINE format representation of the Line.
 */
BOOL Line_GetOutlineData(LPLINE lpLine, LPTEXTLINE lpBuf)
{
   switch (lpLine->m_lineType) {
      case TEXTLINETYPE:
         return TextLine_GetOutlineData((LPTEXTLINE)lpLine, lpBuf);

#if defined( OLE_CNTR )
      case CONTAINERLINETYPE:
         return ContainerLine_GetOutlineData(
               (LPCONTAINERLINE)lpLine,
               lpBuf
         );
#endif

      default:
         return FALSE;       // unknown line type
   }
}


/* Line_CalcTabWidthInHimetric
 * ---------------------------
 *
 *      Recalculate the width for the line's current tab level
 */
static int Line_CalcTabWidthInHimetric(LPLINE lpLine, HDC hDC)
{
   int nTabWidthInHimetric;

   nTabWidthInHimetric=lpLine->m_nTabLevel * TABWIDTH;
   return nTabWidthInHimetric;
}


/* Line_Indent
 * -----------
 *
 *      Increment the tab level for the line
 */
void Line_Indent(LPLINE lpLine, HDC hDC)
{
   lpLine->m_nTabLevel++;
   lpLine->m_nTabWidthInHimetric = Line_CalcTabWidthInHimetric(lpLine, hDC);

#if defined( INPLACE_CNTR )
   if (Line_GetLineType(lpLine) == CONTAINERLINETYPE)
      ContainerLine_UpdateInPlaceObjectRects((LPCONTAINERLINE)lpLine, NULL);
#endif
}


/* Line_Unindent
 * -------------
 *
 * Decrement the tab level for the line
 */
void Line_Unindent(LPLINE lpLine, HDC hDC)
{
   if(lpLine->m_nTabLevel > 0) {
      lpLine->m_nTabLevel--;
      lpLine->m_nTabWidthInHimetric = Line_CalcTabWidthInHimetric(lpLine,
hDC);
   }

#if defined( INPLACE_CNTR )
   if (Line_GetLineType(lpLine) == CONTAINERLINETYPE)
      ContainerLine_UpdateInPlaceObjectRects((LPCONTAINERLINE)lpLine, NULL);
#endif
}


/* Line_GetTotalWidthInHimetric
 * ----------------------------
 *
 *      Calculate the total width of the line
 */
UINT Line_GetTotalWidthInHimetric(LPLINE lpLine)
{
   return lpLine->m_nWidthInHimetric + lpLine->m_nTabWidthInHimetric;
```

```
}


/* Line_SetWidthInHimetric
 * -----------------------
 *
 *      Set the width of the line
 */
void Line_SetWidthInHimetric(LPLINE lpLine, int nWidth)
{
   if (!lpLine)
      return;

   lpLine->m_nWidthInHimetric = nWidth;
}


/* Line_GetWidthInHimetric
 * -----------------------
 *
 *      Return the width of the line
 */
UINT Line_GetWidthInHimetric(LPLINE lpLine)
{
   if (!lpLine)
      return 0;

   return lpLine->m_nWidthInHimetric;
}





/* Line_GetTabLevel
 * ----------------
 *
 * Return the tab level of a line object.
 */
UINT Line_GetTabLevel(LPLINE lpLine)
{
   return lpLine->m_nTabLevel;
}


/* Line_DrawToScreen
 * -----------------
 *
 *      Draw the item in the owner-draw listbox
 */
void Line_DrawToScreen(
      LPLINE      lpLine,
      HDC         hDC,
      LPRECT      lprcPix,
      UINT        itemAction,
      UINT        itemState,
      LPRECT      lprcDevice
)
{
   if (!lpLine || !hDC || !lprcPix || !lprcDevice)
      return;

   /* Draw a list box item in its normal drawing action.
    * Then check if it is selected or has the focus state, and call
    * functions to handle drawing for these states if necessary.
    */
   if(itemAction & (ODA_SELECT | ODA_DRAWENTIRE)) {
      HFONT hfontOld;
      int nMapModeOld;
      RECT rcWindowOld;
      RECT rcViewportOld;
      RECT rcLogical;

      // NOTE: we have to set the device context to HIMETRIC in order
      // draw the line; however, we have to restore the HDC before
      // we draw focus or dragfeedback...

      rcLogical.left = 0;
      rcLogical.bottom = 0;
      rcLogical.right = lpLine->m_nWidthInHimetric;
      rcLogical.top = lpLine->m_nHeightInHimetric;

      {
         HBRUSH hbr;
         RECT    rcDraw;

         lpLine->m_fSelected = (BOOL)(itemState & ODS_SELECTED);

         if (ODS_SELECTED & itemState) {
            /*Get proper txt colors */
            hbr = CreateSolidBrush(GetSysColor(COLOR_HIGHLIGHT));
         }
         else {
            hbr = CreateSolidBrush(GetSysColor(COLOR_WINDOW));
         }

         rcDraw = *lprcPix;
         rcDraw.right = lprcDevice->left;
         FillRect(hDC, lprcPix, hbr);

         rcDraw = *lprcPix;
         rcDraw.left = lprcDevice->right;
         FillRect(hDC, lprcPix, hbr);

         DeleteObject(hbr);
      }

      nMapModeOld=SetDCToAnisotropic(hDC, lprcDevice, &rcLogical,
                     (LPRECT)&rcWindowOld, (LPRECT)&rcViewportOld);
      // Set the default font size, and font face name
      hfontOld = SelectObject(hDC, OutlineApp_GetActiveFont(g_lpApp));

      Line_Draw(lpLine, hDC, &rcLogical, NULL, (ODS_SELECTED & itemState));

      SelectObject(hDC, hfontOld);

      ResetOrigDC(hDC, nMapModeOld, (LPRECT)&rcWindowOld,
            (LPRECT)&rcViewportOld);

#if defined( OLE_CNTR )
      if ((itemState & ODS_SELECTED) &&
         (Line_GetLineType(lpLine)==CONTAINERLINETYPE))
         ContainerLine_DrawSelHilight(
               (LPCONTAINERLINE)lpLine,
               hDC,
               lprcPix,
               ODA_SELECT,
               ODS_SELECTED
         );
#endif

   }

   /* If a list box item just gained or lost the focus,
   * call function (which could check if ODS_FOCUS bit is set)
   * and draws item in focus or non-focus state.
   */
   if(itemAction & ODA_FOCUS )
      Line_DrawFocusRect(lpLine, hDC, lprcPix, itemAction, itemState);


#if defined( OLE_CNTR )
   if (Line_GetLineType(lpLine) == CONTAINERLINETYPE) {
      LPCONTAINERLINE lpContainerLine = (LPCONTAINERLINE)lpLine;
      LPCONTAINERDOC lpDoc = lpContainerLine->m_lpDoc;
      BOOL fIsLink;
      RECT rcObj;

      if (ContainerDoc_GetShowObjectFlag(lpDoc)) {
         ContainerLine_GetOleObjectRectInPixels(lpContainerLine, &rcObj);
         fIsLink = ContainerLine_IsOleLink(lpContainerLine);
         OleUIShowObject(&rcObj, hDC, fIsLink);
      }
   }
#endif

#if defined( USE_DRAGDROP )
   if (lpLine->m_fDragOverLine)
      Line_DrawDragFeedback(lpLine, hDC, lprcPix, itemState );
#endif

}
```

```
/* Line_Draw
 * ---------
 *
 *  Draw a line on a DC.
 *
 * Parameters:
 *      hDC     - DC to which the line will be drawn
 *      lpRect  - the object rect in logical coordinates
 */
void Line_Draw(
      LPLINE      lpLine,
      HDC         hDC,
      LPRECT      lpRect,
      LPRECT      lpRectWBounds,
      BOOL        fHighlight
)
{
   switch (lpLine->m_lineType) {
      case TEXTLINETYPE:
         TextLine_Draw(
             (LPTEXTLINE)lpLine, hDC, lpRect,lpRectWBounds,fHighlight);
         break;

#if defined( OLE_CNTR )
      case CONTAINERLINETYPE:
         ContainerLine_Draw(
             (LPCONTAINERLINE)lpLine,hDC,lpRect,lpRectWBounds,fHighlight);
         break;
#endif

      default:
         return;     // unknown line type
   }
   return;
}


/* Line_DrawSelHilight
 * -------------------
 *
 *      Handles selection of list box item
 */
void Line_DrawSelHilight(LPLINE lpLine, HDC hDC, LPRECT lpRect, UINT
itemAction, UINT itemState)
{
   switch (lpLine->m_lineType) {
      case TEXTLINETYPE:
         TextLine_DrawSelHilight((LPTEXTLINE)lpLine, hDC, lpRect,
            itemAction, itemState);
         break;

#if defined( OLE_CNTR )
      case CONTAINERLINETYPE:
         ContainerLine_DrawSelHilight((LPCONTAINERLINE)lpLine, hDC, lpRect,
            itemAction, itemState);
         break;
#endif

      default:
         return;     // unknown line type
   }
   return;

}

/* Line_DrawFocusRect
 * ------------------
 *
 *      Handles focus state of list box item
 */
void Line_DrawFocusRect(LPLINE lpLine, HDC hDC, LPRECT lpRect, UINT
itemAction, UINT itemState)
{
   if(lpLine)
      DrawFocusRect(hDC, lpRect);
}

#if defined( USE_DRAGDROP )

/* Line_DrawDragFeedback
 * ---------------------
 *
 *      Handles focus state of list box item
 */
void Line_DrawDragFeedback(LPLINE lpLine, HDC hDC, LPRECT lpRect, UINT
itemState )
{
   if(lpLine)
      DrawFocusRect(hDC, lpRect);
}

#endif  // USE_DRAGDROP


/* Line_GetHeightInHimetric
 * ------------------------
 *
 *      Return the height of the item in HIMETRIC units
 */
UINT Line_GetHeightInHimetric(LPLINE lpLine)
{
   if (!lpLine)
      return 0;

   return (UINT)lpLine->m_nHeightInHimetric;
}


/* Line_SetHeightInHimetric
 * ------------------------
 *
 *      Set the height of the item in HIMETRIC units.
 */
void Line_SetHeightInHimetric(LPLINE lpLine, int nHeight)
{
   if (!lpLine)
      return;

   switch (lpLine->m_lineType) {
      case TEXTLINETYPE:
         TextLine_SetHeightInHimetric((LPTEXTLINE)lpLine, nHeight);
         break;

#if defined( OLE_CNTR )
      case CONTAINERLINETYPE:
         ContainerLine_SetHeightInHimetric((LPCONTAINERLINE)lpLine,
               nHeight);
         break;
#endif

   }
}


/* Line_Delete
 * -----------
 *
 *      Delete the Line structure
 */
void Line_Delete(LPLINE lpLine)
{
   switch (lpLine->m_lineType) {
      case TEXTLINETYPE:
         TextLine_Delete((LPTEXTLINE)lpLine);
         break;

#if defined( OLE_CNTR )
      case CONTAINERLINETYPE:
         ContainerLine_Delete((LPCONTAINERLINE)lpLine);
         break;
#endif

      default:
         break;      // unknown line type
   }
}


/* Line_CopyToDoc
 * --------------
 *
 *      Copy a line to another Document (usually ClipboardDoc)
 */
BOOL Line_CopyToDoc(LPLINE lpSrcLine, LPOUTLINEDOC lpDestDoc, int nIndex)
{
   switch (lpSrcLine->m_lineType) {
      case TEXTLINETYPE:
         return TextLine_CopyToDoc((LPTEXTLINE)lpSrcLine,lpDestDoc,nIndex);
         break;

#if defined( OLE_CNTR )
      case CONTAINERLINETYPE:
         return ContainerLine_CopyToDoc(
               (LPCONTAINERLINE)lpSrcLine,
               lpDestDoc,
               nIndex
         );
         break;
#endif

      default:
         return FALSE;       // unknown line type
   }
}


/* Line_SaveToStg
 * --------------
 *
 *      Save a single line object to a storage
 *
 *      Return TRUE if successful, FALSE otherwise
 */
BOOL Line_SaveToStg(LPLINE lpLine, UINT uFormat, LPSTORAGE lpSrcStg,
LPSTORAGE lpDestStg, LPSTREAM lpLLStm, BOOL fRemember)
{
   LINERECORD_ONDISK lineRecord;
   ULONG nWritten;
   HRESULT hrErr;
   BOOL fStatus;
   LARGE_INTEGER dlibSavePos;
   LARGE_INTEGER dlibZeroOffset;
   LISet32( dlibZeroOffset, 0 );

   /* save seek position before line record is written in case of error */
   hrErr = lpLLStm->lpVtbl->Seek(
         lpLLStm,
         dlibZeroOffset,
         STREAM_SEEK_CUR,
         (ULARGE_INTEGER FAR*)&dlibSavePos
   );
   if (hrErr != NOERROR) return FALSE;

#if defined( OLE_CNTR )
   if (lpLine->m_lineType == CONTAINERLINETYPE) {
      /* OLE2NOTE: asking an OLE object to save may cause the
      **    object to send an OnViewChange notification if there are any
      **    outstanding changes to the object. this is particularly true
      **    for objects with coarse update granularity like OLE 1.0
      **    objects. if an OnViewChange notification is received then the
      **    object's presentation cache will be updated BEFORE it is
      **    saved. It is important that the extents stored as part of
      **    the ContainerLine/Line record associated with the OLE object
      **    are updated before the Line data is saved to the storage. it
      **    is important that this extent information matches the data
      **    saved with the OLE object. the Line extent information is
      **    updated in the IAdviseSink::OnViewChange method implementation.
      */
      // only save the OLE object if format is compatible.
      if (uFormat != ((LPCONTAINERAPP)g_lpApp)->m_cfCntrOutl)
         goto error;

      fStatus = ContainerLine_SaveOleObjectToStg(
            (LPCONTAINERLINE)lpLine,
            lpSrcStg,
            lpDestStg,
            fRemember
      );
      if (! fStatus) goto error;
   }
#endif

   // compilers should handle alignment correctly
   lineRecord.m_lineType = (USHORT)lpLine->m_lineType;
   lineRecord.m_nTabLevel = (USHORT)lpLine->m_nTabLevel;
   lineRecord.m_nTabWidthInHimetric = (USHORT)lpLine->m_nTabWidthInHimetric;
   lineRecord.m_nWidthInHimetric = (USHORT)lpLine->m_nWidthInHimetric;
   lineRecord.m_nHeightInHimetric = (USHORT)lpLine->m_nHeightInHimetric;
   lineRecord.m_reserved = 0;

   /* write line record header */
   hrErr = lpLLStm->lpVtbl->Write(
         lpLLStm,
         (LPVOID)&lineRecord,
         sizeof(lineRecord),
         &nWritten
   );

   if (hrErr != NOERROR) {
      OleDbgOutHResult("Write Line header returned", hrErr);
      goto error;
    }

   switch (lpLine->m_lineType) {
      case TEXTLINETYPE:
         fStatus = TextLine_SaveToStm((LPTEXTLINE)lpLine, lpLLStm);
         if (! fStatus) goto error;
         break;

#if defined( OLE_CNTR )
      case CONTAINERLINETYPE:
         fStatus=ContainerLine_SaveToStm((LPCONTAINERLINE)lpLine,lpLLStm);
         if (! fStatus) goto error;
         break;
#endif
```

```
      default:
         goto error;       // unknown line type
   }

   return TRUE;

error:

   /* retore seek position prior to writing Line record */
   lpLLStm->lpVtbl->Seek(
         lpLLStm,
         dlibSavePos,
         STREAM_SEEK_SET,
         NULL
   );

   return FALSE;
}


/* Line_LoadFromStg
 * ----------------
 *
 *      Load a single line object from storage
 */
LPLINE Line_LoadFromStg(LPSTORAGE lpSrcStg, LPSTREAM lpLLStm, LPOUTLINEDOC
lpDestDoc)
{
   LINERECORD_ONDISK lineRecord;
   LPLINE lpLine = NULL;
   ULONG nRead;
   HRESULT hrErr;

   /* read line record header */
   hrErr = lpLLStm->lpVtbl->Read(
         lpLLStm,
         (LPVOID)&lineRecord,
         sizeof(lineRecord),
         &nRead
   );

   if (hrErr != NOERROR) {
      OleDbgOutHResult("Read Line header returned", hrErr);
      return NULL;
    }

   switch ((LINETYPE)lineRecord.m_lineType) {
      case TEXTLINETYPE:
         lpLine = TextLine_LoadFromStg(lpSrcStg, lpLLStm, lpDestDoc);
         break;

#if defined( OLE_CNTR )
      case CONTAINERLINETYPE:
         lpLine = ContainerLine_LoadFromStg(lpSrcStg, lpLLStm, lpDestDoc);
         break;
#endif

      default:
         return NULL;        // unknown line type
   }

   lpLine->m_lineType = (LINETYPE) lineRecord.m_lineType;
   lpLine->m_nTabLevel = (UINT) lineRecord.m_nTabLevel;
   lpLine->m_nTabWidthInHimetric = (UINT) lineRecord.m_nTabWidthInHimetric;
   lpLine->m_nWidthInHimetric = (UINT) lineRecord.m_nWidthInHimetric;
   lpLine->m_nHeightInHimetric = (UINT) lineRecord.m_nHeightInHimetric;

   return lpLine;
}


/* Line_IsSelected
 * ---------------
 *
 *      Return the selection state of the line
 */
BOOL Line_IsSelected(LPLINE lpLine)
{
   if (!lpLine)
      return FALSE;

   return lpLine->m_fSelected;
}
```

OUTLLIST.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    outldata.c
**
**    This file contains LineList and NameTable functions
**    and related support functions.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/


#include "outline.h"

OLEDBGDATA

extern LPOUTLINEAPP g_lpApp;

OLECHAR ErrMsgListBox[] = OLESTR("Can't create ListBox!");

static int g_iMapMode;

/* LineList_Init
 * -------------
 *
 *      Create and Initialize the LineList (owner-drawn listbox)
 */
BOOL LineList_Init(LPLINELIST lpLL, LPOUTLINEDOC lpOutlineDoc)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;

#if defined( INPLACE_CNTR )
   lpLL->m_hWndListBox = CreateWindow(
               "listbox",              /* Window class name           */
               NULL,                   /* Window's title              */

               /* OLE2NOTE: an in-place contanier MUST use
               **    WS_CLIPCHILDREN window style for the window
               **    that it uses as the parent for the server's
               **    in-place active window so that its
               **    painting does NOT interfere with the painting
               **    of the server's in-place active child window.
               */

               WS_CLIPCHILDREN |
               WS_CHILDWINDOW |
               WS_VISIBLE |
               WS_VSCROLL |
               WS_HSCROLL |
               LBS_EXTENDEDSEL |
               LBS_NOTIFY |
               LBS_OWNERDRAWVARIABLE |
               LBS_NOINTEGRALHEIGHT |
               LBS_USETABSTOPS,
               0, 0,                   /* Use default X, Y            */
               0, 0,                   /* Use default X, Y            */
               lpOutlineDoc->m_hWndDoc,/* Parent window's handle      */
               (HMENU)IDC_LINELIST,    /* Child Window ID             */
               lpOutlineApp->m_hInst,  /* Instance of window          */
               NULL);                  /* Create struct for WM_CREATE */
#else
   lpLL->m_hWndListBox = CreateWindow(
               "listbox",              /* Window class name           */
               NULL,                   /* Window's title              */
               WS_CHILDWINDOW |
               WS_VISIBLE |
               WS_VSCROLL |
               WS_HSCROLL |
               LBS_EXTENDEDSEL |
               LBS_NOTIFY |
               LBS_OWNERDRAWVARIABLE |
               LBS_NOINTEGRALHEIGHT |
               LBS_USETABSTOPS,
               0, 0,                   /* Use default X, Y            */
               0, 0,                   /* Use default X, Y            */
               lpOutlineDoc->m_hWndDoc,/* Parent window's handle      */
               (HMENU)IDC_LINELIST,    /* Child Window ID             */
               lpOutlineApp->m_hInst,  /* Instance of window          */
               NULL);                  /* Create struct for WM_CREATE */

#endif


   if(! lpLL->m_hWndListBox) {
      OutlineApp_ErrorMessage(g_lpApp, ErrMsgListBox);
      return FALSE;
   }

   lpOutlineApp->m_ListBoxWndProc =
         (FARPROC) GetWindowLong ( lpLL->m_hWndListBox, GWL_WNDPROC );
   SetWindowLong (lpLL->m_hWndListBox, GWL_WNDPROC, (LONG) LineListWndProc);

#if defined ( USE_DRAGDROP )
   /* m_iDragOverLine saves index of line that has drag/drop target
   **    feedback. we currently use our focus rectangle feedback for
   **    this. it would be better to have a different visual feedback
   **    for potential target of the pending drop.
   */
   lpLL->m_iDragOverLine = -1;
#endif

   lpLL->m_nNumLines = 0;
   lpLL->m_nMaxLineWidthInHimetric = 0;
   lpLL->m_lpDoc = lpOutlineDoc;
   _fmemset(&lpLL->m_lrSaveSel, 0, sizeof(LINERANGE));
   return TRUE;
}


/* LineList_Destroy
 * ----------------
 *
 *      Clear (delete) all Line objects from the list and free supporting
 *      memory (ListBox Window) used by the LineList object itself.
 */
void LineList_Destroy(LPLINELIST lpLL)
{
   int i;
   int linesTotal = lpLL->m_nNumLines;

   // Delete all Line objects
   for (i = 0; i < linesTotal; i++)
      LineList_DeleteLine(lpLL, 0);   // NOTE: always delete line 0

   // Remove all Lines from the ListBox
   SendMessage(lpLL->m_hWndListBox,LB_RESETCONTENT,0,0L);

   lpLL->m_nNumLines=0;
   DestroyWindow(lpLL->m_hWndListBox);
   lpLL->m_hWndListBox = NULL;
}


/* LineList_AddLine
 * ----------------
 *
 *      Add one line to the list box. The line is added following the
 * line with index "nIndex". If nIndex is larger than the number of lines
 * in the ListBox, then the line is appended to the end. The selection
 * is set to the newly added line.
 */
void LineList_AddLine(LPLINELIST lpLL, LPLINE lpLine, int nIndex)
{
   int nAddIndex = (lpLL->m_nNumLines == 0 ?
         0 :
         (nIndex >= lpLL->m_nNumLines ? lpLL->m_nNumLines : nIndex+1));
   LINERANGE lrSel;

#if defined( USE_HEADING )
   int nHeight = Line_GetHeightInHimetric(lpLine);

   nHeight = XformHeightInHimetricToPixels(NULL, nHeight);

   // Add a dummy string to the row heading
   Heading_RH_SendMessage(OutlineDoc_GetHeading(lpLL->m_lpDoc),
         LB_INSERTSTRING, (WPARAM)nAddIndex, MAKELPARAM(nHeight, 0));
#endif
```

```
   lrSel.m_nStartLine = nAddIndex;
   lrSel.m_nEndLine =   nAddIndex;

   if (!lpLine) {
      OutlineApp_ErrorMessage(g_lpApp, OLESTR("Could not create line."));
      return;
   }

   SendMessage(lpLL->m_hWndListBox, LB_INSERTSTRING, (WPARAM)nAddIndex,
         (DWORD)lpLine);

   LineList_SetMaxLineWidthInHimetric(
         lpLL,
         Line_GetTotalWidthInHimetric(lpLine)
   );

   lpLL->m_nNumLines++;

   LineList_SetSel(lpLL, &lrSel);
}


/* LineList_DeleteLine
 * -------------------
 *
 *      Delete one line from listbox and memory
 */
void LineList_DeleteLine(LPLINELIST lpLL, int nIndex)
{
   LPLINE lpLine = LineList_GetLine(lpLL, nIndex);
   BOOL fResetSel;

   fResetSel = (BOOL)SendMessage(lpLL->m_hWndListBox, LB_GETSEL,
(WPARAM)nIndex, 0L);

   if (lpLine)
      Line_Delete(lpLine);    // free memory of Line

   // Remove the Line from the ListBox
   SendMessage(lpLL->m_hWndListBox, LB_DELETESTRING, (WPARAM)nIndex, 0L);
   lpLL->m_nNumLines--;

   if (fResetSel) {
      if (nIndex > 0) {
#if defined( WIN32 )
         SendMessage(
               lpLL->m_hWndListBox,
               LB_SETSEL,
               (WPARAM)TRUE,
               (LPARAM)nIndex-1
         );
#else
         SendMessage(
               lpLL->m_hWndListBox,
               LB_SETSEL,
                (WPARAM)TRUE,
                MAKELPARAM(nIndex-1,0)
         );
#endif
      } else {
         if (lpLL->m_nNumLines > 0) {
#if defined( WIN32 )
            SendMessage(
                  lpLL->m_hWndListBox,
                  LB_SETSEL,
                  (WPARAM)TRUE,
                  (LPARAM)0
            );
#else
            SendMessage(
                  lpLL->m_hWndListBox,
                  LB_SETSEL,
                  (WPARAM)TRUE,
                  MAKELPARAM(0,0)
            );
#endif
         }
      }
   }

#if defined( USE_HEADING )
   // Remove the dummy string from the row heading
   Heading_RH_SendMessage(OutlineDoc_GetHeading(lpLL->m_lpDoc),
         LB_DELETESTRING, (WPARAM)nIndex, 0L);
#endif

}


/* LineList_ReplaceLine
 * --------------------
 *
 *      Replace the line at a given index in the list box with a new
 * line.
 */
void LineList_ReplaceLine(LPLINELIST lpLL, LPLINE lpLine, int nIndex)
{
   LPLINE lpOldLine = LineList_GetLine(lpLL, nIndex);

   if (lpOldLine)
      Line_Delete(lpOldLine);    // free memory of Line
   else
      return;     // if no previous line then invalid index

   SendMessage(
         lpLL->m_hWndListBox,
         LB_SETITEMDATA,
         (WPARAM)nIndex,
         (LPARAM)lpLine
   );
```

```
}


/* LineList_GetLineIndex
 * ---------------------
 *
 *      Return the index of the Line given a pointer to the line.
 *      Return -1 if the line is not found.
 */
int LineList_GetLineIndex(LPLINELIST lpLL, LPLINE lpLine)
{
   LRESULT lRet;

   if (! lpLine) return -1;

   lRet = SendMessage(
         lpLL->m_hWndListBox,
         LB_FINDSTRING,
         (WPARAM)-1,
         (LPARAM)(LPCSTR)lpLine
      );

   return ((lRet == LB_ERR) ? -1 : (int)lRet);
}


/* LineList_GetLine
 * ----------------
 *
 *      Retrieve the pointer to the Line given its index in the LineList
 */
LPLINE LineList_GetLine(LPLINELIST lpLL, int nIndex)
{
   DWORD dWord;
   LRESULT lRet;

   if (lpLL->m_nNumLines == 0 || nIndex > lpLL->m_nNumLines || nIndex < 0)
      return NULL;

   lRet = SendMessage(
         lpLL->m_hWndListBox,LB_GETTEXT,nIndex,(LPARAM)(LPCSTR)&dWord);

   return ((lRet == LB_ERR || lRet == 0) ? NULL : (LPLINE)dWord);
}


/* LineList_SetFocusLine
 * ---------------------
 *
 */

void LineList_SetFocusLine ( LPLINELIST lpLL, WORD wIndex )
{

   SendMessage(lpLL->m_hWndListBox, LB_SETCARETINDEX, (WPARAM)wIndex, 0L );
```

```
}


/* LineList_GetLineRect
 * --------------------
 *
 * Retrieve the rectangle of a Line given its index in the LineList
 */
BOOL LineList_GetLineRect(LPLINELIST lpLL, int nIndex, LPRECT lpRect)
{
   DWORD iReturn = (DWORD)LB_ERR;

   if ( !(lpLL->m_nNumLines == 0 || nIndex > lpLL->m_nNumLines || nIndex <
0) )
      iReturn = SendMessage(lpLL->m_hWndListBox,LB_GETITEMRECT,nIndex,
(LPARAM)lpRect);

   return (iReturn == LB_ERR ? FALSE : TRUE );
}


/* LineList_GetFocusLineIndex
 * --------------------------
 *
 * Get the index of the line that currently has focus (the active line).
 */
int LineList_GetFocusLineIndex(LPLINELIST lpLL)
{
   return (int)SendMessage(lpLL->m_hWndListBox,LB_GETCARETINDEX,0,0L);
}


/* LineList_GetCount
 * -----------------
 *
 *      Return number of line objects
 */
int LineList_GetCount(LPLINELIST lpLL)
{
   if (lpLL)
      return lpLL->m_nNumLines;
   else {
      OleDbgAssert(lpLL!=NULL);
      return 0;
   }
}


/* LineList_SetMaxLineWidthInHimetric
 * ----------------------------------
 *
 *  Adjust the maximum line width for the listbox. The max line width is
 *  used to determine if a horizontal scroll bar is needed.
 *
 *  Parameters:
 *      nWidthInHimetric - if +ve, width of an additional line
 *                       - if -ve, reset Max to be the value
 *
 *  Returns:
 *      TRUE is max line width of LineList changed
 *      FALSE if no change
 */
BOOL LineList_SetMaxLineWidthInHimetric(LPLINELIST lpLL, int
nWidthInHimetric)
{
   int nWidthInPix;
   BOOL fSizeChanged = FALSE;
   LPSCALEFACTOR lpscale;

   if (!lpLL)
      return FALSE;

   lpscale = OutlineDoc_GetScaleFactor(lpLL->m_lpDoc);

   if (nWidthInHimetric < 0) {
      lpLL->m_nMaxLineWidthInHimetric = -1;
      nWidthInHimetric *= -1;
   }

   if (nWidthInHimetric > lpLL->m_nMaxLineWidthInHimetric) {
      lpLL->m_nMaxLineWidthInHimetric = nWidthInHimetric;
      nWidthInPix = XformWidthInHimetricToPixels(NULL, nWidthInHimetric +
            LOWORD(OutlineDoc_GetMargin(lpLL->m_lpDoc)) +
            HIWORD(OutlineDoc_GetMargin(lpLL->m_lpDoc)));

      nWidthInPix = (int)(nWidthInPix * lpscale->dwSxN / lpscale->dwSxD);
      SendMessage(
            lpLL->m_hWndListBox,
            LB_SETHORIZONTALEXTENT,
            nWidthInPix,
            0L
      );
      fSizeChanged = TRUE;

#if defined( USE_HEADING )
      Heading_CH_SetHorizontalExtent(
            OutlineDoc_GetHeading(lpLL->m_lpDoc), lpLL->m_hWndListBox);
#endif

   }
   return fSizeChanged;
}


/* LineList_GetMaxLineWidthInHimetric
 * ----------------------------------
 *
 *      Return the width of the widest line
 */
```

```
int LineList_GetMaxLineWidthInHimetric(LPLINELIST lpLL)
{
   return lpLL->m_nMaxLineWidthInHimetric;
}


/* LineList_RecalcMaxLineWidthInHimetric
 * -------------------------------------
 *
 *  Recalculate the maximum line width in the entire list.
 *
 *  Parameters:
 *      nWidthInHimetric should be set to the width of line being removed.
 *      nWidthInHimetric == 0 forces list to recalculate in all cases.
 *      nWidthInHimetric == current max width => forces recalc.
 *
 *  Returns:
 *      TRUE is max line width of LineList changed
 *      FALSE if no change
 */
BOOL LineList_RecalcMaxLineWidthInHimetric(
      LPLINELIST          lpLL,
   int                 nWidthInHimetric
)
{
   int i;
   LPLINE lpLine;
   BOOL fSizeChanged = FALSE;
   int nOrgMaxLineWidthInHimetric = lpLL->m_nMaxLineWidthInHimetric;

   if (nWidthInHimetric == 0 ||
      nWidthInHimetric == lpLL->m_nMaxLineWidthInHimetric) {

      lpLL->m_nMaxLineWidthInHimetric = -1;

      LineList_SetMaxLineWidthInHimetric(lpLL, 0);

      for(i = 0; i < lpLL->m_nNumLines; i++) {
         lpLine=LineList_GetLine(lpLL, i);
         LineList_SetMaxLineWidthInHimetric(
               lpLL,
               Line_GetTotalWidthInHimetric(lpLine)
         );
      }
   }

   if (nOrgMaxLineWidthInHimetric != lpLL->m_nMaxLineWidthInHimetric)
      fSizeChanged = TRUE;

   return fSizeChanged;
}


/* LineList_CalcSelExtentInHimetric
 * --------------------------------
 *
 *      Calculate the extents (widht and height) of a selection of lines.
 *
 * if lplrSel == NULL, calculate extent of all lines.
 */
void LineList_CalcSelExtentInHimetric(
      LPLINELIST          lpLL,
      LPLINERANGE         lplrSel,
      LPSIZEL             lpsizel
)
{
   int i;
   int nEndLine;
   int nStartLine;
   LPLINE lpLine;
   long lWidth;

   if (lplrSel) {
      nEndLine = lplrSel->m_nEndLine;
      nStartLine = lplrSel->m_nStartLine;
   } else {
      nEndLine = LineList_GetCount(lpLL) - 1;
      nStartLine = 0;
   }

   lpsizel->cx = 0;
   lpsizel->cy = 0;

   for(i = nStartLine; i <= nEndLine; i++) {
      lpLine=LineList_GetLine(lpLL,i);
      if (lpLine) {
         lWidth = (long)Line_GetTotalWidthInHimetric(lpLine);
         lpsizel->cx = max(lpsizel->cx, lWidth);
         lpsizel->cy += lpLine->m_nHeightInHimetric;
      }
   }
}


/* LineList_GetWindow
 * ------------------
 *
 * Return handle of list box
 */
HWND LineList_GetWindow(LPLINELIST lpLL)
{
   return lpLL->m_hWndListBox;
}


/* LineList_GetDC
 * --------------
 *
 * Return DC handle of list box
 */
```

```
HDC LineList_GetDC(LPLINELIST lpLL)
{
   HFONT hfontOld;
   HDC hDC = GetDC(lpLL->m_hWndListBox);
   int     iXppli;     //* pixels per logical inch along width
   int     iYppli;     //* pixels per logical inch along height
   SIZE    size;

   // Setup a mapping mode for the DC which maps physical pixel
   // coordinates to HIMETRIC units. The standard MM_HIMETRIC mapping
   // mode does not work correctly because it does not take into
   // account that a logical inch on the display screen is drawn
   // physically larger than 1 inch. We will setup an anisotropic
   // mapping mode which will perform the transformation properly.

   g_iMapMode = SetMapMode(hDC, MM_ANISOTROPIC);
   iXppli = GetDeviceCaps (hDC, LOGPIXELSX);
   iYppli = GetDeviceCaps (hDC, LOGPIXELSY);
   SetViewportExtEx(hDC, iXppli, iYppli, &size);
   SetWindowExtEx(hDC, HIMETRIC_PER_INCH, HIMETRIC_PER_INCH, &size);

   // Set the default font size, and font face name
   hfontOld = SelectObject(hDC, OutlineApp_GetActiveFont(g_lpApp));

   return hDC;
}


/* LineList_ReleaseDC
 * ------------------
 *
 *      Release DC of list box returned from previous LineList_GetDC call.
 */
void LineList_ReleaseDC(LPLINELIST lpLL, HDC hDC)
{
   SetMapMode(hDC, g_iMapMode);
   ReleaseDC(lpLL->m_hWndListBox, hDC);
}


/* LineList_SetLineHeight
 * ----------------------
 *
 *      Set the height of a line in the LineList list box
 */
void LineList_SetLineHeight(LPLINELIST lpLL,int nIndex,int
nHeightInHimetric)
{
   LPARAM          lParam;
   LPOUTLINEDOC    lpDoc;
   LPSCALEFACTOR   lpscale;
   UINT            uHeightInPix;
   LPHEADING       lphead;

   if (!lpLL)
```

```
      return;

   lpDoc = lpLL->m_lpDoc;
   lphead = OutlineDoc_GetHeading(lpDoc);
   lpscale = OutlineDoc_GetScaleFactor(lpDoc);

   uHeightInPix = XformHeightInHimetricToPixels(NULL, nHeightInHimetric);

   Heading_RH_SendMessage(lphead, LB_SETITEMDATA, (WPARAM)nIndex,
         MAKELPARAM(uHeightInPix, 0));

   uHeightInPix = (UINT)(uHeightInPix * lpscale->dwSyN / lpscale->dwSyD);

   if (uHeightInPix > LISTBOX_HEIGHT_LIMIT)
      uHeightInPix = LISTBOX_HEIGHT_LIMIT;


   lParam = MAKELPARAM(uHeightInPix, 0);
   SendMessage(lpLL->m_hWndListBox,LB_SETITEMHEIGHT,(WPARAM)nIndex, lParam);
   Heading_RH_SendMessage(lphead, LB_SETITEMHEIGHT, (WPARAM)nIndex, lParam);
   Heading_RH_ForceRedraw(lphead, TRUE);
}


/* LineList_ReScale
 * ----------------
 *
 *      Re-scale the LineList list box
 */
void LineList_ReScale(LPLINELIST lpLL, LPSCALEFACTOR lpscale)
{
   int nIndex;
   LPLINE lpLine;
   UINT uWidthInHim;

   if (!lpLL)
      return;

   for (nIndex = 0; nIndex < lpLL->m_nNumLines; nIndex++) {
      lpLine = LineList_GetLine(lpLL, nIndex);
      if (lpLine) {
         LineList_SetLineHeight(
               lpLL,
               nIndex,
               Line_GetHeightInHimetric(lpLine)
         );
      }
   }

   uWidthInHim = LineList_GetMaxLineWidthInHimetric(lpLL);
   LineList_SetMaxLineWidthInHimetric(lpLL, -(int)uWidthInHim);
}

/* LineList_SetSel
 * ---------------
 *
 *      Set the selection in list box
 */
void LineList_SetSel(LPLINELIST lpLL, LPLINERANGE lplrSel)
{
   DWORD dwSel;

   if (lpLL->m_nNumLines <= 0 || lplrSel->m_nStartLine < 0)
      return;     // no lines in list; can't set a selection

   dwSel = MAKELPARAM(lplrSel->m_nStartLine, lplrSel->m_nEndLine);

   lpLL->m_lrSaveSel = *lplrSel;

   /* remove previous selection */
#if defined( WIN32 )
   SendMessage(
         lpLL->m_hWndListBox,
         LB_SETSEL,
         (WPARAM)FALSE,
         (LPARAM)-1
   );
#else
   SendMessage(
         lpLL->m_hWndListBox,
         LB_SETSEL,
         (WPARAM)FALSE,
         MAKELPARAM(-1,0)
   );
#endif

   /* mark selection */
   SendMessage(lpLL->m_hWndListBox,LB_SELITEMRANGE, (WPARAM)TRUE,
(LPARAM)dwSel);
   /* set focus line (caret) */
   LineList_SetFocusLine ( lpLL, (WORD)lplrSel->m_nStartLine );

}


/* LineList_GetSel
 * ---------------
 *
 * Get the selection in list box.
 *
 * Returns the count of items selected
 */
int LineList_GetSel(LPLINELIST lpLL, LPLINERANGE lplrSel)
{
   int nNumSel=(int)SendMessage(lpLL->m_hWndListBox,LB_GETSELCOUNT,0,0L);

   if (nNumSel) {
      SendMessage(lpLL->m_hWndListBox,LB_GETSELITEMS,
         (WPARAM)1,(LPARAM)(int FAR*)&(lplrSel->m_nStartLine));
      lplrSel->m_nEndLine = lplrSel->m_nStartLine + nNumSel - 1;
   } else {
      _fmemset(lplrSel, 0, sizeof(LINERANGE));
   }
   return nNumSel;
}


/* LineList_RemoveSel
 * ------------------
 *
 * Remove the selection in list box but save the selection state so that
 * it can be restored by calling LineList_RestoreSel
 * LineList_RemoveSel is called when the LineList window looses focus.
 */
void LineList_RemoveSel(LPLINELIST lpLL)
{
   LINERANGE lrSel;
   if (LineList_GetSel(lpLL, &lrSel) > 0) {
      lpLL->m_lrSaveSel = lrSel;
#if defined( WIN32 )
      SendMessage(
            lpLL->m_hWndListBox,
            LB_SETSEL,
            (WPARAM)FALSE,
            (LPARAM)-1
      );
#else
      SendMessage(
            lpLL->m_hWndListBox,
            LB_SETSEL,
            (WPARAM)FALSE,
            MAKELPARAM(-1,0)
      );
#endif
   }
}


/* LineList_RestoreSel
 * ------------------
 *
 * Restore the selection in list box that was previously saved by a call to
 * LineList_RemoveSel.
 * LineList_RestoreSel is called when the LineList window gains focus.
 */
void LineList_RestoreSel(LPLINELIST lpLL)
{
   LineList_SetSel(lpLL, &lpLL->m_lrSaveSel);
}


/* LineList_SetRedraw
 * ------------------
 *
 *      Enable/Disable the redraw of the linelist (listbox) on screen
 *
 *  fEnbaleDraw = TRUE      - enable redraw
 *                FALSE     - disable redraw
 */
void LineList_SetRedraw(LPLINELIST lpLL, BOOL fEnableDraw)
{
   SendMessage(lpLL->m_hWndListBox,WM_SETREDRAW,(WPARAM)fEnableDraw,0L);
}


/* LineList_ForceRedraw
 * --------------------
 *
 *      Force redraw of the linelist (listbox) on screen
 */
void LineList_ForceRedraw(LPLINELIST lpLL, BOOL fErase)
{
   InvalidateRect(lpLL->m_hWndListBox, NULL, fErase);
}


/* LineList_ForceLineRedraw
 * ------------------------
 *
 *      Force a particular line of the linelist (listbox) to redraw.
 */
void LineList_ForceLineRedraw(LPLINELIST lpLL, int nIndex, BOOL fErase)
{
   RECT   rect;

   LineList_GetLineRect( lpLL, nIndex, (LPRECT)&rect );
   InvalidateRect( lpLL->m_hWndListBox, (LPRECT)&rect, fErase );
}


/* LineList_ScrollLineIntoView
 * ---------------------------
 *  Make sure that the specified line is in view; if necessary scroll
 *      the listbox. if any portion of the line is visible, then no
 *      scrolling will occur.
 */
void LineList_ScrollLineIntoView(LPLINELIST lpLL, int nIndex)
{
   RECT rcWindow;
   RECT rcLine;
   RECT rcInt;

   if ( lpLL->m_nNumLines == 0 )
      return;

   if (! LineList_GetLineRect( lpLL, nIndex, (LPRECT)&rcLine ) )
      return;

   GetClientRect( lpLL->m_hWndListBox, (LPRECT) &rcWindow );
   if (! IntersectRect((LPRECT)&rcInt, (LPRECT)&rcWindow, (LPRECT)&rcLine))
      SendMessage(
            lpLL->m_hWndListBox,
            LB_SETTOPINDEX,
            (WPARAM)nIndex,
            (LPARAM)NULL
      );
}


/* LineList_CopySelToDoc
 * ---------------------
 *
 *      Copy the selection of the linelist to another document
 *
 *  RETURNS: number of lines copied.
 */
int LineList_CopySelToDoc(
      LPLINELIST              lpSrcLL,
      LPLINERANGE             lplrSel,
      LPOUTLINEDOC            lpDestDoc
)
{
   int              nEndLine;
   int              nStartLine;
   LPLINELIST       lpDestLL = &lpDestDoc->m_LineList;
   signed short     nDestIndex = LineList_GetFocusLineIndex(lpDestLL);
   LPLINE           lpSrcLine;
   int              nCopied = 0;
   int              i;

   if (lplrSel) {
      nEndLine = lplrSel->m_nEndLine;
      nStartLine = lplrSel->m_nStartLine;
   } else {
      nEndLine = LineList_GetCount(lpSrcLL) - 1;
      nStartLine = 0;
   }

   for(i = nStartLine; i <= nEndLine; i++) {
      lpSrcLine = LineList_GetLine(lpSrcLL, i);
      if (lpSrcLine && Line_CopyToDoc(lpSrcLine, lpDestDoc, nDestIndex)) {
         nDestIndex++;
         nCopied++;
      }
   }

   return nCopied;
}


/* LineList_SaveSelToStg
 * ---------------------
 *
 *      Save lines in selection into lpDestStg.
 *
 *      Return TRUE if ok, FALSE if error
 */
BOOL LineList_SaveSelToStg(
      LPLINELIST              lpLL,
      LPLINERANGE             lplrSel,
      UINT                    uFormat,
      LPSTORAGE               lpSrcStg,
      LPSTORAGE               lpDestStg,
      LPSTREAM                lpLLStm,
      BOOL                    fRemember
)
{
   int nEndLine;
   int nStartLine;
   int nNumLinesWritten = 0;
   HRESULT hrErr = NOERROR;
   ULONG nWritten;
   LPLINE lpLine;
   LINELISTHEADER_ONDISK llhRecord;
   int i;
   LARGE_INTEGER dlibSaveHeaderPos;
   LARGE_INTEGER dlibZeroOffset;
   LISet32( dlibZeroOffset, 0 );

   if (lplrSel) {
      nEndLine = lplrSel->m_nEndLine;
      nStartLine = lplrSel->m_nStartLine;
   } else {
      nEndLine = LineList_GetCount(lpLL) - 1;
      nStartLine = 0;
   }

   _fmemset(&llhRecord,0,sizeof(llhRecord));

   /* save seek position for LineList header record */
   hrErr = lpLLStm->lpVtbl->Seek(
         lpLLStm,
         dlibZeroOffset,
         STREAM_SEEK_CUR,
         (ULARGE_INTEGER FAR*)&dlibSaveHeaderPos
   );
   if (hrErr != NOERROR) goto error;

   /* write LineList header record */
   hrErr = lpLLStm->lpVtbl->Write(
         lpLLStm,
         (LPVOID)&llhRecord,
         sizeof(llhRecord),
         &nWritten
    );
   if (hrErr != NOERROR) {
      OleDbgOutHResult("Write LineList header returned", hrErr);
      goto error;
    }
   for(i = nStartLine; i <= nEndLine; i++) {
      lpLine = LineList_GetLine(lpLL, i);
      if(lpLine &&
         Line_SaveToStg(lpLine, uFormat, lpSrcStg, lpDestStg, lpLLStm,
                                                       fRemember))
         llhRecord.m_nNumLines++;
   }

   /* retore seek position for LineList header record */
   hrErr = lpLLStm->lpVtbl->Seek(
         lpLLStm,
         dlibSaveHeaderPos,
         STREAM_SEEK_SET,
         NULL
   );
   if (hrErr != NOERROR) goto error;

   /* write LineList header record */
   hrErr = lpLLStm->lpVtbl->Write(
         lpLLStm,
         (LPVOID)&llhRecord,
         sizeof(llhRecord),
         &nWritten
   );
   if (hrErr != NOERROR) goto error;

   /* reset seek position to end of stream */
   hrErr = lpLLStm->lpVtbl->Seek(
         lpLLStm,
         dlibZeroOffset,
         STREAM_SEEK_END,
         NULL
   );
   if (hrErr != NOERROR) goto error;

   return TRUE;

error:
#if defined( _DEBUG )
   OleDbgAssertSz(
         hrErr == NOERROR,
         "Could not write LineList header to LineList stream"
   );
#endif
   return FALSE;
}


/* LineList_LoadFromStg
 * --------------------
 *
 *      Load lines into linelist from storage.
 *
 *      Return TRUE if ok, FALSE if error
 */
BOOL LineList_LoadFromStg(
      LPLINELIST              lpLL,
      LPSTORAGE               lpSrcStg,
      LPSTREAM                lpLLStm
)
{
   HRESULT hrErr;
   ULONG nRead;
   LPLINE lpLine;
   int i;
   int nNumLines;
   LINELISTHEADER_ONDISK llineRecord;

   /* write LineList header record */
   hrErr = lpLLStm->lpVtbl->Read(
         lpLLStm,
         (LPVOID)&llineRecord,
         sizeof(llineRecord),
         &nRead
   );

   if (hrErr != NOERROR) {
      OleDbgOutHResult("Read LineList header returned", hrErr);
      goto error;
    }

   nNumLines = (int) llineRecord.m_nNumLines;

   for(i = 0; i < nNumLines; i++) {
      lpLine = Line_LoadFromStg(lpSrcStg, lpLLStm, lpLL->m_lpDoc);
      if (! lpLine)
         goto error;

      // Directly add lines to LineList without trying to update a NameTbl
      LineList_AddLine(lpLL, lpLine, i-1);
   }

   return TRUE;

error:
   // Delete any Line objects that were created
   if (lpLL->m_nNumLines > 0) {
      int nNumLines = lpLL->m_nNumLines;
      for (i = 0; i < nNumLines; i++)
         LineList_DeleteLine(lpLL, i);
   }

   return FALSE;
}


#if defined( USE_DRAGDROP )
```

```
/* LineList_SetFocusLineFromPointl
 * -------------------------------
 *
 */

void LineList_SetFocusLineFromPointl( LPLINELIST lpLL, POINTL pointl )
{
   int i = LineList_GetLineIndexFromPointl( lpLL, pointl );

   if ( i == (int)-1)
      return ;
   else
      LineList_SetFocusLine( lpLL, (WORD)i );
}


/* LineList_SetDragOverLineFromPointl
 * ----------------------------------
 *
 */

void LineList_SetDragOverLineFromPointl ( LPLINELIST lpLL, POINTL pointl )
{
   int    nIndex = LineList_GetLineIndexFromPointl( lpLL, pointl );
   LPLINE lpline = LineList_GetLine( lpLL, nIndex );

   if (!lpline)
      return;

   if (! lpline->m_fDragOverLine) {
      /* user has dragged over a new line. force new drop target line
      **    to repaint so that drop feedback will be drawn.
      */
      lpline->m_fDragOverLine = TRUE;
      LineList_ForceLineRedraw( lpLL, nIndex, TRUE /*fErase*/);

      if (lpLL->m_iDragOverLine!= -1 && lpLL->m_iDragOverLine!=nIndex) {

         /* force previous drop target line to repaint so that drop
         **    feedback will be undrawn
         */
         lpline = LineList_GetLine( lpLL, lpLL->m_iDragOverLine );
         if (lpline)
            lpline->m_fDragOverLine = FALSE;

         LineList_ForceLineRedraw(
               lpLL,lpLL->m_iDragOverLine,TRUE /*fErase*/);
      }

      lpLL->m_iDragOverLine = nIndex;

      // Force repaint immediately
      UpdateWindow(lpLL->m_hWndListBox);
   }
}
```

```
/* LineList_Scroll
 * ---------------
 *
 * Scroll the LineList list box in the desired direction by one line.
 *
 *      this function is called during a drag operation.
 */

void LineList_Scroll(LPLINELIST lpLL, DWORD dwScrollDir)
{
   switch (dwScrollDir) {
      case SCROLLDIR_UP:
         SendMessage( lpLL->m_hWndListBox, WM_VSCROLL, SB_LINEUP, 0L );
         break;

      case SCROLLDIR_DOWN:
         SendMessage( lpLL->m_hWndListBox, WM_VSCROLL, SB_LINEDOWN, 0L );
         break;
   }
}


/* LineList_GetLineIndexFromPointl
 * -------------------------------
 *   do hit test to get index of line corresponding to pointl
 */
int LineList_GetLineIndexFromPointl(LPLINELIST lpLL, POINTL pointl)
{
   RECT  rect;
   POINT point;
   DWORD i;

   point.x = (int)pointl.x;
   point.y = (int)pointl.y;

   ScreenToClient( lpLL->m_hWndListBox, &point);

   if ( lpLL->m_nNumLines == 0 )
      return -1;

   GetClientRect( lpLL->m_hWndListBox, (LPRECT) &rect );

   i = SendMessage( lpLL->m_hWndListBox, LB_GETTOPINDEX, (WPARAM)NULL,
(LPARAM)NULL );

   for ( ;; i++){

      RECT rectItem;

      if (!LineList_GetLineRect( lpLL, (int)i, (LPRECT)&rectItem ) )
         return -1;

      if ( rectItem.top > rect.bottom )
```

```
         return -1;

      if ( rectItem.top <= point.y && point.y <= rectItem.bottom)
         return (int)i;

   }

}


/* LineList_RestoreDragFeedback
 * ----------------------------
 *
 * Retore the index of the line that currently has focus (the active line).
 */
void LineList_RestoreDragFeedback(LPLINELIST lpLL)
{
   LPLINE lpLine;

   if (lpLL->m_iDragOverLine < 0 )
      return;

   lpLine = LineList_GetLine( lpLL, lpLL->m_iDragOverLine);

   if (lpLine) {

      lpLine->m_fDragOverLine = FALSE;
      LineList_ForceLineRedraw( lpLL,lpLL->m_iDragOverLine,TRUE /*fErase*/);

      // Force repaint immediately
      UpdateWindow(lpLL->m_hWndListBox);
   }

   lpLL->m_iDragOverLine = -1;

}

#endif
```

OUTLNAME.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    outlname.c
**
**    This file contains OutlineName functions.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/


#include "outline.h"

OLEDBGDATA


/* OutlineName_SetName
 * -------------------
 *
 *      Change the string of a name.
 */
void OutlineName_SetName(LPOUTLINENAME lpOutlineName, LPOLESTR lpszName)
{
   OLESTRCPY(lpOutlineName->m_szName, lpszName);
}


/* OutlineName_SetSel
 * ------------------
 *
 *      Change the line range of a  name.
 */
void OutlineName_SetSel(LPOUTLINENAME lpOutlineName, LPLINERANGE lplrSel,
BOOL fRangeModified)
{
#if defined( OLE_SERVER )
   // Call OLE server specific function instead
   ServerName_SetSel((LPSERVERNAME)lpOutlineName, lplrSel, fRangeModified);
#else

   lpOutlineName->m_nStartLine = lplrSel->m_nStartLine;
   lpOutlineName->m_nEndLine = lplrSel->m_nEndLine;
#endif
}


/* OutlineName_GetSel
 * ------------------
 *
 *      Retrieve the line range of a name.
 */
void OutlineName_GetSel(LPOUTLINENAME lpOutlineName, LPLINERANGE lplrSel)
{
   lplrSel->m_nStartLine = lpOutlineName->m_nStartLine;
   lplrSel->m_nEndLine = lpOutlineName->m_nEndLine;
}


/* OutlineName_SaveToStg
 * ---------------------
 *
 *      Save a name into a storage
 */
BOOL OutlineName_SaveToStg(LPOUTLINENAME lpOutlineName, LPLINERANGE lplrSel,
UINT uFormat, LPSTREAM lpNTStm, BOOL FAR* lpfNameSaved)
{
   HRESULT hrErr = NOERROR;
   ULONG nWritten;

   *lpfNameSaved = FALSE;

   /* if no range given or if the name is completely within the range,
   **      write it out.
   */
   if (!lplrSel ||
      ((lplrSel->m_nStartLine <= lpOutlineName->m_nStartLine) &&
      (lplrSel->m_nEndLine >= lpOutlineName->m_nEndLine))) {

      hrErr = lpNTStm->lpVtbl->Write(
            lpNTStm,
            lpOutlineName,
            sizeof(OUTLINENAME),
            &nWritten
      );
      *lpfNameSaved = TRUE;
   }
   return ((hrErr == NOERROR) ? TRUE : FALSE);
}


/* OutlineName_LoadFromStg
 * -----------------------
 *
 *      Load names from an open stream of a storage. if the name already
 * exits in the OutlineNameTable, it is NOT modified.
 *
 *      Returns TRUE is all ok, else FALSE.
 */
BOOL OutlineName_LoadFromStg(LPOUTLINENAME lpOutlineName, LPSTREAM lpNTStm)
{
   HRESULT hrErr = NOERROR;
   ULONG nRead;

   hrErr = lpNTStm->lpVtbl->Read(
         lpNTStm,
            lpOutlineName,
            sizeof(OUTLINENAME),
            &nRead
    );

    return ((hrErr == NOERROR) ? TRUE : FALSE);
}
```

_

OUTLNTBL.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    outlntbl.c
**
**    This file contains OutlineNameTable functions.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/


#include "outline.h"

OLEDBGDATA

extern LPOUTLINEAPP g_lpApp;

OLECHAR ErrMsgNameTable[] = OLESTR("Can't create NameTable!");


/* OutlineNameTable_Init
 * ---------------------
 *
 *      initialize a name table.
 */
BOOL OutlineNameTable_Init(LPOUTLINENAMETABLE lpOutlineNameTable,
LPOUTLINEDOC lpOutlineDoc)
{
   HWND lpParent = OutlineDoc_GetWindow(lpOutlineDoc);

   lpOutlineNameTable->m_nCount = 0;

   /* We will use an OwnerDraw listbox as our data structure to
   **    maintain the table of Names. this listbox will never be made
   **    visible. the listbox is just a convenient data structure to
   **    manage a collection.
   */
   lpOutlineNameTable->m_hWndListBox = CreateWindow(
               "listbox",              /* Window class name           */
               NULL,                   /* Window's title              */
               WS_CHILDWINDOW |
               LBS_OWNERDRAWFIXED,
               0, 0,                   /* Use default X, Y            */
               0, 0,                   /* Use default X, Y            */
               lpParent,               /* Parent window's handle      */
               (HMENU)IDC_NAMETABLE,   /* Child Window ID             */
               g_lpApp->m_hInst,       /* Instance of window          */
               NULL);                  /* Create struct for WM_CREATE */

   if (! lpOutlineNameTable->m_hWndListBox) {
      OutlineApp_ErrorMessage(g_lpApp, ErrMsgNameTable);
      return FALSE;
   }

   return TRUE;
}


/* OutlineNameTable_Destroy
 * ------------------------
 *
 *      Free memory used by the name table.
 */
void OutlineNameTable_Destroy(LPOUTLINENAMETABLE lpOutlineNameTable)
{
   // Delete all names
   OutlineNameTable_ClearAll(lpOutlineNameTable);

   DestroyWindow(lpOutlineNameTable->m_hWndListBox);
   Delete(lpOutlineNameTable);
}


/* OutlineNameTable_AddName
 * ------------------------
 *
 *      Add a name to the table
 */
void OutlineNameTable_AddName(LPOUTLINENAMETABLE lpOutlineNameTable,
LPOUTLINENAME lpOutlineName)
{
   SendMessage(
         lpOutlineNameTable->m_hWndListBox,
         LB_ADDSTRING,
         0,
         (DWORD)lpOutlineName
   );
   lpOutlineNameTable->m_nCount++;
}


/* OutlineNameTable_DeleteName
 * ---------------------------
 *
 *      Delete a name from table
 */
void OutlineNameTable_DeleteName(LPOUTLINENAMETABLE lpOutlineNameTable,int
nIndex)
{
   LPOUTLINENAME lpOutlineName =
OutlineNameTable_GetName(lpOutlineNameTable, nIndex);

#if defined( OLE_SERVER )
   /* OLE2NOTE: if there is a pseudo object attached to this name, it
   **    must first be closed before deleting the Name. this will
   **    cause OnClose notification to be sent to all linking clients.
   */
   ServerName_ClosePseudoObj((LPSERVERNAME)lpOutlineName);
#endif

   if (lpOutlineName)
      Delete(lpOutlineName);      // free memory for name

   SendMessage(
         lpOutlineNameTable->m_hWndListBox,
         LB_DELETESTRING,
         (WPARAM)nIndex,
         0L
   );
   lpOutlineNameTable->m_nCount--;
}


/* OutlineNameTable_GetNameIndex
 * -----------------------------
 *
 *      Return the index of the Name given a pointer to the Name.
 *      Return -1 if the Name is not found.
 */
int OutlineNameTable_GetNameIndex(LPOUTLINENAMETABLE lpOutlineNameTable,
LPOUTLINENAME lpOutlineName)
{
   LRESULT lReturn;

   if (! lpOutlineName) return -1;

   lReturn = SendMessage(
         lpOutlineNameTable->m_hWndListBox,
         LB_FINDSTRING,
         (WPARAM)-1,
         (LPARAM)(LPCSTR)lpOutlineName
      );

   return ((lReturn == LB_ERR) ? -1 : (int)lReturn);
}


/* OutlineNameTable_GetName
 * ------------------------
 *
 *      Retrieve the pointer to the Name given its index in the NameTable
 */
LPOUTLINENAME OutlineNameTable_GetName(LPOUTLINENAMETABLE
lpOutlineNameTable, int nIndex)
{
   LPOUTLINENAME lpOutlineName = NULL;
    LRESULT lResult;

   if (lpOutlineNameTable->m_nCount == 0 ||
      nIndex > lpOutlineNameTable->m_nCount ||
      nIndex < 0) {
      return NULL;
   }

   lResult = SendMessage(
         lpOutlineNameTable->m_hWndListBox,
         LB_GETTEXT,
         nIndex,
         (LPARAM)(LPCSTR)&lpOutlineName
   );
    OleDbgAssert(lResult != LB_ERR);
   return lpOutlineName;
}


/* OutlineNameTable_FindName
 * -------------------------
 *
 *      Find a name in the name table given a string.
 */
LPOUTLINENAME OutlineNameTable_FindName(LPOUTLINENAMETABLE
lpOutlineNameTable, LPOLESTR lpszName)
{
   LPOUTLINENAME lpOutlineName;
   BOOL fFound = FALSE;
   int i;

   for (i = 0; i < lpOutlineNameTable->m_nCount; i++) {
      lpOutlineName = OutlineNameTable_GetName(lpOutlineNameTable, i);
      if (OLESTRCPY(lpOutlineName->m_szName, lpszName) == 0) {
         fFound = TRUE;
         break;      // FOUND MATCH!
      }
   }

   return (fFound ? lpOutlineName : NULL);
}


/* OutlineNameTable_FindNamedRange
 * -------------------------------
 *
 *      Find a name in the name table which matches a given line range.
 */
LPOUTLINENAME OutlineNameTable_FindNamedRange(LPOUTLINENAMETABLE
lpOutlineNameTable, LPLINERANGE lplrSel)
{
   LPOUTLINENAME lpOutlineName;
   BOOL fFound = FALSE;
   int i;

   for (i = 0; i < lpOutlineNameTable->m_nCount; i++) {
      lpOutlineName = OutlineNameTable_GetName(lpOutlineNameTable, i);
      if ((lpOutlineName->m_nStartLine == lplrSel->m_nStartLine) &&
         (lpOutlineName->m_nEndLine == lplrSel->m_nEndLine) ) {
```

```
         fFound = TRUE;
         break;      // FOUND MATCH!
      }
   }

   return (fFound ? lpOutlineName : NULL);
}


/* OutlineNameTable_GetCount
 * -------------------------
 *
 * Return number of names in nametable
 */
int OutlineNameTable_GetCount(LPOUTLINENAMETABLE lpOutlineNameTable)
{
   if (!lpOutlineNameTable)
      return 0;

   return lpOutlineNameTable->m_nCount;
}


/* OutlineNameTable_ClearAll
 * -------------------------
 *
 *      Remove all names from table
 */
void OutlineNameTable_ClearAll(LPOUTLINENAMETABLE lpOutlineNameTable)
{
   LPOUTLINENAME lpOutlineName;
   int i;
   int nCount = lpOutlineNameTable->m_nCount;

   for (i = 0; i < nCount; i++) {
      lpOutlineName = OutlineNameTable_GetName(lpOutlineNameTable, i);
      Delete(lpOutlineName);      // free memory for name
   }

   lpOutlineNameTable->m_nCount = 0;
   SendMessage(lpOutlineNameTable->m_hWndListBox,LB_RESETCONTENT,0,0L);
}


/* OutlineNameTable_AddLineUpdate
 * ------------------------------
 *
 *      Update table when a new line is added at nAddIndex
 * The line used to be at nAddIndex is pushed down
 */
void OutlineNameTable_AddLineUpdate(LPOUTLINENAMETABLE lpOutlineNameTable,
int nAddIndex)
{
   LPOUTLINENAME lpOutlineName;
   LINERANGE lrSel;
   int i;
   BOOL fRangeModified = FALSE;

   for(i = 0; i < lpOutlineNameTable->m_nCount; i++) {
      lpOutlineName=OutlineNameTable_GetName(lpOutlineNameTable, i);
      OutlineName_GetSel(lpOutlineName, &lrSel);

      if((int)lrSel.m_nStartLine > nAddIndex) {
         lrSel.m_nStartLine++;
         fRangeModified = !fRangeModified;
      }
      if((int)lrSel.m_nEndLine > nAddIndex) {
         lrSel.m_nEndLine++;
         fRangeModified = !fRangeModified;
      }

      OutlineName_SetSel(lpOutlineName, &lrSel, fRangeModified);
   }
}


/* OutlineNameTable_DeleteLineUpdate
 * ---------------------------------
 *
 *      Update the table when a line at nDeleteIndex is removed
 */
void OutlineNameTable_DeleteLineUpdate(LPOUTLINENAMETABLE
lpOutlineNameTable, int nDeleteIndex)
{
   LPOUTLINENAME lpOutlineName;
   LINERANGE lrSel;
   int i;
   BOOL fRangeModified = FALSE;

   for(i = 0; i < lpOutlineNameTable->m_nCount; i++) {
      lpOutlineName=OutlineNameTable_GetName(lpOutlineNameTable, i);
      OutlineName_GetSel(lpOutlineName, &lrSel);

      if((int)lrSel.m_nStartLine > nDeleteIndex) {
         lrSel.m_nStartLine--;
         fRangeModified = !fRangeModified;
      }
      if((int)lrSel.m_nEndLine >= nDeleteIndex) {
         lrSel.m_nEndLine--;
         fRangeModified = !fRangeModified;
      }

      // delete the name if its entire range is deleted
      if(lrSel.m_nStartLine > lrSel.m_nEndLine) {
         OutlineNameTable_DeleteName(lpOutlineNameTable, i);
         i--;  // re-examine this name
      } else {
         OutlineName_SetSel(lpOutlineName, &lrSel, fRangeModified);
      }
   }
```

```
}


/* OutlineNameTable_SaveSelToStg
 * -----------------------------
 *
 *      Save only the names that refer to lines completely contained in the
 * specified selection range.
 */
BOOL OutlineNameTable_SaveSelToStg(
      LPOUTLINENAMETABLE      lpOutlineNameTable,
      LPLINERANGE             lplrSel,
      UINT                    uFormat,
      LPSTREAM                lpNTStm
)
{
   HRESULT hrErr;
   ULONG nWritten;
   LPOUTLINENAME lpOutlineName;
   short nNameCount = 0;
   BOOL fNameSaved;
   BOOL fStatus;
   int i;
   LARGE_INTEGER dlibZeroOffset;
   LISet32( dlibZeroOffset, 0 );

   /* initially write 0 for count of names. the correct count will be
   **    written at the end when we know how many names qualified to
   **    be written (within the selection).
   */
   hrErr = lpNTStm->lpVtbl->Write(
         lpNTStm,
         (short FAR*)&nNameCount,
         sizeof(nNameCount),
         &nWritten
   );
   if (hrErr != NOERROR) {
      OleDbgOutHResult("Write NameTable header returned", hrErr);
      goto error;
    }

   for(i = 0; i < lpOutlineNameTable->m_nCount; i++) {
      lpOutlineName=OutlineNameTable_GetName(lpOutlineNameTable, i);
      fStatus = OutlineName_SaveToStg(
            lpOutlineName,
            lplrSel,
            uFormat,
            lpNTStm,
            (BOOL FAR*)&fNameSaved
      );
      if (! fStatus) goto error;
      if (fNameSaved) nNameCount++;
   }

   /* write the final count of names written. */
   hrErr = lpNTStm->lpVtbl->Seek(
         lpNTStm,
         dlibZeroOffset,
         STREAM_SEEK_SET,
         NULL
   );
   if (hrErr != NOERROR) {
      OleDbgOutHResult("Seek to NameTable header returned", hrErr);
      goto error;
    }

   hrErr = lpNTStm->lpVtbl->Write(
         lpNTStm,
         (short FAR*)&nNameCount,
         sizeof(nNameCount),
         &nWritten
   );
   if (hrErr != NOERROR) {
      OleDbgOutHResult("Write NameTable count in header returned", hrErr);
      goto error;
    }

   OleStdRelease((LPUNKNOWN)lpNTStm);
   return TRUE;

error:
   if (lpNTStm)
      OleStdRelease((LPUNKNOWN)lpNTStm);

   return FALSE;
}


/* OutlineNameTable_LoadFromStg
 * ----------------------------
 *
 *      Load Name Table from file
 *
 *      Return TRUE if ok, FALSE if error
 */
BOOL OutlineNameTable_LoadFromStg(LPOUTLINENAMETABLE lpOutlineNameTable,
LPSTORAGE lpSrcStg)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   HRESULT hrErr;
   IStream FAR* lpNTStm;
   ULONG nRead;
   short nCount;
   LPOUTLINENAME lpOutlineName;
   BOOL fStatus;
   short i;

   hrErr = lpSrcStg->lpVtbl->OpenStream(
         lpSrcStg,
         OLESTR("NameTable"),
         NULL,
         STGM_READ | STGM_SHARE_EXCLUSIVE,
         0,
         &lpNTStm
   );
   if (hrErr != NOERROR) {
      OleDbgOutHResult("OpenStream NameTable returned", hrErr);
      goto error;
    }

   hrErr = lpNTStm->lpVtbl->Read(lpNTStm,&nCount,sizeof(nCount),&nRead);
   if (hrErr != NOERROR) {
      OleDbgOutHResult("Read NameTable header returned", hrErr);
      goto error;
    }

   for (i = 0; i < nCount; i++) {
      lpOutlineName = OutlineApp_CreateName(lpOutlineApp);
      if (! lpOutlineName) goto error;
      fStatus = OutlineName_LoadFromStg(lpOutlineName, lpNTStm);
      if (! fStatus) goto error;
      OutlineNameTable_AddName(lpOutlineNameTable, lpOutlineName);
   }

   OleStdRelease((LPUNKNOWN)lpNTStm);
   return TRUE;

error:
   if (lpNTStm)
      OleStdRelease((LPUNKNOWN)lpNTStm);

   return FALSE;
}
```

OUTLRC.H   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Sample Code
**
**    outlrc.h
**
**    This file containes constants used in rc file for Outline.exe
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#if !defined( _OUTLRC_H_ )
#define _OUTLRC_H_

#ifndef RC_INVOKED
#pragma message ("INCLUDING OUTLRC.H from " __FILE__)
#endif  /* RC_INVOKED */

#if defined( OLE_SERVER ) && ! defined( INPLACE_SVR )

#ifdef TEST32
#define APPNAME                 OLESTR("SvrOut32")
#define APPDESC                 "OLE 2.0 Server Sample Code 32"
#else
#define APPNAME                 OLESTR("SvrOutl")
#define APPDESC                 "OLE 2.0 Server Sample Code"
#endif   //TEST32

#define APPMENU                 "SvrOutlMenu"
#define APPACCEL                "SvrOutlAccel"
#define FB_EDIT_ACCEL           "SvrOutlAccelFocusEdit"
#define APPICON                 "SvrOutlIcon"
#define APPWNDCLASS             "SvrOutlApp"
#define DOCWNDCLASS             "SvrOutlDoc"

#endif  // OLE_SERVER && ! INPLACE_SVR

#if defined( INPLACE_SVR )

#ifdef TEST32
#define APPNAME                 OLESTR("ISvrOu32")
#define APPDESC                 "OLE 2.0 In-Place Server Sample Code 32"
#else
#define APPNAME                 OLESTR("ISvrOtl")
#define APPDESC                 "OLE 2.0 In-Place Server Sample Code"
#endif //TEST32

#define APPMENU                 "SvrOutlMenu"
#define APPACCEL                "SvrOutlAccel"
#define FB_EDIT_ACCEL           "SvrOutlAccelFocusEdit"
#define APPICON                 "SvrOutlIcon"
```

```
#define APPWNDCLASS              "SvrOutlApp"
#define DOCWNDCLASS              "SvrOutlDoc"
#endif  // INPLACE_SVR

#if defined( OLE_CNTR ) && ! defined( INPLACE_CNTR )

#ifdef TEST32
#define APPNAME                  OLESTR("CntrOu32")
#define APPDESC                  "OLE 2.0 Container Sample Code 32"
#else
#define APPNAME                  OLESTR("CntrOutl")
#define APPDESC                  "OLE 2.0 Container Sample Code"
#endif //TEST32

#define APPMENU                  "CntrOutlMenu"
#define APPACCEL                 "CntrOutlAccel"
#define FB_EDIT_ACCEL            "CntrOutlAccelFocusEdit"
#define APPICON                  "CntrOutlIcon"
#define APPWNDCLASS              "CntrOutlApp"
#define DOCWNDCLASS              "CntrOutlDoc"
#endif  // OLE_CNTR && ! INPLACE_CNTR

#if defined( INPLACE_CNTR )

#ifdef TEST32
#define APPNAME                  OLESTR("ICntrO32")
#define APPDESC                  "OLE 2.0 In-Place Container Sample Code 32"
#else
#define APPNAME                  OLESTR("ICntrOtl")
#define APPDESC                  "OLE 2.0 In-Place Container Sample Code"
#endif

#define APPMENU                  "CntrOutlMenu"
#define APPACCEL                 "CntrOutlAccel"
#define FB_EDIT_ACCEL            "CntrOutlAccelFocusEdit"
#define APPICON                  "CntrOutlIcon"
#define APPWNDCLASS              "CntrOutlApp"
#define DOCWNDCLASS              "CntrOutlDoc"
#endif  // INPLACE_CNTR

#if !defined( OLE_VERSION )
#define APPNAME                  OLESTR("Outline")
#define APPMENU                  "OutlineMenu"
#define APPACCEL                 "OutlineAccel"
#define FB_EDIT_ACCEL            "OutlineAccelFocusEdit"
#define APPICON                  "OutlineIcon"
#define APPWNDCLASS              "OutlineApp"
#define DOCWNDCLASS              "OutlineDoc"
#define APPDESC                  "OLE 2.0 Sample Code"
#endif  // OLE_VERSION

#define IDM_FILE                        1000
#define IDM_F_NEW                       1050
#define IDM_F_OPEN                      1100
#define IDM_F_SAVE                      1150
```

```
#define IDM_F_SAVEAS                   1200
#define IDM_F_PRINT                    1300
#define IDM_F_PRINTERSETUP             1350
#define IDM_F_EXIT                     1450
#define IDM_EDIT                       2000
#define IDM_E_UNDO                     2050
#define IDM_E_CUT                      2150
#define IDM_E_COPY                     2200
#define IDM_E_PASTE                    2250
#define IDM_E_PASTESPECIAL             2255
#define IDM_E_CLEAR                    2300
#define IDM_E_SELECTALL                2560
#define IDM_LINE                       3000
#define IDM_L_ADDLINE                  3400
#define IDM_L_EDITLINE                 3450
#define IDM_L_INDENTLINE               3500
#define IDM_L_UNINDENTLINE             3550
#define IDM_L_SETLINEHEIGHT            3560
#define IDM_NAME                       4000
#define IDM_N_DEFINENAME               4050
#define IDM_N_GOTONAME                 4100
#define IDM_HELP                       5000
#define IDM_H_ABOUT                    5050
#define IDM_DEBUG                      6000
#define IDM_D_DEBUGLEVEL               6050
#define IDM_D_INSTALLMSGFILTER         6060
#define IDM_D_REJECTINCOMING            6070
#define IDM_O_BB_TOP                   6100
#define IDM_O_BB_BOTTOM                6150
#define IDM_O_BB_POPUP                 6200
#define IDM_O_BB_HIDE                  6210
#define IDM_O_FB_TOP                   6250
#define IDM_O_FB_BOTTOM                6300
#define IDM_O_FB_POPUP                 6350
#define IDM_O_HEAD_SHOW                6400
#define IDM_O_HEAD_HIDE                6450
#define IDM_O_SHOWOBJECT               6460
#define IDM_V_ZOOM_400                 6500
#define IDM_V_ZOOM_300                 6510
#define IDM_V_ZOOM_200                 6520
#define IDM_V_ZOOM_100                 6550
#define IDM_V_ZOOM_75                  6600
#define IDM_V_ZOOM_50                  6650
#define IDM_V_ZOOM_25                  6700
#define IDM_V_SETMARGIN_0              6750
#define IDM_V_SETMARGIN_1              6800
#define IDM_V_SETMARGIN_2              6850
#define IDM_V_SETMARGIN_3              6860
#define IDM_V_SETMARGIN_4              6870
#define IDM_V_ADDTOP_1                 6900
#define IDM_V_ADDTOP_2                 6910
#define IDM_V_ADDTOP_3                 6920
#define IDM_V_ADDTOP_4                 6930
```

```
#define IDM_FB_EDIT                       7000
#define IDM_FB_CANCEL                     7005
#define IDM_F2                            7010
#define IDM_ESCAPE                        7015


#define IDD_LINELISTBOX                   101
#define IDD_EDIT                          102
#define IDD_COMBO                         103
#define IDD_DELETE                        104
#define IDD_CLOSE                         105
#define IDD_APPTEXT                       106
#define IDD_FROM                          107
#define IDD_TO                            108
#define IDD_BITMAPLOCATION                109
#define IDD_CHECK                         110
#define IDD_TEXT                          111
#define IDD_LIMIT                         112


#define IDC_LINELIST                      201
#define IDC_NAMETABLE                     202


#define WM_U_INITFRAMETOOLS               WM_USER

#ifdef RC_INVOKED
#include "debug.rc"
#endif  /* RC_INVOKED */

#endif // _OUTLRC_H_
_
```

OUTLTXTL.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Sample Code
**
**    outltxtl.c
**
**    This file contains TextLine methods and related support functions.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/


#include "outline.h"

OLEDBGDATA

extern LPOUTLINEAPP g_lpApp;


/* TextLine_Create
 * ---------------
 *
 *      Create a text line object and return the pointer
 */
LPTEXTLINE TextLine_Create(HDC hDC, UINT nTab, LPSTR lpszText)
{
        LPTEXTLINE lpTextLine;

        lpTextLine=(LPTEXTLINE) New((DWORD)sizeof(TEXTLINE));
        if (lpTextLine == NULL) {
                OleDbgAssertSz(lpTextLine!=NULL,"Error allocating
TextLine");
                return NULL;
        }

        TextLine_Init(lpTextLine, nTab, hDC);

        if (lpszText) {
                lpTextLine->m_nLength = lstrlen(lpszText);
                lstrcpy((LPSTR)lpTextLine->m_szText, lpszText);
        } else {
                lpTextLine->m_nLength = 0;
                lpTextLine->m_szText[0] = '\0';
        }

        TextLine_CalcExtents(lpTextLine, hDC);

        return(lpTextLine);
}
```

```
/* TextLine_Init
 * -------------
 *
 *      Calculate the width/height of a text line object.
 */
void TextLine_Init(LPTEXTLINE lpTextLine, int nTab, HDC hDC)
{
        Line_Init((LPLINE)lpTextLine, nTab, hDC);   // init the base class
fields

        ((LPLINE)lpTextLine)->m_lineType = TEXTLINETYPE;
        lpTextLine->m_nLength = 0;
        lpTextLine->m_szText[0] = '\0';
}


/* TextLine_Delete
 * ---------------
 *
 *      Delete the TextLine structure
 */
void TextLine_Delete(LPTEXTLINE lpTextLine)
{
        Delete((LPVOID)lpTextLine);
}


/* TextLine_Edit
 * -------------
 *
 *      Edit the text line object.
 *
 *      Returns TRUE if line was changed
 *              FALSE if the line was NOT changed
 */
BOOL TextLine_Edit(LPTEXTLINE lpLine, HWND hWndDoc, HDC hDC)
{
#if defined( USE_FRAMETOOLS )
        LPFRAMETOOLS lptb = OutlineApp_GetFrameTools(g_lpApp);
#endif
        BOOL fStatus = FALSE;

#if defined( USE_FRAMETOOLS )
        FrameTools_FB_GetEditText(lptb, lpLine->m_szText, sizeof(lpLine-
>m_szText));
#else
        if (! InputTextDlg(hWndDoc, lpLine->m_szText, "Edit Line"))
                return FALSE;
#endif

        lpLine->m_nLength = lstrlen(lpLine->m_szText);
        TextLine_CalcExtents(lpLine, hDC);
        fStatus = TRUE;

        return fStatus;
```

```
}


/* TextLine_CalcExtents
 * --------------------
 *
 *      Calculate the width/height of a text line object.
 */
void TextLine_CalcExtents(LPTEXTLINE lpTextLine, HDC hDC)
{
        SIZE size;
        LPLINE lpLine = (LPLINE)lpTextLine;

        if (lpTextLine->m_nLength) {
                GetTextExtentPoint(hDC, lpTextLine->m_szText,
                                                        lpTextLine-
>m_nLength, &size);
                lpLine->m_nWidthInHimetric=size.cx;
                lpLine->m_nHeightInHimetric=size.cy;
        } else {
                // we still need to calculate proper height even for NULL
string
                TEXTMETRIC tm;
                GetTextMetrics(hDC, &tm);

                // required to set height
                lpLine->m_nHeightInHimetric = tm.tmHeight;
                lpLine->m_nWidthInHimetric = 0;
        }

#if defined( _DEBUG )
        {
                RECT rc;
                rc.left = 0;
                rc.top = 0;
                rc.right = XformWidthInHimetricToPixels(hDC,
                                lpLine->m_nWidthInHimetric);
                rc.bottom = XformHeightInHimetricToPixels(hDC,
                                lpLine->m_nHeightInHimetric);

                OleDbgOutRect3("TextLine_CalcExtents", (LPRECT)&rc);
        }
#endif
}



/* TextLine_SetHeightInHimetric
 * ----------------------------
 *
 *      Set the height of a textline object.
 */
void TextLine_SetHeightInHimetric(LPTEXTLINE lpTextLine, int nHeight)
{
        if (!lpTextLine)
```

```
                return;

        ((LPLINE)lpTextLine)->m_nHeightInHimetric = nHeight;
}



/* TextLine_GetTextLen
 * -------------------
 *
 * Return length of string of the TextLine (not considering the tab level).
 */
int TextLine_GetTextLen(LPTEXTLINE lpTextLine)
{
        return lstrlen((LPSTR)lpTextLine->m_szText);
}


/* TextLine_GetTextData
 * --------------------
 *
 * Return the string of the TextLine (not considering the tab level).
 */
void TextLine_GetTextData(LPTEXTLINE lpTextLine, LPSTR lpszBuf)
{
        lstrcpy(lpszBuf, (LPSTR)lpTextLine->m_szText);
}


/* TextLine_GetOutlineData
 * -----------------------
 *
 * Return the CF_OUTLINE format data for the TextLine.
 */
BOOL TextLine_GetOutlineData(LPTEXTLINE lpTextLine, LPTEXTLINE lpBuf)
{
        TextLine_Copy((LPTEXTLINE)lpTextLine, lpBuf);
        return TRUE;
}


/* TextLine_Draw
 * -------------
 *
 *      Draw a text line object on a DC.
 * Parameters:
 *      hDC     - DC to which the line will be drawn
 *      lpRect  - the object rectangle in logical coordinates
 *      lpRectWBounds - bounding rect of the metafile underneath hDC
 *                      (NULL if hDC is not a metafile DC)
 *                      this is used by ContainerLine_Draw to draw the OLE
obj
 *      fHighlight    - TRUE use selection highlight text color
 */
void TextLine_Draw(
                LPTEXTLINE  lpTextLine,
                HDC         hDC,
                LPRECT      lpRect,
                LPRECT      lpRectWBounds,
                BOOL        fHighlight
)
{
        RECT rc;
        int nBkMode;
        COLORREF clrefOld;

        if (!lpTextLine)
                return;

        rc = *lpRect;
        rc.left += ((LPLINE)lpTextLine)->m_nTabWidthInHimetric;
        rc.right += ((LPLINE)lpTextLine)->m_nTabWidthInHimetric;

        nBkMode = SetBkMode(hDC, TRANSPARENT);

        if (fHighlight) {
                /*Get proper txt colors */
                clrefOld =
SetTextColor(hDC,GetSysColor(COLOR_HIGHLIGHTTEXT));
        }
        else {
                clrefOld = SetTextColor(hDC, GetSysColor(COLOR_WINDOWTEXT));
        }

        ExtTextOut(
                        hDC,
                        rc.left,
                        rc.top,
                        ETO_CLIPPED,
                        (LPRECT)&rc,
                        lpTextLine->m_szText,
                        lpTextLine->m_nLength,
                        (LPINT) NULL /* default char spacing */
        );

        SetTextColor(hDC, clrefOld);
        SetBkMode(hDC, nBkMode);
}

/* TextLine_DrawSelHilight
 * -----------------------
 *
 *      Handles selection of textline
 */
void TextLine_DrawSelHilight(LPTEXTLINE lpTextLine, HDC hDC, LPRECT lpRect,
UINT itemAction, UINT itemState)
{
        if (itemAction & ODA_SELECT) {
                // check if there is a selection state change, ==> invert
rect
```

```
                if (itemState & ODS_SELECTED) {
                        if (!((LPLINE)lpTextLine)->m_fSelected) {
                                ((LPLINE)lpTextLine)->m_fSelected = TRUE;
                                InvertRect(hDC, (LPRECT)lpRect);
                        }
                } else {
                        if (((LPLINE)lpTextLine)->m_fSelected) {
                                ((LPLINE)lpTextLine)->m_fSelected = FALSE;
                                InvertRect(hDC, lpRect);
                        }
                }
        } else if (itemAction & ODA_DRAWENTIRE) {
                ((LPLINE)lpTextLine)->m_fSelected=((itemState &
ODS_SELECTED) ? TRUE : FALSE);
                InvertRect(hDC, lpRect);
        }
}

/* TextLine_Copy
 * -------------
 *
 *      Duplicate a textline
 */
BOOL TextLine_Copy(LPTEXTLINE lpSrcLine, LPTEXTLINE lpDestLine)
{
        _fmemcpy(lpDestLine, lpSrcLine, sizeof(TEXTLINE));
        return TRUE;
}


/* TextLine_CopyToDoc
 * ------------------
 *
 *      Copy a textline to another Document (usually ClipboardDoc)
 */
BOOL TextLine_CopyToDoc(LPTEXTLINE lpSrcLine, LPOUTLINEDOC lpDestDoc, int
nIndex)
{
        LPTEXTLINE  lpDestLine;
        BOOL        fStatus = FALSE;

        lpDestLine = (LPTEXTLINE) New((DWORD)sizeof(TEXTLINE));
        if (lpDestLine == NULL) {
                OleDbgAssertSz(lpDestLine!=NULL,"Error allocating
TextLine");
                return FALSE;
        }

        if (TextLine_Copy(lpSrcLine, lpDestLine)) {
                OutlineDoc_AddLine(lpDestDoc, (LPLINE)lpDestLine, nIndex);
                fStatus = TRUE;
        }

        return fStatus;
}
```

```
/* TextLine_SaveToStg
 * ------------------
 *
 *      Save a textline into a storage
 *
 *      Return TRUE if successful, FALSE otherwise
 */
BOOL TextLine_SaveToStm(LPTEXTLINE lpTextLine, LPSTREAM lpLLStm)
{
        HRESULT hrErr;
        ULONG nWritten;
        USHORT nLengthOnDisk;

        nLengthOnDisk = (USHORT) lpTextLine->m_nLength;

        hrErr = lpLLStm->lpVtbl->Write(
                        lpLLStm,
                        (LPVOID)&nLengthOnDisk,
                        sizeof(nLengthOnDisk),
                        &nWritten
        );
        if (hrErr != NOERROR) {
                OleDbgOutHResult("Write TextLine data (1) returned", hrErr);
                return FALSE;
    }

        hrErr = lpLLStm->lpVtbl->Write(
                        lpLLStm,
                        (LPVOID)lpTextLine->m_szText,
                        lpTextLine->m_nLength,
                        &nWritten
        );
        if (hrErr != NOERROR) {
                OleDbgOutHResult("Write TextLine data (2) returned", hrErr);
                return FALSE;
    }

        return TRUE;
}


/* TextLine_LoadFromStg
 * --------------------
 *
 *      Load a textline from storage
 */
LPLINE TextLine_LoadFromStg(LPSTORAGE lpSrcStg, LPSTREAM lpLLStm,
LPOUTLINEDOC lpDestDoc)
{
        HRESULT hrErr;
        ULONG nRead;
        LPTEXTLINE lpTextLine;
        USHORT nLengthOnDisk;
        lpTextLine=(LPTEXTLINE) New((DWORD)sizeof(TEXTLINE));
        if (lpTextLine == NULL) {
                OleDbgAssertSz(lpTextLine!=NULL,"Error allocating
TextLine");
                return NULL;
        }

        TextLine_Init(lpTextLine, 0, NULL);

        hrErr = lpLLStm->lpVtbl->Read(
                        lpLLStm,
                        (LPVOID)&nLengthOnDisk,
                        sizeof(nLengthOnDisk),
                        &nRead
        );
        if (hrErr != NOERROR) {
                OleDbgOutHResult("Read TextLine data (1) returned", hrErr);
                return NULL;
    }

        lpTextLine->m_nLength = (UINT) nLengthOnDisk;
        OleDbgAssert(lpTextLine->m_nLength < sizeof(lpTextLine->m_szText));

        hrErr = lpLLStm->lpVtbl->Read(
                        lpLLStm,
                        (LPVOID)&lpTextLine->m_szText,
                        lpTextLine->m_nLength,
                        &nRead
        );
        if (hrErr != NOERROR) {
                OleDbgOutHResult("Read TextLine data (1) returned", hrErr);
                return NULL;
    }

        lpTextLine->m_szText[lpTextLine->m_nLength] = '\0'; // add str
terminator

        return (LPLINE)lpTextLine;
}
```

STATUS.H   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Sample Code
**
**    status.h
**
**    This file contains typedefs, defines, global variable declarations,
**    and function prototypes for the status bar window.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

// Sizes of statusbar items
#if defined( USE_STATUSBAR )
   #define STATUS_HEIGHT   23
#else
   #define STATUS_HEIGHT   0
#endif
#define STATUS_RLEFT    8
#define STATUS_RRIGHT   400
#define STATUS_RTOP     3
#define STATUS_RBOTTOM  20
#define STATUS_TTOP     4
#define STATUS_TLEFT    11
#define STATUS_THEIGHT  18


typedef enum {
   STATUS_READY,
   STATUS_BLANK
} STATCONTROL;

// Window for status bar.
extern HWND hwndStatusbar;

BOOL RegisterStatusClass(HINSTANCE hInstance);
HWND CreateStatusWindow(HWND hWndApp, HINSTANCE hInst);
void DestroyStatusWindow(HWND hWndStatusBar);

void AssignPopupMessage(HMENU hmenuPopup, char *szMessage);

void SetStatusText(HWND hWndStatusBar, LPSTR lpszMessage);
void GetItemMessage(UINT wIDItem, LPSTR FAR* lplpszMessage);
void GetPopupMessage(HMENU hmenuPopup, LPSTR FAR* lplpszMessage);
void GetSysMenuMessage(UINT wIDItem, LPSTR FAR* lplpszMessage);
void GetControlMessage(STATCONTROL scCommand, LPSTR FAR* lplpszMessage);
_
```

STATUS.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Sample Code
**
**    status.c
**
**    This file contains the window handlers, and various initialization
**    and utility functions for an application status bar.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

// Application specific include files
#include "outline.h"
#include "message.h"
#include "status.h"

// Current status message.
static char lpszStatusMessage[256];

// Window proc for status window.
LRESULT FAR PASCAL StatusWndProc
   (HWND hwnd, unsigned message, WPARAM wParam, LPARAM lParam);

// List of all constant messages.
static STATMESG ControlList[2] =
{
   {   STATUS_READY,   "Ready."    },
   {   STATUS_BLANK,   " "         }
};

// List of all system menu messages.
static STATMESG SysMenuList[16] =
{
   {   SC_SIZE,        "Change the size of the window."            },
   {   SC_MOVE,        "Move the window."                          },
   {   SC_MINIMIZE,    "Make the window iconic."                   },
   {   SC_MAXIMIZE,    "Make the window the size of the screen."   },
   {   SC_NEXTWINDOW,  "Activate the next window."                 },
   {   SC_PREVWINDOW,  "Activate the previous window."             },
   {   SC_CLOSE,       "Close this window."                        },
   {   SC_VSCROLL,     "Vertical scroll?"                          },
   {   SC_HSCROLL,     "Horizontal scroll?"                        },
   {   SC_MOUSEMENU,   "A menu for mice."                          },
   {   SC_KEYMENU,     "A menu for keys (I guess)."                },
   {   SC_ARRANGE,     "Arrange something."                        },
   {   SC_RESTORE,     "Make the window noramally sized."          },
   {   SC_TASKLIST,    "Put up the task list dialog."              },
   {   SC_SCREENSAVE,  "Save the screen!  Run for your life!"      },
   {   SC_HOTKEY,      "Boy, is this key hot!"                     }
};
```

```
// Message type for popup messages.
typedef struct {
   HMENU hmenu;
   char string[MAX_MESSAGE];
} STATPOPUP;

// List of all popup messages.
static STATPOPUP PopupList[NUM_POPUP];

static UINT nCurrentPopup = 0;



/* RegisterStatusClass
 * -------------------
 *
 * Creates classes for status window.
 *
 * HINSTANCE hInstance
 *
 * RETURNS: TRUE if class successfully registered.
 *          FALSE otherwise.
 *
 * CUSTOMIZATION: Change class name.
 *
 */
BOOL RegisterStatusClass(HINSTANCE hInstance)
{
   WNDCLASS  wc;

   wc.lpszClassName = "ObjStatus";
   wc.lpfnWndProc   = StatusWndProc;
   wc.style         = 0;
   wc.hInstance     = hInstance;
   wc.hIcon         = NULL;
   wc.cbClsExtra    = 4;
   wc.cbWndExtra    = 0;
   wc.lpszMenuName  = NULL;
   wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground = GetStockObject(LTGRAY_BRUSH);

   if (!RegisterClass(&wc))
      return FALSE;

   return TRUE;
}


/* CreateStatusWindow
 * ------------------
 *
 * Creates status window.
 *
 * HWND hwndMain
 *
 * RETURNS: HWND of status window if creation is successful.
 *          NULL otherwise.
 *
 * CUSTOMIZATION: Change class name.
 *
 */
HWND CreateStatusWindow(HWND hWndApp, HINSTANCE hInst)
{
   RECT rect;
   int width, height;
   HWND hWndStatusBar;

   lstrcpy (lpszStatusMessage, ControlList[0].string);
   GetClientRect(hWndApp, &rect);
   width = rect.right - rect.left;
   height = rect.bottom - rect.top;

   hWndStatusBar = CreateWindow (
      "ObjStatus",
      "SvrStatus",
      WS_CHILD |
      WS_CLIPSIBLINGS |
      WS_VISIBLE,
      0, height - STATUS_HEIGHT,
      width,
      STATUS_HEIGHT,
      hWndApp,
      NULL,
      hInst,
      NULL
   );

   return hWndStatusBar;
}


/* DestroyStatusWindow
 * -------------------
 *
 * Destroys status window.
 *
 * CUSTOMIZATION: None.
 *
 */
void DestroyStatusWindow(HWND hWndStatusBar)
{
   DestroyWindow(hWndStatusBar);
}


/* AssignPopupMessage
 * ------------------
 *
 * Associates a string with a popup menu handle.
 *
 * HMENU hmenuPopup
 * char *szMessage
 *
 * CUSTOMIZATION: None.
 *
 */
void AssignPopupMessage(HMENU hmenuPopup, char *szMessage)
{
   if (nCurrentPopup < NUM_POPUP) {
      PopupList[nCurrentPopup].hmenu = hmenuPopup;
      lstrcpy(PopupList[nCurrentPopup].string, szMessage);
      ++nCurrentPopup;
   }
}


/* SetStatusText
 * -------------
 *
 * Show the message in the status line.
 */
void SetStatusText(HWND hWndStatusBar, LPSTR lpszMessage)
{
   lstrcpy (lpszStatusMessage, lpszMessage);
   InvalidateRect (hWndStatusBar, (LPRECT)NULL,  TRUE);
   UpdateWindow (hWndStatusBar);
}


/* GetItemMessage
 * --------------
 *
 * Retrieve the message associated with the given menu command item number.
 *
 * UINT wIDItem
 * LPVOID lpDoc
 *
 * CUSTOMIZATION: None.
 *
 */
void GetItemMessage(UINT wIDItem, LPSTR FAR* lplpszMessage)
{
   UINT i;

   *lplpszMessage = ControlList[1].string;
   for (i = 0; i < NUM_STATS; ++i) {
      if (wIDItem == MesgList[i].wIDItem) {
         *lplpszMessage = MesgList[i].string;
         break;
      }
   }
}
```

```
/* GetPopupMessage
 * ---------------
 *
 * Retrieve the message associated with the given popup menu.
 *
 * HMENU hmenuPopup
 * LPVOID lpDoc
 *
 * CUSTOMIZATION: None.
 *
 */
void GetPopupMessage(HMENU hmenuPopup, LPSTR FAR* lplpszMessage)
{
   UINT i;

   *lplpszMessage = ControlList[1].string;
   for (i = 0; i < nCurrentPopup; ++i) {
      if (hmenuPopup == PopupList[i].hmenu) {
         *lplpszMessage = PopupList[i].string;
         break;
      }
   }
}


/* GetSysMenuMessage
 * -----------------
 *
 * Retrieves the messages to correspond to items on the system menu.
 *
 *
 * UINT wIDItem
 * LPVOID lpDoc
 *
 * CUSTOMIZATION: None.
 *
 */
void GetSysMenuMessage(UINT wIDItem, LPSTR FAR* lplpszMessage)
{
   UINT i;

   *lplpszMessage = ControlList[1].string;
   for (i = 0; i < 16; ++i) {
      if (wIDItem == SysMenuList[i].wIDItem) {
         *lplpszMessage = SysMenuList[i].string;
         break;
      }
   }
}


/* GetControlMessage
 * -----------------
 *
 * Retrieves the general system messages.
 *
 *
 * STATCONTROL scCommand
 * LPVOID lpDoc
 *
 * CUSTOMIZATION: Add new messages.
 *
 */
void GetControlMessage(STATCONTROL scCommand, LPSTR FAR* lplpszMessage)
{
   UINT i;

   *lplpszMessage = ControlList[1].string;
   for (i = 0; i < 2; ++i) {
      if ((UINT)scCommand == ControlList[i].wIDItem) {
         *lplpszMessage = ControlList[i].string;
         break;
      }
   }
}



/* StatusWndProc
 * -------------
 *
 * Message handler for the statusbar window.
 *
 *
 * CUSTOMIZATION: None
 *
 */
LRESULT FAR PASCAL StatusWndProc
   (HWND hwnd, unsigned message, WPARAM wParam, LPARAM lParam)
{
   if (message == WM_PAINT) {
      RECT        rc;
      HDC         hdc;
      PAINTSTRUCT paintstruct;
      HPEN        hpenOld;
      HPEN        hpen;
      HFONT       hfontOld;
      HFONT       hfont;
      HPALETTE    hpalOld = NULL;
      POINT       point;

      BeginPaint (hwnd, &paintstruct);
      hdc = GetDC (hwnd);

      GetClientRect (hwnd, (LPRECT) &rc);

      hpenOld = SelectObject (hdc, GetStockObject (BLACK_PEN));

      MoveToEx (hdc, 0, 0, &point);
      LineTo (hdc, rc.right, 0);
      SelectObject (hdc, GetStockObject (WHITE_PEN));

      MoveToEx (hdc, STATUS_RRIGHT, STATUS_RTOP, &point);
      LineTo (hdc, STATUS_RRIGHT, STATUS_RBOTTOM);
      LineTo (hdc, STATUS_RLEFT-1, STATUS_RBOTTOM);

      hpen = CreatePen (PS_SOLID, 1, /* DKGRAY */ 0x00808080);
      SelectObject (hdc, hpen);

      MoveToEx (hdc, STATUS_RLEFT, STATUS_RBOTTOM-1, &point);
      LineTo (hdc, STATUS_RLEFT, STATUS_RTOP);
      LineTo (hdc, STATUS_RRIGHT, STATUS_RTOP);

      SetBkMode (hdc, TRANSPARENT);
      SetTextAlign (hdc, TA_LEFT | TA_TOP);
      hfont = CreateFont (STATUS_THEIGHT, 0, 0, 0, FW_NORMAL, FALSE, FALSE,
                     FALSE, ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                     CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                     DEFAULT_PITCH | FF_DONTCARE, "MS Sans Serif");

      hfontOld = SelectObject(hdc, hfont);

      TextOut (hdc, STATUS_TLEFT, STATUS_TTOP,
             lpszStatusMessage,
             lstrlen(lpszStatusMessage));

      // Restore original objects
      SelectObject (hdc, hfontOld);
      SelectObject (hdc, hpenOld);
      DeleteObject (hpen);
      DeleteObject (hfont);

      ReleaseDC (hwnd, hdc);
      EndPaint (hwnd, &paintstruct);

      return 0;
   }
   else {
      return DefWindowProc(hwnd, message, wParam, lParam);
   }
}

_
```

SVRBASE.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Server Sample Code
**
**    svrbase.c
**
**    This file contains all interfaces, methods and related support
**    functions for the basic OLE Object (Server) application. The
**    basic OLE Object application supports embedding an object and
**    linking to a file-based or embedded object as a whole. The basic
**    Object application includes the following implementation objects:
**
**    ClassFactory (aka. ClassObject) Object    (see file classfac.c)
**      exposed interfaces:
**          IClassFactory interface
**
**    ServerDoc Object
**      exposed interfaces:
**          IUnknown
**          IOleObject interface
**          IPersistStorage interface
**          IDataObject interface
**
**    ServerApp Object
**      exposed interfaces:
**          IUnknown
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/


#include "outline.h"

OLEDBGDATA

extern LPOUTLINEAPP             g_lpApp;
extern IOleObjectVtbl           g_SvrDoc_OleObjectVtbl;
extern IPersistStorageVtbl      g_SvrDoc_PersistStorageVtbl;

#if defined( INPLACE_SVR )
extern IOleInPlaceObjectVtbl        g_SvrDoc_OleInPlaceObjectVtbl;
extern IOleInPlaceActiveObjectVtbl  g_SvrDoc_OleInPlaceActiveObjectVtbl;
#endif  // INPLACE_SVR

#if defined( SVR_TREATAS )
extern IStdMarshalInfoVtbl      g_SvrDoc_StdMarshalInfoVtbl;
#endif  // SVR_TREATAS


// REVIEW: should use string resource for messages
extern OLECHAR ErrMsgSaving[];
extern OLECHAR ErrMsgFormatNotSupported[];
static OLECHAR ErrMsgPSSaveFail[] = OLESTR("PSSave failed");
static OLECHAR ErrMsgLowMemNClose[] = OLESTR("Warning OUT OF MEMORY! We must
close down");
extern char g_szUpdateCntrDoc[] = "&Update %s";
extern char g_szExitNReturnToCntrDoc[] = "E&xit && Return to %s";


/*************************************************************************
** ServerDoc::IOleObject interface implementation
*************************************************************************/

// IOleObject::QueryInterface method

STDMETHODIMP SvrDoc_OleObj_QueryInterface(
      LPOLEOBJECT             lpThis,
      REFIID                  riid,
      LPVOID FAR*             lplpvObj
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;

   return OleDoc_QueryInterface((LPOLEDOC)lpServerDoc, riid, lplpvObj);
}


// IOleObject::AddRef method

STDMETHODIMP_(ULONG) SvrDoc_OleObj_AddRef(LPOLEOBJECT lpThis)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;

   OleDbgAddRefMethod(lpThis, "IOleObject");

   return OleDoc_AddRef((LPOLEDOC)lpServerDoc);
}


// IOleObject::Release method

STDMETHODIMP_(ULONG) SvrDoc_OleObj_Release(LPOLEOBJECT lpThis)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;

   OleDbgReleaseMethod(lpThis, "IOleObject");

   return OleDoc_Release((LPOLEDOC)lpServerDoc);
}


// IOleObject::SetClientSite method
```

```
STDMETHODIMP SvrDoc_OleObj_SetClientSite(
      LPOLEOBJECT             lpThis,
      LPOLECLIENTSITE         lpclientSite
)
{
      LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;

   // artificial AddRef in case object is destroyed during call
   SvrDoc_OleObj_AddRef(lpThis);

   OLEDBG_BEGIN2("SvrDoc_OleObj_SetClientSite\r\n")

   // SetClientSite is only valid to call on an embedded object
   if (lpOutlineDoc->m_docInitType != DOCTYPE_EMBEDDED) {
      OleDbgAssert(lpOutlineDoc->m_docInitType == DOCTYPE_EMBEDDED);
      OLEDBG_END2
      // release artificial AddRef
      SvrDoc_OleObj_Release(lpThis);
      return ResultFromScode(E_UNEXPECTED);
   }

   /* if we currently have a client site ptr, then release it. */
   if (lpServerDoc->m_lpOleClientSite)
      OleStdRelease((LPUNKNOWN)lpServerDoc->m_lpOleClientSite);

   lpServerDoc->m_lpOleClientSite = (LPOLECLIENTSITE) lpclientSite;
   // OLE2NOTE: to be able to hold onto clientSite pointer, we must AddRef
it
   if (lpclientSite)
      lpclientSite->lpVtbl->AddRef(lpclientSite);

   OLEDBG_END2
   // release artificial AddRef
   SvrDoc_OleObj_Release(lpThis);
   return NOERROR;
}


// IOleObject::GetClientSite method

STDMETHODIMP SvrDoc_OleObj_GetClientSite(
      LPOLEOBJECT             lpThis,
      LPOLECLIENTSITE FAR*    lplpClientSite
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;

   // artificial AddRef in case object is destroyed during call
   SvrDoc_OleObj_AddRef(lpThis);

   OleDbgOut2("SvrDoc_OleObj_GetClientSite\r\n");
```

```
    /* OLE2NOTE: we MUST AddRef this interface pointer to give the
    **    caller a personal copy of the pointer
    */
    lpServerDoc->m_lpOleClientSite->lpVtbl->AddRef(
          lpServerDoc->m_lpOleClientSite
    );
    *lplpClientSite = lpServerDoc->m_lpOleClientSite;

    // release artificial AddRef
    SvrDoc_OleObj_Release(lpThis);

    return NOERROR;

}


// IOleObject::SetHostNames method

STDMETHODIMP SvrDoc_OleObj_SetHostNames(
      LPOLEOBJECT            lpThis,
      LPCOLESTR              szContainerApp,
      LPCOLESTR              szContainerObj
)
{
    LPSERVERDOC lpServerDoc =
          ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
    LPOUTLINEDOC    lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
    char szAnsiStr1[256], szAnsiStr2[256], szAnsiStr3[256];

    // artificial AddRef in case object is destroyed during call
    SvrDoc_OleObj_AddRef(lpThis);

    OleDbgOut2("SvrDoc_OleObj_SetHostNames\r\n");

    OLESTRCPY((LPOLESTR)lpServerDoc->m_szContainerApp, szContainerApp);
    OLESTRCPY((LPOLESTR)lpServerDoc->m_szContainerObj, szContainerObj);

    /* The Window title for an embedded object is constructed as
    **    follows:
    **      <server app name> - <obj short type> in <cont. doc name>
    **
    **    here we construct the current document title portion of the
    **    name which follows the '-'. OutlineDoc_SetTitle prepends the
    **    "<server app name> - " to the document title.
    */
    // REVIEW: this string should be loaded from string resource
    W2A (lpServerDoc->m_szContainerObj, szAnsiStr1, 256);
    W2A (SHORTUSERTYPENAME, szAnsiStr3, 256);
    wsprintf(szAnsiStr2, "%s in %s",
          szAnsiStr3, (LPSTR)szAnsiStr1);
    A2W (szAnsiStr2, lpOutlineDoc->m_szFileName, OLEUI_CCHPATHMAX);

    lpOutlineDoc->m_lpszDocTitle = lpOutlineDoc->m_szFileName;
    OutlineDoc_SetTitle(lpOutlineDoc, FALSE /*fMakeUpperCase*/);
    /* OLE2NOTE: update the application menus correctly for an embedded
    **    object. the changes include:
    **      1 Remove File/New and File/Open (SDI ONLY)
    **      2 Change File/Save As.. to File/Save Copy As..
    **      3 Change File menu so it contains "Update" instead of "Save"
    **      4 Change File/Exit to File/Exit & Return to <client doc>"
    */
    ServerDoc_UpdateMenu(lpServerDoc);

    // release artificial AddRef
    SvrDoc_OleObj_Release(lpThis);

    return NOERROR;
}


// IOleObject::Close method

STDMETHODIMP SvrDoc_OleObj_Close(
      LPOLEOBJECT             lpThis,
      DWORD                   dwSaveOption
)
{
    LPSERVERDOC lpServerDoc =
          ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
    BOOL fStatus;

    OLEDBG_BEGIN2("SvrDoc_OleObj_Close\r\n")

    /* OLE2NOTE: the OLE 2.0 user model is that embedded objects should
    **    always be saved when closed WITHOUT any prompting to the
    **    user. this is the recommendation irregardless of whether the
    **    object is activated in-place or open in its own window.
    **    this is a CHANGE from the OLE 1.0 user model where it
    **    was the guideline that servers always prompt to save changes.
    **    thus OLE 2.0 compound document oriented container's should
    **    always pass dwSaveOption==OLECLOSE_SAVEIFDIRTY. it is
    **    possible that for programmatic uses a container may want to
    **    specify a different dwSaveOption. the implementation of
    **    various save options can be tricky, particularly considering
    **    cases involving in-place activation. the following would be
    **    reasonable behavior:
    **
    **      (1) OLECLOSE_SAVEIFDIRTY: if dirty, save. close.
    **      (2) OLECLOSE_NOSAVE: close.
    **      (3) OLECLOSE_PROMPTSAVE:
    **        (a) object visible, but not in-place:
    **              if not dirty, close.
    **              switch(prompt)
    **                 case IDYES: save. close.
    **                 case IDNO: close.
    **                 case IDCANCEL: return OLE_E_PROMPTSAVECANCELLED
    **        (b) object invisible (includes UIDeactivated object)
    **              if dirty, save. close.
    **              NOTE: NO PROMPT. it is not appropriate to prompt
```

```
   **                       if the object is not visible.
   **         (c) object is in-place active:
   **                if dirty, save. close.
   **                NOTE: NO PROMPT. it is not appropriate to prompt
   **                      if the object is active in-place.
   */
   fStatus = OutlineDoc_Close((LPOUTLINEDOC)lpServerDoc, dwSaveOption);
   OleDbgAssertSz(fStatus == TRUE, "SvrDoc_OleObj_Close failed\r\n");

   OLEDBG_END2

   return (fStatus ? NOERROR : ResultFromScode(E_FAIL));
}


// IOleObject::SetMoniker method

STDMETHODIMP SvrDoc_OleObj_SetMoniker(
      LPOLEOBJECT             lpThis,
      DWORD                   dwWhichMoniker,
      LPMONIKER               lpmk
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpServerDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPMONIKER lpmkFull = NULL;
   HRESULT hrErr;
   SCODE sc;

   // artificial AddRef in case object is destroyed during call
   SvrDoc_OleObj_AddRef(lpThis);

   OLEDBG_BEGIN2("SvrDoc_OleObj_SetMoniker\r\n")

   /* OLE2NOTE: if our full moniker is passed then we can use it,
   **    otherwise we must call back to our ClientSite to get our full
   **    moniker.
   */
   if (dwWhichMoniker == OLEWHICHMK_OBJFULL) {

      /* Register the document as running with the new moniker and
      **      notify any clients that our moniker has changed.
      */
      OleDoc_DocRenamedUpdate(lpOleDoc, lpmk);

      if (lpOutlineDoc->m_docInitType != DOCTYPE_EMBEDDED) {
         IBindCtx  FAR  *pbc = NULL;
         LPOLESTR lpszName = NULL;

         /* OLE2NOTE: if this is a FILE-based or untitled document
         **    then we should accept this new moniker as our document's
         **    moniker. we will remember this moniker instead of the
         **    FileMoniker that we have by default. this allows
         **    systems that use special monikers to track the
         **    location of documents to inform a document that is a
         **    link source of its special moniker. this enables the
         **    document to use this special moniker when building
         **    composite monikers to identify contained objects and
         **    pseudo objects (ranges).
         **
         **    we should also use the DisplayName form of this
         **    moniker as our document name in our window title.
         */
         if (lpOleDoc->m_lpFileMoniker) {
            lpOleDoc->m_lpFileMoniker->lpVtbl->Release(
                  lpOleDoc->m_lpFileMoniker);
         }
         lpOleDoc->m_lpFileMoniker = lpmk;
         // we must AddRef the moniker to hold on to it
         lpmk->lpVtbl->AddRef(lpmk);

         /* we should also use the DisplayName form of this
         **    moniker as our document name in our window title.
         */
         CreateBindCtx(0, (LPBC FAR*)&pbc);
         lpmk->lpVtbl->GetDisplayName(lpmk,pbc,NULL,&lpszName);
         pbc->lpVtbl->Release(pbc);
         if (lpszName) {
            OLESTRCPY(lpOutlineDoc->m_szFileName, lpszName);
            lpOutlineDoc->m_lpszDocTitle = lpOutlineDoc->m_szFileName;
            OutlineDoc_SetTitle(lpOutlineDoc, FALSE /*fMakeUpperCase*/);
            OleStdFreeString(lpszName, NULL);
         }
      }

      OLEDBG_END2
      // release artificial AddRef
      SvrDoc_OleObj_Release(lpThis);
      return NOERROR;
   }

   /* if the passed moniker was NOT a full moniker then we must call
   **    back to our ClientSite to get our full moniker. this is
   **    needed in order to register in the RunningObjectTable. if we
   **    don't have a ClientSite then this is an error.
   */
   if (lpServerDoc->m_lpOleClientSite == NULL) {
      sc = E_FAIL;
      goto error;
   }

   hrErr = lpServerDoc->m_lpOleClientSite->lpVtbl->GetMoniker(
         lpServerDoc->m_lpOleClientSite,
         OLEGETMONIKER_ONLYIFTHERE,
         OLEWHICHMK_OBJFULL,
         &lpmkFull
   );
   if (hrErr != NOERROR) {
      sc = GetScode(hrErr);
      goto error;
   }

   /* Register the document as running with the new moniker and
   **      notify any clients that our moniker has changed.
   */
   OleDoc_DocRenamedUpdate(lpOleDoc, lpmkFull);

   if (lpmkFull)
      OleStdRelease((LPUNKNOWN)lpmkFull);

   OLEDBG_END2

   // release artificial AddRef
   SvrDoc_OleObj_Release(lpThis);
   return NOERROR;

error:
   OLEDBG_END2

   // release artificial AddRef
   SvrDoc_OleObj_Release(lpThis);
   return ResultFromScode(sc);
}


// IOleObject::GetMoniker method

STDMETHODIMP SvrDoc_OleObj_GetMoniker(
      LPOLEOBJECT             lpThis,
      DWORD                   dwAssign,
      DWORD                   dwWhichMoniker,
      LPMONIKER FAR*          lplpmk
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpServerDoc;
   SCODE sc;

   // artificial AddRef in case object is destroyed during call
   SvrDoc_OleObj_AddRef(lpThis);

   OLEDBG_BEGIN2("SvrDoc_OleObj_GetMoniker\r\n")

   /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
   *lplpmk = NULL;

   if (lpServerDoc->m_lpOleClientSite) {

      /* document is an embedded object. retrieve our moniker from
      **    our container.
      */
      OLEDBG_BEGIN2("IOleClientSite::GetMoniker called\r\n")
      sc = GetScode( lpServerDoc->m_lpOleClientSite->lpVtbl->GetMoniker(
            lpServerDoc->m_lpOleClientSite,
            dwAssign,
            dwWhichMoniker,
            lplpmk
      ) );
      OLEDBG_END2

   } else if (lpOleDoc->m_lpFileMoniker) {

      /* document is a top-level user document (either
      **    file-based or untitled). return the FileMoniker stored
      **    with the document; it uniquely identifies the document.
      */
      if (dwWhichMoniker == OLEWHICHMK_CONTAINER)
         sc = E_INVALIDARG;  // file-based object has no CONTAINER moniker
      else {
         *lplpmk = lpOleDoc->m_lpFileMoniker;
         (*lplpmk)->lpVtbl->AddRef(*lplpmk); // must AddRef to pass out ptr
         sc = S_OK;
      }

   } else {
      // document is not yet fully initialized => no moniker
      sc = E_FAIL;
   }

   OLEDBG_END2

   // release artificial AddRef
   SvrDoc_OleObj_Release(lpThis);
   return ResultFromScode(sc);
}


// IOleObject::InitFromData method

STDMETHODIMP SvrDoc_OleObj_InitFromData(
      LPOLEOBJECT             lpThis,
      LPDATAOBJECT            lpDataObject,
      BOOL                    fCreation,
      DWORD                   reserved
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;

   OLEDBG_BEGIN2("SvrDoc_OleObj_InitFromData\r\n")

   // REVIEW: NOT YET IMPLEMENTED

   OLEDBG_END2
   return ResultFromScode(E_NOTIMPL);
}
```

```
// IOleObject::GetClipboardData method

STDMETHODIMP SvrDoc_OleObj_GetClipboardData(
      LPOLEOBJECT             lpThis,
      DWORD                   reserved,
      LPDATAOBJECT FAR*       lplpDataObject
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;

   OLEDBG_BEGIN2("SvrDoc_OleObj_GetClipboardData\r\n")

   // REVIEW: NOT YET IMPLEMENTED

   OLEDBG_END2
   return ResultFromScode(E_NOTIMPL);
}


// IOleObject::DoVerb method

STDMETHODIMP SvrDoc_OleObj_DoVerb(
      LPOLEOBJECT             lpThis,
      LONG                    lVerb,
      LPMSG                   lpmsg,
      LPOLECLIENTSITE         lpActiveSite,
      LONG                    lindex,
      HWND                    hwndParent,
      LPCRECT                 lprcPosRect
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   SCODE sc = S_OK;

   // artificial AddRef in case object is destroyed during call
   SvrDoc_OleObj_AddRef(lpThis);

   OLEDBG_BEGIN2("SvrDoc_OleObj_DoVerb\r\n")

   switch (lVerb) {

      default:
         /* OLE2NOTE: when an unknown verb number is given, the
         **    server must take careful action:
         **    1. if it is one of the specially defined OLEIVERB
         **    (negative numbered) verbs, the app should return an
         **    error (E_NOTIMPL) and perform no action.
         **
         **    2. if the verb is a application specific verb
         **    (positive numbered verb), then the app should
         **    return the special scode (OLEOBJ_S_INVALIDVERB). BUT,
         **    we should still perform our normal primary verb action.
         */
         if (lVerb < 0) {
            OLEDBG_END2
            return ResultFromScode(E_NOTIMPL);
         } else {
            sc = OLEOBJ_S_INVALIDVERB;
         }

         // deliberatly fall through to Primary Verb

#if !defined( INPLACE_SVR )
      case 0:
      case OLEIVERB_SHOW:
      case OLEIVERB_OPEN:
         OutlineDoc_ShowWindow(lpOutlineDoc);
         break;

      case OLEIVERB_HIDE:
         OleDoc_HideWindow((LPOLEDOC)lpServerDoc, FALSE /*fShutdown*/);
         break;
#endif  // ! INPLACE_SVR
#if defined( INPLACE_SVR )
      case 0:
      case OLEIVERB_SHOW:

         /* OLE2NOTE: if our window is already open (visible) then
         **    we should simply surface the open window. if not,
         **    then we can do our primary action of in-place
         **    activation.
         */
         if ( lpServerDoc->m_lpOleClientSite
               && ! (IsWindowVisible(lpOutlineDoc->m_hWndDoc) &&
                     ! lpServerDoc->m_fInPlaceActive) ) {
            ServerDoc_DoInPlaceActivate(
                  lpServerDoc, lVerb, lpmsg, lpActiveSite);
         }
         OutlineDoc_ShowWindow(lpOutlineDoc);
         break;

      case 1:
      case OLEIVERB_OPEN:
         ServerDoc_DoInPlaceDeactivate(lpServerDoc);
         OutlineDoc_ShowWindow(lpOutlineDoc);
         break;


      case OLEIVERB_HIDE:
         if (lpServerDoc->m_fInPlaceActive) {

            SvrDoc_IPObj_UIDeactivate(
                  (LPOLEINPLACEOBJECT)&lpServerDoc->m_OleInPlaceObject);

#if defined( SVR_INSIDEOUT )
            /* OLE2NOTE: an inside-out style in-place server will
            **    NOT hide its window in UIDeactive (an outside-in
            **    style object will hide its window in
            **    UIDeactivate). thus we need to explicitly hide
            **    our window now.
            */
            ServerDoc_DoInPlaceHide(lpServerDoc);
#endif // INSIEDOUT

         } else {
            OleDoc_HideWindow((LPOLEDOC)lpServerDoc, FALSE /*fShutdown*/);
         }
         break;

      case OLEIVERB_UIACTIVATE:

#if defined( SVR_INSIDEOUT )
      /* OLE2NOTE: only an inside-out style object supports
      **    INPLACEACTIVATE verb
      */
      case OLEIVERB_INPLACEACTIVATE:
#endif // SVR_INSIDEOUT

         /* OLE2NOTE: if our window is already open (visible) then
         **    we can NOT activate in-place.
         */
         if (IsWindowVisible(lpOutlineDoc->m_hWndDoc) &&
                  ! lpServerDoc->m_fInPlaceActive ) {
            sc = OLE_E_NOT_INPLACEACTIVE;
         } else {
            sc = GetScode( ServerDoc_DoInPlaceActivate(
                  lpServerDoc, lVerb, lpmsg, lpActiveSite) );
            if (SUCCEEDED(sc))
               OutlineDoc_ShowWindow(lpOutlineDoc);
         }
         break;
#endif  // INPLACE_SVR
   }

   OLEDBG_END2

   // release artificial AddRef
   SvrDoc_OleObj_Release(lpThis);
   return ResultFromScode(sc);
}


// IOleObject::EnumVerbs method

STDMETHODIMP SvrDoc_OleObj_EnumVerbs(
      LPOLEOBJECT             lpThis,
      LPENUMOLEVERB FAR*      lplpenumOleVerb
)
{
   OleDbgOut2("SvrDoc_OleObj_EnumVerbs\r\n");
   /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
   *lplpenumOleVerb = NULL;

   /* An object implemented as a server EXE (as this sample
   **    is) may simply return OLE_S_USEREG to instruct the OLE
   **    DefHandler to call the OleReg* helper API which uses info in
   **    the registration database. Alternatively, the OleRegEnumVerbs
   **    API may be called directly. Objects implemented as a server
   **    DLL may NOT return OLE_S_USEREG; they must call the OleReg*
   **    API or provide their own implementation. For EXE based
   **    objects it is more efficient to return OLE_S_USEREG, because
   **    in then the verb enumerator is instantiated in the callers
   **    process space and no LRPC remoting is required.
   */
   return ResultFromScode(OLE_S_USEREG);
}


// IOleObject::Update method

STDMETHODIMP SvrDoc_OleObj_Update(LPOLEOBJECT lpThis)
{
   OleDbgOut2("SvrDoc_OleObj_Update\r\n");

   /* OLE2NOTE: a server-only app is always "up-to-date".
   **    a container-app which contains links where the link source
   **    has changed since the last update of the link would be
   **    considered "out-of-date". the "Update" method instructs the
   **    object to get an update from any out-of-date links.
   */

   return NOERROR;
}


// IOleObject::IsUpToDate method

STDMETHODIMP SvrDoc_OleObj_IsUpToDate(LPOLEOBJECT lpThis)
{
   OleDbgOut2("SvrDoc_OleObj_IsUpToDate\r\n");

   /* OLE2NOTE: a server-only app is always "up-to-date".
   **    a container-app which contains links where the link source
   **    has changed since the last update of the link would be
   **    considered "out-of-date".
   */
   return NOERROR;
}


// IOleObject::GetUserClassID method

STDMETHODIMP SvrDoc_OleObj_GetUserClassID(
      LPOLEOBJECT             lpThis,
      LPCLSID                 lpClassID
```

```
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   OleDbgOut2("SvrDoc_OleObj_GetClassID\r\n");

   /* OLE2NOTE: we must be carefull to return the correct CLSID here.
   **    if we are currently preforming a "TreatAs (aka. ActivateAs)"
   **    operation then we need to return the class of the object
   **    written in the storage of the object. otherwise we would
   **    return our own class id.
   */
   return ServerDoc_GetClassID(lpServerDoc, lpClassID);
}


// IOleObject::GetUserType method

STDMETHODIMP SvrDoc_OleObj_GetUserType(
      LPOLEOBJECT             lpThis,
      DWORD                   dwFormOfType,
      LPOLESTR FAR*           lpszUserType
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   OleDbgOut2("SvrDoc_OleObj_GetUserType\r\n");

   /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
   *lpszUserType = NULL;

   /* OLE2NOTE: we must be carefull to return the correct user type here.
   **    if we are currently preforming a "TreatAs (aka. ActivateAs)"
   **    operation then we need to return the user type name that
   **    corresponds to the class of the object we are currently
   **    emmulating. otherwise we should return our normal user type
   **    name corresponding to our own class. This routine determines
   **    the current clsid in effect.
   **
   **    An object implemented as a server EXE (as this sample
   **    is) may simply return OLE_S_USEREG to instruct the OLE
   **    DefHandler to call the OleReg* helper API which uses info in
   **    the registration database. Alternatively, the OleRegGetUserType
   **    API may be called directly. Objects implemented as a server
   **    DLL may NOT return OLE_S_USEREG; they must call the OleReg*
   **    API or provide their own implementation. For EXE based
   **    objects it is more efficient to return OLE_S_USEREG, because
   **    in then the return string is instantiated in the callers
   **    process space and no LRPC remoting is required.
   */
#if defined( SVR_TREATAS )
   if (! IsEqualCLSID(&lpServerDoc->m_clsidTreatAs, &CLSID_NULL) )
      return OleRegGetUserType(
         (REFCLSID)&lpServerDoc->m_clsidTreatAs,dwFormOfType,lpszUserType);
   else
```

```
#endif  // SVR_TREATAS

   return ResultFromScode(OLE_S_USEREG);
}


// IOleObject::SetExtent method

STDMETHODIMP SvrDoc_OleObj_SetExtent(
      LPOLEOBJECT             lpThis,
      DWORD                   dwDrawAspect,
      LPSIZEL                 lplgrc
)
{
   OleDbgOut2("SvrDoc_OleObj_SetExtent\r\n");

   /* SVROUTL does NOT allow the object's size to be set by its
   **    container. the size of the ServerDoc object is determined by
   **    the data contained within the document.
   */
   return ResultFromScode(E_FAIL);
}


// IOleObject::GetExtent method

STDMETHODIMP SvrDoc_OleObj_GetExtent(
      LPOLEOBJECT             lpThis,
      DWORD                   dwDrawAspect,
      LPSIZEL                 lpsizel
)
{
   LPOLEDOC lpOleDoc =
         (LPOLEDOC)((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;

   // artificial AddRef in case object is destroyed during call
   SvrDoc_OleObj_AddRef(lpThis);

   OleDbgOut2("SvrDoc_OleObj_GetExtent\r\n");

   /* OLE2NOTE: it is VERY important to check which aspect the caller
   **    is asking about. an object implemented by a server EXE MAY
   **    fail to return extents when asked for DVASPECT_ICON.
   */
   if (dwDrawAspect == DVASPECT_CONTENT) {
      OleDoc_GetExtent(lpOleDoc, lpsizel);
      // release artificial AddRef
      SvrDoc_OleObj_Release(lpThis);
      return NOERROR;
   }

#if defined( LATER )

   else if (dwDrawAspect == DVASPECT_THUMBNAIL)
   {
      /* as our thumbnail we will render only the first page of the
      **    document. calculate extents of our thumbnail rendering.
      **
      ** OLE2NOTE: thumbnails are most often used by applications in
      **    FindFile or FileOpen type dialogs to give the user a
      **    quick view of the contents of the file or object.
      */
      OleDoc_GetThumbnailExtent(lpOleDoc, lpsizel);
      // release artificial AddRef
      SvrDoc_OleObj_Release(lpThis);
      return NOERROR;
   }
#endif

   else
   {
      // release artificial AddRef
      SvrDoc_OleObj_Release(lpThis);
      return ResultFromScode(E_FAIL);
   }
}


// IOleObject::Advise method

STDMETHODIMP SvrDoc_OleObj_Advise(
      LPOLEOBJECT             lpThis,
      LPADVISESINK            lpAdvSink,
      LPDWORD                 lpdwConnection
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   HRESULT hrErr;
   SCODE   sc;

   // artificial AddRef in case object is destroyed during call
   SvrDoc_OleObj_AddRef(lpThis);

   OLEDBG_BEGIN2("SvrDoc_OleObj_Advise\r\n");

   if (lpServerDoc->m_OleDoc.m_fObjIsClosing)
   {
       //  We don't accept any more Advise's once we're closing
       sc = OLE_E_ADVISENOTSUPPORTED;
       goto error;
   }

   if (lpServerDoc->m_lpOleAdviseHldr == NULL &&
      CreateOleAdviseHolder(&lpServerDoc->m_lpOleAdviseHldr) != NOERROR) {
      sc = E_OUTOFMEMORY;
      goto error;
   }

   OLEDBG_BEGIN2("IOleAdviseHolder::Advise called\r\n")
   hrErr = lpServerDoc->m_lpOleAdviseHldr->lpVtbl->Advise(
         lpServerDoc->m_lpOleAdviseHldr,
         lpAdvSink,
         lpdwConnection
   );
   OLEDBG_END2

   OLEDBG_END2
   // release artificial AddRef
   SvrDoc_OleObj_Release(lpThis);
   return hrErr;

error:
   OLEDBG_END2
   *lpdwConnection = 0;
   // release artificial AddRef
   SvrDoc_OleObj_Release(lpThis);
   return ResultFromScode(sc);
}


// IOleObject::Unadvise method

STDMETHODIMP SvrDoc_OleObj_Unadvise(LPOLEOBJECT lpThis, DWORD dwConnection)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   HRESULT hrErr;
   SCODE   sc;

   // artificial AddRef in case object is destroyed during call
   SvrDoc_OleObj_AddRef(lpThis);

   OLEDBG_BEGIN2("SvrDoc_OleObj_Unadvise\r\n");

   if (lpServerDoc->m_lpOleAdviseHldr == NULL) {
      sc = E_FAIL;
      goto error;
   }

   OLEDBG_BEGIN2("IOleAdviseHolder::Unadvise called\r\n")
   hrErr = lpServerDoc->m_lpOleAdviseHldr->lpVtbl->Unadvise(
         lpServerDoc->m_lpOleAdviseHldr,
         dwConnection
   );
   OLEDBG_END2

   OLEDBG_END2

   // release artificial AddRef
   SvrDoc_OleObj_Release(lpThis);
   return hrErr;

error:
   OLEDBG_END2
   // release artificial AddRef
   SvrDoc_OleObj_Release(lpThis);
   return ResultFromScode(sc);
}


// IOleObject::EnumAdvise method

STDMETHODIMP SvrDoc_OleObj_EnumAdvise(
      LPOLEOBJECT             lpThis,
      LPENUMSTATDATA FAR*     lplpenumAdvise
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   HRESULT hrErr;
   SCODE   sc;

   // artificial AddRef in case object is destroyed during call
   SvrDoc_OleObj_AddRef(lpThis);

   OLEDBG_BEGIN2("SvrDoc_OleObj_EnumAdvise\r\n");

   /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
   *lplpenumAdvise = NULL;

   if (lpServerDoc->m_lpOleAdviseHldr == NULL) {
      sc = E_FAIL;
      goto error;
   }

   OLEDBG_BEGIN2("IOleAdviseHolder::EnumAdvise called\r\n")
   hrErr = lpServerDoc->m_lpOleAdviseHldr->lpVtbl->EnumAdvise(
         lpServerDoc->m_lpOleAdviseHldr,
         lplpenumAdvise
   );
   OLEDBG_END2

   OLEDBG_END2

   // release artificial AddRef
   SvrDoc_OleObj_Release(lpThis);
   return hrErr;

error:
   OLEDBG_END2

   // release artificial AddRef
   SvrDoc_OleObj_Release(lpThis);
   return ResultFromScode(sc);
}


// IOleObject::GetMiscStatus method
```

```
STDMETHODIMP SvrDoc_OleObj_GetMiscStatus(
      LPOLEOBJECT             lpThis,
      DWORD                   dwAspect,
      DWORD FAR*              lpdwStatus
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;

   // artificial AddRef in case object is destroyed during call
   SvrDoc_OleObj_AddRef(lpThis);

   OleDbgOut2("SvrDoc_OleObj_GetMiscStatus\r\n");

   /* Get our default MiscStatus for the given Aspect. this
   **    information is registered in the RegDB. We query the RegDB
   **    here to guarantee that the value returned from this method
   **    agrees with the values in RegDB. in this way we only have to
   **    maintain the info in one place (in the RegDB). Alternatively
   **    we could have the values hard coded here.
   */
   OleRegGetMiscStatus((REFCLSID)&CLSID_APP, dwAspect, lpdwStatus);

   /* OLE2NOTE: check if the data copied is compatible to be
   **    linked by an OLE 1.0 container. it is compatible if
   **    either the data is an untitled document, a file, or a
   **    selection of data within a file. if the data is part of
   **    an embedded object, then it is NOT compatible to be
   **    linked by an OLE 1.0 container. if it is compatible then
   **    we must include OLEMISC_CANLINKBYOLE1 as part of the
   **    dwStatus flags transfered via CF_OBJECTDESCRIPTOR or
   **    CF_LINKSRCDESCRIPTOR.
   */
   if (lpOutlineDoc->m_docInitType == DOCTYPE_NEW ||
      lpOutlineDoc->m_docInitType == DOCTYPE_FROMFILE)
      *lpdwStatus |= OLEMISC_CANLINKBYOLE1;

#if defined( INPLACE_SVR )
   if (dwAspect == DVASPECT_CONTENT)
      *lpdwStatus |= (OLEMISC_INSIDEOUT | OLEMISC_ACTIVATEWHENVISIBLE);
#endif  // INPLACE_SVR

   // release artificial AddRef
   SvrDoc_OleObj_Release(lpThis);
   return NOERROR;
}


// IOleObject::SetColorScheme method

STDMETHODIMP SvrDoc_OleObj_SetColorScheme(
      LPOLEOBJECT             lpThis,
      LPLOGPALETTE            lpLogpal
)
{
   OleDbgOut2("SvrDoc_OleObj_SetColorScheme\r\n");

   // REVIEW: NOT YET IMPLEMENTED

   return ResultFromScode(E_NOTIMPL);
}


/*************************************************************************
** ServerDoc::IPersistStorage interface implementation
*************************************************************************/

// IPersistStorage::QueryInterface method

STDMETHODIMP SvrDoc_PStg_QueryInterface(
      LPPERSISTSTORAGE        lpThis,
      REFIID                  riid,
      LPVOID FAR*             lplpvObj
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocPersistStorageImpl FAR*)lpThis)->lpServerDoc;

   return OleDoc_QueryInterface((LPOLEDOC)lpServerDoc, riid, lplpvObj);
}


// IPersistStorage::AddRef method

STDMETHODIMP_(ULONG) SvrDoc_PStg_AddRef(LPPERSISTSTORAGE lpThis)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocPersistStorageImpl FAR*)lpThis)->lpServerDoc;

   OleDbgAddRefMethod(lpThis, "IPersistStorage");

   return OleDoc_AddRef((LPOLEDOC)lpServerDoc);
}


// IPersistStorage::Release method

STDMETHODIMP_(ULONG) SvrDoc_PStg_Release(LPPERSISTSTORAGE lpThis)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocPersistStorageImpl FAR*)lpThis)->lpServerDoc;

   OleDbgReleaseMethod(lpThis, "IPersistStorage");

   return OleDoc_Release((LPOLEDOC)lpServerDoc);
}
```

```
// IPersistStorage::GetClassID method

STDMETHODIMP SvrDoc_PStg_GetClassID(
      LPPERSISTSTORAGE        lpThis,
      LPCLSID                 lpClassID
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocPersistStorageImpl FAR*)lpThis)->lpServerDoc;
   OleDbgOut2("SvrDoc_PStg_GetClassID\r\n");

   /* OLE2NOTE: we must be carefull to return the correct CLSID here.
   **    if we are currently preforming a "TreatAs (aka. ActivateAs)"
   **    operation then we need to return the class of the object
   **    written in the storage of the object. otherwise we would
   **    return our own class id.
   */
   return ServerDoc_GetClassID(lpServerDoc, lpClassID);
}


// IPersistStorage::IsDirty method

STDMETHODIMP  SvrDoc_PStg_IsDirty(LPPERSISTSTORAGE  lpThis)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocPersistStorageImpl FAR*)lpThis)->lpServerDoc;
   OleDbgOut2("SvrDoc_PStg_IsDirty\r\n");

   if (OutlineDoc_IsModified((LPOUTLINEDOC)lpServerDoc))
      return NOERROR;
   else
      return ResultFromScode(S_FALSE);
}



// IPersistStorage::InitNew method

STDMETHODIMP SvrDoc_PStg_InitNew(
      LPPERSISTSTORAGE        lpThis,
      LPSTORAGE               lpStg
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocPersistStorageImpl FAR*)lpThis)->lpServerDoc;
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpServerDoc;
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPOLESTR    lpszUserType = (LPOLESTR)FULLUSERTYPENAME;
   HRESULT hrErr;
   SCODE sc;

   OLEDBG_BEGIN2("SvrDoc_PStg_InitNew\r\n")

#if defined( SVR_TREATAS )
```

```
    {
       LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
       CLSID       clsid;
       CLIPFORMAT  cfFmt;
       LPOLESTR       lpszType;

       /* OLE2NOTE: if the Server is capable of supporting "TreatAs"
       **    (aka. ActivateAs), it must read the class that is written
       **    into the storage. if this class is NOT the app's own
       **    class ID, then this is a TreatAs operation. the server
       **    then must faithfully pretend to be the class that is
       **    written into the storage. it must also faithfully write
       **    the data back to the storage in the SAME format as is
       **    written in the storage.
       **
       **    SVROUTL and ISVROTL can emulate each other. they have the
       **    simplification that they both read/write the identical
       **    format. thus for these apps no actual conversion of the
       **    native bits is actually required.
       */
       lpServerDoc->m_clsidTreatAs = CLSID_NULL;
       if (OleStdGetTreatAsFmtUserType(&CLSID_APP, lpStg, &clsid,
                     (CLIPFORMAT FAR*)&cfFmt, (LPOLESTR FAR*)&lpszType)) {

          if (cfFmt == lpOutlineApp->m_cfOutline) {
             // We should perform TreatAs operation
             if (lpServerDoc->m_lpszTreatAsType)
                OleStdFreeString(lpServerDoc->m_lpszTreatAsType, NULL);

             lpServerDoc->m_clsidTreatAs = clsid;
             ((LPOUTLINEDOC)lpServerDoc)->m_cfSaveFormat = cfFmt;
             lpServerDoc->m_lpszTreatAsType = lpszType;
             lpszUserType = lpServerDoc->m_lpszTreatAsType;

             OleDbgOut3("SvrDoc_PStg_InitNew: TreateAs ==> '");
//             OleDbgOutNoPrefix3(lpServerDoc->m_lpszTreatAsType);
             OleDbgOutNoPrefix3("'\r\n");
          } else {
             // ERROR: we ONLY support TreatAs for CF_OUTLINE format
             OleDbgOut("SvrDoc_PStg_InitNew: INVALID TreatAs Format\r\n");
             OleStdFreeString(lpszType, NULL);
          }
       }
    }
#endif  // SVR_TREATAS

    /* OLE2NOTE: a server EXE object should write its format tag to its
    **    storage in InitNew so that the DefHandler can know the format
    **    of the object. this is particularly important if the objects
    **    uses CF_METATFILE or CF_DIB as its format. the DefHandler
    **    automatically avoids separately storing presentation cache
    **    data when the object's native data is a standard presentation
    **    format.
    */
```

```
   WriteFmtUserTypeStg(lpStg,(CLIPFORMAT)lpOutlineApp-
>m_cfOutline,lpszUserType);

   // set the doc to a new embedded object.
   if (! ServerDoc_InitNewEmbed(lpServerDoc)) {
      sc = E_FAIL;
      goto error;
   }

   /* OLE2NOTE: An embedded object must guarantee that it can save
   **    even in low memory situations. it must be able to
   **    successfully save itself without consuming any additional
   **    memory. this means that a server is NOT supposed to open or
   **    create any streams or storages when
   **    IPersistStorage::Save(fSameAsLoad==TRUE) is called. thus an
   **    embedded object should hold onto its storage and pre-open and
   **    hold open any streams that it will need later when it is time
   **    to save.
   */
   hrErr = lpStg->lpVtbl->CreateStream(
         lpStg,
         OLESTR("LineList"),
         STGM_WRITE | STGM_SHARE_EXCLUSIVE | STGM_CREATE,
         0,
         0,
         &lpOleDoc->m_lpLLStm
   );

   if (hrErr != NOERROR) {
      OleDbgAssertSz(hrErr==NOERROR,"Could not create LineList stream");
      OleDbgOutHResult("LineList CreateStream returned", hrErr);
      sc = GetScode(hrErr);
      goto error;
   }

   hrErr = lpStg->lpVtbl->CreateStream(
         lpStg,
         OLESTR("NameTable"),
         STGM_WRITE | STGM_SHARE_EXCLUSIVE | STGM_CREATE,
         0,
         0,
         &lpOleDoc->m_lpNTStm
   );

   if (hrErr != NOERROR) {
      OleDbgAssertSz(hrErr==NOERROR,"Could not create NameTable stream");
      OleDbgOutHResult("NameTable CreateStream returned", hrErr);
      sc = GetScode(hrErr);
      goto error;
   }

   lpOleDoc->m_lpStg = lpStg;

   // OLE2NOTE: to be able to hold onto IStorage* pointer, we must AddRef it
   lpStg->lpVtbl->AddRef(lpStg);
   OLEDBG_END2

   return NOERROR;

error:
   OLEDBG_END2

   return ResultFromScode(sc);
}


// IPersistStorage::Load method

STDMETHODIMP SvrDoc_PStg_Load(
      LPPERSISTSTORAGE        lpThis,
      LPSTORAGE               lpStg
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocPersistStorageImpl FAR*)lpThis)->lpServerDoc;
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpServerDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   SCODE sc;
   HRESULT hrErr;

   OLEDBG_BEGIN2("SvrDoc_PStg_Load\r\n")

   if (OutlineDoc_LoadFromStg((LPOUTLINEDOC)lpServerDoc, lpStg)) {

      ((LPOUTLINEDOC)lpServerDoc)->m_docInitType = DOCTYPE_EMBEDDED;

      /* OLE2NOTE: we need to check if the ConvertStg bit is on. if
      **    so, we need to clear the ConvertStg bit and mark the
      **    document as dirty so as to force a save when the document
      **    is closed. the actual conversion of the bits should be
      **    performed when the data is loaded from the IStorage*. in
      **    our case any conversion of data formats would be done in
      **    OutlineDoc_LoadFromStg function. in reality both SVROUTL
      **    and ISVROTL read and write the same format so no actual
      **    conversion of data bits is necessary.
      */
      if (GetConvertStg(lpStg) == NOERROR) {
         SetConvertStg(lpStg, FALSE);

         OleDbgOut3("SvrDoc_PStg_Load: ConvertStg==TRUE\r\n");
         OutlineDoc_SetModified(lpOutlineDoc, TRUE, FALSE, FALSE);
      }

   } else {
      sc = E_FAIL;
      goto error;
   }

   /* OLE2NOTE: An embedded object must guarantee that it can save
```

```
**    even in low memory situations. it must be able to
**    successfully save itself without consuming any additional
**    memory. this means that a server is NOT supposed to open or
**    create any streams or storages when
**    IPersistStorage::Save(fSameAsLoad==TRUE) is called. thus an
**    embedded object should hold onto its storage and pre-open and
**    hold open any streams that it will need later when it is time
**    to save.
*/
if (lpOleDoc->m_lpLLStm)
   OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpLLStm);
hrErr = lpStg->lpVtbl->OpenStream(
      lpStg,
      OLESTR("LineList"),
      NULL,
      STGM_READWRITE | STGM_SHARE_EXCLUSIVE,
      0,
      &lpOleDoc->m_lpLLStm
);

if (hrErr != NOERROR) {
   OleDbgAssertSz(hrErr==NOERROR,"Could not create LineList stream");
   OleDbgOutHResult("LineList CreateStream returned", hrErr);
   sc = GetScode(hrErr);
   goto error;
}

if (lpOleDoc->m_lpNTStm)
   OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpNTStm);
hrErr = lpStg->lpVtbl->OpenStream(
      lpStg,
      OLESTR("NameTable"),
      NULL,
      STGM_READWRITE | STGM_SHARE_EXCLUSIVE,
      0,
      &lpOleDoc->m_lpNTStm
);

if (hrErr != NOERROR) {
   OleDbgAssertSz(hrErr==NOERROR,"Could not create NameTable stream");
   OleDbgOutHResult("NameTable CreateStream returned", hrErr);
   sc = GetScode(hrErr);
   goto error;
}

lpOleDoc->m_lpStg = lpStg;

// OLE2NOTE: to be able to hold onto IStorage* pointer, we must AddRef it
lpStg->lpVtbl->AddRef(lpStg);

OLEDBG_END2
return NOERROR;

error:
OLEDBG_END2
```

```
   return ResultFromScode(sc);
}


// IPersistStorage::Save method

STDMETHODIMP SvrDoc_PStg_Save(
      LPPERSISTSTORAGE        lpThis,
      LPSTORAGE               lpStg,
      BOOL                    fSameAsLoad
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocPersistStorageImpl FAR*)lpThis)->lpServerDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   BOOL fStatus;
   SCODE sc;

   OLEDBG_BEGIN2("SvrDoc_PStg_Save\r\n")

   fStatus = OutlineDoc_SaveSelToStg(
         (LPOUTLINEDOC)lpServerDoc,
         NULL,
         lpOutlineDoc->m_cfSaveFormat,
         lpStg,
         fSameAsLoad,
         FALSE
   );

   if (! fStatus) {
      OutlineApp_ErrorMessage(g_lpApp, ErrMsgPSSaveFail);
      sc = E_FAIL;
      goto error;
   }

   lpServerDoc->m_fSaveWithSameAsLoad = fSameAsLoad;
   lpServerDoc->m_fNoScribbleMode = TRUE;

   OLEDBG_END2
   return NOERROR;

error:
   OLEDBG_END2
   return ResultFromScode(sc);
}



// IPersistStorage::SaveCompleted method

STDMETHODIMP SvrDoc_PStg_SaveCompleted(
      LPPERSISTSTORAGE        lpThis,
      LPSTORAGE               lpStgNew
)
{
```

```
LPSERVERDOC lpServerDoc =
      ((struct CDocPersistStorageImpl FAR*)lpThis)->lpServerDoc;
LPOLEDOC lpOleDoc = (LPOLEDOC)lpServerDoc;
LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
HRESULT hrErr;

OLEDBG_BEGIN2("SvrDoc_PStg_SaveCompleted\r\n")

/* OLE2NOTE: this sample application is a pure server application.
**    a container/server application would have to call SaveCompleted
**    for each of its contained compound document objects. if a new
**    storage was given, then the container/server would have to
**    open the corresponding new sub-storage for each compound
**    document object and pass as an argument in the SaveCompleted
**    call.
*/

/* OLE2NOTE: it is only legal to perform a Save or SaveAs operation
**    on an embedded object. if the document is a file-based document
**    then we can not be changed to a IStorage-base object.
**
**      fSameAsLoad   lpStgNew     Type of Save     Send OnSave
**    ---------------------------------------------------------
**         TRUE        NULL        SAVE             YES
**         TRUE        ! NULL      SAVE *           YES
**         FALSE       ! NULL      SAVE AS          YES
**         FALSE       NULL        SAVE COPY AS     NO
**
**    * this is a strange case that is possible. it is inefficient
**    for the caller; it would be better to pass lpStgNew==NULL for
**    the Save operation.
*/
if ( ((lpServerDoc->m_fSaveWithSameAsLoad && lpStgNew==NULL) || lpStgNew)
      && (lpOutlineDoc->m_docInitType != DOCTYPE_EMBEDDED) ) {
   OLEDBG_END2
   return ResultFromScode(E_INVALIDARG);
}

/* OLE2NOTE: inform any linking clients that the document has been
**    saved. in addition, any currently active pseudo objects
**    should also inform their clients. we should only broadcast an
**    OnSave notification if a Save or SaveAs operation was
**    performed. we do NOT want to send the notification if a
**    SaveCopyAs operation was performed.
*/
if (lpStgNew || lpServerDoc->m_fSaveWithSameAsLoad) {

   /* OLE2NOTE: if IPersistStorage::Save has been called, then we
   **    need to clear the dirty bit and send OnSave notification.
   **    if HandsOffStorage is called directly without first
   **    calling Save, then we do NOT want to clear the dirty bit
   **    and send OnSave when SaveCompleted is called.
   */
   if (lpServerDoc->m_fNoScribbleMode) {
      OutlineDoc_SetModified(lpOutlineDoc, FALSE, FALSE, FALSE);
      ServerDoc_SendAdvise (
            lpServerDoc,
            OLE_ONSAVE,
            NULL,   /* lpmkDoc -- not relevant here */
            0       /* advf -- not relevant here */
      );
   }
   lpServerDoc->m_fSaveWithSameAsLoad = FALSE;
}
lpServerDoc->m_fNoScribbleMode = FALSE;

/* OLE2NOTE: An embedded object must guarantee that it can save
**    even in low memory situations. it must be able to
**    successfully save itself without consuming any additional
**    memory. this means that a server is NOT supposed to open or
**    create any streams or storages when
**    IPersistStorage::Save(fSameAsLoad==TRUE) is called. thus an
**    embedded object should hold onto its storage and pre-open and
**    hold open any streams that it will need later when it is time
**    to save. if this is a SaveAs situtation, then we want to
**    pre-open and hold open our streams to guarantee that a
**    subsequent save will be successful in low-memory. if we fail
**    to open these streams then we want to force ourself to close
**    to make sure the can't make editing changes that can't be
**    later saved.
*/
if ( lpStgNew && !lpServerDoc->m_fSaveWithSameAsLoad ) {

   // release previous streams
   if (lpOleDoc->m_lpLLStm) {
      OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpLLStm);
      lpOleDoc->m_lpLLStm = NULL;
   }
   if (lpOleDoc->m_lpNTStm) {
      OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpNTStm);
      lpOleDoc->m_lpNTStm = NULL;
   }
   if (lpOleDoc->m_lpStg) {
      OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpStg);
      lpOleDoc->m_lpStg = NULL;
   }

   hrErr = lpStgNew->lpVtbl->OpenStream(
         lpStgNew,
         OLESTR("LineList"),
         NULL,
         STGM_READWRITE | STGM_SHARE_EXCLUSIVE,
         0,
         &lpOleDoc->m_lpLLStm
   );

   if (hrErr != NOERROR) {
      OleDbgAssertSz(hrErr==NOERROR,"Could not create LineList stream");
      OleDbgOutHResult("LineList CreateStream returned", hrErr);
```

```
         goto error;
      }

      hrErr = lpStgNew->lpVtbl->OpenStream(
            lpStgNew,
            OLESTR("NameTable"),
            NULL,
            STGM_READWRITE | STGM_SHARE_EXCLUSIVE,
            0,
            &lpOleDoc->m_lpNTStm
      );

      if (hrErr != NOERROR) {
         OleDbgAssertSz(hrErr==NOERROR,"Could not create NameTable stream");
         OleDbgOutHResult("NameTable CreateStream returned", hrErr);
         goto error;
      }

      lpOleDoc->m_lpStg = lpStgNew;

      // OLE2NOTE: to hold onto IStorage* pointer, we must AddRef it
      lpStgNew->lpVtbl->AddRef(lpStgNew);
   }

   OLEDBG_END2
   return NOERROR;

error:
   OLEDBG_END2
   return ResultFromScode(E_OUTOFMEMORY);
}


// IPersistStorage::HandsOffStorage method

STDMETHODIMP SvrDoc_PStg_HandsOffStorage(LPPERSISTSTORAGE lpThis)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocPersistStorageImpl FAR*)lpThis)->lpServerDoc;
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpServerDoc;

   OLEDBG_BEGIN2("SvrDoc_PStg_HandsOffStorage\r\n")

   /* OLE2NOTE: An embedded object must guarantee that it can save
   **    even in low memory situations. it must be able to
   **    successfully save itself without consuming any additional
   **    memory. this means that a server is NOT supposed to open or
   **    create any streams or storages when
   **    IPersistStorage::Save(fSameAsLoad==TRUE) is called. thus an
   **    embedded object should hold onto its storage and pre-open and
   **    hold open any streams that it will need later when it is time
   **    to save. Now when HandsOffStorage is called the object must
   **    release its storage and any streams that is holds open.
   **    later when SaveCompleted is called, it will be given back its
   **    storage.
   */
   if (lpOleDoc->m_lpLLStm) {
      OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpLLStm);
      lpOleDoc->m_lpLLStm = NULL;
   }
   if (lpOleDoc->m_lpNTStm) {
      OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpNTStm);
      lpOleDoc->m_lpNTStm = NULL;
   }
   if (lpOleDoc->m_lpStg) {
      OleStdRelease((LPUNKNOWN)lpOleDoc->m_lpStg);
      lpOleDoc->m_lpStg = NULL;
   }

   OLEDBG_END2
   return NOERROR;
}



#if defined( SVR_TREATAS )

/*************************************************************************
** ServerDoc::IStdMarshalInfo interface implementation
*************************************************************************/

// IStdMarshalInfo::QueryInterface method

STDMETHODIMP SvrDoc_StdMshl_QueryInterface(
      LPSTDMARSHALINFO        lpThis,
      REFIID                  riid,
      LPVOID FAR*             lplpvObj
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocStdMarshalInfoImpl FAR*)lpThis)->lpServerDoc;

   return OleDoc_QueryInterface((LPOLEDOC)lpServerDoc, riid, lplpvObj);
}


// IStdMarshalInfo::AddRef method

STDMETHODIMP_(ULONG) SvrDoc_StdMshl_AddRef(LPSTDMARSHALINFO lpThis)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocStdMarshalInfoImpl FAR*)lpThis)->lpServerDoc;

   OleDbgAddRefMethod(lpThis, "IStdMarshalInfo");

   return OleDoc_AddRef((LPOLEDOC)lpServerDoc);
}


// IStdMarshalInfo::Release method
```

```
STDMETHODIMP_(ULONG) SvrDoc_StdMshl_Release(LPSTDMARSHALINFO lpThis)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocStdMarshalInfoImpl FAR*)lpThis)->lpServerDoc;

   OleDbgReleaseMethod(lpThis, "IStdMarshalInfo");

   return OleDoc_Release((LPOLEDOC)lpServerDoc);
}


// IStdMarshalInfo::GetClassForHandler

STDMETHODIMP SvrDoc_StdMshl_GetClassForHandler(
      LPSTDMARSHALINFO        lpThis,
      DWORD                   dwDestContext,
      LPVOID                  pvDestContext,
      LPCLSID                 lpClassID
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocStdMarshalInfoImpl FAR*)lpThis)->lpServerDoc;
   OleDbgOut2("SvrDoc_StdMshl_GetClassForHandler\r\n");

   // OLE2NOTE: we only handle LOCAL marshal context.
   if (dwDestContext != MSHCTX_LOCAL || pvDestContext != NULL)
      return ResultFromScode(E_INVALIDARG);

   /* OLE2NOTE: we must return our REAL clsid, NOT the clsid that we
   **    are pretending to be if a "TreatAs" is in effect.
   */
   *lpClassID = CLSID_APP;
   return NOERROR;
}
#endif  // SVR_TREATAS



/*************************************************************************
** ServerDoc Support Functions
*************************************************************************/


/* ServerDoc_Init
 * --------------
 *
 *  Initialize the fields of a new ServerDoc object. The object is initially
 *  not associated with a file or an (Untitled) document. This function sets
 *  the docInitType to DOCTYPE_UNKNOWN. After calling this function the
 *  caller should call:
 *      1.) OutlineDoc_InitNewFile to set the ServerDoc to (Untitled)
 *      2.) OutlineDoc_LoadFromFile to associate the ServerDoc with a file.
 *  This function creates a new window for the document.
 *
 *  NOTE: the window is initially created with a NIL size. it must be
 *        sized and positioned by the caller. also the document is initially
 *        created invisible. the caller must call OutlineDoc_ShowWindow
 *        after sizing it to make the document window visible.
 */
BOOL ServerDoc_Init(LPSERVERDOC lpServerDoc, BOOL fDataTransferDoc)
{
   lpServerDoc->m_cPseudoObj                  = 0;
   lpServerDoc->m_lpOleClientSite             = NULL;
   lpServerDoc->m_lpOleAdviseHldr             = NULL;
   lpServerDoc->m_lpDataAdviseHldr            = NULL;

   // initialy doc does not have any storage
   lpServerDoc->m_fNoScribbleMode             = FALSE;
   lpServerDoc->m_fSaveWithSameAsLoad         = FALSE;
   lpServerDoc->m_szContainerApp[0]           = '\0';
   lpServerDoc->m_szContainerObj[0]           = '\0';
   lpServerDoc->m_nNextRangeNo                = 0L;
   lpServerDoc->m_lrSrcSelOfCopy.m_nStartLine  = -1;
   lpServerDoc->m_lrSrcSelOfCopy.m_nEndLine    = -1;
   lpServerDoc->m_fDataChanged                = FALSE;
   lpServerDoc->m_fSizeChanged                = FALSE;
   lpServerDoc->m_fSendDataOnStop             = FALSE;

#if defined( SVR_TREATAS )
   lpServerDoc->m_clsidTreatAs                = CLSID_NULL;
   lpServerDoc->m_lpszTreatAsType             = NULL;
#endif  // SVR_TREATAS

#if defined( INPLACE_SVR )
   lpServerDoc->m_hWndHatch                   =
         CreateHatchWindow(
               OutlineApp_GetWindow(g_lpApp),
               OutlineApp_GetInstance(g_lpApp)
         );
   if (!lpServerDoc->m_hWndHatch)
      return FALSE;

   lpServerDoc->m_fInPlaceActive              = FALSE;
   lpServerDoc->m_fInPlaceVisible             = FALSE;
   lpServerDoc->m_fUIActive                   = FALSE;
   lpServerDoc->m_lpIPData                    = NULL;
   lpServerDoc->m_fMenuHelpMode               = FALSE; // F1 pressed in
menu

   INIT_INTERFACEIMPL(
         &lpServerDoc->m_OleInPlaceObject,
         &g_SvrDoc_OleInPlaceObjectVtbl,
         lpServerDoc
   );
   INIT_INTERFACEIMPL(
         &lpServerDoc->m_OleInPlaceActiveObject,
         &g_SvrDoc_OleInPlaceActiveObjectVtbl,
         lpServerDoc
   );
```

```
#endif // INPLACE_SVR

   INIT_INTERFACEIMPL(
         &lpServerDoc->m_OleObject,
         &g_SvrDoc_OleObjectVtbl,
         lpServerDoc
   );

   INIT_INTERFACEIMPL(
         &lpServerDoc->m_PersistStorage,
         &g_SvrDoc_PersistStorageVtbl,
         lpServerDoc
   );

#if defined( SVR_TREATAS )

   INIT_INTERFACEIMPL(
         &lpServerDoc->m_StdMarshalInfo,
         &g_SvrDoc_StdMarshalInfoVtbl,
         lpServerDoc
   );
#endif  // SVR_TREATAS
   return TRUE;
}


/* ServerDoc_InitNewEmbed
 * ----------------------
 *
 *  Initialize the ServerDoc object to be a new embedded object document.
 *  This function sets the docInitType to DOCTYPE_EMBED.
 */
BOOL ServerDoc_InitNewEmbed(LPSERVERDOC lpServerDoc)
{
   char szAnsiStr[256];
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;

   OleDbgAssert(lpOutlineDoc->m_docInitType == DOCTYPE_UNKNOWN);

   lpOutlineDoc->m_docInitType = DOCTYPE_EMBEDDED;

   /* The Window title for an embedded object is constructed as
   **    follows:
   **      <server app name> - <obj short type> in <cont. doc name>
   **
   **    here we construct the current document title portion of the
   **    name which follows the '-'. OutlineDoc_SetTitle prepends the
   **    "<server app name> - " to the document title.
   */
   // REVIEW: this string should be loaded from string resource
   wsprintf(szAnsiStr, "%s in %s",
      (LPSTR)SHORTUSERTYPENAME,
      (LPSTR)DEFCONTAINERNAME);
   A2W (szAnsiStr, lpOutlineDoc->m_szFileName, OLEUI_CCHPATHMAX);
   lpOutlineDoc->m_lpszDocTitle = lpOutlineDoc->m_szFileName;
   /* OLE2NOTE: an embedding should be marked as initially dirty so
   **    that on close we always call IOleClientSite::SaveObject.
   */
   OutlineDoc_SetModified(lpOutlineDoc, TRUE, FALSE, FALSE);

   OutlineDoc_SetTitle(lpOutlineDoc, FALSE /*fMakeUpperCase*/);

   return TRUE;
}


/* ServerDoc_SendAdvise
 * --------------------
 *
 * This function sends an advise notification on behalf of a specific
 *  doc object to all its clients.
 */
void ServerDoc_SendAdvise(
      LPSERVERDOC     lpServerDoc,
      WORD            wAdvise,
      LPMONIKER       lpmkDoc,
      DWORD           dwAdvf
)
{
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpServerDoc;


   switch (wAdvise) {

      case OLE_ONDATACHANGE:

         // inform clients that the data of the object has changed

         if (lpOutlineDoc->m_nDisableDraw == 0) {
            /* drawing is currently enabled. inform clients that
            **    the data of the object has changed
            */

            lpServerDoc->m_fDataChanged = FALSE;

            /* OLE2NOTE: we must note the time of last change
            **    for our object in the RunningObjectTable.
            **    this is used as the basis to answer
            **    IOleObject::IsUpToDate. we only want to note
            **    the change time when an actual change takes
            **    place. we do NOT want to set it when we are
            **    notifying clients of ADVF_DATAONSTOP
            */
            if (dwAdvf == 0)
               OleStdNoteObjectChangeTime(lpOleDoc->m_dwRegROT);

            if (lpServerDoc->m_lpDataAdviseHldr) {
               OLEDBG_BEGIN2("IDataAdviseHolder::SendOnDataChange
called\r\n");
               lpServerDoc->m_lpDataAdviseHldr->lpVtbl->SendOnDataChange(
                     lpServerDoc->m_lpDataAdviseHldr,
                     (LPDATAOBJECT)&lpOleDoc->m_DataObject,
                     0,
                     dwAdvf
               );
               OLEDBG_END2

            }

#if defined( INPLACE_SVR )
            /* OLE2NOTE: if the ServerDoc is currently in-place UI active,
            **    then is it important to renegotiate the size for the
            **    in-place document window BEFORE sending OnDataChange
            **    (which will cause the window to repaint).
            */
            if (lpServerDoc->m_fSizeChanged) {
               lpServerDoc->m_fSizeChanged = FALSE;
               if (lpServerDoc->m_fInPlaceActive)
                  ServerDoc_UpdateInPlaceWindowOnExtentChange(lpServerDoc);
            }
#endif

            /* OLE2NOTE: we do NOT need to tell our pseudo objects to
            **    broadcast OnDataChange notification because
            **    they will do it automatically when an editing
            **    change in the document affects a PseudoObj.
            **    (see OutlineNameTable_AddLineUpdate,
            **         OutlineNameTable_DeleteLineUpdate,
            **    and  ServerNameTable_EditLineUpdate)
            */

         } else {
            /* drawing is currently disabled. do not send
            **    notifications or call
            **    IOleInPlaceObject::OnPosRectChange until drawing
            **    is re-enabled.
            */
         }
         break;

      case OLE_ONCLOSE:

         // inform clients that the document is shutting down

         if (lpServerDoc->m_lpOleAdviseHldr) {
            OLEDBG_BEGIN2("IOleAdviseHolder::SendOnClose called\r\n");
            lpServerDoc->m_lpOleAdviseHldr->lpVtbl->SendOnClose(
                  lpServerDoc->m_lpOleAdviseHldr
            );
            OLEDBG_END2
         }
         /* OLE2NOTE: we do NOT need to tell our pseudo objects to
         **    broadcast OnClose notification because they will do
         **    it automatically when the pseudo object is closed.
         **    (see PseudoObj_Close)
         */

         break;

      case OLE_ONSAVE:

         // inform clients that the object has been saved

         OLEDBG_BEGIN3("ServerDoc_SendAdvise ONSAVE\r\n");

         if (lpServerDoc->m_lpOleAdviseHldr) {
            OLEDBG_BEGIN2("IOleAdviseHolder::SendOnSave called\r\n");
            lpServerDoc->m_lpOleAdviseHldr->lpVtbl->SendOnSave(
                  lpServerDoc->m_lpOleAdviseHldr
            );
            OLEDBG_END2
         }

         /* OLE2NOTE: inform any clients of pseudo objects
         **    within our document, that our document has been
         **    saved.
         */
         ServerNameTable_InformAllPseudoObjectsDocSaved(
               (LPSERVERNAMETABLE)lpOutlineDoc->m_lpNameTable,
               lpmkDoc
         );
         OLEDBG_END3
         break;

      case OLE_ONRENAME:

         // inform clients that the object's name has changed

         OLEDBG_BEGIN3("ServerDoc_SendAdvise ONRENAME\r\n");

         if (lpmkDoc && lpServerDoc->m_lpOleAdviseHldr) {
            OLEDBG_BEGIN2("IOleAdviseHolder::SendOnRename called\r\n");
            lpServerDoc->m_lpOleAdviseHldr->lpVtbl->SendOnRename(
                  lpServerDoc->m_lpOleAdviseHldr,
                  lpmkDoc
            );
            OLEDBG_END2
         }

         OLEDBG_END3
         break;
   }
}


/* ServerDoc_GetClassID
```

```
** --------------------
**    Return the class ID corresponding to the bits in the storage.
**    normally this will be our application's given CLSID. but if a
**    "TreateAs (aka. ActivateAs)" operation is taking place, then our
**    application needs to pretend to be the class of the object that
**    we are emulating. this is also the class that will be written
**    into the storage.
*/
HRESULT ServerDoc_GetClassID(LPSERVERDOC lpServerDoc, LPCLSID lpclsid)
{
#if defined( SVR_TREATAS )
   if (! IsEqualCLSID(&lpServerDoc->m_clsidTreatAs, &CLSID_NULL))
      *lpclsid = lpServerDoc->m_clsidTreatAs;
   else
#endif  // SVR_TREATAS
      *lpclsid = CLSID_APP;

   return NOERROR;
}



/* ServerDoc_UpdateMenu
 * --------------------
 *
 *  Update menu for embedding mode. the changes include:
 *      1 Remove File/New and File/Open (SDI ONLY)
 *      2 Change File/Save As.. to File/Save Copy As..
 *      3 Change File menu so it contains "Update" instead of "Save"
 *      4 Change File/Exit to File/Exit & Return to <client doc>"
 */
void ServerDoc_UpdateMenu(LPSERVERDOC lpServerDoc)
{
   char    str[256];
   HWND    hWndMain;
   HMENU   hMenu;
   char    szAnsiStr[256];


   OleDbgOut2("ServerDoc_UpdateMenu\r\n");

   hWndMain=g_lpApp->m_hWndApp;
   hMenu=GetMenu(hWndMain);

#if defined( SDI_VERSION )
   /* SDI ONLY: Remove File/New and File/Open */
   DeleteMenu(hMenu, IDM_F_NEW, MF_BYCOMMAND);
   DeleteMenu(hMenu, IDM_F_OPEN, MF_BYCOMMAND);
#endif

   // Change File.Save As.. to File.Save Copy As.. */
   ModifyMenu(hMenu,IDM_F_SAVEAS, MF_STRING, IDM_F_SAVEAS, "Save Copy
As..");

   // Change File.Save to "&Update <container doc>"
   W2A (lpServerDoc->m_szContainerObj, szAnsiStr, 256);
   wsprintf(str, g_szUpdateCntrDoc, szAnsiStr);
   ModifyMenu(hMenu, IDM_F_SAVE, MF_STRING, IDM_F_SAVE, str);

   // Change File/Exit to File/Exit & Return to <container doc>" */
   W2A (lpServerDoc->m_szContainerObj, szAnsiStr, 256);
   wsprintf(str, g_szExitNReturnToCntrDoc, szAnsiStr);
   ModifyMenu(hMenu, IDM_F_EXIT, MF_STRING, IDM_F_EXIT, str);

   DrawMenuBar(hWndMain);
}

#if defined( MDI_VERSION )

// NOTE: ServerDoc_RestoreMenu is actually redundant because the
//          app is dying when the function is called.  (In SDI, the
//          app will terminate when the ref counter of the server doc
//          is zero). However, it is important for MDI.

/* ServerDoc_RestoreMenu
 * ---------------------
 *
 *      Reset the menu to non-embedding mode
 */
void ServerDoc_RestoreMenu(LPSERVERDOC lpServerDoc)
{
   LPOUTLINEAPP    lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   HWND            hWndMain;
   HMENU           hMenu;


   OleDbgOut2("ServerDoc_RestoreMenu\r\n");

   hWndMain = lpOutlineApp->m_hWndApp;
   hMenu = GetMenu(hWndMain);

   /* Add back File/New, File/Open.. and File/Save */
   InsertMenu(hMenu, IDM_F_SAVEAS, MF_BYCOMMAND | MF_ENABLED | MF_STRING,
      IDM_F_NEW, "&New");
   InsertMenu(hMenu, IDM_F_SAVEAS, MF_BYCOMMAND | MF_ENABLED | MF_STRING,
      IDM_F_OPEN, "&Open...");

   /* Change File menu so it contains "Save As..." instead of */
   /* "Save Copy As..." */
   ModifyMenu(hMenu, IDM_F_SAVEAS, MF_STRING, IDM_F_SAVEAS, "Save &As..");

   /* Change File menu so it contains "Save" instead of "Update" */
   ModifyMenu(hMenu, IDM_F_SAVE, MF_STRING, IDM_F_SAVE, "&Save");

   /* Change File menu so it contains "Exit" */
   /* instead of just "Exit & Return to <client doc>" */
   ModifyMenu(hMenu, IDM_F_EXIT, MF_STRING, IDM_F_EXIT, "E&xit");

   DrawMenuBar (hWndMain);
```

```
}

#endif  // MDI_VERSION
```

SVRINPL.C   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Server Sample Code
**
**    svrinpl.c
**
**    This file contains all interfaces, methods and related support
**    functions for an In-Place Object (Server) application (aka. Visual
**    Editing). The in-place Object application includes the following
**    implementation objects:
**
**    ServerDoc Object
**      exposed interfaces:
**          IOleInPlaceObject
**          IOleInPlaceActiveObject
**
**    ServerApp Object
**      exposed interfaces:
**          IUnknown
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/


#include "outline.h"

OLEDBGDATA

extern LPOUTLINEAPP             g_lpApp;


/* OLE2NOTE: the object should compose a string that is used by
**    in-place containers to be used for the window titles. this string
**    is passed to the container application via
**    IOleInPlaceUIWindow::SetActiveObject. the string should have the
**    following form:
**          <application name> - <object short type name>
**    SDI containers can use the string directly to display in the
**    frame window title. the container would concatenate the string
**    " in <container doc name>".
**    an MDI container with the MDI child window maximized can do the
**    same as the SDI container. an MDI container with the MDI child
**    windows NOT maximized can look for the " - " in the string from
**    the object. the first part of the string (app name) would be put
**    as the frame window title; the second part would be composed with
**    " in <container doc name>" and used as the MDI child window
**    title.
*/

// REVIEW: should use string resource for messages
OLECHAR g_szIPObjectTitle[] = APPNAME OLESTR(" - ") SHORTUSERTYPENAME;
```

```
extern RECT g_rectNull;



/*************************************************************************
** ServerDoc::IOleInPlaceObject interface implementation
*************************************************************************/

// IOleInPlaceObject::QueryInterface method

STDMETHODIMP SvrDoc_IPObj_QueryInterface(
      LPOLEINPLACEOBJECT  lpThis,
      REFIID              riid,
      LPVOID FAR *        lplpvObj
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;

   return OleDoc_QueryInterface((LPOLEDOC)lpServerDoc, riid, lplpvObj);
}


// IOleInPlaceObject::AddRef method

STDMETHODIMP_(ULONG) SvrDoc_IPObj_AddRef(LPOLEINPLACEOBJECT lpThis)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;

   OleDbgAddRefMethod(lpThis, "IOleInPlaceObject");

   return OleDoc_AddRef((LPOLEDOC)lpServerDoc);
}


// IOleInPlaceObject::Release method

STDMETHODIMP_(ULONG) SvrDoc_IPObj_Release(LPOLEINPLACEOBJECT lpThis)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;

   OleDbgReleaseMethod(lpThis, "IOleInPlaceObject");

   return OleDoc_Release((LPOLEDOC)lpServerDoc);
}


// IOleInPlaceObject::GetWindow method

STDMETHODIMP SvrDoc_IPObj_GetWindow(
      LPOLEINPLACEOBJECT  lpThis,
      HWND FAR*           lphwnd
```

```
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;

   OLEDBG_BEGIN2("SvrDoc_IPObj_GetWindow\r\n")

   *lphwnd = ((LPOUTLINEDOC)lpServerDoc)->m_hWndDoc;

   OLEDBG_END2
   return S_OK;
}


// IOleInPlaceObject::ContextSensitiveHelp method

STDMETHODIMP SvrDoc_IPObj_ContextSensitiveHelp(
      LPOLEINPLACEOBJECT  lpThis,
      BOOL                fEnable
)
{
   LPOLEDOC lpOleDoc =
         (LPOLEDOC)((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   OleDbgOut2("SvrDoc_IPObj_ContextSensitiveHelp\r\n");

   /* OLE2NOTE: see context sensitive help technote (CSHELP.DOC).
   **    This method is called when SHIFT-F1 context sensitive help is
   **    entered. the cursor should then change to a question mark
   **    cursor and the app should enter a modal state where the next
   **    mouse click does not perform its normal action but rather
   **    gives help corresponding to the location clicked. if the app
   **    does not implement a help system, it should at least eat the
   **    click and do nothing.
   */
   lpOleDoc->m_fCSHelpMode = fEnable;

   return S_OK;
}


// IOleInPlaceObject::InPlaceDeactivate method

STDMETHODIMP SvrDoc_IPObj_InPlaceDeactivate(LPOLEINPLACEOBJECT lpThis)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   HRESULT hrErr;

   // artificial AddRef in case object is deleted during call
   SvrDoc_IPObj_AddRef(lpThis);

   OLEDBG_BEGIN2("SvrDoc_IPObj_InPlaceDeactivate\r\n")

   hrErr = ServerDoc_DoInPlaceDeactivate(lpServerDoc);
   OLEDBG_END2

   // release artificial AddRef
   SvrDoc_IPObj_Release(lpThis);

   return hrErr;
}


// IOleInPlaceObject::UIDeactivate method

STDMETHODIMP SvrDoc_IPObj_UIDeactivate(LPOLEINPLACEOBJECT lpThis)
{
   LPSERVERDOC     lpServerDoc =
                  ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   LPSERVERAPP     lpServerApp = (LPSERVERAPP)g_lpApp;
   LPOUTLINEDOC    lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPINPLACEDATA   lpIPData = lpServerDoc->m_lpIPData;
   LPLINELIST      lpLL = (LPLINELIST)&((LPOUTLINEDOC)lpServerDoc)-
>m_LineList;
   HWND            hWndApp = OutlineApp_GetWindow(g_lpApp);

   OLEDBG_BEGIN2("SvrDoc_IPObj_UIDeactivate\r\n");

   if (!lpServerDoc->m_fUIActive) {
      OLEDBG_END2
      return NOERROR;
   }

   // artificial AddRef in case object is deleted during call
   SvrDoc_IPObj_AddRef(lpThis);

   lpServerDoc->m_fUIActive = FALSE;

   // Clip the hatch window to the size of pos rect so, that the object
   // adornments and hatch border will not be visible.
   ServerDoc_ResizeInPlaceWindow(lpServerDoc,
         (LPRECT)&(lpServerDoc->m_lpIPData->rcPosRect),
         (LPRECT)&(lpServerDoc->m_lpIPData->rcPosRect)
   );

   if (lpIPData->lpDoc)
      lpIPData->lpDoc->lpVtbl->SetActiveObject(lpIPData->lpDoc, NULL, NULL);

   if (lpIPData->lpFrame) {
      lpIPData->lpFrame->lpVtbl->SetActiveObject(
         lpIPData->lpFrame,
         NULL,
         NULL
      );
   }

#if defined( USE_FRAMETOOLS )
   /* OLE2NOTE: we must hide our frame tools here but NOT call
   **    IOleInPlaceFrame::SetBorderSpace(NULL) or SetMenu(NULL).
   **    we must hide our tools BEFORE calling
   **    IOleInPlaceSite::OnUIDeactivate. the container will put
   **    his menus and tools back when OnUIDeactivate is called.
   */
   ServerDoc_RemoveFrameLevelTools(lpServerDoc);
#endif

   OLEDBG_BEGIN2("IOleInPlaceSite::OnUIDeactivate called\r\n");
   lpIPData->lpSite->lpVtbl->OnUIDeactivate(lpIPData->lpSite, FALSE);
   OLEDBG_END2

   /* Reset to use our normal app's accelerator table */
   g_lpApp->m_hAccelApp = lpServerApp->m_hAccelBaseApp;
   g_lpApp->m_hAccel = lpServerApp->m_hAccelBaseApp;
   g_lpApp->m_hWndAccelTarget = hWndApp;

   OLEDBG_END2

#if !defined( SVR_INSIDEOUT )
   /* OLE2NOTE: an "outside-in" style in-place server would hide its
   **    window here. an "inside-out" style server leaves its window
   **    visible when it is UIDeactivated. it would only hide its
   **    window when InPlaceDeactivated. this app is an "inside-out"
   **    style server. it is recommended for most server to support
   **    inside-out behavior if possible.
   */
   ServerDoc_DoInPlaceHide(lpServerDoc);
#endif // INSIEDOUT
   // release artificial AddRef
   SvrDoc_IPObj_Release(lpThis);

   return NOERROR;
}


// IOleInPlaceObject::SetObjectRects method

STDMETHODIMP SvrDoc_IPObj_SetObjectRects(
      LPOLEINPLACEOBJECT  lpThis,
      LPCRECT             lprcPosRect,
      LPCRECT             lprcClipRect
)
{
   LPSERVERDOC  lpServerDoc =
               ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   LPINPLACEDATA lpIPData = lpServerDoc->m_lpIPData;
   LPLINELIST   lpLL = OutlineDoc_GetLineList((LPOUTLINEDOC)lpServerDoc);


   // artificial AddRef in case object is deleted during call
   SvrDoc_IPObj_AddRef(lpThis);

   OLEDBG_BEGIN2("SvrDoc_IPObj_SetObjectRects\r\n")

#if defined( _DEBUG )
```

```
   OleDbgOutRect3("SvrDoc_IPObj_SetObjectRects (PosRect)",
         (LPRECT)lprcPosRect);
   OleDbgOutRect3("SvrDoc_IPObj_SetObjectRects (ClipRect)",
         (LPRECT)lprcClipRect);
#endif
   // save the current PosRect and ClipRect
   lpIPData->rcPosRect = *lprcPosRect;
   lpIPData->rcClipRect = *lprcClipRect;

   if (! lpServerDoc->m_fUIActive) // hatch and adornaments must not be
drawn
      lprcClipRect = lprcPosRect;

   ServerDoc_ResizeInPlaceWindow(
         lpServerDoc, (LPRECT)lprcPosRect, (LPRECT)lprcClipRect);

   OLEDBG_END2

   // release artificial AddRef
   SvrDoc_IPObj_Release(lpThis);

   return NOERROR;
}


// IOleInPlaceObject::ReactivateAndUndo method

STDMETHODIMP SvrDoc_IPObj_ReactivateAndUndo(LPOLEINPLACEOBJECT lpThis)
{
   OLEDBG_BEGIN2("SvrDoc_IPObj_ReactivateAndUndo\r\n")

   // We do not support support UNDO.

   /* REVIEW: for debugging purposes it would be useful to give a
   **    message box indicating that this method has been called.
   */

   OLEDBG_END2
   return NOERROR;
}


/*************************************************************************
** ServerDoc::IOleInPlaceActiveObject interface implementation
*************************************************************************/

// IOleInPlaceActiveObject::QueryInterface method

STDMETHODIMP SvrDoc_IPActiveObj_QueryInterface(
      LPOLEINPLACEACTIVEOBJECT    lpThis,
      REFIID                      riid,
      LPVOID FAR *                lplpvObj
)
{
   SCODE sc = E_NOINTERFACE;
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;

   /* The container should not be able to access the other interfaces
   ** of our object by doing QI on this interface.
   */

   *lplpvObj = NULL;
   if (IsEqualIID(riid, &IID_IUnknown) ||
      IsEqualIID(riid, &IID_IOleWindow) ||
      IsEqualIID(riid, &IID_IOleInPlaceActiveObject)) {
      OleDbgOut4("OleDoc_QueryInterface: IOleInPlaceActiveObject*
RETURNED\r\n");

      *lplpvObj = lpThis;
      OleDoc_AddRef((LPOLEDOC)lpServerDoc);
      sc = NOERROR;
   }

   OleDbgQueryInterfaceMethod(*lplpvObj);

   return ResultFromScode(sc);
}


// IOleInPlaceActiveObject::AddRef method

STDMETHODIMP_(ULONG) SvrDoc_IPActiveObj_AddRef(
      LPOLEINPLACEACTIVEOBJECT lpThis
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;

   OleDbgAddRefMethod(lpThis, "IOleInPlaceActiveObject");

   return OleDoc_AddRef((LPOLEDOC)lpServerDoc);
}


// IOleInPlaceActiveObject::Release method

STDMETHODIMP_(ULONG) SvrDoc_IPActiveObj_Release(
      LPOLEINPLACEACTIVEOBJECT lpThis
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;

   OleDbgReleaseMethod(lpThis, "IOleInPlaceActiveObject");

   return OleDoc_Release((LPOLEDOC)lpServerDoc);
}
```

```
// IOleInPlaceActiveObject::GetWindow method

STDMETHODIMP SvrDoc_IPActiveObj_GetWindow(
      LPOLEINPLACEACTIVEOBJECT    lpThis,
      HWND FAR*                   lphwnd
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;

   OLEDBG_BEGIN2("SvrDoc_IPActiveObj_GetWindow\r\n")

   *lphwnd = ((LPOUTLINEDOC)lpServerDoc)->m_hWndDoc;

   OLEDBG_END2
   return NOERROR;
}


// IOleInPlaceActiveObject::ContextSensitiveHelp method

STDMETHODIMP SvrDoc_IPActiveObj_ContextSensitiveHelp(
      LPOLEINPLACEACTIVEOBJECT    lpThis,
      BOOL                        fEnterMode
)
{
   LPSERVERDOC lpServerDoc =
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   OleDbgOut2("SvrDoc_IPActiveObj_ContextSensitiveHelp\r\n");

   /* OLE2NOTE: see context sensitive help technote (CSHELP.DOC)
   **    This method is called when F1 is pressed when a menu item is
   **    selected. this tells the in-place server application to give
   **    help rather than execute the next menu command. at a minimum,
   **    even if the in-place server application does not implement a
   **    help system, it should NOT execute the next command when
   **    fEnable==TRUE. We set the active object's m_fMenuMode flag here.
   **    later, in WM_COMMAND processing in the DocWndProc, if this
   **    flag is set then the command is NOT executed (and help could
   **    be given if we had a help system....but we don't.)
   */
   lpServerDoc->m_fMenuHelpMode = fEnterMode;

#if !defined( HACK )
   ((LPOLEDOC)lpServerDoc)->m_fCSHelpMode = fEnterMode;
#endif
   return NOERROR;
}


// IOleInPlaceActiveObject::TranslateAccelerator method

STDMETHODIMP SvrDoc_IPActiveObj_TranslateAccelerator(
      LPOLEINPLACEACTIVEOBJECT    lpThis,
      LPMSG                       lpmsg
```

```
)
{
   // This will never be called because this server is implemented as an EXE
   return NOERROR;
}


// IOleInPlaceActiveObject::OnFrameWindowActivate method

STDMETHODIMP SvrDoc_IPActiveObj_OnFrameWindowActivate(
      LPOLEINPLACEACTIVEOBJECT    lpThis,
      BOOL                        fActivate
)
{
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)
         ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   HWND hWndDoc = OutlineDoc_GetWindow(lpOutlineDoc);
#if defined( _DEBUG )
   if (fActivate)
      OleDbgOut2("SvrDoc_IPActiveObj_OnFrameWindowActivate(TRUE)\r\n");
   else
      OleDbgOut2("SvrDoc_IPActiveObj_OnFrameWindowActivate(FALSE)\r\n");
#endif  // _DEBUG

   /* OLE2NOTE: this is a notification of the container application's
   **    WM_ACTIVATEAPP status. some applications may find this
   **    important. we need to update the enable/disable status of our
   **    tool bar buttons.
   */

   // OLE2NOTE: We can't call OutlineDoc_UpdateFrameToolButtons
   //           right away which
   //           would generate some OLE calls and eventually
   //           WM_ACTIVATEAPP and a loop was formed. Therefore, we
   //           should delay the frame tool initialization until
   //           WM_ACTIVATEAPP is finished by posting a message
   //           to ourselves.

   /* Update enable/disable state of buttons in toolbar */
   if (fActivate)
      PostMessage(hWndDoc, WM_U_INITFRAMETOOLS, 0, 0L);

   return NOERROR;
}


// IOleInPlaceActiveObject::OnDocWindowActivate method

STDMETHODIMP SvrDoc_IPActiveObj_OnDocWindowActivate(
      LPOLEINPLACEACTIVEOBJECT    lpThis,
      BOOL                        fActivate
)
{
   LPSERVERDOC     lpServerDoc =
                  ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   LPINPLACEDATA   lpIPData = lpServerDoc->m_lpIPData;
#if defined( _DEBUG )
   if (fActivate)
      OleDbgOut2("SvrDoc_IPActiveObj_OnDocWindowActivate(TRUE)\r\n");
   else
      OleDbgOut2("SvrDoc_IPActiveObj_OnDocWindowActivate(FALSE)\r\n");
#endif

   if (fActivate) {
      ServerDoc_AddFrameLevelUI(lpServerDoc);
   }
   else {
#if defined( USE_FRAMETOOLS )
      /* OLE2NOTE: we must NOT call IOleInPlaceFrame::SetBorderSpace(NULL)
      **    or SetMenu(NULL) here. we should simply hide our tools.
      */
      ServerDoc_RemoveFrameLevelTools(lpServerDoc);
#endif
   }

   OLEDBG_END2
   return NOERROR;
}


// IOleInPlaceActiveObject::ResizeBorder method

STDMETHODIMP SvrDoc_IPActiveObj_ResizeBorder(
      LPOLEINPLACEACTIVEOBJECT    lpThis,
      LPCRECT                     lprectBorder,
      LPOLEINPLACEUIWINDOW        lpIPUiWnd,
      BOOL                        fFrameWindow
)
{
   LPSERVERDOC lpServerDoc =
            ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;

   OLEDBG_BEGIN2("SvrDoc_IPActiveObj_ResizeBorder\r\n")


#if defined( USE_FRAMETOOLS )

   if (fFrameWindow) {
      FrameTools_NegotiateForSpaceAndShow(
            lpOutlineDoc->m_lpFrameTools,
            (LPRECT)lprectBorder,
            (LPOLEINPLACEFRAME)lpIPUiWnd
      );
   }

#endif

   OLEDBG_END2
   return NOERROR;
```

```
}


// IOleInPlaceActiveObject::EnableModeless method

STDMETHODIMP SvrDoc_IPActiveObj_EnableModeless(
      LPOLEINPLACEACTIVEOBJECT    lpThis,
      BOOL                        fEnable
)
{
#if defined( USE_FRAMETOOLS )
   LPSERVERDOC lpServerDoc =
            ((struct CDocOleObjectImpl FAR*)lpThis)->lpServerDoc;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPFRAMETOOLS lptb;

   /* OLE2NOTE: we must enable/disable mouse and keyboard input to our
   **    floating tool palette
   */
   if (lpOutlineDoc) {
      lptb = lpOutlineDoc->m_lpFrameTools;
      if (lptb)
         FrameTools_EnableWindow(lptb, fEnable);
   }
#endif  // USE_FRAMETOOLS

#if defined( _DEBUG )
   if (fEnable)
      OleDbgOut2("SvrDoc_IPActiveObj_EnableModeless(TRUE)\r\n");
   else
      OleDbgOut2("SvrDoc_IPActiveObj_EnableModeless(FALSE)\r\n");
#endif  // _DEBUG

   /* OLE2NOTE: this method is called when the top-level, in-place
   **    container puts up a modal dialog. it tells the UIActive
   **    object to disable it modeless dialogs for the duration that
   **    the container is displaying a modal dialog.
   **
   **    ISVROTL does not use any modeless dialogs, thus we can
   **    ignore this method.
   */
   return NOERROR;
}


/*************************************************************************
** Support Functions
*************************************************************************/


HRESULT ServerDoc_DoInPlaceActivate(
      LPSERVERDOC     lpServerDoc,
      LONG            lVerb,
      LPMSG           lpmsg,
      LPOLECLIENTSITE lpActiveSite
```

```
)
{
   LPOUTLINEAPP            lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPSERVERAPP             lpServerApp = (LPSERVERAPP)g_lpApp;
   LPOLEAPP                lpOleApp = (LPOLEAPP)g_lpApp;
   SCODE                   sc = E_FAIL;
   RECT                    rcPos;
   RECT                    rcClip;
   LPINPLACEDATA           lpIPData = lpServerDoc->m_lpIPData;
   LPOUTLINEDOC            lpOutlineDoc=(LPOUTLINEDOC)lpServerDoc;
   HWND                    hWndDoc = lpOutlineDoc->m_hWndDoc;
   HWND                    hWndHatch = lpServerDoc->m_hWndHatch;
   HRESULT                 hrErr;
   LPLINELIST              lpLL=(LPLINELIST)&lpOutlineDoc->m_LineList;
   LPOLEINPLACESITE    lpIPSite = NULL;

   /* OLE2NOTE: lpActiveSite should be used only for InPlace PLAYing.
   **    This app does not do inplace PLAYing, so it never uses
   **    lpActiveSite.
   */

   /* InPlace activation can only be done if the ClientSite is non-NULL. */
   if (! lpServerDoc->m_lpOleClientSite)
      return NOERROR;

   if (! lpServerDoc->m_fInPlaceActive) {

      // if the object is in open mode then we do not want to do inplace
      // activation.
      if (IsWindowVisible(lpOutlineDoc->m_hWndDoc))
      {
         return NOERROR;
      }

      lpIPSite = (LPOLEINPLACESITE)OleStdQueryInterface(
            (LPUNKNOWN)lpServerDoc->m_lpOleClientSite,
            &IID_IOleInPlaceSite
      );

      if (! lpIPSite)
         goto errActivate;

      OLEDBG_BEGIN2("IOleInPlaceSite::CanInPlaceActivate called\r\n");
      hrErr = lpIPSite->lpVtbl->CanInPlaceActivate(lpIPSite);
      OLEDBG_END2
      if (hrErr != NOERROR)
         goto errActivate;

      lpServerDoc->m_fInPlaceActive = TRUE;
      OLEDBG_BEGIN2("IOleInPlaceSite::OnInPlaceActivate called\r\n");
      hrErr = lpIPSite->lpVtbl->OnInPlaceActivate(lpIPSite);
      OLEDBG_END2
      if (hrErr != NOERROR)
         goto errActivate;
      if (! ServerDoc_AllocInPlaceData(lpServerDoc)) {
         sc = E_OUTOFMEMORY;
         OLEDBG_BEGIN2("IOleInPlaceSite::OnInPlaceDeactivate called\r\n");
         lpIPSite->lpVtbl->OnInPlaceDeactivate(lpIPSite);
         OLEDBG_END2
         goto errActivate;
      }

      (lpIPData = lpServerDoc->m_lpIPData)->lpSite = lpIPSite;
      goto InPlaceActive;

   errActivate:
      lpServerDoc->m_fInPlaceActive = FALSE;
      if (lpIPSite)
         OleStdRelease((LPUNKNOWN)lpIPSite);

      return ResultFromScode(sc);
   }


InPlaceActive:

   if (! lpServerDoc->m_fInPlaceVisible) {
      lpServerDoc->m_fInPlaceVisible = TRUE;

      OLEDBG_BEGIN2("IOleInPlaceSite::GetWindow called\r\n");
      hrErr = lpIPData->lpSite->lpVtbl->GetWindow(
               lpIPData->lpSite, &lpServerDoc->m_hWndParent);
      OLEDBG_END2
      if (hrErr != NOERROR) {
         sc = GetScode(hrErr);
         goto errRtn;
      }

      if (! lpServerDoc->m_hWndParent)
         goto errRtn;

      /* OLE2NOTE: The server should fill in the "cb" field so that the
      **    container can tell what size structure the server is
      **    expecting. this enables this structure to be easily extended
      **    in future releases of OLE. the container should check this
      **    field so that it doesn't try to use fields that do not exist
      **    since the server may be using an old structure definition.
      */
      _fmemset(
         (LPOLEINPLACEFRAMEINFO)&lpIPData->frameInfo,
         0,
         sizeof(OLEINPLACEFRAMEINFO)
      );
      lpIPData->frameInfo.cb = sizeof(OLEINPLACEFRAMEINFO);

      OLEDBG_BEGIN2("IOleInPlaceSite::GetWindowContext called\r\n");
      hrErr = lpIPData->lpSite->lpVtbl->GetWindowContext(lpIPData->lpSite,
               (LPOLEINPLACEFRAME FAR*) &lpIPData->lpFrame,
               (LPOLEINPLACEUIWINDOW FAR*)&lpIPData->lpDoc,
               (LPRECT)&rcPos,
               (LPRECT)&rcClip,
               (LPOLEINPLACEFRAMEINFO)&lpIPData->frameInfo);
      OLEDBG_END2

      if (hrErr != NOERROR) {
         sc = GetScode(hrErr);
         goto errRtn;
      }

      lpServerApp->m_lpIPData = lpIPData;
      ShowWindow(hWndDoc, SW_HIDE);   // make sure we are hidden

      /* OLE2NOTE: reparent in-place server document's window to the
      **    special in-place hatch border window. set the in-place site's
      **    window as the parent of the hatch window. position the
      **    in-place and hatch border windows using the PosRect and
      **    ClipRect.
      **    it is important to properly parent and position the in-place
      **    server window BEFORE calling IOleInPlaceFrame::SetMenu and
      **    SetBorderSpace.
      */
      ShowWindow(lpServerDoc->m_hWndHatch, SW_SHOW);
      // make sure App busy/blocked dialogs are parented to our
      // new hWndFrame
      OleStdMsgFilter_SetParentWindow(
         lpOleApp->m_lpMsgFilter,lpIPData->frameInfo.hwndFrame);
      SetParent(lpServerDoc->m_hWndHatch, lpServerDoc->m_hWndParent);
      SetParent(hWndDoc, lpServerDoc->m_hWndHatch);

#if defined( _DEBUG )
      OleDbgOutRect3("IOleInPlaceSite::GetWindowContext (PosRect)",
            (LPRECT)&rcPos);
      OleDbgOutRect3("IOleInPlaceSite::GetWindowContext (ClipRect)",
            (LPRECT)&rcClip);
#endif
      // save the current PosRect and ClipRect
      lpIPData->rcPosRect  = rcPos;
      lpIPData->rcClipRect = rcClip;

      /* OLE2NOTE: build the shared menu for the in-place container and
      **    the server.
      */
      if (ServerDoc_AssembleMenus (lpServerDoc) != NOERROR)
         goto errRtn;

#if defined( SVR_INSIDEOUT )
      if (lVerb == OLEIVERB_INPLACEACTIVATE) {
         // Clip the hatch window to the size of pos rect so, that
         // hatch and object adornments  will not be visible.
         ServerDoc_ResizeInPlaceWindow(lpServerDoc,
            (LPRECT)&(lpServerDoc->m_lpIPData->rcPosRect),
            (LPRECT)&(lpServerDoc->m_lpIPData->rcPosRect)
         );
      }
```

```
#endif  // SVR_INSIDEOUT
   }

#if defined( SVR_INSIDEOUT )
   // OLE2NOTE: if verb is OLEIVERB_INPLACEACTIVATE we do NOT want to
   // show our UI
   if (lVerb == OLEIVERB_INPLACEACTIVATE) {

      return NOERROR;
   }
#endif  // SVR_INSIDEOUT

   if (! lpServerDoc->m_fUIActive) {
      lpServerDoc->m_fUIActive = TRUE;
      OLEDBG_BEGIN2("IOleInPlaceSite::OnUIActivate called\r\n");
      hrErr = lpIPData->lpSite->lpVtbl->OnUIActivate(lpIPData->lpSite);
      OLEDBG_END2
      if (hrErr != NOERROR) {
         lpServerDoc->m_fUIActive = FALSE;
         goto errRtn;
      }

      SetFocus(hWndDoc);

      // Show the object adornments and hacth border around them.
      ServerDoc_ResizeInPlaceWindow(lpServerDoc,
               (LPRECT)&lpIPData->rcPosRect,
               (LPRECT)&lpIPData->rcClipRect
      );

      /* OLE2NOTE: IOleInPlaceFrame::SetActiveObject must be called BEFORE
      **    IOleInPlaceFrame::SetMenu.
      */
      OLEDBG_BEGIN2("IOleInPlaceSite::SetActiveObject called\r\n");
      lpIPData->lpFrame->lpVtbl->SetActiveObject(
         lpIPData->lpFrame,
         (LPOLEINPLACEACTIVEOBJECT) &lpServerDoc->m_OleInPlaceActiveObject,
         (LPOLESTR)g_szIPObjectTitle
      );
      OLEDBG_END2

      /* OLE2NOTE: If the container wants to give ownership of the
      **    palette then he would sendmessage WM_QUEYNEWPALETTE to
      **    the object window proc, before returning from
      **    IOleInPlaceFrame::SetActiveObject. Those objects which
      **    want to be edited inplace only if they have the ownership of
      **    the palette, can check at this point in the code whether
      **    they got WM_QUERYNEWPALETTE or not. If they didn't get
      **    the message, then they can inplace deactivate and do open
      **    editing instead.
      */



      if (lpIPData->lpDoc) {
         lpIPData->lpDoc->lpVtbl->SetActiveObject(
            lpIPData->lpDoc,
            (LPOLEINPLACEACTIVEOBJECT)&lpServerDoc-
>m_OleInPlaceActiveObject,
            (LPOLESTR)g_szIPObjectTitle
         );
      }

      /* OLE2NOTE: install the menu and frame-level tools on the in-place
      **    frame.
      */
      ServerDoc_AddFrameLevelUI(lpServerDoc);
   }


   return NOERROR;

errRtn:
   ServerDoc_DoInPlaceDeactivate(lpServerDoc);

   return ResultFromScode(sc);
}



HRESULT ServerDoc_DoInPlaceDeactivate(LPSERVERDOC lpServerDoc)
{
   LPINPLACEDATA   lpIPData = lpServerDoc->m_lpIPData;
   LPOUTLINEDOC    lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;

   if (!lpServerDoc->m_fInPlaceActive)
      return S_OK;

   lpServerDoc->m_fInPlaceActive = FALSE;

   SvrDoc_IPObj_UIDeactivate(
         (LPOLEINPLACEOBJECT)&lpServerDoc->m_OleInPlaceObject);

   /* OLE2NOTE: an inside-out style in-place server will
   **    NOT hide its window in UIDeactive (an outside-in
   **    style object will hide its window in UIDeactivate).
   **    thus, an inside-out server must explicitly hide
   **    its window in InPlaceDeactivate. it is ALSO important for an
   **    outside-in style object to call ServerDoc_DoInPlaceHide here
   **    BEFORE freeing the InPlaceData structure. it will be common
   **    for in-place containers to call IOleInPlaceObject::
   **    InPlaceDeactivate in their IOleInPlaceSite::OnUIDeactiate
   **    implementation.
   */
   ServerDoc_DoInPlaceHide(lpServerDoc);

   OLEDBG_BEGIN2("IOleInPlaceSite::OnInPlaceDeactivate called\r\n");
   lpIPData->lpSite->lpVtbl->OnInPlaceDeactivate(lpIPData->lpSite);
   OLEDBG_END2
   OleStdRelease((LPUNKNOWN)lpIPData->lpSite);
   lpIPData->lpSite = NULL;

   ServerDoc_FreeInPlaceData(lpServerDoc);

   return NOERROR;
}


HRESULT ServerDoc_DoInPlaceHide(LPSERVERDOC lpServerDoc)
{
   LPINPLACEDATA   lpIPData = lpServerDoc->m_lpIPData;
   LPOUTLINEDOC    lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPOLEAPP        lpOleApp = (LPOLEAPP)g_lpApp;
   HWND            hWndApp = OutlineApp_GetWindow(g_lpApp);

   if (! lpServerDoc->m_fInPlaceVisible)
      return NOERROR;

   // Set the parent back to server app's window
   OleDoc_HideWindow((LPOLEDOC)lpServerDoc, FALSE /* fShutdown */);

   /* we need to enusure that our window is set to normal 100% zoom.
   **    if the window is next shown in open mode it should start out
   **    at normal zoom factor. our window may have been set to a
   **    different zoom factor while it was in-place active.
   */
   OutlineDoc_SetCurrentZoomCommand(lpOutlineDoc,IDM_V_ZOOM_100);

   lpServerDoc->m_fInPlaceVisible = FALSE;

   lpServerDoc->m_hWndParent = hWndApp;
   SetParent(
      lpOutlineDoc->m_hWndDoc,
      lpServerDoc->m_hWndParent
   );

   // make sure App busy/blocked dialogs are parented to our own hWndApp
   OleStdMsgFilter_SetParentWindow(lpOleApp->m_lpMsgFilter, hWndApp);

   // Hide the in-place hatch border window.
   ShowWindow(lpServerDoc->m_hWndHatch, SW_HIDE);

   ServerDoc_DisassembleMenus(lpServerDoc);

   /* we no longer need the IOleInPlaceFrame* or the doc's
   **    IOleInPlaceWindow* interface pointers.
   */
   if (lpIPData->lpDoc) {
      OleStdRelease((LPUNKNOWN)lpIPData->lpDoc);
      lpIPData->lpDoc = NULL;
   }

   if (lpIPData->lpFrame) {
      OleStdRelease((LPUNKNOWN)lpIPData->lpFrame);
      lpIPData->lpFrame = NULL;
   }

   ((LPSERVERAPP)g_lpApp)->m_lpIPData = NULL;

   return NOERROR;
}


BOOL ServerDoc_AllocInPlaceData(LPSERVERDOC lpServerDoc)
{
   LPINPLACEDATA   lpIPData;

   if (!(lpIPData = (LPINPLACEDATA) New(sizeof(INPLACEDATA))))
      return FALSE;

   lpIPData->lpFrame       = NULL;
   lpIPData->lpDoc         = NULL;
   lpIPData->lpSite        = NULL;
   lpIPData->hOlemenu      = NULL;
   lpIPData->hMenuShared   = NULL;

   lpServerDoc->m_lpIPData = lpIPData;
   return TRUE;
}


void ServerDoc_FreeInPlaceData(LPSERVERDOC lpServerDoc)
{
   Delete(lpServerDoc->m_lpIPData);
   lpServerDoc->m_lpIPData = NULL;
}


HRESULT ServerDoc_AssembleMenus(LPSERVERDOC lpServerDoc)
{
   HMENU           hMenuShared;
   LONG FAR*       lpWidths;
   UINT            uPosition;
   UINT            uPositionStart;
   LPSERVERAPP     lpServerApp = (LPSERVERAPP) g_lpApp;
   LPINPLACEDATA   lpIPData = lpServerDoc->m_lpIPData;
   HRESULT         hresult;
   BOOL            fNoError = TRUE;

   lpWidths = lpIPData->menuGroupWidths.width;
   hMenuShared = CreateMenu();

   if (hMenuShared &&
      (hresult = lpIPData->lpFrame->lpVtbl->InsertMenus(
         lpIPData->lpFrame, hMenuShared,
         &lpIPData->menuGroupWidths)) == NOERROR) {

      /* Insert EDIT group menus */
```

```
uPosition = (UINT)lpWidths[0]; /* # of menus in the FILE group */
uPositionStart = uPosition;

fNoError &= InsertMenu(
      hMenuShared,
      (UINT)uPosition,
      (UINT)(MF_BYPOSITION | MF_POPUP),
      (UINT)lpServerApp->m_hMenuEdit,
      (LPCSTR)"&Edit"
);
uPosition++;

lpWidths[1] = uPosition - uPositionStart;

/* Insert OBJECT group menus */

uPosition += (UINT)lpWidths[2];
uPositionStart = uPosition;

fNoError &= InsertMenu(
      hMenuShared,
      (UINT)uPosition,
      (UINT)(MF_BYPOSITION | MF_POPUP),
      (UINT)lpServerApp->m_hMenuLine,
      (LPCSTR)"&Line"
);
uPosition++;

fNoError &= InsertMenu(
      hMenuShared,
      (UINT)uPosition,
      (UINT)(MF_BYPOSITION | MF_POPUP),
      (UINT)lpServerApp->m_hMenuName,
      (LPCSTR)"&Name"
);
uPosition++;

fNoError &= InsertMenu(
      hMenuShared,
      (UINT)uPosition,
      (UINT)(MF_BYPOSITION | MF_POPUP),
      (UINT)lpServerApp->m_hMenuOptions,
      (LPCSTR)"&Options"
);
uPosition++;

fNoError &= InsertMenu(
      hMenuShared,
      (UINT)uPosition,
      (UINT)(MF_BYPOSITION | MF_POPUP),
      (UINT)lpServerApp->m_hMenuDebug,
      (LPCSTR)"DbgI&Svr"
);
uPosition++;
```

```
      lpWidths[3] = uPosition - uPositionStart;

      /* Insert HELP group menus */

      uPosition += (UINT) lpWidths[4]; /* # of menus in WINDOW group */
      uPositionStart = uPosition;

      fNoError &= InsertMenu(
            hMenuShared,
            (UINT)uPosition,
            (UINT)(MF_BYPOSITION | MF_POPUP),
            (UINT)lpServerApp->m_hMenuHelp,
            (LPCSTR)"&Help"
      );
      uPosition++;

      lpWidths[5] = uPosition - uPositionStart;

      OleDbgAssert(fNoError == TRUE);

   } else {
      /* In-place container does not allow us to add menus to the
      **    frame.
      ** OLE2NOTE: even when the in-place container does NOT allow
      **    the building of a merged menu bar, it is CRITICAL that
      **    the in-place server still call OleCreateMenuDescriptor
      **    passing NULL for hMenuShared.
      */
      if (hMenuShared) {
         DestroyMenu(hMenuShared);
         hMenuShared = NULL;
      }
   }

   lpIPData->hMenuShared = hMenuShared;

   if (!(lpIPData->hOlemenu = OleCreateMenuDescriptor(hMenuShared,
                                 &lpIPData->menuGroupWidths)))
    {
      return ResultFromScode(E_OUTOFMEMORY);
    }

   return NOERROR;
}


void ServerDoc_DisassembleMenus(LPSERVERDOC lpServerDoc)
{
   UINT             uCount;
   UINT            uGroup;
   UINT            uDeleteAt;
   LPINPLACEDATA   lpIPData = lpServerDoc->m_lpIPData;
   LONG FAR*       lpWidths = lpIPData->menuGroupWidths.width;
   BOOL            fNoError = TRUE;
   /* OLE2NOTE: even when hMenuShared is NULL (ie. the server has no
   **    Menu), there is still an hOleMenu created that must be destroyed.
   */
   if (lpIPData->hOlemenu) {
      OleDestroyMenuDescriptor (lpIPData->hOlemenu);
      lpIPData->hOlemenu = NULL;
   }

   if (! lpIPData->hMenuShared)
      return;     // no menus to be destroyed

   /* Remove server group menus. */
   uDeleteAt = 0;
   for (uGroup = 0; uGroup < 6; uGroup++) {
      uDeleteAt += (UINT)lpWidths[uGroup++];
      for (uCount = 0; uCount < (UINT)lpWidths[uGroup]; uCount++)
         fNoError &= RemoveMenu(lpIPData->hMenuShared, uDeleteAt,
                        MF_BYPOSITION);
   }

   /* Remove container group menus */
   fNoError &= (lpIPData->lpFrame->lpVtbl->RemoveMenus(
      lpIPData->lpFrame,
      lpIPData->hMenuShared) == NOERROR);

   OleDbgAssert(fNoError == TRUE);

   DestroyMenu(lpIPData->hMenuShared);
   lpIPData->hMenuShared = NULL;
}


/* ServerDoc_UpdateInPlaceWindowOnExtentChange
** -------------------------------------------
**    The size of the in-place window needs to be changed.
**    calculate the size required in Client coordinates (taking into
**    account the current scale factor imposed by the in-place
**    container) and ask our in-place container to allow us to resize.
**    our container must call us back via
**    IOleInPlaceObject::SetObjectRects for the actual sizing to take
**    place.
**
**    OLE2NOTE: the rectangle that we ask for from our in-place
**    container is always the rectangle required for the object display
**    itself (in our case the size of the LineList contents). it does
**    NOT include the space we require for object frame adornments.
*/
void ServerDoc_UpdateInPlaceWindowOnExtentChange(LPSERVERDOC lpServerDoc)
{
   SIZEL       sizelHim;
   SIZEL       sizelPix;
   RECT        rcPosRect;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPLINELIST  lpLL=(LPLINELIST)&lpOutlineDoc->m_LineList;
   HWND        hWndLL = lpLL->m_hWndListBox;
   LPSCALEFACTOR lpscale = (LPSCALEFACTOR)&lpOutlineDoc->m_scale;

   if (!lpServerDoc->m_fInPlaceActive)
      return;

   OleDoc_GetExtent((LPOLEDOC)lpServerDoc, (LPSIZEL)&sizelHim);

   // apply current scale factor
   sizelHim.cx = sizelHim.cx * lpscale->dwSxN / lpscale->dwSxD;
   sizelHim.cy = sizelHim.cy * lpscale->dwSxN / lpscale->dwSxD;
   XformSizeInHimetricToPixels(NULL, (LPSIZEL)&sizelHim,
(LPSIZEL)&sizelPix);

   GetWindowRect(hWndLL, (LPRECT)&rcPosRect);
   ScreenToClient(lpServerDoc->m_hWndParent, (POINT FAR *)&rcPosRect);

   rcPosRect.right = rcPosRect.left + (int) sizelPix.cx;
   rcPosRect.bottom = rcPosRect.top + (int) sizelPix.cy;
   OleDbgOutRect3("ServerDoc_UpdateInPlaceWindowOnExtentChange: (PosRect)",
(LPRECT)&rcPosRect);

   OLEDBG_BEGIN2("IOleInPlaceSite::OnPosRectChange called\r\n");
   lpServerDoc->m_lpIPData->lpSite->lpVtbl->OnPosRectChange(
         lpServerDoc->m_lpIPData->lpSite,
         (LPRECT) &rcPosRect
   );
   OLEDBG_END2
}


/* ServerDoc_CalcInPlaceWindowPos
 * ------------------------------
 *
 *  Move (and re-scale) the ServerDoc to the specified rectangle.
 *
 *  Parameters:
 *      lprcListBox - rect in client coordinate in which the listbox will
fit
 *      lprcDoc     - corresponding size of the Doc in client coordinate
 *
 */
void ServerDoc_CalcInPlaceWindowPos(
      LPSERVERDOC         lpServerDoc,
      LPRECT              lprcListBox,
      LPRECT              lprcDoc,
      LPSCALEFACTOR       lpscale
)
{
   SIZEL sizelHim;
   SIZEL sizelPix;
   LPLINELIST lpLL;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPHEADING lphead;

   if (!lpServerDoc || !lprcListBox || !lprcDoc)
```

```
      return;
   lphead = (LPHEADING)&lpOutlineDoc->m_heading;

   lpLL = OutlineDoc_GetLineList(lpOutlineDoc);
   OleDoc_GetExtent((LPOLEDOC)lpServerDoc, (LPSIZEL)&sizelHim);
   XformSizeInHimetricToPixels(NULL, &sizelHim, &sizelPix);

   if (sizelHim.cx == 0 || sizelPix.cx == 0) {
      lpscale->dwSxN = 1;
      lpscale->dwSxD = 1;
   } else {
      lpscale->dwSxN = lprcListBox->right - lprcListBox->left;
      lpscale->dwSxD = sizelPix.cx;
   }

   if (sizelHim.cy == 0 || sizelPix.cy == 0) {
      lpscale->dwSyN = 1;
      lpscale->dwSyD = 1;
   } else {
      lpscale->dwSyN = lprcListBox->bottom - lprcListBox->top;
      lpscale->dwSyD = sizelPix.cy;
   }

   lprcDoc->left = lprcListBox->left - Heading_RH_GetWidth(lphead,lpscale);
   lprcDoc->right = lprcListBox->right;
   lprcDoc->top = lprcListBox->top - Heading_CH_GetHeight(lphead,lpscale);
   lprcDoc->bottom = lprcListBox->bottom;
}


/* ServerDoc_ResizeInPlaceWindow
** -----------------------------
**    Actually resize the in-place ServerDoc windows according to the
**    PosRect and ClipRect allowed by our in-place container.
**
**    OLE2NOTE: the PosRect rectangle that our in-place container tells
**    us is always the rectangle required for the object display
**    itself (in our case the size of the LineList contents). it does
**    NOT include the space we require for object frame adornments.
*/
void ServerDoc_ResizeInPlaceWindow(
      LPSERVERDOC         lpServerDoc,
      LPCRECT             lprcPosRect,
      LPCRECT             lprcClipRect
)
{
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPLINELIST   lpLL = (LPLINELIST)&lpOutlineDoc->m_LineList;
   SCALEFACTOR  scale;
   RECT         rcDoc;
   POINT        ptOffset;

   /* OLE2NOTE: calculate the space needed for our object frame
   **    adornments. our in-place container tells us the size that our
   **    object should take in window client coordinates
   **    (lprcPosRect). the rectangle cooresponds to the size that our
   **    LineList ListBox should be. our Doc window must the correct
   **    amount larger to accomodate our row/column headings.
   **    then move all windows into position.
   */
   ServerDoc_CalcInPlaceWindowPos(
         lpServerDoc,
         (LPRECT)lprcPosRect,
         (LPRECT)&rcDoc,
         (LPSCALEFACTOR)&scale
   );

   /* OLE2NOTE: we need to honor the lprcClipRect specified by our
   **    in-place container. we must NOT draw outside of the ClipRect.
   **    in order to achieve this, we will size the hatch window to be
   **    exactly the size that should be visible (rcVisRect). the
   **    rcVisRect is defined as the intersection of the full size of
   **    the in-place server window and the lprcClipRect.
   **    the ClipRect could infact clip the HatchRect on the
   **    right/bottom and/or on the top/left. if it is clipped on the
   **    right/bottom then it is sufficient to simply resize the hatch
   **    window. but if the HatchRect is clipped on the top/left then
   **    we must "move" the ServerDoc window (child of HatchWindow) by
   **    the delta that was clipped. the window origin of the
   **    ServerDoc window will then have negative coordinates relative
   **    to its parent HatchWindow.
   */
   SetHatchWindowSize(
         lpServerDoc->m_hWndHatch,
         (LPRECT)&rcDoc,
         (LPRECT)lprcClipRect,
         (LPPOINT)&ptOffset
   );

   // shift Doc window to account for hatch frame being drawn
   OffsetRect((LPRECT)&rcDoc, ptOffset.x, ptOffset.y);

   // move/size/set scale factor of ServerDoc window.
   OutlineDoc_SetScaleFactor(
         lpOutlineDoc, (LPSCALEFACTOR)&scale, (LPRECT)&rcDoc);

   /* reset the horizontal extent of the listbox. this makes
   **    the listbox realize that a scroll bar is not needed.
   */
   SendMessage(
         lpLL->m_hWndListBox,
         LB_SETHORIZONTALEXTENT,
         (int) 0,
         0L
   );
   SendMessage(
         lpLL->m_hWndListBox,
         LB_SETHORIZONTALEXTENT,
         (int) (lprcPosRect->right - lprcPosRect->left),
         0L
```

```
   );
}


/* ServerDoc_SetStatusText
**    Tell the active in-place frame to display a status message.
*/
void ServerDoc_SetStatusText(LPSERVERDOC lpServerDoc, LPOLESTR lpszMessage)
{
   if (lpServerDoc && lpServerDoc->m_fUIActive &&
      lpServerDoc->m_lpIPData != NULL) {

      OLEDBG_BEGIN2("IOleInPlaceFrame::SetStatusText called\r\n")
      lpServerDoc->m_lpIPData->lpFrame->lpVtbl->SetStatusText
            (lpServerDoc->m_lpIPData->lpFrame, lpszMessage);
      OLEDBG_END2
   }
}


/* ServerDoc_GetTopInPlaceFrame
** ----------------------------
**    returns NON-AddRef'ed pointer to Top In-Place Frame interface
*/
LPOLEINPLACEFRAME ServerDoc_GetTopInPlaceFrame(LPSERVERDOC lpServerDoc)
{
   if (lpServerDoc->m_lpIPData)
      return lpServerDoc->m_lpIPData->lpFrame;
   else
      return NULL;
}

void ServerDoc_GetSharedMenuHandles(
      LPSERVERDOC lpServerDoc,
      HMENU FAR*      lphSharedMenu,
      HOLEMENU FAR*   lphOleMenu
)
{
   if (lpServerDoc->m_lpIPData) {
      *lphSharedMenu = lpServerDoc->m_lpIPData->hMenuShared;
      *lphOleMenu = lpServerDoc->m_lpIPData->hOlemenu;
   } else {
      *lphSharedMenu = NULL;
      *lphOleMenu = NULL;
   }
}


void ServerDoc_AddFrameLevelUI(LPSERVERDOC lpServerDoc)
{
   LPOUTLINEAPP lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPSERVERAPP lpServerApp = (LPSERVERAPP)g_lpApp;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPOLEINPLACEFRAME lpTopIPFrame=ServerDoc_GetTopInPlaceFrame(lpServerDoc);
   HMENU           hSharedMenu;            // combined obj/cntr menu
   HOLEMENU        hOleMenu;               // returned by OleCreateMenuDesc.

   ServerDoc_GetSharedMenuHandles(
         lpServerDoc,
         &hSharedMenu,
         &hOleMenu
   );

   lpTopIPFrame->lpVtbl->SetMenu(
         lpTopIPFrame,
         hSharedMenu,
         hOleMenu,
         lpOutlineDoc->m_hWndDoc
   );

   // save normal accelerator table
   lpServerApp->m_hAccelBaseApp = lpOutlineApp->m_hAccelApp;

   // install accelerator table for UIActive server (w/ active editor cmds)
   lpOutlineApp->m_hAccel = lpServerApp->m_hAccelIPSvr;
   lpOutlineApp->m_hAccelApp = lpServerApp->m_hAccelIPSvr;
   lpOutlineApp->m_hWndAccelTarget = lpOutlineDoc->m_hWndDoc;

#if defined( USE_FRAMETOOLS )
   ServerDoc_AddFrameLevelTools(lpServerDoc);

   // update toolbar button enable states
   OutlineDoc_UpdateFrameToolButtons(lpOutlineDoc);
#endif
}


void ServerDoc_AddFrameLevelTools(LPSERVERDOC lpServerDoc)
{
   LPOUTLINEAPP    lpOutlineApp = (LPOUTLINEAPP)g_lpApp;
   LPSERVERAPP     lpServerApp = (LPSERVERAPP)g_lpApp;
   LPOUTLINEDOC    lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPOLEINPLACEFRAME lpTopIPFrame=ServerDoc_GetTopInPlaceFrame(lpServerDoc);

#if defined( USE_FRAMETOOLS )
   HWND            hWndFrame;

   FrameTools_Enable(lpOutlineDoc->m_lpFrameTools, TRUE);

   // if not in-place UI active, add our tools to our own frame.
   if (! lpServerDoc->m_fUIActive) {
      OutlineDoc_AddFrameLevelTools(lpOutlineDoc);
      return;
   }

   if ((hWndFrame = OutlineApp_GetFrameWindow(lpOutlineApp)) == NULL) {
      /* we could NOT get a valid frame window, so POP our tools up. */

      /* OLE2NOTE: since we are poping up our tools, we MUST inform
      **    the top in-place frame window that we need NO tool space
      **    BUT that it should NOT put its own tools up. if we were
      **    to pass NULL instead of (0,0,0,0), then the container
      **    would have the option to leave its own tools up.
      */
      lpTopIPFrame->lpVtbl->SetBorderSpace(
            lpTopIPFrame,
            (LPCBORDERWIDTHS)&g_rectNull
      );
      FrameTools_PopupTools(lpOutlineDoc->m_lpFrameTools);
   } else {

      /* OLE2NOTE: we need to negotiate for space and attach our frame
      **    level tools to the top-level in-place container's frame window.
      */
      FrameTools_AttachToFrame(lpOutlineDoc->m_lpFrameTools, hWndFrame);

      FrameTools_NegotiateForSpaceAndShow(
            lpOutlineDoc->m_lpFrameTools,
            NULL,
            lpTopIPFrame
      );
   }

#else   // ! USE_FRAMETOOLS
   /* OLE2NOTE: if you do NOT use frame tools, you MUST inform the top
   **    in-place frame window so that it can put back its own tools.
   */
   lpTopIPFrame->lpVtbl->SetBorderSpace(lpIPData->lpFrame, NULL);
#endif  // ! USE_FRAMETOOLS
}


#if defined( USE_FRAMETOOLS )

void ServerDoc_RemoveFrameLevelTools(LPSERVERDOC lpServerDoc)
{
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   OleDbgAssert(lpOutlineDoc->m_lpFrameTools != NULL);

    // Reparent our tools back to one of our own windows
    FrameTools_AttachToFrame(lpOutlineDoc->m_lpFrameTools,g_lpApp-
>m_hWndApp);

   FrameTools_Enable(lpOutlineDoc->m_lpFrameTools, FALSE);
}
#endif  // USE_FRAMETOOLS



void ServerDoc_UIActivate (LPSERVERDOC lpServerDoc)
{
   if (lpServerDoc->m_fInPlaceActive && !lpServerDoc->m_fUIActive) {
      ServerDoc_DoInPlaceActivate(lpServerDoc,
            OLEIVERB_UIACTIVATE,
            NULL /*lpmsg*/,
                lpServerDoc->m_lpOleClientSite
        );
        OutlineDoc_ShowWindow((LPOUTLINEDOC)lpServerDoc);
    }
}
```

SVROUTL.DEF   (OUTLINE Sample)

```
;;
;;
;;    OLE 2.0 Server Sample Code
;;
;;    svroutl.def
;;
;;    Definition file for svroutl.exe
;;
;;    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
;;
;;

NAME            SvrOutl
DESCRIPTION             'Microsoft OLE 2.0 Server Sample code'
EXETYPE         WINDOWS
STUB            'WINSTUB.EXE'
CODE            PRELOAD MOVEABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        4096

;; NOTE: Do not add exports to this file.  Use __export
;; in your function prototype and definition instead.
```

SVROUTL.H (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Server Sample Code
**
**    svroutl.h
**
**    This file contains file contains data structure defintions,
**    function prototypes, constants, etc. used by the OLE 2.0 server
**    app version of the Outline series of sample applications:
**          Outline -- base version of the app (without OLE functionality)
**          SvrOutl -- OLE 2.0 Server sample app
**          CntrOutl -- OLE 2.0 Containter sample app
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#if !defined( _SVROUTL_H_ )
#define _SVROUTL_H_

#ifndef RC_INVOKED
#pragma message ("INCLUDING SVROUTL.H from " __FILE__)
#endif  /* RC_INVOKED */

#include "oleoutl.h"
#include "ansiapi.h"

/* Defines */

// Enable SVROUTL and ISVROTL to emulate each other (TreatAs aka.
ActivateAs)
#define SVR_TREATAS     1

// Enable SVROUTL and ISVROTL to convert each other (TreatAs aka.
ActivateAs)
#define SVR_CONVERTTO   1

// Enable ISVROTL to operate as in inside-out style in-place object
#define SVR_INSIDEOUT   1

/* Default name used for container of the embedded object. used if
**    container forgets to call IOleObject::SetHostNames
*/
// REVIEW: should load from string resource
#define DEFCONTAINERNAME    "Unknown Document"

/* Default prefix for auto-generated range names. This is used with
**    links to unnamed ranges (pseudo objects).
*/
// REVIEW: should load from string resource
#define DEFRANGENAMEPREFIX  "Range"
```

```
// Maximum length of strings accepted through IOleObject::SetHostNames
//      (note: this is rather arbitrary; a better strategy would be to
//             dynamically allocated buffers for these strings.)
#define MAXAPPNAME  80
#define MAXCONTAINERNAME    80

// Menu option in embedding mode
#define IDM_F_UPDATE    1151

/* Types */

/* Codes for CallBack events */
typedef enum tagOLE_NOTIFICATION {
   OLE_ONDATACHANGE,        // 0
   OLE_ONSAVE,              // 1
   OLE_ONRENAME,            // 2
   OLE_ONCLOSE              // 3
} OLE_NOTIFICATION;

/* Codes to indicate mode of storage for an object.
**    Mode of the storage is modified by the IPersistStorage methods:
**      Save, HandsOffStorage, and SaveCompleted.
*/
typedef enum tagSTGMODE {
   STGMODE_NORMAL      = 0,
   STGMODE_NOSCRIBBLE  = 1,
   STGMODE_HANDSOFF    = 2
} STGMODE;


/* Forward type definitions */
typedef struct tagSERVERAPP FAR* LPSERVERAPP;
typedef struct tagSERVERDOC FAR* LPSERVERDOC;
typedef struct tagPSEUDOOBJ FAR* LPPSEUDOOBJ;

typedef struct tagINPLACEDATA {
   OLEMENUGROUPWIDTHS      menuGroupWidths;
   HOLEMENU                hOlemenu;
   HMENU                   hMenuShared;
   LPOLEINPLACESITE        lpSite;
   LPOLEINPLACEUIWINDOW    lpDoc;
   LPOLEINPLACEFRAME       lpFrame;
   OLEINPLACEFRAMEINFO     frameInfo;
   HWND                    hWndFrame;
   BOOL                    fBorderOn;
   RECT                    rcPosRect;
   RECT                    rcClipRect;
} INPLACEDATA, FAR* LPINPLACEDATA;


/*************************************************************************
** class SERVERDOC : OLEDOC
**    SERVERDOC is an extention to the abstract base OLEDOC class.
**    The OLEDOC class defines the fields, methods and interfaces that
**    are common to both server and client implementations. The
**    SERVERDOC class adds the fields, methods and interfaces that are
**    specific to OLE 2.0 Server functionality. There is one instance
**    of SERVERDOC object created per document open in the app. The SDI
**    version of the app supports one SERVERDOC at a time. The MDI
**    version of the app can manage multiple documents at one time.
**    The SERVERDOC class inherits all fields from the OLEDOC class.
**    This inheritance is achieved by including a member variable of
**    type OLEDOC as the first field in the SERVERDOC structure. Thus a
**    pointer to a SERVERDOC object can be cast to be a pointer to a
**    OLEDOC object or an OUTLINEDOC object
*************************************************************************/

typedef struct tagSERVERDOC {
   OLEDOC              m_OleDoc;           // ServerDoc inherits from OleDoc
   ULONG               m_cPseudoObj;       // total count of pseudo obj's
   LPOLECLIENTSITE     m_lpOleClientSite;  // Client associated with the obj
   LPOLEADVISEHOLDER   m_lpOleAdviseHldr;  // helper obj to hold ole advises
   LPDATAADVISEHOLDER  m_lpDataAdviseHldr; // helper obj to hold data
advises
   BOOL                m_fNoScribbleMode;  // was IPS::Save called
   BOOL                m_fSaveWithSameAsLoad;  // was IPS::Save called with
                                 // fSameAsLoad==TRUE.
   OLECHAR             m_szContainerApp[MAXAPPNAME];
   OLECHAR             m_szContainerObj[MAXCONTAINERNAME];
   ULONG               m_nNextRangeNo;     // next no. for unnamed range
   LINERANGE           m_lrSrcSelOfCopy;   // src sel if doc created for
copy
   BOOL                m_fDataChanged;     // data changed when draw
disabled
   BOOL                m_fSizeChanged;     // size changed when draw
disabled
   BOOL                m_fSendDataOnStop;  // did data ever change?
#if defined( SVR_TREATAS )
   CLSID               m_clsidTreatAs;     // clsid to pretend to be
   LPOLESTR            m_lpszTreatAsType;  // user type name to pretend to
be
#endif  // SVR_TREATAS

#if defined( LATER )
   // REVIEW: is it necessary to register a WildCard Moniker
   DWORD               m_dwWildCardRegROT; // key if wildcard reg'ed in ROT
#endif

#if defined( INPLACE_SVR )
   BOOL                m_fInPlaceActive;
   BOOL                m_fInPlaceVisible;
   BOOL                m_fUIActive;
   HWND                m_hWndParent;
   HWND                m_hWndHatch;
   LPINPLACEDATA       m_lpIPData;
   BOOL                m_fMenuHelpMode;// is F1 pressed in menu, give help

   struct CDocOleInPlaceObjectImpl {
      IOleInPlaceObjectVtbl FAR*  lpVtbl;
      LPSERVERDOC                 lpServerDoc;
      int                         cRef;   // interface specific ref count.
   } m_OleInPlaceObject;

   struct CDocOleInPlaceActiveObjectImpl {
      IOleInPlaceActiveObjectVtbl FAR* lpVtbl;
      LPSERVERDOC                      lpServerDoc;
      int                              cRef;// interface specific ref count.
   } m_OleInPlaceActiveObject;
#endif // INPLACE_SVR

   struct CDocOleObjectImpl {
      IOleObjectVtbl FAR*         lpVtbl;
      LPSERVERDOC                 lpServerDoc;
      int                         cRef;   // interface specific ref count.
   } m_OleObject;

   struct CDocPersistStorageImpl {
      IPersistStorageVtbl FAR*    lpVtbl;
      LPSERVERDOC                 lpServerDoc;
      int                         cRef;   // interface specific ref count.
   } m_PersistStorage;

#if defined( SVR_TREATAS )
   struct CDocStdMarshalInfoImpl {
      IStdMarshalInfoVtbl FAR*    lpVtbl;
      LPSERVERDOC                 lpServerDoc;
      int                         cRef;   // interface specific ref count.
   } m_StdMarshalInfo;
#endif  // SVR_TREATAS

} SERVERDOC;

/* ServerDoc methods (functions) */
BOOL ServerDoc_Init(LPSERVERDOC lpServerDoc, BOOL fDataTransferDoc);
BOOL ServerDoc_InitNewEmbed(LPSERVERDOC lpServerDoc);
void ServerDoc_PseudoObjUnlockDoc(
      LPSERVERDOC         lpServerDoc,
      LPPSEUDOOBJ         lpPseudoObj
);
void ServerDoc_PseudoObjLockDoc(LPSERVERDOC lpServerDoc);
BOOL ServerDoc_PasteFormatFromData(
      LPSERVERDOC             lpServerDoc,
      CLIPFORMAT              cfFormat,
      LPDATAOBJECT            lpSrcDataObj,
      BOOL                    fLocalDataObj,
      BOOL                    fLink
);
BOOL ServerDoc_QueryPasteFromData(
      LPSERVERDOC             lpServerDoc,
      LPDATAOBJECT            lpSrcDataObj,
      BOOL                    fLink
);
HRESULT ServerDoc_GetClassID(LPSERVERDOC lpServerDoc, LPCLSID lpclsid);
void ServerDoc_UpdateMenu(LPSERVERDOC lpServerDoc);
void ServerDoc_RestoreMenu(LPSERVERDOC lpServerDoc);
```

```
HRESULT ServerDoc_GetData (
      LPSERVERDOC             lpServerDoc,
      LPFORMATETC             lpformatetc,
      LPSTGMEDIUM             lpMedium
);
HRESULT ServerDoc_GetDataHere (
      LPSERVERDOC             lpServerDoc,
      LPFORMATETC             lpformatetc,
      LPSTGMEDIUM             lpMedium
);
HRESULT ServerDoc_QueryGetData(LPSERVERDOC lpServerDoc,LPFORMATETC
lpformatetc);
HRESULT ServerDoc_EnumFormatEtc(
      LPSERVERDOC             lpServerDoc,
      DWORD                   dwDirection,
      LPENUMFORMATETC FAR*    lplpenumFormatEtc
);
HANDLE ServerDoc_GetMetafilePictData(
      LPSERVERDOC             lpServerDoc,
      LPLINERANGE             lplrSel
);
void ServerDoc_SendAdvise(
      LPSERVERDOC     lpServerDoc,
      WORD            wAdvise,
      LPMONIKER       lpmkDoc,
      DWORD           dwAdvf
);
HRESULT ServerDoc_GetObject(
      LPSERVERDOC             lpServerDoc,
      LPOLESTR                   lpszItem,
      REFIID                  riid,
      LPVOID FAR*             lplpvObject
);
HRESULT ServerDoc_IsRunning(LPSERVERDOC lpServerDoc, LPOLESTR lpszItem);
LPMONIKER ServerDoc_GetSelRelMoniker(
      LPSERVERDOC             lpServerDoc,
      LPLINERANGE             lplrSel,
      DWORD                   dwAssign
);
LPMONIKER ServerDoc_GetSelFullMoniker(
      LPSERVERDOC             lpServerDoc,
      LPLINERANGE             lplrSel,
      DWORD                   dwAssign
);


#if defined( INPLACE_SVR )
HRESULT ServerDoc_DoInPlaceActivate(
      LPSERVERDOC     lpServerDoc,
      LONG            lVerb,
      LPMSG           lpmsg,
      LPOLECLIENTSITE lpActiveSite
);
HRESULT ServerDoc_DoInPlaceDeactivate(LPSERVERDOC lpServerDoc);
HRESULT ServerDoc_DoInPlaceHide(LPSERVERDOC lpServerDoc);
```

```
BOOL ServerDoc_AllocInPlaceData(LPSERVERDOC lpServerDoc);
void ServerDoc_FreeInPlaceData(LPSERVERDOC lpServerDoc);

HRESULT ServerDoc_AssembleMenus(LPSERVERDOC lpServerDoc);
void    ServerDoc_DisassembleMenus(LPSERVERDOC lpServerDoc);
void ServerDoc_CalcInPlaceWindowPos(
      LPSERVERDOC         lpServerDoc,
      LPRECT              lprcListBox,
      LPRECT              lprcDoc,
      LPSCALEFACTOR       lpscale
);
void ServerDoc_UpdateInPlaceWindowOnExtentChange(LPSERVERDOC lpServerDoc);
void ServerDoc_ResizeInPlaceWindow(
      LPSERVERDOC         lpServerDoc,
      LPCRECT             lprcPosRect,
      LPCRECT             lprcClipRect
);
void ServerDoc_ShadeInPlaceBorder(LPSERVERDOC lpServerDoc, BOOL fShadeOn);
void ServerDoc_SetStatusText(LPSERVERDOC lpServerDoc, LPOLESTR lpszMessage);
LPOLEINPLACEFRAME ServerDoc_GetTopInPlaceFrame(LPSERVERDOC lpServerDoc);
void ServerDoc_GetSharedMenuHandles(
      LPSERVERDOC lpServerDoc,
      HMENU FAR*      lphSharedMenu,
      HOLEMENU FAR*   lphOleMenu
);
void ServerDoc_AddFrameLevelUI(LPSERVERDOC lpServerDoc);
void ServerDoc_AddFrameLevelTools(LPSERVERDOC lpServerDoc);
void ServerDoc_UIActivate (LPSERVERDOC lpServerDoc);

#if defined( USE_FRAMETOOLS )
void ServerDoc_RemoveFrameLevelTools(LPSERVERDOC lpServerDoc);
#endif // USE_FRAMETOOLS

#endif // INPLACE_SVR


/* ServerDoc::IOleObject methods (functions) */
STDMETHODIMP SvrDoc_OleObj_QueryInterface(
      LPOLEOBJECT             lpThis,
      REFIID                  riid,
      LPVOID FAR*             lplpvObj
);
STDMETHODIMP_(ULONG) SvrDoc_OleObj_AddRef(LPOLEOBJECT lpThis);
STDMETHODIMP_(ULONG) SvrDoc_OleObj_Release(LPOLEOBJECT lpThis);
STDMETHODIMP SvrDoc_OleObj_SetClientSite(
      LPOLEOBJECT             lpThis,
      LPOLECLIENTSITE         lpclientSite
);
STDMETHODIMP SvrDoc_OleObj_GetClientSite(
      LPOLEOBJECT             lpThis,
      LPOLECLIENTSITE FAR*    lplpClientSite
);
STDMETHODIMP SvrDoc_OleObj_SetHostNames(
      LPOLEOBJECT             lpThis,
      LPCOLESTR               szContainerApp,
      LPCOLESTR               szContainerObj
);
STDMETHODIMP SvrDoc_OleObj_Close(
      LPOLEOBJECT             lpThis,
      DWORD                   dwSaveOption
);
STDMETHODIMP SvrDoc_OleObj_SetMoniker(
      LPOLEOBJECT             lpThis,
      DWORD                   dwWhichMoniker,
      LPMONIKER               lpmk
);
STDMETHODIMP SvrDoc_OleObj_GetMoniker(
      LPOLEOBJECT             lpThis,
      DWORD                   dwAssign,
      DWORD                   dwWhichMoniker,
      LPMONIKER FAR*          lplpmk
);
STDMETHODIMP SvrDoc_OleObj_InitFromData(
      LPOLEOBJECT             lpThis,
      LPDATAOBJECT            lpDataObject,
      BOOL                    fCreation,
      DWORD                   reserved
);
STDMETHODIMP SvrDoc_OleObj_GetClipboardData(
      LPOLEOBJECT             lpThis,
      DWORD                   reserved,
      LPDATAOBJECT FAR*       lplpDataObject
);
STDMETHODIMP SvrDoc_OleObj_DoVerb(
      LPOLEOBJECT             lpThis,
      LONG                    lVerb,
      LPMSG                   lpmsg,
      LPOLECLIENTSITE         lpActiveSite,
      LONG                    lindex,
      HWND                    hwndParent,
      LPCRECT                 lprcPosRect
);
STDMETHODIMP SvrDoc_OleObj_EnumVerbs(
      LPOLEOBJECT             lpThis,
      LPENUMOLEVERB FAR*      lplpenumOleVerb
);
STDMETHODIMP SvrDoc_OleObj_Update(LPOLEOBJECT lpThis);
STDMETHODIMP SvrDoc_OleObj_IsUpToDate(LPOLEOBJECT lpThis);
STDMETHODIMP SvrDoc_OleObj_GetUserClassID(
      LPOLEOBJECT             lpThis,
      LPCLSID                 lpclsid
);
STDMETHODIMP SvrDoc_OleObj_GetUserType(
      LPOLEOBJECT             lpThis,
      DWORD                   dwFormOfType,
      LPOLESTR FAR*           lpszUserType
);
STDMETHODIMP SvrDoc_OleObj_SetExtent(
      LPOLEOBJECT             lpThis,
      DWORD                   dwDrawAspect,
      LPSIZEL                lplgrc
);
STDMETHODIMP SvrDoc_OleObj_GetExtent(
      LPOLEOBJECT            lpThis,
      DWORD                  dwDrawAspect,
      LPSIZEL                lplgrc
);
STDMETHODIMP SvrDoc_OleObj_Advise(
      LPOLEOBJECT            lpThis,
      LPADVISESINK           lpAdvSink,
      LPDWORD                lpdwConnection
);
STDMETHODIMP SvrDoc_OleObj_Unadvise(LPOLEOBJECT lpThis, DWORD dwConnection);
STDMETHODIMP SvrDoc_OleObj_EnumAdvise(
      LPOLEOBJECT            lpThis,
      LPENUMSTATDATA FAR*    lplpenumAdvise
);
STDMETHODIMP SvrDoc_OleObj_GetMiscStatus(
      LPOLEOBJECT            lpThis,
      DWORD                  dwAspect,
      DWORD FAR*             lpdwStatus
);
STDMETHODIMP SvrDoc_OleObj_SetColorScheme(
      LPOLEOBJECT            lpThis,
      LPLOGPALETTE           lpLogpal
);
STDMETHODIMP SvrDoc_OleObj_LockObject(
      LPOLEOBJECT            lpThis,
      BOOL                   fLock
);

/* ServerDoc::IPersistStorage methods (functions) */
STDMETHODIMP SvrDoc_PStg_QueryInterface(
      LPPERSISTSTORAGE       lpThis,
      REFIID                 riid,
      LPVOID FAR*            lplpvObj
);
STDMETHODIMP_(ULONG) SvrDoc_PStg_AddRef(LPPERSISTSTORAGE lpThis);
STDMETHODIMP_(ULONG) SvrDoc_PStg_Release(LPPERSISTSTORAGE lpThis);
STDMETHODIMP SvrDoc_PStg_GetClassID(
      LPPERSISTSTORAGE       lpThis,
      LPCLSID                lpClassID
);
STDMETHODIMP  SvrDoc_PStg_IsDirty(LPPERSISTSTORAGE  lpThis);
STDMETHODIMP SvrDoc_PStg_InitNew(
      LPPERSISTSTORAGE       lpThis,
      LPSTORAGE              lpStg
);
STDMETHODIMP SvrDoc_PStg_Load(
      LPPERSISTSTORAGE       lpThis,
      LPSTORAGE              lpStg
);
STDMETHODIMP SvrDoc_PStg_Save(
      LPPERSISTSTORAGE       lpThis,
      LPSTORAGE              lpStg,
      BOOL                    fSameAsLoad
);
STDMETHODIMP SvrDoc_PStg_SaveCompleted(
      LPPERSISTSTORAGE        lpThis,
      LPSTORAGE               lpStgNew
);
STDMETHODIMP SvrDoc_PStg_HandsOffStorage(LPPERSISTSTORAGE lpThis);


#if defined( SVR_TREATAS )

/* ServerDoc::IStdMarshalInfo methods (functions) */
STDMETHODIMP SvrDoc_StdMshl_QueryInterface(
      LPSTDMARSHALINFO        lpThis,
      REFIID                  riid,
      LPVOID FAR*             lplpvObj
);
STDMETHODIMP_(ULONG) SvrDoc_StdMshl_AddRef(LPSTDMARSHALINFO lpThis);
STDMETHODIMP_(ULONG) SvrDoc_StdMshl_Release(LPSTDMARSHALINFO lpThis);
STDMETHODIMP SvrDoc_StdMshl_GetClassForHandler(
      LPSTDMARSHALINFO        lpThis,
      DWORD                   dwDestContext,
      LPVOID                  pvDestContext,
      LPCLSID                 lpClassID
);
#endif  // SVR_TREATAS

/*************************************************************************
** class SERVERAPP : OLEAPP
**    SERVERAPP is an extention to the abstract base OLEAPP class.
**    The OLEAPP class defines the fields, methods and interfaces that
**    are common to both server and client implementations. The
**    SERVERAPP class adds the fields and methods that are specific to
**    OLE 2.0 Server functionality. There is one instance of
**    SERVERAPP object created per running application instance. This
**    object holds many fields that could otherwise be organized as
**    global variables. The SERVERAPP class inherits all fields
**    from the OLEAPP class. This inheritance is achieved by including a
**    member variable of type OLEAPP as the first field in the SERVERAPP
**    structure. OLEAPP inherits from OLEAPP. This inheritance is
**    achieved in the same manner. Thus a pointer to a SERVERAPP object
**    can be cast to be a pointer to an OLEAPP or an OUTLINEAPP object
*************************************************************************/

typedef struct tagSERVERAPP {
   OLEAPP      m_OleApp;       // ServerApp inherits all fields of OleApp

#if defined( INPLACE_SVR )
   HACCEL  m_hAccelIPSvr; // accelerators for server's active object
commands
   HACCEL  m_hAccelBaseApp;    // normal accel for non-inplace server mode
   HMENU   m_hMenuEdit;   // handle to Edit menu of the server app
   HMENU   m_hMenuLine;   // handle to Line menu of the server app
   HMENU   m_hMenuName;   // handle to Name menu of the server app
   HMENU   m_hMenuOptions; // handle to Options menu of the server app
   HMENU   m_hMenuDebug;       // handle to Debug menu of the server app
   HMENU   m_hMenuHelp;   // handle to Help menu of the server app
   LPINPLACEDATA   m_lpIPData;
#endif

} SERVERAPP;

/* ServerApp methods (functions) */
BOOL ServerApp_InitInstance(
      LPSERVERAPP            lpServerApp,
      HINSTANCE              hInst,
      int                    nCmdShow
);
BOOL ServerApp_InitVtbls (LPSERVERAPP lpServerApp);



/*************************************************************************
** class SERVERNAME : OUTLINENAME
**    SERVERNAME class is an extension to the OUTLINENAME base class that
**    adds functionallity required to support linking to ranges (pseudo
**    objects). Pseudo objects are used to allow linking to a range
**    (sub-selection) of a SERVERDOC document. The base class OUTLINENAME
**    stores a particular named selection in the document. The
**    NAMETABLE class holds all of the names defined in a particular
**    document. Each OUTLINENAME object has a string as its key and a
**    starting line index and an ending line index for the named range.
**    The SERVERNAME class, also, stores a pointer to a PSEUDOOBJ if one
**    has been allocated that corresponds to the named selection.
**    The SERVERNAME class inherits all fields from the OUTLINENAME class.
**    This inheritance is achieved by including a member variable of
**    type OUTLINENAME as the first field in the SERVERNAME
**    structure. Thus a pointer to an SERVERNAME object can be cast to be
**    a pointer to a OUTLINENAME object.
*************************************************************************/

typedef struct tagSERVERNAME {
   OUTLINENAME     m_Name;         // ServerName inherits all fields of Name
   LPPSEUDOOBJ m_lpPseudoObj;  // ptr to pseudo object if allocated
} SERVERNAME, FAR* LPSERVERNAME;

/* ServerName methods (functions) */
void ServerName_SetSel(
      LPSERVERNAME           lpServerName,
      LPLINERANGE            lplrSel,
      BOOL                   fRangeModified
);
void ServerName_SendPendingAdvises(LPSERVERNAME lpServerName);
LPPSEUDOOBJ ServerName_GetPseudoObj(
      LPSERVERNAME           lpServerName,
      LPSERVERDOC            lpServerDoc
);
void ServerName_ClosePseudoObj(LPSERVERNAME lpServerName);
```

```
/*************************************************************************
** class PSEUDOOBJ
**    The PSEUDOOBJ (pseudo object) is a concrete class. A pseudo object
**    is created when a link is made to a range of lines within an
**    SERVERDOC document. A pseudo object is dependent on the existance
**    of the SERVERDOC which represents the whole document.
*************************************************************************/

typedef struct tagPSEUDOOBJ {
   ULONG               m_cRef;             // total ref count for obj
   BOOL                m_fObjIsClosing;    // flag to guard recursive close
   LPSERVERNAME        m_lpName;           // named range for this pseudo
obj
   LPSERVERDOC         m_lpDoc;            // ptr to whole document
   LPOLEADVISEHOLDER   m_lpOleAdviseHldr;  // helper obj to hold ole advises
   LPDATAADVISEHOLDER  m_lpDataAdviseHldr; // helper obj to hold data
advises
   BOOL                m_fDataChanged;     // data changed when draw
disabled

   struct CPseudoObjUnknownImpl {
      IUnknownVtbl FAR*       lpVtbl;
      LPPSEUDOOBJ             lpPseudoObj;
      int                     cRef;   // interface specific ref count.
   } m_Unknown;

   struct CPseudoObjOleObjectImpl {
      IOleObjectVtbl FAR*     lpVtbl;
      LPPSEUDOOBJ             lpPseudoObj;
      int                     cRef;   // interface specific ref count.
   } m_OleObject;

   struct CPseudoObjDataObjectImpl {
      IDataObjectVtbl FAR*    lpVtbl;
      LPPSEUDOOBJ             lpPseudoObj;
      int                     cRef;   // interface specific ref count.
   } m_DataObject;

} PSEUDOOBJ;

/* PseudoObj methods (functions) */
void PseudoObj_Init(
      LPPSEUDOOBJ             lpPseudoObj,
      LPSERVERNAME            lpServerName,
      LPSERVERDOC             lpServerDoc
);
ULONG PseudoObj_AddRef(LPPSEUDOOBJ lpPseudoObj);
ULONG PseudoObj_Release(LPPSEUDOOBJ lpPseudoObj);
HRESULT PseudoObj_QueryInterface(
      LPPSEUDOOBJ         lpPseudoObj,
      REFIID              riid,
      LPVOID FAR*         lplpUnk
);
BOOL PseudoObj_Close(LPPSEUDOOBJ lpPseudoObj);
void PseudoObj_Destroy(LPPSEUDOOBJ lpPseudoObj);
void PseudoObj_GetSel(LPPSEUDOOBJ lpPseudoObj, LPLINERANGE lplrSel);
void PseudoObj_GetExtent(LPPSEUDOOBJ lpPseudoObj, LPSIZEL lpsizel);
void PseudoObj_GetExtent(LPPSEUDOOBJ lpPseudoObj, LPSIZEL lpsizel);
void PseudoObj_SendAdvise(
      LPPSEUDOOBJ lpPseudoObj,
      WORD        wAdvise,
      LPMONIKER   lpmkObj,
      DWORD       dwAdvf
);
LPMONIKER PseudoObj_GetFullMoniker(LPPSEUDOOBJ lpPseudoObj, LPMONIKER
lpmkDoc);

/* PseudoObj::IUnknown methods (functions) */
STDMETHODIMP PseudoObj_Unk_QueryInterface(
      LPUNKNOWN         lpThis,
      REFIID            riid,
      LPVOID FAR*       lplpvObj
);
STDMETHODIMP_(ULONG) PseudoObj_Unk_AddRef(LPUNKNOWN lpThis);
STDMETHODIMP_(ULONG) PseudoObj_Unk_Release (LPUNKNOWN lpThis);

/* PseudoObj::IOleObject methods (functions) */
STDMETHODIMP PseudoObj_OleObj_QueryInterface(
      LPOLEOBJECT     lpThis,
      REFIID          riid,
      LPVOID FAR*     lplpvObj
);
STDMETHODIMP_(ULONG) PseudoObj_OleObj_AddRef(LPOLEOBJECT lpThis);
STDMETHODIMP_(ULONG) PseudoObj_OleObj_Release(LPOLEOBJECT lpThis);
STDMETHODIMP PseudoObj_OleObj_SetClientSite(
      LPOLEOBJECT         lpThis,
      LPOLECLIENTSITE     lpClientSite
);
STDMETHODIMP PseudoObj_OleObj_GetClientSite(
      LPOLEOBJECT             lpThis,
      LPOLECLIENTSITE FAR*    lplpClientSite
);
STDMETHODIMP PseudoObj_OleObj_SetHostNames(
      LPOLEOBJECT             lpThis,
      LPCOLESTR               szContainerApp,
      LPCOLESTR               szContainerObj
);
STDMETHODIMP PseudoObj_OleObj_Close(
      LPOLEOBJECT             lpThis,
      DWORD                   dwSaveOption
);
STDMETHODIMP PseudoObj_OleObj_SetMoniker(
      LPOLEOBJECT lpThis,
      DWORD       dwWhichMoniker,
      LPMONIKER   lpmk
);
STDMETHODIMP PseudoObj_OleObj_GetMoniker(
      LPOLEOBJECT     lpThis,
      DWORD           dwAssign,
      DWORD           dwWhichMoniker,
      LPMONIKER FAR*  lplpmk
);
STDMETHODIMP PseudoObj_OleObj_InitFromData(
      LPOLEOBJECT             lpThis,
      LPDATAOBJECT            lpDataObject,
      BOOL                    fCreation,
      DWORD                   reserved
);
STDMETHODIMP PseudoObj_OleObj_GetClipboardData(
      LPOLEOBJECT             lpThis,
      DWORD                   reserved,
      LPDATAOBJECT FAR*       lplpDataObject
);
STDMETHODIMP PseudoObj_OleObj_DoVerb(
      LPOLEOBJECT             lpThis,
      LONG                    lVerb,
      LPMSG                   lpmsg,
      LPOLECLIENTSITE         lpActiveSite,
      LONG                    lindex,
      HWND                    hwndParent,
      LPCRECT                 lprcPosRect
);
STDMETHODIMP PseudoObj_OleObj_EnumVerbs(
      LPOLEOBJECT         lpThis,
      LPENUMOLEVERB FAR*  lplpenumOleVerb
);
STDMETHODIMP PseudoObj_OleObj_Update(LPOLEOBJECT lpThis);
STDMETHODIMP PseudoObj_OleObj_IsUpToDate(LPOLEOBJECT lpThis);
STDMETHODIMP PseudoObj_OleObj_GetUserClassID(
      LPOLEOBJECT             lpThis,
      LPCLSID                 lpclsid
);
STDMETHODIMP PseudoObj_OleObj_GetUserType(
      LPOLEOBJECT             lpThis,
      DWORD                   dwFormOfType,
      LPOLESTR FAR*           lpszUserType
);
STDMETHODIMP PseudoObj_OleObj_SetExtent(
      LPOLEOBJECT             lpThis,
      DWORD                   dwDrawAspect,
      LPSIZEL                 lplgrc
);
STDMETHODIMP PseudoObj_OleObj_GetExtent(
      LPOLEOBJECT             lpThis,
      DWORD                   dwDrawAspect,
      LPSIZEL                 lplgrc
);
STDMETHODIMP PseudoObj_OleObj_Advise(
      LPOLEOBJECT lpThis,
      LPADVISESINK lpAdvSink,
      LPDWORD lpdwConnection
);
STDMETHODIMP PseudoObj_OleObj_Unadvise(LPOLEOBJECT lpThis,DWORD
dwConnection);
STDMETHODIMP PseudoObj_OleObj_EnumAdvise(
      LPOLEOBJECT lpThis,
      LPENUMSTATDATA FAR* lplpenumAdvise
);
STDMETHODIMP PseudoObj_OleObj_GetMiscStatus(
      LPOLEOBJECT             lpThis,
      DWORD                   dwAspect,
      DWORD FAR*              lpdwStatus
);
STDMETHODIMP PseudoObj_OleObj_SetColorScheme(
      LPOLEOBJECT             lpThis,
      LPLOGPALETTE            lpLogpal
);
STDMETHODIMP PseudoObj_OleObj_LockObject(
      LPOLEOBJECT             lpThis,
      BOOL                    fLock
);

/* PseudoObj::IDataObject methods (functions) */
STDMETHODIMP PseudoObj_DataObj_QueryInterface (
      LPDATAOBJECT      lpThis,
      REFIID            riid,
      LPVOID FAR*       lplpvObj
);
STDMETHODIMP_(ULONG) PseudoObj_DataObj_AddRef(LPDATAOBJECT lpThis);
STDMETHODIMP_(ULONG) PseudoObj_DataObj_Release (LPDATAOBJECT lpThis);
STDMETHODIMP PseudoObj_DataObj_GetData (
      LPDATAOBJECT    lpThis,
      LPFORMATETC     lpformatetc,
      LPSTGMEDIUM     lpMedium
);
STDMETHODIMP PseudoObj_DataObj_GetDataHere (
      LPDATAOBJECT    lpThis,
      LPFORMATETC     lpformatetc,
      LPSTGMEDIUM     lpMedium
);
STDMETHODIMP PseudoObj_DataObj_QueryGetData (
      LPDATAOBJECT    lpThis,
      LPFORMATETC     lpformatetc
);
STDMETHODIMP PseudoObj_DataObj_GetCanonicalFormatEtc (
      LPDATAOBJECT    lpThis,
      LPFORMATETC     lpformatetc,
      LPFORMATETC     lpformatetcOut
);
STDMETHODIMP PseudoObj_DataObj_SetData (
      LPDATAOBJECT    lpThis,
      LPFORMATETC     lpformatetc,
      LPSTGMEDIUM     lpmedium,
      BOOL            fRelease
);
STDMETHODIMP PseudoObj_DataObj_EnumFormatEtc(
      LPDATAOBJECT            lpThis,
      DWORD                   dwDirection,
      LPENUMFORMATETC FAR*    lplpenumFormatEtc
);
```

```
STDMETHODIMP PseudoObj_DataObj_DAdvise(
      LPDATAOBJECT    lpThis,
      FORMATETC FAR*  lpFormatetc,
      DWORD           advf,
      LPADVISESINK    lpAdvSink,
      DWORD FAR*      lpdwConnection
);
STDMETHODIMP PseudoObj_DataObj_DUnadvise(LPDATAOBJECT lpThis, DWORD
dwConnection);
STDMETHODIMP PseudoObj_DataObj_EnumAdvise(
      LPDATAOBJECT lpThis,
      LPENUMSTATDATA FAR* lplpenumAdvise
);


/*************************************************************************
** class SERVERNAMETABLE : OUTLINENAMETABLE
**    SERVERNAMETABLE class is an extension to the OUTLINENAMETABLE
**    base class that adds functionallity required to support linking
**    to ranges (pseudo objects). The name table manages the table of
**    named selections in the document. Each name table entry has a
**    string as its key and a starting line index and an ending line
**    index for the named range. The SERVERNAMETABLE entries, in
**    addition, maintain a pointer to a PSEUDOOBJ pseudo object if one
**    has been already allocated. There is always one instance of
**    SERVERNAMETABLE for each SERVERDOC object created.
**    The SERVERNAME class inherits all fields from the NAME class.
**    This inheritance is achieved by including a member variable of
**    type NAME as the first field in the SERVERNAME
**    structure. Thus a pointer to an SERVERNAME object can be cast to be
**    a pointer to a NAME object.
*************************************************************************/

typedef struct tagSERVERNAMETABLE {
   OUTLINENAMETABLE    m_NameTable;    // we inherit from OUTLINENAMETABLE

   // ServerNameTable does NOT add any fields

} SERVERNAMETABLE, FAR* LPSERVERNAMETABLE;

/* ServerNameTable methods (functions) */
void ServerNameTable_EditLineUpdate(
      LPSERVERNAMETABLE       lpServerNameTable,
      int                     nEditIndex
);
void ServerNameTable_InformAllPseudoObjectsDocRenamed(
      LPSERVERNAMETABLE       lpServerNameTable,
      LPMONIKER               lpmkDoc
);
void ServerNameTable_InformAllPseudoObjectsDocSaved(
      LPSERVERNAMETABLE       lpServerNameTable,
      LPMONIKER               lpmkDoc
);
void ServerNameTable_SendPendingAdvises(LPSERVERNAMETABLE
lpServerNameTable);
```

```
LPPSEUDOOBJ ServerNameTable_GetPseudoObj(
      LPSERVERNAMETABLE       lpServerNameTable,
      LPOLESTR                lpszItem,
      LPSERVERDOC             lpServerDoc
);
void ServerNameTable_CloseAllPseudoObjs(LPSERVERNAMETABLE
lpServerNameTable);


#if defined( INPLACE_SVR)

/* ServerDoc::IOleInPlaceObject methods (functions) */

STDMETHODIMP SvrDoc_IPObj_QueryInterface(
      LPOLEINPLACEOBJECT  lpThis,
      REFIID              riid,
      LPVOID FAR *        lplpvObj
);
STDMETHODIMP_(ULONG) SvrDoc_IPObj_AddRef(LPOLEINPLACEOBJECT lpThis);
STDMETHODIMP_(ULONG) SvrDoc_IPObj_Release(LPOLEINPLACEOBJECT lpThis);
STDMETHODIMP SvrDoc_IPObj_GetWindow(
      LPOLEINPLACEOBJECT  lpThis,
      HWND FAR*           lphwnd
);
STDMETHODIMP SvrDoc_IPObj_ContextSensitiveHelp(
      LPOLEINPLACEOBJECT  lpThis,
      BOOL                fEnable
);
STDMETHODIMP SvrDoc_IPObj_InPlaceDeactivate(LPOLEINPLACEOBJECT lpThis);
STDMETHODIMP SvrDoc_IPObj_UIDeactivate(LPOLEINPLACEOBJECT lpThis);
STDMETHODIMP SvrDoc_IPObj_SetObjectRects(
      LPOLEINPLACEOBJECT  lpThis,
      LPCRECT             lprcPosRect,
      LPCRECT             lprcClipRect
);
STDMETHODIMP SvrDoc_IPObj_ReactivateAndUndo(LPOLEINPLACEOBJECT lpThis);

/* ServerDoc::IOleInPlaceActiveObject methods (functions) */

STDMETHODIMP SvrDoc_IPActiveObj_QueryInterface(
      LPOLEINPLACEACTIVEOBJECT    lpThis,
      REFIID                      riidReq,
      LPVOID FAR *                lplpUnk
);
STDMETHODIMP_(ULONG) SvrDoc_IPActiveObj_AddRef(
      LPOLEINPLACEACTIVEOBJECT lpThis
);
STDMETHODIMP_(ULONG) SvrDoc_IPActiveObj_Release(
      LPOLEINPLACEACTIVEOBJECT lpThis
);
STDMETHODIMP SvrDoc_IPActiveObj_GetWindow(
      LPOLEINPLACEACTIVEOBJECT    lpThis,
      HWND FAR*                   lphwnd
);
STDMETHODIMP SvrDoc_IPActiveObj_ContextSensitiveHelp(
      LPOLEINPLACEACTIVEOBJECT    lpThis,
      BOOL                        fEnable
);
STDMETHODIMP SvrDoc_IPActiveObj_TranslateAccelerator(
      LPOLEINPLACEACTIVEOBJECT    lpThis,
      LPMSG                       lpmsg
);
STDMETHODIMP SvrDoc_IPActiveObj_OnFrameWindowActivate(
      LPOLEINPLACEACTIVEOBJECT    lpThis,
      BOOL                        fActivate
);
STDMETHODIMP SvrDoc_IPActiveObj_OnDocWindowActivate(
      LPOLEINPLACEACTIVEOBJECT    lpThis,
      BOOL                        fActivate
);
STDMETHODIMP SvrDoc_IPActiveObj_ResizeBorder(
      LPOLEINPLACEACTIVEOBJECT    lpThis,
      LPCRECT                     lprectBorder,
      LPOLEINPLACEUIWINDOW        lpIPUiWnd,
      BOOL                        fFrameWindow
);
STDMETHODIMP SvrDoc_IPActiveObj_EnableModeless(
      LPOLEINPLACEACTIVEOBJECT    lpThis,
      BOOL                        fEnable
);

#endif // INPLACE_SVR

#endif // _SVROUTL_H_
```

SVROUTL.RC   (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2.0 Sample Code
**
**    precomp.c
**
**    This file is used to precompile the OUTLINE.H header file
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/

#include "outline.h"
```

SVRPSOBJ.C (OUTLINE Sample)

```
/*************************************************************************
**
**    OLE 2 Server Sample Code
**
**    svrpsobj.c
**
**    This file contains all PseudoObj methods and related support
**    functions.
**
**    (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
**
*************************************************************************/


#include "outline.h"

OLEDBGDATA

extern LPOUTLINEAPP             g_lpApp;
extern IUnknownVtbl             g_PseudoObj_UnknownVtbl;
extern IOleObjectVtbl           g_PseudoObj_OleObjectVtbl;
extern IDataObjectVtbl          g_PseudoObj_DataObjectVtbl;


/* PseudoObj_Init
** --------------
**  Initialize fields in a newly constructed PseudoObj.
**  NOTE: ref cnt of PseudoObj initialized to 0
*/
void PseudoObj_Init(
      LPPSEUDOOBJ             lpPseudoObj,
      LPSERVERNAME            lpServerName,
      LPSERVERDOC             lpServerDoc
)
{
   OleDbgOut2("++PseudoObj Created\r\n");

   lpPseudoObj->m_cRef             = 0;
   lpPseudoObj->m_lpName           = lpServerName;
   lpPseudoObj->m_lpDoc            = lpServerDoc;
   lpPseudoObj->m_lpOleAdviseHldr  = NULL;
   lpPseudoObj->m_lpDataAdviseHldr = NULL;
   lpPseudoObj->m_fObjIsClosing    = FALSE;

   INIT_INTERFACEIMPL(
         &lpPseudoObj->m_Unknown,
         &g_PseudoObj_UnknownVtbl,
         lpPseudoObj
   );

   INIT_INTERFACEIMPL(
         &lpPseudoObj->m_OleObject,
         &g_PseudoObj_OleObjectVtbl,
         lpPseudoObj
   );

   INIT_INTERFACEIMPL(
         &lpPseudoObj->m_DataObject,
         &g_PseudoObj_DataObjectVtbl,
         lpPseudoObj
   );

   /* OLE2NOTE: Increment the refcnt of the Doc on behalf of the
   **    PseudoObj. the Document should not shut down unless all
   **    pseudo objects are closed. when a pseudo object is destroyed,
   **    it calls ServerDoc_PseudoObjUnlockDoc to release this hold on
   **    the document.
   */
   ServerDoc_PseudoObjLockDoc(lpServerDoc);
}



/* PseudoObj_AddRef
** ----------------
**
**  increment the ref count of the PseudoObj object.
**
**    Returns the new ref count on the object
*/
ULONG PseudoObj_AddRef(LPPSEUDOOBJ lpPseudoObj)
{
   ++lpPseudoObj->m_cRef;

#if defined( _DEBUG )
   OleDbgOutRefCnt4(
         "PseudoObj_AddRef: cRef++\r\n",
         lpPseudoObj,
         lpPseudoObj->m_cRef
   );
#endif
   return lpPseudoObj->m_cRef;
}


/* PseudoObj_Release
** -----------------
**
**  decrement the ref count of the PseudoObj object.
**    if the ref count goes to 0, then the PseudoObj is destroyed.
**
**    Returns the remaining ref count on the object
*/
ULONG PseudoObj_Release(LPPSEUDOOBJ lpPseudoObj)
{
   ULONG cRef;
   /*****************************************************************
   ** OLE2NOTE: when the obj refcnt == 0, then destroy the object.     **
   **     otherwise the object is still in use.                        **
   *****************************************************************/

   cRef = --lpPseudoObj->m_cRef;

#if defined( _DEBUG )
   OleDbgAssertSz(lpPseudoObj->m_cRef >= 0,"Release called with cRef == 0");

   OleDbgOutRefCnt4(
         "PseudoObj_Release: cRef--\r\n", lpPseudoObj,cRef);
#endif

   if (cRef == 0)
      PseudoObj_Destroy(lpPseudoObj);

   return cRef;
}


/* PseudoObj_QueryInterface
** ------------------------
**
** Retrieve a pointer to an interface on the PseudoObj object.
**
**    Returns S_OK if interface is successfully retrieved.
**            E_NOINTERFACE if the interface is not supported
*/
HRESULT PseudoObj_QueryInterface(
      LPPSEUDOOBJ         lpPseudoObj,
      REFIID              riid,
      LPVOID FAR*         lplpvObj
)
{
   SCODE sc = E_NOINTERFACE;

   /* OLE2NOTE: we must make sure to set all out ptr parameters to NULL. */
   *lplpvObj = NULL;

   if (IsEqualIID(riid, &IID_IUnknown)) {
      OleDbgOut4("PseudoObj_QueryInterface: IUnknown* RETURNED\r\n");

      *lplpvObj = (LPVOID) &lpPseudoObj->m_Unknown;
      PseudoObj_AddRef(lpPseudoObj);
      sc = S_OK;
   }
   else if (IsEqualIID(riid, &IID_IOleObject)) {
      OleDbgOut4("PseudoObj_QueryInterface: IOleObject* RETURNED\r\n");

      *lplpvObj = (LPVOID) &lpPseudoObj->m_OleObject;
      PseudoObj_AddRef(lpPseudoObj);
      sc = S_OK;
   }
   else if (IsEqualIID(riid, &IID_IDataObject)) {
      OleDbgOut4("PseudoObj_QueryInterface: IDataObject* RETURNED\r\n");

      *lplpvObj = (LPVOID) &lpPseudoObj->m_DataObject;
      PseudoObj_AddRef(lpPseudoObj);
      sc = S_OK;
   }

   OleDbgQueryInterfaceMethod(*lplpvObj);

   return ResultFromScode(sc);
}


/* PseudoObj_Close
 * ---------------
 *
 *  Close the pseudo object. Force all external connections to close
 *      down. This causes link clients to release this PseudoObj. when
 *      the refcount actually reaches 0, then the PseudoObj will be
 *      destroyed.
 *
 *  Returns:
 *      FALSE -- user canceled the closing of the doc.
 *      TRUE -- the doc was successfully closed
 */

BOOL PseudoObj_Close(LPPSEUDOOBJ lpPseudoObj)
{
   LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpPseudoObj->m_lpDoc;
   LPSERVERNAME lpServerName = (LPSERVERNAME)lpPseudoObj->m_lpName;
   LPOLEAPP lpOleApp = (LPOLEAPP)g_lpApp;
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpServerDoc;
   BOOL fStatus = TRUE;

   if (lpPseudoObj->m_fObjIsClosing)
      return TRUE;    // Closing is already in progress

   lpPseudoObj->m_fObjIsClosing = TRUE;   // guard against recursive call

   OLEDBG_BEGIN3("PseudoObj_Close\r\n")

   /* OLE2NOTE: in order to have a stable App, Doc, AND pseudo object
   **    during the process of closing, we intially AddRef the App,
   **    Doc, and PseudoObj ref counts and later Release them. These
   **    initial AddRefs are artificial; they are simply done to
   **    guarantee that these objects do not get destroyed until the
   **    end of this routine.
   */
   OleApp_AddRef(lpOleApp);
   OleDoc_AddRef(lpOleDoc);
   PseudoObj_AddRef(lpPseudoObj);

   if (lpPseudoObj->m_lpDataAdviseHldr) {
      /* OLE2NOTE: send last OnDataChange notification to clients
      **    that have registered for data notifications when object
      **    stops running (ADVF_DATAONSTOP)
      */
      PseudoObj_SendAdvise(
            lpPseudoObj,
            OLE_ONDATACHANGE,
            NULL,   /* lpmkObj -- not relevant here */
            ADVF_DATAONSTOP
      );

      /* OLE2NOTE: we just sent the last data notification that we
      **    need to send; release our DataAdviseHolder. we SHOULD be
      **    the only one using it.
      */
      OleStdVerifyRelease(
            (LPUNKNOWN)lpPseudoObj->m_lpDataAdviseHldr,
            OLESTR("DataAdviseHldr not released properly")
      );
      lpPseudoObj->m_lpDataAdviseHldr = NULL;
   }

   if (lpPseudoObj->m_lpOleAdviseHldr) {
      // OLE2NOTE: inform all of our linking clients that we are closing.
      PseudoObj_SendAdvise(
            lpPseudoObj,
            OLE_ONCLOSE,
            NULL,   /* lpmkObj -- not relevant here */
            0       /* advf -- not relevant here */
      );

      /* OLE2NOTE: OnClose is the last notification that we need to
      **    send; release our OleAdviseHolder. we SHOULD be the only
      **    one using it. this will make our destructor realize that
      **    OnClose notification has already been sent.
      */
      OleStdVerifyRelease(
            (LPUNKNOWN)lpPseudoObj->m_lpOleAdviseHldr,
            OLESTR("OleAdviseHldr not released properly")
      );
      lpPseudoObj->m_lpOleAdviseHldr = NULL;
   }

   /* OLE2NOTE: this call forces all external connections to our
   **    object to close down and therefore guarantees that we receive
   **    all releases associated with those external connections.
   */
   OLEDBG_BEGIN2("CoDisconnectObject called\r\n")
   CoDisconnectObject((LPUNKNOWN)&lpPseudoObj->m_Unknown, 0);
   OLEDBG_END2

   PseudoObj_Release(lpPseudoObj);     // release artificial AddRef above
   OleDoc_Release(lpOleDoc);           // release artificial AddRef above
   OleApp_Release(lpOleApp);           // release artificial AddRef above
```

```
   OLEDBG_END3
   return fStatus;
}


/* PseudoObj_Destroy
** -----------------
**    Destroy (Free) the memory used by a PseudoObj structure.
**    This function is called when the ref count of the PseudoObj goes
**    to zero. the ref cnt goes to zero after PseudoObj_Delete forces
**    the OleObject to unload and release its pointers to the
**    PseudoObj IOleClientSite and IAdviseSink interfaces.
*/

void PseudoObj_Destroy(LPPSEUDOOBJ lpPseudoObj)
{
   LPSERVERDOC lpServerDoc = lpPseudoObj->m_lpDoc;
   LPOLEAPP    lpOleApp = (LPOLEAPP)g_lpApp;
   LPOLEDOC    lpOleDoc = (LPOLEDOC)lpServerDoc;

   OLEDBG_BEGIN3("PseudoObj_Destroy\r\n")

   /* OLE2NOTE: in order to have a stable App, Doc, AND pseudo object
   **    during the process of closing, we intially AddRef the App,
   **    Doc ref counts and later Release them. These
   **    initial AddRefs are artificial; they are simply done to
   **    guarantee that these objects do not get destroyed until the
   **    end of this routine.
   */
   OleApp_AddRef(lpOleApp);
   OleDoc_AddRef(lpOleDoc);

   /*****************************************************************
   ** OLE2NOTE: we no longer need the advise and enum holder objects,
   **    so release them.
   *****************************************************************/

   if (lpPseudoObj->m_lpDataAdviseHldr) {
      /* release DataAdviseHldr; we SHOULD be the only one using it. */
      OleStdVerifyRelease(
            (LPUNKNOWN)lpPseudoObj->m_lpDataAdviseHldr,
            OLESTR("DataAdviseHldr not released properly")
         );
      lpPseudoObj->m_lpDataAdviseHldr = NULL;
   }

   if (lpPseudoObj->m_lpOleAdviseHldr) {
      /* release OleAdviseHldr; we SHOULD be the only one using it. */
      OleStdVerifyRelease(
            (LPUNKNOWN)lpPseudoObj->m_lpOleAdviseHldr,
            OLESTR("OleAdviseHldr not released properly")
         );
      lpPseudoObj->m_lpOleAdviseHldr = NULL;
   }
   /* forget the pointer to destroyed PseudoObj in NameTable */
   if (lpPseudoObj->m_lpName)
      lpPseudoObj->m_lpName->m_lpPseudoObj = NULL;

   /* OLE2NOTE: release the lock on the Doc held on behalf of the
   **    PseudoObj. the Document should not shut down unless all
   **    pseudo objects are closed. when a pseudo object is first
   **    created, it calls ServerDoc_PseudoObjLockDoc to guarantee
   **    that the document stays alive (called from PseudoObj_Init).
   */
   ServerDoc_PseudoObjUnlockDoc(lpServerDoc, lpPseudoObj);

   Delete(lpPseudoObj);        // Free the memory for the structure itself

   OleDoc_Release(lpOleDoc);       // release artificial AddRef above
   OleApp_Release(lpOleApp);       // release artificial AddRef above

   OLEDBG_END3
}


/* PseudoObj_GetSel
** ----------------
**    Return the line range for the pseudo object
*/
void PseudoObj_GetSel(LPPSEUDOOBJ lpPseudoObj, LPLINERANGE lplrSel)
{
   LPOUTLINENAME lpOutlineName = (LPOUTLINENAME)lpPseudoObj->m_lpName;
   lplrSel->m_nStartLine = lpOutlineName->m_nStartLine;
   lplrSel->m_nEndLine = lpOutlineName->m_nEndLine;
}


/* PseudoObj_GetExtent
 * -------------------
 *
 *      Get the extent (width, height) of the entire document.
 */
void PseudoObj_GetExtent(LPPSEUDOOBJ lpPseudoObj, LPSIZEL lpsizel)
{
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpPseudoObj->m_lpDoc;
   LPLINELIST lpLL = (LPLINELIST)&((LPOUTLINEDOC)lpOleDoc)->m_LineList;
   LINERANGE lrSel;

   PseudoObj_GetSel(lpPseudoObj, (LPLINERANGE)&lrSel);

   LineList_CalcSelExtentInHimetric(lpLL, (LPLINERANGE)&lrSel, lpsizel);
}


/* PseudoObj_SendAdvise
 * --------------------
 *
 * This function sends an advise notification on behalf of a specific
 *  doc object to all its clients.
 */
void PseudoObj_SendAdvise(
      LPPSEUDOOBJ lpPseudoObj,
      WORD        wAdvise,
      LPMONIKER   lpmkObj,
      DWORD       dwAdvf
)
{
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpPseudoObj->m_lpDoc;

   // artificial AddRef in case object is destroyed during call
   PseudoObj_AddRef(lpPseudoObj);

   switch (wAdvise) {

      case OLE_ONDATACHANGE:

         // inform clients that the data of the object has changed

         if (lpOutlineDoc->m_nDisableDraw == 0) {
            /* drawing is currently enabled. inform clients that
            **    the data of the object has changed
            */

            lpPseudoObj->m_fDataChanged = FALSE;
            if (lpPseudoObj->m_lpDataAdviseHldr) {

               OLEDBG_BEGIN2("IDataAdviseHolder::SendOnDataChange
called\r\n");
               lpPseudoObj->m_lpDataAdviseHldr->lpVtbl->SendOnDataChange(
                     lpPseudoObj->m_lpDataAdviseHldr,
                     (LPDATAOBJECT)&lpPseudoObj->m_DataObject,
                     0,
                     dwAdvf
               );
               OLEDBG_END2
            }

         } else {
            /* drawing is currently disabled. do not send
            **    notifications until drawing is re-enabled.
            */
            lpPseudoObj->m_fDataChanged = TRUE;
         }
         break;

      case OLE_ONCLOSE:

         // inform clients that the object is shutting down

         if (lpPseudoObj->m_lpOleAdviseHldr) {

            OLEDBG_BEGIN2("IOleAdviseHolder::SendOnClose called\r\n");
            lpPseudoObj->m_lpOleAdviseHldr->lpVtbl->SendOnClose(
                  lpPseudoObj->m_lpOleAdviseHldr
            );
            OLEDBG_END2
         }
         break;

      case OLE_ONSAVE:

         // inform clients that the object has been saved

         if (lpPseudoObj->m_lpOleAdviseHldr) {

            OLEDBG_BEGIN2("IOleAdviseHolder::SendOnClose called\r\n");
            lpPseudoObj->m_lpOleAdviseHldr->lpVtbl->SendOnSave(
                  lpPseudoObj->m_lpOleAdviseHldr
            );
            OLEDBG_END2
         }
         break;

      case OLE_ONRENAME:

         // inform clients that the object's name has changed
         if (lpmkObj && lpPseudoObj->m_lpOleAdviseHldr) {

            OLEDBG_BEGIN2("IOleAdviseHolder::SendOnRename called\r\n");
            if (lpPseudoObj->m_lpOleAdviseHldr)
               lpPseudoObj->m_lpOleAdviseHldr->lpVtbl->SendOnRename(
                     lpPseudoObj->m_lpOleAdviseHldr,
                     lpmkObj
               );
            OLEDBG_END2
         }
         break;
   }
   // release artificial AddRef
   PseudoObj_Release(lpPseudoObj);
}


/* PseudoObj_GetFullMoniker
 * ------------------------
 *
 * Returns the Full, absolute Moniker which identifies this pseudo object.
 */
LPMONIKER PseudoObj_GetFullMoniker(LPPSEUDOOBJ lpPseudoObj, LPMONIKER
lpmkDoc)
{
   LPOUTLINENAME lpOutlineName = (LPOUTLINENAME)lpPseudoObj->m_lpName;
   LPMONIKER lpmkItem = NULL;
   LPMONIKER lpmkPseudoObj = NULL;

   if (lpmkDoc != NULL) {

      CreateItemMoniker(OLESTDDELIM,lpOutlineName->m_szName,&lpmkItem);
      /* OLE2NOTE: create an absolute moniker which identifies the
      **    pseudo object. this moniker is created as a composite of
      **    the absolute moniker for the entire document appended
      **    with an item moniker which identifies the selection of
      **    the pseudo object relative to the document.
      */
      CreateGenericComposite(lpmkDoc, lpmkItem, &lpmkPseudoObj);

      if (lpmkItem)
         OleStdRelease((LPUNKNOWN)lpmkItem);

      return lpmkPseudoObj;
   } else {
      return NULL;
   }
}


/*************************************************************************
** PseudoObj::IUnknown interface implementation
*************************************************************************/

STDMETHODIMP PseudoObj_Unk_QueryInterface(
      LPUNKNOWN         lpThis,
      REFIID            riid,
      LPVOID FAR*       lplpvObj
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjUnknownImpl FAR*)lpThis)->lpPseudoObj;

   return PseudoObj_QueryInterface(lpPseudoObj, riid, lplpvObj);
}


STDMETHODIMP_(ULONG) PseudoObj_Unk_AddRef(LPUNKNOWN lpThis)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjUnknownImpl FAR*)lpThis)->lpPseudoObj;

   OleDbgAddRefMethod(lpThis, "IUnknown");

   return PseudoObj_AddRef(lpPseudoObj);
}


STDMETHODIMP_(ULONG) PseudoObj_Unk_Release (LPUNKNOWN lpThis)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjUnknownImpl FAR*)lpThis)->lpPseudoObj;

   OleDbgReleaseMethod(lpThis, "IUnknown");

   return PseudoObj_Release(lpPseudoObj);
```

```
}


/*************************************************************************
** PseudoObj::IOleObject interface implementation
*************************************************************************/

STDMETHODIMP PseudoObj_OleObj_QueryInterface(
      LPOLEOBJECT     lpThis,
      REFIID          riid,
      LPVOID FAR*     lplpvObj
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjOleObjectImpl FAR*)lpThis)->lpPseudoObj;

   return PseudoObj_QueryInterface(lpPseudoObj, riid, lplpvObj);
}


STDMETHODIMP_(ULONG) PseudoObj_OleObj_AddRef(LPOLEOBJECT lpThis)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjOleObjectImpl FAR*)lpThis)->lpPseudoObj;

   OleDbgAddRefMethod(lpThis, "IOleObject");

   return PseudoObj_AddRef((LPPSEUDOOBJ)lpPseudoObj);
}


STDMETHODIMP_(ULONG) PseudoObj_OleObj_Release(LPOLEOBJECT lpThis)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjOleObjectImpl FAR*)lpThis)->lpPseudoObj;

   OleDbgReleaseMethod(lpThis, "IOleObject");

   return PseudoObj_Release((LPPSEUDOOBJ)lpPseudoObj);
}


STDMETHODIMP PseudoObj_OleObj_SetClientSite(
      LPOLEOBJECT         lpThis,
      LPOLECLIENTSITE     lpClientSite
)
{
   OleDbgOut2("PseudoObj_OleObj_SetClientSite\r\n");

   // OLE2NOTE: a pseudo object does NOT support SetExtent

   return ResultFromScode(E_FAIL);
}
```

```
STDMETHODIMP PseudoObj_OleObj_GetClientSite(
      LPOLEOBJECT             lpThis,
      LPOLECLIENTSITE FAR*    lplpClientSite
)
{
   OleDbgOut2("PseudoObj_OleObj_GetClientSite\r\n");

   *lplpClientSite = NULL;

   // OLE2NOTE: a pseudo object does NOT support SetExtent

   return ResultFromScode(E_FAIL);
}



STDMETHODIMP PseudoObj_OleObj_SetHostNames(
      LPOLEOBJECT             lpThis,
      LPCOLESTR               szContainerApp,
      LPCOLESTR               szContainerObj
)
{
   OleDbgOut2("PseudoObj_OleObj_SetHostNames\r\n");

   // OLE2NOTE: a pseudo object does NOT support SetExtent

   return ResultFromScode(E_FAIL);
}


STDMETHODIMP PseudoObj_OleObj_Close(
      LPOLEOBJECT             lpThis,
      DWORD                   dwSaveOption
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjOleObjectImpl FAR*)lpThis)->lpPseudoObj;
   BOOL fStatus;

   OLEDBG_BEGIN2("PseudoObj_OleObj_Close\r\n")

   /* OLE2NOTE: a pseudo object's implementation of IOleObject::Close
   **    should ignore the dwSaveOption parameter. it is NOT
   **    applicable to pseudo objects.
   */

   fStatus = PseudoObj_Close(lpPseudoObj);
   OleDbgAssertSz(fStatus == TRUE, "PseudoObj_OleObj_Close failed\r\n");

   OLEDBG_END2
   return NOERROR;
}
```

```
STDMETHODIMP PseudoObj_OleObj_SetMoniker(
      LPOLEOBJECT lpThis,
      DWORD       dwWhichMoniker,
      LPMONIKER   lpmk
)
{
   OleDbgOut2("PseudoObj_OleObj_SetMoniker\r\n");

   // OLE2NOTE: a pseudo object does NOT support SetMoniker

   return ResultFromScode(E_FAIL);
}


STDMETHODIMP PseudoObj_OleObj_GetMoniker(
      LPOLEOBJECT     lpThis,
      DWORD           dwAssign,
      DWORD           dwWhichMoniker,
      LPMONIKER FAR*  lplpmk
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjOleObjectImpl FAR*)lpThis)->lpPseudoObj;
   LPOLEDOC lpOleDoc = (LPOLEDOC)lpPseudoObj->m_lpDoc;
   LPMONIKER lpmkDoc;

   // artificial AddRef in case object is destroyed during call
   PseudoObj_OleObj_AddRef(lpThis);

   OLEDBG_BEGIN2("PseudoObj_OleObj_GetMoniker\r\n")

   lpmkDoc = OleDoc_GetFullMoniker(lpOleDoc, GETMONIKER_ONLYIFTHERE);
   *lplpmk = PseudoObj_GetFullMoniker(lpPseudoObj, lpmkDoc);

   OLEDBG_END2

   if (*lplpmk != NULL)
   {
      // release artificial AddRef
      PseudoObj_OleObj_Release(lpThis);
      return NOERROR;
   }
   else
   {
      // release artificial AddRef
      PseudoObj_OleObj_Release(lpThis);
      return ResultFromScode(E_FAIL);
   }
}


STDMETHODIMP PseudoObj_OleObj_InitFromData(
      LPOLEOBJECT             lpThis,
      LPDATAOBJECT            lpDataObject,
      BOOL                    fCreation,
      DWORD                   reserved
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjOleObjectImpl FAR*)lpThis)->lpPseudoObj;
   OleDbgOut2("PseudoObj_OleObj_InitFromData\r\n");

   // REVIEW: NOT YET IMPLEMENTED

   return ResultFromScode(E_NOTIMPL);
}


STDMETHODIMP PseudoObj_OleObj_GetClipboardData(
      LPOLEOBJECT             lpThis,
      DWORD                   reserved,
      LPDATAOBJECT FAR*       lplpDataObject
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjOleObjectImpl FAR*)lpThis)->lpPseudoObj;
   OleDbgOut2("PseudoObj_OleObj_GetClipboardData\r\n");

   // REVIEW: NOT YET IMPLEMENTED

   return ResultFromScode(E_NOTIMPL);
}


STDMETHODIMP PseudoObj_OleObj_DoVerb(
      LPOLEOBJECT             lpThis,
      LONG                    lVerb,
      LPMSG                   lpmsg,
      LPOLECLIENTSITE         lpActiveSite,
      LONG                    lindex,
      HWND                    hwndParent,
      LPCRECT                 lprcPosRect
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjOleObjectImpl FAR*)lpThis)->lpPseudoObj;
   LPOUTLINEDOC lpOutlineDoc = (LPOUTLINEDOC)lpPseudoObj->m_lpDoc;
   LPSERVERDOC lpServerDoc = lpPseudoObj->m_lpDoc;
   LINERANGE lrSel;
   HRESULT hrErr;

   // artificial AddRef in case object is destroyed during call
   PseudoObj_OleObj_AddRef(lpThis);

   OLEDBG_BEGIN2("PseudoObj_OleObj_DoVerb\r\n");

   /* OLE2NOTE: we must first ask our Document to perform the same
   **    verb. then if the verb is NOT OLEIVERB_HIDE we should also
   **    select the range of our pseudo object.
   **    however, we must give our document its own embedding site as
   **    its active site.
   */
   hrErr = SvrDoc_OleObj_DoVerb(
         (LPOLEOBJECT)&lpServerDoc->m_OleObject,
         lVerb,
         lpmsg,
         lpServerDoc->m_lpOleClientSite,
         lindex,
         NULL,   /* we have no hwndParent to give */
         NULL    /* we have no lprcPosRect to give */
   );
   if (FAILED(hrErr)) {
      OLEDBG_END2
      // release artificial AddRef
      PseudoObj_OleObj_Release(lpThis);
      return hrErr;
   }

   if (lVerb != OLEIVERB_HIDE) {
      PseudoObj_GetSel(lpPseudoObj, &lrSel);
      OutlineDoc_SetSel(lpOutlineDoc, &lrSel);
   }

   OLEDBG_END2
   // release artificial AddRef
   PseudoObj_OleObj_Release(lpThis);
   return NOERROR;
}



STDMETHODIMP PseudoObj_OleObj_EnumVerbs(
      LPOLEOBJECT         lpThis,
      LPENUMOLEVERB FAR*  lplpenumOleVerb
)
{
   OleDbgOut2("PseudoObj_OleObj_EnumVerbs\r\n");

   /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
   *lplpenumOleVerb = NULL;

   /* A pseudo object may NOT return OLE_S_USEREG; they must call the
   **    OleReg* API or provide their own implementation. Because this
   **    pseudo object does NOT implement IPersist, simply a low-level
   **    remoting handler (ProxyManager) object as opposed to a
   **    DefHandler object is used as the handler for the pseudo
   **    object in a clients process space. The ProxyManager does NOT
   **    handle the OLE_S_USEREG return values.
   */
   return OleRegEnumVerbs((REFCLSID)&CLSID_APP, lplpenumOleVerb);
}


STDMETHODIMP PseudoObj_OleObj_Update(LPOLEOBJECT lpThis)
```

```
{
   OleDbgOut2("PseudoObj_OleObj_Update\r\n");

   /* OLE2NOTE: a server-only app is always "up-to-date".
   **    a container-app which contains links where the link source
   **    has changed since the last update of the link would be
   **    considered "out-of-date". the "Update" method instructs the
   **    object to get an update from any out-of-date links.
   */

   return NOERROR;
}


STDMETHODIMP PseudoObj_OleObj_IsUpToDate(LPOLEOBJECT lpThis)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjOleObjectImpl FAR*)lpThis)->lpPseudoObj;
   OleDbgOut2("PseudoObj_OleObj_IsUpToDate\r\n");

   /* OLE2NOTE: a server-only app is always "up-to-date".
   **    a container-app which contains links where the link source
   **    has changed since the last update of the link would be
   **    considered "out-of-date".
   */
   return NOERROR;
}


STDMETHODIMP PseudoObj_OleObj_GetUserClassID(
      LPOLEOBJECT             lpThis,
      LPCLSID                 lpclsid
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjOleObjectImpl FAR*)lpThis)->lpPseudoObj;
   LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpPseudoObj->m_lpDoc;
   OleDbgOut2("PseudoObj_OleObj_GetUserClassID\r\n");

   /* OLE2NOTE: we must be carefull to return the correct CLSID here.
   **    if we are currently preforming a "TreatAs (aka. ActivateAs)"
   **    operation then we need to return the class of the object
   **    written in the storage of the object. otherwise we would
   **    return our own class id.
   */
   return ServerDoc_GetClassID(lpServerDoc, lpclsid);
}


STDMETHODIMP PseudoObj_OleObj_GetUserType(
      LPOLEOBJECT             lpThis,
      DWORD                   dwFormOfType,
      LPOLESTR FAR*           lpszUserType
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjOleObjectImpl FAR*)lpThis)->lpPseudoObj;
   LPSERVERDOC lpServerDoc = (LPSERVERDOC)lpPseudoObj->m_lpDoc;
   OleDbgOut2("PseudoObj_OleObj_GetUserType\r\n");

   /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
   *lpszUserType = NULL;

   /* OLE2NOTE: we must be carefull to return the correct user type here.
   **    if we are currently preforming a "TreatAs (aka. ActivateAs)"
   **    operation then we need to return the user type name that
   **    corresponds to the class of the object we are currently
   **    emmulating. otherwise we should return our normal user type
   **    name corresponding to our own class. This routine determines
   **    the current clsid in effect.
   **
   **    A pseudo object may NOT return OLE_S_USEREG; they must call the
   **    OleReg* API or provide their own implementation. Because this
   **    pseudo object does NOT implement IPersist, simply a low-level
   **    remoting handler (ProxyManager) object as opposed to a
   **    DefHandler object is used as the handler for the pseudo
   **    object in a clients process space. The ProxyManager does NOT
   **    handle the OLE_S_USEREG return values.
   */
#if defined( SVR_TREATAS )
   if (! IsEqualCLSID(&lpServerDoc->m_clsidTreatAs, &CLSID_NULL) )
      return OleRegGetUserType(
         (REFCLSID)&lpServerDoc->m_clsidTreatAs,dwFormOfType,lpszUserType);
   else
#endif  // SVR_TREATAS

   return OleRegGetUserType((REFCLSID)&CLSID_APP,dwFormOfType,lpszUserType);
}


STDMETHODIMP PseudoObj_OleObj_SetExtent(
      LPOLEOBJECT             lpThis,
      DWORD                   dwDrawAspect,
      LPSIZEL                 lplgrc
)
{
   OleDbgOut2("PseudoObj_OleObj_SetExtent\r\n");

   // OLE2NOTE: a pseudo object does NOT support SetExtent

   return ResultFromScode(E_FAIL);
}


STDMETHODIMP PseudoObj_OleObj_GetExtent(
      LPOLEOBJECT             lpThis,
      DWORD                   dwDrawAspect,
      LPSIZEL                 lpsizel
)
```

```
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjOleObjectImpl FAR*)lpThis)->lpPseudoObj;
   OleDbgOut2("PseudoObj_OleObj_GetExtent\r\n");

   /* OLE2NOTE: it is VERY important to check which aspect the caller
   **    is asking about. an object implemented by a server EXE MAY
   **    fail to return extents when asked for DVASPECT_ICON.
   */
   if (dwDrawAspect == DVASPECT_CONTENT) {
      PseudoObj_GetExtent(lpPseudoObj, lpsizel);
      return NOERROR;
   }
   else
   {
      return ResultFromScode(E_FAIL);
   }
}


STDMETHODIMP PseudoObj_OleObj_Advise(
      LPOLEOBJECT lpThis,
      LPADVISESINK lpAdvSink,
      LPDWORD lpdwConnection
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjOleObjectImpl FAR*)lpThis)->lpPseudoObj;
   HRESULT hrErr;
   SCODE   sc;
   OLEDBG_BEGIN2("PseudoObj_OleObj_Advise\r\n");

   if (lpPseudoObj->m_lpOleAdviseHldr == NULL &&
      CreateOleAdviseHolder(&lpPseudoObj->m_lpOleAdviseHldr) != NOERROR) {
      sc = E_OUTOFMEMORY;
      goto error;
   }

   OLEDBG_BEGIN2("IOleAdviseHolder::Advise called\r\n")
   hrErr = lpPseudoObj->m_lpOleAdviseHldr->lpVtbl->Advise(
         lpPseudoObj->m_lpOleAdviseHldr,
         lpAdvSink,
         lpdwConnection
   );
   OLEDBG_END2

   OLEDBG_END2
   return hrErr;

error:
   OLEDBG_END2
   return ResultFromScode(sc);
}
```

```
STDMETHODIMP PseudoObj_OleObj_Unadvise(LPOLEOBJECT lpThis, DWORD
dwConnection)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjOleObjectImpl FAR*)lpThis)->lpPseudoObj;
   HRESULT hrErr;
   SCODE   sc;

   // artificial AddRef in case object is destroyed during call
   PseudoObj_OleObj_AddRef(lpThis);

   OLEDBG_BEGIN2("PseudoObj_OleObj_Unadvise\r\n");


   if (lpPseudoObj->m_lpOleAdviseHldr == NULL) {
      sc = E_FAIL;
      goto error;
   }

   OLEDBG_BEGIN2("IOleAdviseHolder::Unadvise called\r\n")
   hrErr = lpPseudoObj->m_lpOleAdviseHldr->lpVtbl->Unadvise(
         lpPseudoObj->m_lpOleAdviseHldr,
         dwConnection
   );
   OLEDBG_END2

   OLEDBG_END2
   // release artificial AddRef
   PseudoObj_OleObj_Release(lpThis);
   return hrErr;

error:
   OLEDBG_END2
   // release artificial AddRef
   PseudoObj_OleObj_Release(lpThis);
   return ResultFromScode(sc);
}


STDMETHODIMP PseudoObj_OleObj_EnumAdvise(
      LPOLEOBJECT lpThis,
      LPENUMSTATDATA FAR* lplpenumAdvise
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjOleObjectImpl FAR*)lpThis)->lpPseudoObj;
   HRESULT hrErr;
   SCODE   sc;

   // artificial AddRef in case object is destroyed during call
   PseudoObj_OleObj_AddRef(lpThis);

   OLEDBG_BEGIN2("PseudoObj_OleObj_EnumAdvise\r\n");

   /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
   *lplpenumAdvise = NULL;

   if (lpPseudoObj->m_lpOleAdviseHldr == NULL) {
      sc = E_FAIL;
      goto error;
   }

   OLEDBG_BEGIN2("IOleAdviseHolder::EnumAdvise called\r\n")
   hrErr = lpPseudoObj->m_lpOleAdviseHldr->lpVtbl->EnumAdvise(
         lpPseudoObj->m_lpOleAdviseHldr,
         lplpenumAdvise
   );
   OLEDBG_END2

   OLEDBG_END2
   // release artificial AddRef
   PseudoObj_OleObj_Release(lpThis);
   return hrErr;

error:
   OLEDBG_END2
   // release artificial AddRef
   PseudoObj_OleObj_Release(lpThis);
   return ResultFromScode(sc);
}


STDMETHODIMP PseudoObj_OleObj_GetMiscStatus(
      LPOLEOBJECT             lpThis,
      DWORD                   dwAspect,
      DWORD FAR*              lpdwStatus
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjOleObjectImpl FAR*)lpThis)->lpPseudoObj;
   LPOUTLINEDOC  lpOutlineDoc = (LPOUTLINEDOC)lpPseudoObj->m_lpDoc;
   OleDbgOut2("PseudoObj_OleObj_GetMiscStatus\r\n");

   /* Get our default MiscStatus for the given Aspect. this
   **    information is registered in the RegDB. We query the RegDB
   **    here to guarantee that the value returned from this method
   **    agrees with the values in RegDB. in this way we only have to
   **    maintain the info in one place (in the RegDB). Alternatively
   **    we could have the values hard coded here.
   **
   ** OLE2NOTE: A pseudo object may NOT return OLE_S_USEREG; they must
   **    call the
   **    OleReg* API or provide their own implementation. Because this
   **    pseudo object does NOT implement IPersist, simply a low-level
   **    remoting handler (ProxyManager) object as opposed to a
   **    DefHandler object is used as the handler for the pseudo
   **    object in a clients process space. The ProxyManager does NOT
   **    handle the OLE_S_USEREG return values.
   */
   OleRegGetMiscStatus((REFCLSID)&CLSID_APP, dwAspect, lpdwStatus);
   /* OLE2NOTE: check if the pseudo object is compatible to be
   **    linked by an OLE 1.0 container. it is compatible if
   **    either the pseudo object is an untitled document or a
   **    file-based document. if the pseudo object is part of
   **    an embedded object, then it is NOT compatible to be
   **    linked by an OLE 1.0 container. if it is compatible then
   **    we should include OLEMISC_CANLINKBYOLE1 as part of the
   **    dwStatus flags.
   */
   if (lpOutlineDoc->m_docInitType == DOCTYPE_NEW ||
      lpOutlineDoc->m_docInitType == DOCTYPE_FROMFILE)
      *lpdwStatus |= OLEMISC_CANLINKBYOLE1;

   return NOERROR;
}


STDMETHODIMP PseudoObj_OleObj_SetColorScheme(
      LPOLEOBJECT             lpThis,
      LPLOGPALETTE            lpLogpal
)
{
   OleDbgOut2("PseudoObj_OleObj_SetColorScheme\r\n");

   // REVIEW: NOT YET IMPLEMENTED

   return ResultFromScode(E_NOTIMPL);
}


/*************************************************************************
** PseudoObj::IDataObject interface implementation
*************************************************************************/

STDMETHODIMP PseudoObj_DataObj_QueryInterface (
      LPDATAOBJECT      lpThis,
      REFIID            riid,
      LPVOID FAR*       lplpvObj
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjDataObjectImpl FAR*)lpThis)->lpPseudoObj;

   return PseudoObj_QueryInterface(lpPseudoObj, riid, lplpvObj);
}


STDMETHODIMP_(ULONG) PseudoObj_DataObj_AddRef(LPDATAOBJECT lpThis)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjDataObjectImpl FAR*)lpThis)->lpPseudoObj;

   OleDbgAddRefMethod(lpThis, "IDataObject");
   return PseudoObj_AddRef((LPPSEUDOOBJ)lpPseudoObj);
}


STDMETHODIMP_(ULONG) PseudoObj_DataObj_Release (LPDATAOBJECT lpThis)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjDataObjectImpl FAR*)lpThis)->lpPseudoObj;

   OleDbgReleaseMethod(lpThis, "IDataObject");

   return PseudoObj_Release((LPPSEUDOOBJ)lpPseudoObj);
}


STDMETHODIMP PseudoObj_DataObj_GetData (
      LPDATAOBJECT    lpThis,
      LPFORMATETC     lpformatetc,
      LPSTGMEDIUM     lpMedium
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjDataObjectImpl FAR*)lpThis)->lpPseudoObj;
   LPSERVERDOC  lpServerDoc = lpPseudoObj->m_lpDoc;
   LPOUTLINEDOC  lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPSERVERAPP lpServerApp = (LPSERVERAPP)g_lpApp;
   LPOLEAPP  lpOleApp = (LPOLEAPP)lpServerApp;
   LPOUTLINEAPP  lpOutlineApp = (LPOUTLINEAPP)lpServerApp;
   LINERANGE lrSel;
   SCODE sc = S_OK;

   // artificial AddRef in case object is destroyed during call
   PseudoObj_DataObj_AddRef(lpThis);

   OLEDBG_BEGIN2("PseudoObj_DataObj_GetData\r\n")

   PseudoObj_GetSel(lpPseudoObj, &lrSel);

   /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
   lpMedium->tymed = TYMED_NULL;
   lpMedium->pUnkForRelease = NULL;    // we transfer ownership to caller
   lpMedium->u.hGlobal = NULL;

   if (lpformatetc->cfFormat == lpOutlineApp->m_cfOutline) {
      // Verify caller asked for correct medium
      if (!(lpformatetc->tymed & TYMED_HGLOBAL)) {
         sc = DATA_E_FORMATETC;
         goto error;
      }

      lpMedium->u.hGlobal = OutlineDoc_GetOutlineData (lpOutlineDoc,&lrSel);
      if (! lpMedium->u.hGlobal) return ResultFromScode(E_OUTOFMEMORY);
      lpMedium->tymed = TYMED_HGLOBAL;
      OleDbgOut3("PseudoObj_DataObj_GetData: rendered CF_OUTLINE\r\n");
   } else if(lpformatetc->cfFormat == CF_METAFILEPICT &&
      (lpformatetc->dwAspect & DVASPECT_CONTENT) ) {
      // Verify caller asked for correct medium
      if (!(lpformatetc->tymed & TYMED_MFPICT)) {
         sc = DATA_E_FORMATETC;
         goto error;
      }

      lpMedium->u.hGlobal=ServerDoc_GetMetafilePictData(lpServerDoc,&lrSel);
      if (! lpMedium->u.hGlobal) {
         sc = E_OUTOFMEMORY;
         goto error;
      }
      lpMedium->tymed = TYMED_MFPICT;
      OleDbgOut3("PseudoObj_DataObj_GetData: rendered CF_METAFILEPICT\r\n");

   } else if (lpformatetc->cfFormat == CF_METAFILEPICT &&
      (lpformatetc->dwAspect & DVASPECT_ICON) ) {
      CLSID clsid;
      // Verify caller asked for correct medium
      if (!(lpformatetc->tymed & TYMED_MFPICT)) {
         sc = DATA_E_FORMATETC;
         goto error;
      }

      /* OLE2NOTE: we should return the default icon for our class.
      **    we must be carefull to use the correct CLSID here.
      **    if we are currently preforming a "TreatAs (aka. ActivateAs)"
      **    operation then we need to use the class of the object
      **    written in the storage of the object. otherwise we would
      **    use our own class id.
      */
      if (ServerDoc_GetClassID(lpServerDoc, (LPCLSID)&clsid) != NOERROR) {
         sc = DATA_E_FORMATETC;
         goto error;
      }


      lpMedium->u.hGlobal=GetIconOfClass(
            g_lpApp->m_hInst,(REFCLSID)&clsid, NULL, FALSE);
      if (! lpMedium->u.hGlobal) {
         sc = E_OUTOFMEMORY;
         goto error;
      }

      lpMedium->tymed = TYMED_MFPICT;
      OleDbgOut3("PseudoObj_DataObj_GetData: rendered CF_METAFILEPICT (icon)
\r\n");
      // release artificial AddRef
      PseudoObj_DataObj_Release(lpThis);
      return NOERROR;

   } else if (lpformatetc->cfFormat == CF_TEXT) {
      // Verify caller asked for correct medium
      if (!(lpformatetc->tymed & TYMED_HGLOBAL)) {
         sc = DATA_E_FORMATETC;
         goto error;
      }

      lpMedium->u.hGlobal = OutlineDoc_GetTextData (lpOutlineDoc, &lrSel);
      if (! lpMedium->u.hGlobal) {
         sc = E_OUTOFMEMORY;
         goto error;
      }
      lpMedium->tymed = TYMED_HGLOBAL;
      OleDbgOut3("PseudoObj_DataObj_GetData: rendered CF_TEXT\r\n");

   } else {
      sc = DATA_E_FORMATETC;
      goto error;
   }

   OLEDBG_END2
   // release artificial AddRef
   PseudoObj_DataObj_Release(lpThis);
   return NOERROR;

error:
   OLEDBG_END2
   // release artificial AddRef
   PseudoObj_DataObj_Release(lpThis);
   return ResultFromScode(sc);
}


STDMETHODIMP PseudoObj_DataObj_GetDataHere (
      LPDATAOBJECT    lpThis,
      LPFORMATETC     lpformatetc,
      LPSTGMEDIUM     lpMedium
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjDataObjectImpl FAR*)lpThis)->lpPseudoObj;
   LPSERVERDOC  lpServerDoc = lpPseudoObj->m_lpDoc;
   LPOUTLINEDOC  lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPSERVERAPP lpServerApp = (LPSERVERAPP)g_lpApp;
   LPOLEAPP  lpOleApp = (LPOLEAPP)lpServerApp;
   LPOUTLINEAPP  lpOutlineApp = (LPOUTLINEAPP)lpServerApp;


   OleDbgOut("PseudoObj_DataObj_GetDataHere\r\n");

   /* Caller is requesting data to be returned in Caller allocated
   **    medium, but we do NOT support this. we only support
   **    global memory blocks that WE allocate for the caller.
   */
   return ResultFromScode(DATA_E_FORMATETC);
}
```

```
STDMETHODIMP PseudoObj_DataObj_QueryGetData (
      LPDATAOBJECT    lpThis,
      LPFORMATETC     lpformatetc
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjDataObjectImpl FAR*)lpThis)->lpPseudoObj;
   LPSERVERDOC  lpServerDoc = lpPseudoObj->m_lpDoc;
   LPOUTLINEDOC  lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPSERVERAPP lpServerApp = (LPSERVERAPP)g_lpApp;
   LPOLEAPP  lpOleApp = (LPOLEAPP)lpServerApp;
   LPOUTLINEAPP  lpOutlineApp = (LPOUTLINEAPP)lpServerApp;

   // artificial AddRef in case object is destroyed during call
   PseudoObj_DataObj_AddRef(lpThis);

   OleDbgOut2("PseudoObj_DataObj_QueryGetData\r\n");

   /* Caller is querying if we support certain format but does not
   **    want any data actually returned.
   */
   if (lpformatetc->cfFormat == CF_METAFILEPICT &&
      (lpformatetc->dwAspect & (DVASPECT_CONTENT | DVASPECT_ICON)) ) {
      // release artificial AddRef
      PseudoObj_DataObj_Release(lpThis);
      return OleStdQueryFormatMedium(lpformatetc, TYMED_MFPICT);

   } else if (lpformatetc->cfFormat == (lpOutlineApp)->m_cfOutline ||
         lpformatetc->cfFormat == CF_TEXT) {
      // release artificial AddRef
      PseudoObj_DataObj_Release(lpThis);
      return OleStdQueryFormatMedium(lpformatetc, TYMED_HGLOBAL);
   }

   // release artificial AddRef
   PseudoObj_DataObj_Release(lpThis);
   return ResultFromScode(DATA_E_FORMATETC);
}


STDMETHODIMP PseudoObj_DataObj_GetCanonicalFormatEtc(
      LPDATAOBJECT    lpThis,
      LPFORMATETC     lpformatetc,
      LPFORMATETC     lpformatetcOut
)
{
   HRESULT hrErr;
   OleDbgOut2("PseudoObj_DataObj_GetCanonicalFormatEtc\r\n");

   if (!lpformatetcOut)
      return ResultFromScode(E_INVALIDARG);

   /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
   lpformatetcOut->ptd = NULL;
   if (!lpformatetc)
      return ResultFromScode(E_INVALIDARG);

   // OLE2NOTE: we must validate that the format requested is supported
   if ((hrErr=lpThis->lpVtbl->QueryGetData(lpThis,lpformatetc)) != NOERROR)
      return hrErr;

   /* OLE2NOTE: an app that is insensitive to target device (as the
   **    Outline Sample is) should fill in the lpformatOut parameter
   **    but NULL out the "ptd" field; it should return NOERROR if the
   **    input formatetc->ptd what non-NULL. this tells the caller
   **    that it is NOT necessary to maintain a separate screen
   **    rendering and printer rendering. if should return
   **    DATA_S_SAMEFORMATETC if the input and output formatetc's are
   **    identical.
   */

   *lpformatetcOut = *lpformatetc;
   if (lpformatetc->ptd == NULL)
      return ResultFromScode(DATA_S_SAMEFORMATETC);
   else {
      lpformatetcOut->ptd = NULL;
      return NOERROR;
   }
}


STDMETHODIMP PseudoObj_DataObj_SetData (
      LPDATAOBJECT    lpThis,
      LPFORMATETC     lpformatetc,
      LPSTGMEDIUM     lpmedium,
      BOOL            fRelease
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjDataObjectImpl FAR*)lpThis)->lpPseudoObj;
   LPSERVERDOC  lpServerDoc = lpPseudoObj->m_lpDoc;
   LPOUTLINEDOC  lpOutlineDoc = (LPOUTLINEDOC)lpServerDoc;
   LPSERVERAPP lpServerApp = (LPSERVERAPP)g_lpApp;

   OleDbgOut2("PseudoObj_DataObj_SetData\r\n");

   // REVIEW: NOT-YET-IMPLEMENTED
   return ResultFromScode(E_NOTIMPL);
}


STDMETHODIMP PseudoObj_DataObj_EnumFormatEtc(
      LPDATAOBJECT            lpThis,
      DWORD                   dwDirection,
      LPENUMFORMATETC FAR*    lplpenumFormatEtc
)
{
   SCODE sc;
   OleDbgOut2("PseudoObj_DataObj_EnumFormatEtc\r\n");

   /* OLE2NOTE: a pseudo object only needs to enumerate the static list
   **    of formats that are registered for our app in the
   **    registration database. it is NOT
   **    required that a pseudo object (ie. non-DataTransferDoc)
   **    enumerate the OLE formats: CF_LINKSOURCE, CF_EMBEDSOURCE, or
   **    CF_EMBEDDEDOBJECT. we do NOT use pseudo objects for data
   **    transfers.
   **
   **    A pseudo object may NOT return OLE_S_USEREG; they must call the
   **    OleReg* API or provide their own implementation. Because this
   **    pseudo object does NOT implement IPersist, simply a low-level
   **    remoting handler (ProxyManager) object as opposed to a
   **    DefHandler object is used as the handler for the pseudo
   **    object in a clients process space. The ProxyManager does NOT
   **    handle the OLE_S_USEREG return values.
   */
   if (dwDirection == DATADIR_GET)
      return OleRegEnumFormatEtc(
            (REFCLSID)&CLSID_APP, dwDirection, lplpenumFormatEtc);
   else if (dwDirection == DATADIR_SET)
      sc = E_NOTIMPL;
   else
      sc = E_INVALIDARG;

   return ResultFromScode(sc);
}


STDMETHODIMP PseudoObj_DataObj_DAdvise(
      LPDATAOBJECT    lpThis,
      FORMATETC FAR*  lpFormatetc,
      DWORD           advf,
      LPADVISESINK    lpAdvSink,
      DWORD FAR*      lpdwConnection
)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjDataObjectImpl FAR*)lpThis)->lpPseudoObj;
   HRESULT hrErr;
   SCODE   sc;

   // artificial AddRef in case object is destroyed during call
   PseudoObj_DataObj_AddRef(lpThis);

   OLEDBG_BEGIN2("PseudoObj_DataObj_DAdvise\r\n")

   /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
   *lpdwConnection = 0;

   /* OLE2NOTE: we should validate if the caller is setting up an
   **    Advise for a data type that we support. we must
   **    explicitly allow an advise for the "wildcard" advise.
   */
#ifdef WIN32
   if ( !( lpFormatetc->cfFormat == 0 &&
      lpFormatetc->ptd == 0 &&
#else
   if ( !( lpFormatetc->cfFormat == NULL &&
      lpFormatetc->ptd == NULL &&
#endif
      lpFormatetc->dwAspect == -1L &&
      lpFormatetc->lindex == -1L &&
      lpFormatetc->tymed == -1L) &&
      (hrErr = PseudoObj_DataObj_QueryGetData(lpThis, lpFormatetc))
         != NOERROR) {
      sc = GetScode(hrErr);
      goto error;
   }

   if (lpPseudoObj->m_lpDataAdviseHldr == NULL &&
      CreateDataAdviseHolder(&lpPseudoObj->m_lpDataAdviseHldr) != NOERROR) {
      sc = E_OUTOFMEMORY;
      goto error;
   }

   OLEDBG_BEGIN2("IOleAdviseHolder::Advise called\r\n")
   hrErr = lpPseudoObj->m_lpDataAdviseHldr->lpVtbl->Advise(
         lpPseudoObj->m_lpDataAdviseHldr,
         (LPDATAOBJECT)&lpPseudoObj->m_DataObject,
         lpFormatetc,
         advf,
         lpAdvSink,
         lpdwConnection
   );
   OLEDBG_END2

   OLEDBG_END2
   // release artificial AddRef
   PseudoObj_DataObj_Release(lpThis);
   return hrErr;

error:
   OLEDBG_END2
   // release artificial AddRef
   PseudoObj_DataObj_Release(lpThis);
   return ResultFromScode(sc);
}


STDMETHODIMP PseudoObj_DataObj_DUnadvise(LPDATAOBJECT lpThis, DWORD
dwConnection)
{
   LPPSEUDOOBJ lpPseudoObj =
         ((struct CPseudoObjDataObjectImpl FAR*)lpThis)->lpPseudoObj;
   HRESULT hrErr;
   SCODE   sc;
```

```
    // artificial AddRef in case object is destroyed during call
    PseudoObj_DataObj_AddRef(lpThis);
    OLEDBG_BEGIN2("PseudoObj_DataObj_Unadvise\r\n");

    // no one registered
    if (lpPseudoObj->m_lpDataAdviseHldr == NULL) {
       sc = E_FAIL;
       goto error;
    }

    OLEDBG_BEGIN2("IOleAdviseHolder::DUnadvise called\r\n")
    hrErr = lpPseudoObj->m_lpDataAdviseHldr->lpVtbl->Unadvise(
          lpPseudoObj->m_lpDataAdviseHldr,
          dwConnection
    );
    OLEDBG_END2

    OLEDBG_END2
    // release artificial AddRef
    PseudoObj_DataObj_Release(lpThis);
    return hrErr;

error:
    OLEDBG_END2
    // release artificial AddRef
    PseudoObj_DataObj_Release(lpThis);
    return ResultFromScode(sc);
}


STDMETHODIMP PseudoObj_DataObj_EnumAdvise(
      LPDATAOBJECT lpThis,
      LPENUMSTATDATA FAR* lplpenumAdvise
)
{
    LPPSEUDOOBJ lpPseudoObj =
          ((struct CPseudoObjDataObjectImpl FAR*)lpThis)->lpPseudoObj;
    HRESULT hrErr;
    SCODE   sc;

    // artificial AddRef in case object is destroyed during call
    PseudoObj_DataObj_AddRef(lpThis);

    OLEDBG_BEGIN2("PseudoObj_DataObj_EnumAdvise\r\n");

    /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
    *lplpenumAdvise = NULL;

    if (lpPseudoObj->m_lpDataAdviseHldr == NULL) {
       sc = E_FAIL;
       goto error;
    }

    OLEDBG_BEGIN2("IOleAdviseHolder::EnumAdvise called\r\n")
```

```
   hrErr = lpPseudoObj->m_lpDataAdviseHldr->lpVtbl->EnumAdvise(
         lpPseudoObj->m_lpDataAdviseHldr,
         lplpenumAdvise
   );
   OLEDBG_END2

   OLEDBG_END2
   // release artificial AddRef
   PseudoObj_DataObj_Release(lpThis);
   return hrErr;

error:
   OLEDBG_END2
   // release artificial AddRef
   PseudoObj_DataObj_Release(lpThis);
   return ResultFromScode(sc);
}
```

SIMPCNTR

Simpcntr -------- This sample is the simplest OLE 2.0 container that can be written and still support the visual editing feature. The sample has no native file format, and can only support one OLE object at a time. The Insert Object command is used to create an object.

See the MAKEFILE for compilation instructions.

MAKEFILE   (SIMPCNTR Sample)

```
!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

all: simpcntr.exe

simpcntr.exe:  app.obj         \
            doc.obj         \
            ias.obj         \
            iocs.obj        \
            ioipf.obj        \
            ioips.obj        \
            simpcntr.obj     \
            simpcntr.res     \
            site.obj
  $(link) $(linkdebug) $(guiflags) -machine:$(CPU) -out:$*.exe $** $
(olelibs) ole2ui.lib

.cpp.obj:
    $(cc) $(cdebug) $(cflags) $(cvars) $*.cpp

simpcntr.res: simpcntr.rc simpcntr.h
    $(rc) -r -I..\ole2ui -I..\ole2ui\res\usa -I..\ole2ui\res\static
simpcntr.rc
```

APP.H (SIMPCNTR Sample)

```
//**********************************************************************
// File name: app.h
//
//      Definition of CSimpleApp
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#if !defined( _APP_H_)
#define _APP_H_

#include <ole2.h>
#include <stablize.h>
#include "ioipf.h"

class CSimpleDoc;

class CSimpleApp : public IUnknown, public CSafeRefCount
{
public:

        HWND m_hAppWnd;         // main window handle
        HINSTANCE m_hInst;          // application instance
        HMENU           m_hMainMenu;
        HMENU           m_hFileMenu;
        HMENU           m_hEditMenu;
        HMENU           m_hHelpMenu;
        HMENU           m_hCascadeMenu;


        COleInPlaceFrame m_OleInPlaceFrame; // IOleInPlaceFrame
Implementation

        CSimpleDoc FAR * m_lpDoc;   // pointer to document object
        BOOL m_fInitialized;        // OLE initialization flag
        BOOL m_fCSHMode;
        BOOL m_fMenuMode;
        HWND m_hwndUIActiveObj; // HWND of UIActive Object
        HPALETTE m_hStdPal;     // Color palette used by container
        BOOL m_fAppActive;      // TRUE if app is active

        CSimpleApp();           // Constructor
        ~CSimpleApp();          // Destructor
        RECT nullRect;


        // IUnknown Interfaces
        STDMETHODIMP QueryInterface(REFIID riid, LPVOID FAR* ppvObj);
        STDMETHODIMP_(ULONG) AddRef();
        STDMETHODIMP_(ULONG) Release();

        // Initialization methods
        BOOL fInitApplication (HANDLE hInstance);
        BOOL fInitInstance (HANDLE hInstance, int nCmdShow);

        // Message handling methods

        long lCommandHandler (HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam);
        long lSizeHandler (HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam);
        long lCreateDoc (HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam);
        BOOL HandleAccelerators (LPMSG lpMsg);
        void PaintApp(HDC hDC);
        void DestroyDocs();

        // In-Place support functions
        void AddFrameLevelUI();
        void AddFrameLevelTools();
        void ContextSensitiveHelp (BOOL fEnterMode);
        LRESULT QueryNewPalette(void);
};

LRESULT wSelectPalette(HWND hWnd, HPALETTE hPal, BOOL fBackground);

#endif
```

DOC.H (SIMPCNTR Sample)

```
//**********************************************************************
// File name: doc.h
//
//      Definition of CSimpleDoc
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#if !defined( _DOC_H_ )
#define _DOC_H_

#include <stablize.h>

class CSimpleSite;
class CSimpleApp;

class CSimpleDoc : public IUnknown, public CSafeRefCount
{
public:
        LPSTORAGE       m_lpStorage;
        LPOLEINPLACEACTIVEOBJECT m_lpActiveObject;
        BOOL            m_fInPlaceActive;
        BOOL            m_fAddMyUI;
        BOOL            m_fModifiedMenu;

        CSimpleSite FAR * m_lpSite;
        CSimpleApp FAR * m_lpApp;

        HWND m_hDocWnd;

        static CSimpleDoc FAR * Create(CSimpleApp FAR *lpApp, LPRECT
lpRect,HWND hWnd);

        void Close(void);

        CSimpleDoc();
        CSimpleDoc(CSimpleApp FAR *lpApp, HWND hWnd);
        ~CSimpleDoc();

        // IUnknown Interface
        STDMETHODIMP QueryInterface(REFIID riid, LPVOID FAR* ppvObj);
        STDMETHODIMP_(ULONG) AddRef();
        STDMETHODIMP_(ULONG) Release();

        void InsertObject(void);
        void DisableInsertObject(void);
        long lResizeDoc(LPRECT lpRect);
        long lAddVerbs(void);
        void PaintDoc(HDC hDC);
};

#endif
```

IAS.H (SIMPCNTR Sample)

```
//**********************************************************************
// File name: IAS.H
//
//      Definition of CAdviseSink
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************
#if !defined( _IAS_H_ )
#define _IAS_H_

#include <assert.h>

class CSimpleSite;

interface CAdviseSink : public IAdviseSink
{
     int m_nCount;
     CSimpleSite FAR * m_pSite;

     CAdviseSink(CSimpleSite FAR * pSite) {
           OutputDebugString("In IAS's constructor\r\n");
           m_pSite = pSite;
           m_nCount = 0;
           };

     ~CAdviseSink() {
           OutputDebugString("In IAS's destructor\r\n");
           assert(m_nCount == 0);
           } ;

     STDMETHODIMP QueryInterface (REFIID riid, LPVOID FAR* ppv);
     STDMETHODIMP_(ULONG) AddRef ();
     STDMETHODIMP_(ULONG) Release ();

     // *** IAdviseSink methods ***
     STDMETHODIMP_(void) OnDataChange (FORMATETC FAR* pFormatetc, STGMEDIUM 
FAR* pStgmed);
     STDMETHODIMP_(void) OnViewChange (DWORD dwAspect, LONG lindex);
     STDMETHODIMP_(void) OnRename (LPMONIKER pmk);
     STDMETHODIMP_(void) OnSave ();
     STDMETHODIMP_(void) OnClose ();
};


#endif
```

IOCS.H (SIMPCNTR Sample)

```
//**********************************************************************
// File name: IOCS.H
//
//      Definition of COleClientSite
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************
#if !defined( _IOCS_H_ )
#define _IOCS_H_

#include <assert.h>

class CSimpleSite;

interface COleClientSite : public IOleClientSite
{
     int m_nCount;
     CSimpleSite FAR * m_pSite;

     COleClientSite(CSimpleSite FAR * pSite) {
           OutputDebugString("In IOCS's constructor\r\n");
           m_pSite = pSite;
           m_nCount = 0;
           }

     ~COleClientSite() {
           OutputDebugString("In IOCS's destructor\r\n");
           assert(m_nCount == 0);
           }

     STDMETHODIMP QueryInterface(REFIID riid, LPVOID FAR* ppvObj);
     STDMETHODIMP_(ULONG) AddRef();
     STDMETHODIMP_(ULONG) Release();

     // *** IOleClientSite methods ***
     STDMETHODIMP SaveObject();
     STDMETHODIMP GetMoniker(DWORD dwAssign, DWORD dwWhichMoniker, LPMONIKER
FAR* ppmk);
     STDMETHODIMP GetContainer(LPOLECONTAINER FAR* ppContainer);
     STDMETHODIMP ShowObject();
     STDMETHODIMP OnShowWindow(BOOL fShow);
     STDMETHODIMP RequestNewObjectLayout();
};

#endif
```

IOIPF.H (SIMPCNTR Sample)

```
//**********************************************************************
// File name: IOIPF.H
//
//      Definition of COleInPlaceFrame
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************
#if !defined( _IOIPF_H_ )
#define _IOIPF_H_


class CSimpleApp;

interface COleInPlaceFrame : public IOleInPlaceFrame
{
     int m_nCount;
     CSimpleApp FAR * m_pApp;

     COleInPlaceFrame(CSimpleApp FAR * pApp) {
           OutputDebugString("In IOIPF's constructor\r\n");
           m_pApp = pApp;
           m_nCount = 0;
           };

     ~COleInPlaceFrame() {
           OutputDebugString("In IOIPFS's destructor\r\n");
           assert(m_nCount == 0);
           };

     STDMETHODIMP QueryInterface (REFIID riid, LPVOID FAR* ppv);
     STDMETHODIMP_(ULONG) AddRef ();
     STDMETHODIMP_(ULONG) Release ();

     STDMETHODIMP GetWindow (HWND FAR* lphwnd);
     STDMETHODIMP ContextSensitiveHelp (BOOL fEnterMode);

     // *** IOleInPlaceUIWindow methods ***
     STDMETHODIMP GetBorder (LPRECT lprectBorder);
     STDMETHODIMP RequestBorderSpace (LPCBORDERWIDTHS lpborderwidths);
     STDMETHODIMP SetBorderSpace (LPCBORDERWIDTHS lpborderwidths);
  //@@WTK WIN32, UNICODE
     //STDMETHODIMP SetActiveObject (LPOLEINPLACEACTIVEOBJECT
lpActiveObject,LPCSTR lpszObjName);
     STDMETHODIMP SetActiveObject (LPOLEINPLACEACTIVEOBJECT
lpActiveObject,LPCOLESTR lpszObjName);

     // *** IOleInPlaceFrame methods ***
     STDMETHODIMP InsertMenus (HMENU hmenuShared, LPOLEMENUGROUPWIDTHS
lpMenuWidths);
     STDMETHODIMP SetMenu (HMENU hmenuShared, HOLEMENU holemenu, HWND
hwndActiveObject);
     STDMETHODIMP RemoveMenus (HMENU hmenuShared);
  //@@WTK WIN32, UNICODE
     //STDMETHODIMP SetStatusText (LPCSTR lpszStatusText);
     STDMETHODIMP SetStatusText (LPCOLESTR lpszStatusText);
     STDMETHODIMP EnableModeless (BOOL fEnable);
     STDMETHODIMP TranslateAccelerator (LPMSG lpmsg, WORD wID);
};

#endif
```

IOIPS.H (SIMPCNTR Sample)

```
//**********************************************************************
// File name: IOIPS.H
//
//      Definition of COleInPlaceSite
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************
#if !defined( _IOIPS_H_ )
#define _IOIPS_H_


class CSimpleSite;

interface COleInPlaceSite : public IOleInPlaceSite
{
     int m_nCount;
     CSimpleSite FAR * m_pSite;

     COleInPlaceSite(CSimpleSite FAR *pSite) {
           OutputDebugString("In IOIPS's constructor\r\n");
           m_pSite = pSite;
           m_nCount = 0;
           };

     ~COleInPlaceSite() {
           OutputDebugString("In IOIPS;s destructor\r\n");
           assert(m_nCount == 0);
           };

     STDMETHODIMP QueryInterface (REFIID riid, LPVOID FAR* ppv);
     STDMETHODIMP_(ULONG) AddRef ();
     STDMETHODIMP_(ULONG) Release ();

     STDMETHODIMP GetWindow (HWND FAR* lphwnd);
     STDMETHODIMP ContextSensitiveHelp (BOOL fEnterMode);

     // *** IOleInPlaceSite methods ***
     STDMETHODIMP CanInPlaceActivate ();
     STDMETHODIMP OnInPlaceActivate ();
     STDMETHODIMP OnUIActivate ();
     STDMETHODIMP GetWindowContext (LPOLEINPLACEFRAME FAR* lplpFrame,
                                                  LPOLEINPLACEUIWINDOW FAR*
lplpDoc,
                                                  LPRECT lprcPosRect,
                                                  LPRECT lprcClipRect,
                                                  LPOLEINPLACEFRAMEINFO
lpFrameInfo);
     STDMETHODIMP Scroll (SIZE scrollExtent);
     STDMETHODIMP OnUIDeactivate (BOOL fUndoable);
     STDMETHODIMP OnInPlaceDeactivate ();
     STDMETHODIMP DiscardUndoState ();
     STDMETHODIMP DeactivateAndUndo ();
     STDMETHODIMP OnPosRectChange (LPCRECT lprcPosRect);
};

#endif
```

PRE.H   (SIMPCNTR Sample)

```
//**********************************************************************
// File name: pre.h
//
//      Used for precompiled headers
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#if !defined( _PRE_H_)
#define _PRE_H_

#include <windows.h>
#include <ole2.h>
#include <ole2ui.h>
#include <assert.h>
#include <string.h>
#include "simpcntr.h"
#include "resource.h"
#include <ole2ver.h>


#endif
```

RESOURCE.H (SIMPCNTR Sample)

```
//{{NO_DEPENDENCIES}}
// App Studio generated include file.
// Used by SIMPCNTR.RC
//
#define IDM_OPEN                        102
#define IDM_SAVE                        103
#define IDM_SAVEAS                      104
#define IDM_PRINT                       105
#define IDM_EXIT                        106
#define IDM_UNDO                        107
#define IDM_CUT                         108
#define IDM_COPY                        109
#define IDM_PASTE                       110
#define ID_EDIT_INSERTOBJECT            111
#define IDM_INSERTOBJECT                111
#define IDM_NEW                         112

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS

#define _APS_NEXT_RESOURCE_VALUE        101
#define _APS_NEXT_COMMAND_VALUE         113
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

SIMPCNTR.H (SIMPCNTR Sample)

```
//**********************************************************************
// File name: simple.h
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************
#define IDM_ABOUT 100
#define IDM_INSERT  101
#define IDM_VERB0 1000

//@@WTK WIN32, UNICODE
#ifdef WIN32
#define EXPORT
#else
#define EXPORT __export
#endif

int PASCAL WinMain(HANDLE hInstance,HANDLE hPrevInstance,LPSTR lpCmdLine,int
nCmdShow);
BOOL InitApplication(HANDLE hInstance);
BOOL InitInstance(HANDLE hInstance, int nCmdShow);
//@@WTK WIN32, UNICODE _export ==> EXPORT, portable params
long FAR PASCAL EXPORT MainWndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam);
long FAR PASCAL EXPORT DocWndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam);
BOOL FAR PASCAL EXPORT About(HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam);
```

SITE.H (SIMPCNTR Sample)

```
//**********************************************************************
// File name: SITE.H
//
//      Definition of CSimpleSite
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************
#if !defined( _SITE_H_ )
#define _SITE_H_

#include <ole2.h>
#include <stablize.h>
#include "ias.h"
#include "ioips.h"
#include "iocs.h"

class CSimpleDoc;

class CSimpleSite : public IUnknown, public CSafeRefCount
{
public:
        DWORD m_dwConnection;
        LPOLEOBJECT m_lpOleObject;
        LPOLEINPLACEOBJECT m_lpInPlaceObject;
        HWND m_hwndIPObj;
        DWORD m_dwDrawAspect;
        SIZEL m_sizel;
        BOOL m_fInPlaceActive;
        BOOL m_fObjectOpen;
        LPSTORAGE m_lpObjStorage;

        CAdviseSink m_AdviseSink;
        COleInPlaceSite m_OleInPlaceSite;
        COleClientSite m_OleClientSite;

        CSimpleDoc FAR * m_lpDoc;

        // IUnknown Interfaces
        STDMETHODIMP QueryInterface(REFIID riid, LPVOID FAR* ppvObj);
        STDMETHODIMP_(ULONG) AddRef();
        STDMETHODIMP_(ULONG) Release();

        void InitObject(BOOL fCreateNew);
        static CSimpleSite FAR * Create(CSimpleDoc FAR *lpDoc);
        CSimpleSite(CSimpleDoc FAR *lpDoc);
        ~CSimpleSite();
        void PaintObj(HDC hDC);
        void GetObjRect(LPRECT lpRect);
        void CloseOleObject(void);
        void UnloadOleObject(void);
};
```

```
#endif
```

APP.CPP (SIMPCNTR Sample)

```
//**********************************************************************
// File name: app.cpp
//
//    Implementation file for the CSimpleApp Class
//
// Functions:
//
//    See app.h for a list of member functions.
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "iocs.h"
#include "ias.h"
#include "ioipf.h"
#include "ioips.h"
#include "app.h"
#include "site.h"
#include "doc.h"
```

CSimpleApp::CSimpleApp (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::CSimpleApp()
//
// Purpose:
//
//      Constructor for CSimpleApp
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      SetRectEmpty                Windows API
//
// Comments:
//
//      CSimpleApp has a contained COleInPlaceFrame.  On construction
//      of CSimpleApp, we explicitly call the constructor of this
//      contained class and pass a copy of the this pointer, so that
//      COleInPlaceFrame can refer back to this class
//
//********************************************************************
#pragma warning(disable : 4355)  // turn off this warning.  This warning
                                                                // tells us
that we are passing this in
                                                                // an
initializer, before "this" is through
                                                                //
initializing.  This is ok, because
                                                                // we just
store the ptr in the other
                                                                //
constructor

CSimpleApp::CSimpleApp() : m_OleInPlaceFrame(this)
#pragma warning (default : 4355)  // Turn the warning back on
{
        OutputDebugString("In CSimpleApp's Constructor \r\n");

        // clear members
        m_hAppWnd = NULL;
        m_hInst = NULL;
        m_lpDoc = NULL;
    m_hwndUIActiveObj = NULL;
```

```
        // clear flags
        m_fInitialized = FALSE;
        m_fCSHMode = FALSE;
        m_fMenuMode = FALSE;

        // used for inplace
        SetRectEmpty(&nullRect);
}
```

CSimpleApp::~CSimpleApp (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::~CSimpleApp()
//
// Purpose:
//
//      Destructor for CSimpleApp Class.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      OleUninitialize             OLE API
//
// Comments:
//
//********************************************************************

CSimpleApp::~CSimpleApp()
{
        OutputDebugString("In CSimpleApp's Destructor\r\n");

        if (m_hStdPal)
                DeleteObject(m_hStdPal);

        // need to uninit the library...
        if (m_fInitialized)
                OleUninitialize();
}
```

CSimpleApp::DestroyDocs (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::DestroyDocs()
//
// Purpose:
//
//      Destroys all of the open documents in the application (Only one
//      since this is an SDI app, but could easily be modified to
//      support MDI).
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
// Comments:
//
//********************************************************************

void CSimpleApp::DestroyDocs()
{
        CStabilize stabilize(this);
        m_lpDoc->Close();   // we have only 1 document
}
```

CSimpleApp::QueryInterface   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::QueryInterface
//
// Purpose:
//
//      Used for interface negotiation at the Frame level.
//
// Parameters:
//
//      REFIID riid         -   A reference to the interface that is
//                              being queried.
//
//      LPVOID FAR* ppvObj  -   An out parameter to return a pointer to
//                              the interface.
//
// Return Value:
//
//      S_OK    -   The interface is supported.
//      S_FALSE -   The interface is not supported
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      IsEqualIID                  OLE API
//      ResultFromScode             OLE API
//      COleInPlaceFrame::AddRef    IOIPF.CPP
//
// Comments:
//
//      Note that this QueryInterface is associated with the frame.
//      Since the application could potentially have multiple documents
//      and multiple objects, a lot of the interfaces are ambiguous.
//      (ie. which IOleObject is returned?).  For this reason, only
//      pointers to interfaces associated with the frame are returned.
//      In this implementation, Only IOleInPlaceFrame (or one of the
//      interfaces it is derived from) can be returned.
//
//********************************************************************

STDMETHODIMP CSimpleApp::QueryInterface(REFIID riid, LPVOID FAR* ppvObj)
{
        OutputDebugString("In CSimpleApp::QueryInterface\r\n");

        *ppvObj = NULL;     // must set out pointer parameters to NULL

        // looking for IUnknown
        if ( riid == IID_IUnknown)
                {
                AddRef();
                *ppvObj = this;
                return ResultFromScode(S_OK);
                }

        // looking for IOleWindow
        if ( riid == IID_IOleWindow)
                {
                m_OleInPlaceFrame.AddRef();
                *ppvObj=&m_OleInPlaceFrame;
                return ResultFromScode(S_OK);
                }

        // looking for IOleInPlaceUIWindow
        if ( riid == IID_IOleInPlaceUIWindow)
                {
                m_OleInPlaceFrame.AddRef();
                *ppvObj=&m_OleInPlaceFrame;
                return ResultFromScode(S_OK);
                }

        // looking for IOleInPlaceFrame
        if ( riid == IID_IOleInPlaceFrame)
                {
                m_OleInPlaceFrame.AddRef();
                *ppvObj=&m_OleInPlaceFrame;
                return ResultFromScode(S_OK);
                }

        // Not a supported interface
        return ResultFromScode(E_NOINTERFACE);
}
```

CSimpleApp::AddRef (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::AddRef
//
// Purpose:
//
//      Adds to the reference count at the Application level.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the application.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Due to the reference counting model that is used in this
//      implementation, this reference count is the sum of the
//      reference counts on all interfaces of all objects open
//      in the application.
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpleApp::AddRef()
{
        OutputDebugString("In CSimpleApp::AddRef\r\n");
        return SafeAddRef();
}
```

CSimpleApp::Release   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::Release
//
// Purpose:
//
//      Decrements the reference count at this level
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the application.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpleApp::Release()
{
        OutputDebugString("In CSimpleApp::Release\r\n");

        return SafeRelease();
}
```

CSimpleApp::fInitApplication (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::fInitApplication
//
// Purpose:
//
//      Initializes the application
//
// Parameters:
//
//      HANDLE hInstance    -   Instance handle of the application.
//
// Return Value:
//
//      TRUE    -   Application was successfully initialized.
//      FALSE   -   Application could not be initialized
//
// Function Calls:
//      Function                    Location
//
//      LoadIcon                    Windows API
//      LoadCursor                  Windows API
//      GetStockObject              Windows API
//      RegisterClass               Windows API
//
// Comments:
//
//********************************************************************

BOOL CSimpleApp::fInitApplication(HANDLE hInstance)
{
        WNDCLASS  wc;

        // Fill in window class structure with parameters that describe the
        // main window.

        wc.style = NULL;                    // Class style(s).
        wc.lpfnWndProc = MainWndProc;       // Function to retrieve messages
for

// windows of this class.
        wc.cbClsExtra = 0;                  // No per-class extra data.
        wc.cbWndExtra = 0;                  // No per-window extra data.
        wc.hInstance = hInstance;           // Application that owns the
class.
        wc.hIcon = LoadIcon(hInstance, "SimpCntr");
        wc.hCursor = LoadCursor(NULL, IDC_ARROW);
        wc.hbrBackground = GetStockObject(WHITE_BRUSH);
        wc.lpszMenuName =  "SIMPLEMENU";    // Name of menu resource in .RC
file.
        wc.lpszClassName = "SimpCntrAppWClass";  // Name used in
CreateWindow call
```

```
        if (!RegisterClass(&wc))
                return FALSE;

        wc.style = CS_DBLCLKS;              // Class style(s). allow
DBLCLK's
        wc.lpfnWndProc = DocWndProc;        // Function to retrieve messages
for

// windows of this class.
        wc.cbClsExtra = 0;                  // No per-class extra data.
        wc.cbWndExtra = 0;                  // No per-window extra data.
        wc.hInstance = hInstance;           // Application that owns the
class.
        wc.hIcon = NULL;
        wc.hCursor = LoadCursor(NULL, IDC_ARROW);
        wc.hbrBackground = GetStockObject(WHITE_BRUSH);
        wc.lpszMenuName =  NULL;
        wc.lpszClassName = "SimpCntrDocWClass"; // Name used in CreateWindow
call.

        // Register the window class and return success/failure code.

        return (RegisterClass(&wc));
}
```

CSimpleApp::fInitInstance   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::fInitInstance
//
// Purpose:
//
//      Instance initialization.
//
// Parameters:
//
//      HANDLE hInstance    -   App. Instance Handle.
//
//      int nCmdShow        -   Show parameter from WinMain
//
// Return Value:
//
//      TRUE    -   Initialization Successful
//      FALSE   -   Initialization Failed.
//
//
// Function Calls:
//      Function                    Location
//
//      CreateWindow                Windows API
//      ShowWindow                  Windows API
//      UpdateWindow                Windows API
//      OleBuildVersion             OLE API
//      OleInitialize               OLE API
//
// Comments:
//
//      Note that successful Initalization of the OLE libraries
//      is remembered so the UnInit is only called if needed.
//
//********************************************************************

BOOL CSimpleApp::fInitInstance (HANDLE hInstance, int nCmdShow)
{
        DWORD dwVer = OleBuildVersion();
        LPMALLOC lpMalloc = NULL;

        // check to see if we are compatible with this version of the
libraries
        if (HIWORD(dwVer) != rmm || LOWORD(dwVer) < rup) {
#ifdef _DEBUG
                OutputDebugString("WARNING: Incompatible OLE library
version\r\n");
#else
                return FALSE;
#endif
        }
```

```
#if defined( _DEBUG )
        /* OLE2NOTE: Use a special debug allocator to help track down
        **    memory leaks.
        */
        OleStdCreateDbAlloc(0, &lpMalloc);
#endif

        if (SUCCEEDED(OleInitialize(lpMalloc)))
        {
                m_fInitialized = TRUE;
        }
        else
        {
            // Replacing the standard allocator may not be legal.
            // Try again using the default allocator.
            if (SUCCEEDED(OleInitialize(NULL)))
            {
                m_fInitialized = TRUE;
            }
        }

#if defined( _DEBUG )
        /* OLE2NOTE: release the special debug allocator so that only OLE is
        **    holding on to it. later when OleUninitialize is called, then
        **    the debug allocator object will be destroyed. when the debug
        **    allocator object is destoyed, it will report (to the Output
        **    Debug Terminal) whether there are any memory leaks.
        */
        if (lpMalloc) lpMalloc->Release();
#endif

        m_hInst = hInstance;

        // Create the "application" windows
        m_hAppWnd = CreateWindow ("SimpCntrAppWClass",
                                                          "Simple OLE 2.0
In-Place Container",

WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
                                                          CW_USEDEFAULT,
                                                          CW_USEDEFAULT,
                                                          CW_USEDEFAULT,
                                                          CW_USEDEFAULT,
                                                          NULL,
                                                          NULL,
                                                          hInstance,
                                                          NULL);

        if (!m_hAppWnd)
                return FALSE;

        m_hStdPal = OleStdCreateStandardPalette();

        ShowWindow (m_hAppWnd, nCmdShow);
        UpdateWindow (m_hAppWnd);
        return m_fInitialized;
}
```

CSimpleApp::lCommandHandler (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::lCommandHandler
//
// Purpose:
//
//      Handles the processing of WM_COMMAND.
//
// Parameters:
//
//      HWND hWnd       -   Handle to the application Window
//
//      UINT message    -   message (always WM_COMMAND)
//
//      WPARAM wParam   -   Same as passed to the WndProc
//
//      LPARAM lParam   -   Same as passed to the WndProc
//
// Return Value:
//
//      NULL
//
// Function Calls:
//      Function                                    Location
//
//      IOleInPlaceActiveObject::QueryInterface     Object
//      IOleInPlaceObject::ContextSensitiveHelp     Object
//      IOleInPlaceObject::Release                  Object
//      IOleObject::DoVerb                          Object
//      GetClientRect                               Windows API
//      MessageBox                                  Windows API
//      DialogBox                                   Windows API
//      MakeProcInstance                            Windows API
//      FreeProcInstance                            Windows API
//      SendMessage                                 Windows API
//      DefWindowProc                               Windows API
//      CSimpleDoc::InsertObject                    DOC.CPP
//
// Comments:
//
//********************************************************************

long CSimpleApp::lCommandHandler (HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
        CStabilize stabilize(this);
        RECT rect;

        // context sensitive help...
        if (m_fMenuMode || m_fCSHMode)
                {
                if (m_fCSHMode)
```

```
                        {
                        // clear context sensitive help flag
                        m_fCSHMode = FALSE;

                        // if there is an InPlace active object, call its
context sensitive help
                        // method with the FALSE parameter to bring the
object out of the
                        // csh state.  See the technotes for details.
                        if (m_lpDoc->m_lpActiveObject)
                                {
                                LPOLEINPLACEOBJECT lpInPlaceObject;
                                m_lpDoc->m_lpActiveObject-
>QueryInterface(IID_IOleInPlaceObject, (LPVOID FAR *)&lpInPlaceObject);
                                lpInPlaceObject-
>ContextSensitiveHelp(FALSE);
                                lpInPlaceObject->Release();
                                }
                        }

                // see the technotes for details on implementing context
sensitive
                // help
                if (m_fMenuMode)
                        {
                        m_fMenuMode = FALSE;

                        if (m_lpDoc->m_lpActiveObject)
                                m_lpDoc->m_lpActiveObject-
>ContextSensitiveHelp(FALSE);
                        }
                // if we provided help, we would do it here...
                MessageBox (hWnd, "Help", "Help", MB_OK);

                return NULL;
                }

        // see if the command is a verb selections
        //@@WTK WIN32, UNICODE
        //if (wParam >= IDM_VERB0)
        if (LOWORD(wParam) >= IDM_VERB0)
                {
                // get the rectangle of the object
                m_lpDoc->m_lpSite->GetObjRect(&rect);

                //@@WTK WIN32, UNICODE
                //m_lpDoc->m_lpSite->m_lpOleObject->DoVerb(wParam -
IDM_VERB0, NULL, &m_lpDoc->m_lpSite->m_OleClientSite, -1, m_lpDoc-
>m_hDocWnd, &rect);
                m_lpDoc->m_lpSite->m_lpOleObject->DoVerb(LOWORD(wParam) -
IDM_VERB0, NULL,
                                &m_lpDoc->m_lpSite->m_OleClientSite, -1,
m_lpDoc->m_hDocWnd, &rect);
                }
        else
```

```
                {
                //@@WTK WIN32, UNICODE
                //switch (wParam) {
                switch (LOWORD(wParam)) {
                        // bring up the About box
                        case IDM_ABOUT:
                                {
                                FARPROC lpProcAbout =
MakeProcInstance((FARPROC)About, m_hInst);

                                DialogBox(m_hInst,               // current
instance
                                        "AboutBox",                  //
resource to use
                                        m_hAppWnd,                   //
parent handle
                                        lpProcAbout);                //
About() instance address

                                FreeProcInstance(lpProcAbout);
                                break;
                                }

                        // bring up the InsertObject Dialog
                        case IDM_INSERTOBJECT:
                                m_lpDoc->InsertObject();
                                break;

                        // exit the application
                        case IDM_EXIT:
                                SendMessage(hWnd, WM_SYSCOMMAND, SC_CLOSE,
0L);
                                break;

                        case IDM_NEW:
                                m_lpDoc->Close();
                                m_lpDoc = NULL;
                                lCreateDoc(hWnd, 0, 0, 0);
                                break;

                        default:
                                return (DefWindowProc(hWnd, message, wParam,
lParam));
                        }   // end of switch
                }  // end of else
        return NULL;
}
```

CSimpleApp::lSizeHandler   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::lSizeHandler
//
// Purpose:
//
//      Handles the WM_SIZE message
//
// Parameters:
//
//      HWND hWnd       -   Handle to the application Window
//
//      UINT message    -   message (always WM_SIZE)
//
//      WPARAM wParam   -   Same as passed to the WndProc
//
//      LPARAM lParam   -   Same as passed to the WndProc
//
// Return Value:
//
//      LONG    -   returned from the "document" resizing
//
// Function Calls:
//      Function                    Location
//
//      GetClientRect               Windows API
//      CSimpleDoc::lResizeDoc      DOC.CPP
//
// Comments:
//
//********************************************************************

long CSimpleApp::lSizeHandler (HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
        CStabilize stabilize(this);
        RECT rect;

        GetClientRect(m_hAppWnd, &rect);
        return m_lpDoc->lResizeDoc(&rect);
}
```

CSimpleApp::lCreateDoc   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::lCreateDoc
//
// Purpose:
//
//      Handles the creation of a document.
//
// Parameters:
//
//      HWND hWnd        -   Handle to the application Window
//
//      UINT message     -   message (always WM_CREATE)
//
//      WPARAM wParam    -   Same as passed to the WndProc
//
//      LPARAM lParam    -   Same as passed to the WndProc
//
// Return Value:
//
//      NULL
//
// Function Calls:
//      Function                     Location
//
//      GetClientRect                Windows API
//      CSimpleDoc::CSimpleDoc       DOC.CPP
//
// Comments:
//
//********************************************************************

long CSimpleApp::lCreateDoc (HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
        CStabilize stabilize(this);
        RECT rect;

        GetClientRect(hWnd, &rect);

        m_lpDoc = CSimpleDoc::Create(this, &rect, hWnd);

        return NULL;
}
```

CSimpleApp::AddFrameLevelUI (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::AddFrameLevelUI
//
// Purpose:
//
//      Used during InPlace negotiation.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                            Location
//
//      COleInPlaceFrame::SetMenu           IOIPF.CPP
//      CSimpleApp::AddFrameLevelTools      APP.CPP
//
// Comments:
//
//      Be sure to read the Technotes included in the OLE 2.0 toolkit
//
//********************************************************************

void CSimpleApp::AddFrameLevelUI()
{
        CStabilize stabilize(this);
        m_OleInPlaceFrame.SetMenu(NULL, NULL, NULL);
        AddFrameLevelTools();
}
```

CSimpleApp::AddFrameLevelTools (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::AddFrameLevelTools
//
// Purpose:
//
//      Used during InPlace negotiation.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                              Location
//
//      COleInPlaceFrame::SetBorderSpace      IOIPF.CPP
//      InvalidateRect                        Windows API
//
// Comments:
//
//      Be sure to read the Technotes included in the OLE 2.0 toolkit
//
//********************************************************************

void CSimpleApp::AddFrameLevelTools()
{
        CStabilize stabilize(this);
        m_OleInPlaceFrame.SetBorderSpace(&nullRect);
        InvalidateRect(m_hAppWnd, NULL, TRUE);
}
```

CSimpleApp::HandleAccelerators (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::HandleAccelerators
//
// Purpose:
//
//      To properly handle accelerators in the Message Loop
//
// Parameters:
//
//      LPMSG lpMsg -   A pointer to the message structure.
//
// Return Value:
//
//      TRUE    -   The accelerator was handled
//      FALSE   -   The accelerator was not handled
//
// Function Calls:
//      Function                                        Location
//
//      IOleInPlaceActiveObject::TranslateAccelerator   Object
//
// Comments:
//
//      If an object is InPlace active, it gets the first shot at
//      handling the accelerators.
//
//********************************************************************

BOOL CSimpleApp::HandleAccelerators(LPMSG lpMsg)
{
        CStabilize stabilize(this);
        HRESULT hResult;
        BOOL retval = FALSE;

        // if we have an InPlace Active Object
        if (m_lpDoc->m_lpActiveObject)
                {
                // Pass the accelerator on...
                hResult = m_lpDoc->m_lpActiveObject-
>TranslateAccelerator(lpMsg);
                if (hResult == NOERROR)
                        retval = TRUE;
                }

        return retval;
}
```

CSimpleApp::PaintApp   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::PaintApp
//
// Purpose:
//
//      Handles the painting of the doc window.
//
//
// Parameters:
//
//      HDC hDC -   hDC to the Doc Window.
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      CSimpleDoc::PaintDoc        DOC.CPP
//
// Comments:
//
//      This is an app level function in case we want to do palette
//      management.
//
//********************************************************************

void CSimpleApp::PaintApp (HDC hDC)
{
        CStabilize stabilize(this);
        // at this level, we could enumerate through all of the
        // visible objects in the application, so that a palette
        // that best fits all of the objects can be built.

        // This app is designed to take on the same palette
        // functionality that was provided in OLE 1.0, the palette
        // of the last object drawn is realized.  Since we only
        // support one object at a time, it shouldn't be a big
        // deal.

        // if we supported multiple documents, we would enumerate
        // through each of the open documents and call paint.

        if (m_lpDoc)
                m_lpDoc->PaintDoc(hDC);

}
```

CSimpleApp::ContextSensitiveHelp (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::ContextSensitiveHelp
//
// Purpose:
//      Used in supporting context sensitive haelp at the app level.
//
//
// Parameters:
//
//      BOOL fEnterMode    -   Entering/Exiting Context Sensitive
//                             help mode.
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                                    Location
//
//      IOleInPlaceActiveObject::QueryInterface     Object
//      IOleInPlaceObject::ContextSensitiveHelp     Object
//      IOleInPlaceObject::Release                  Object
//
// Comments:
//
//      This function isn't used because we don't support Shift+F1
//      context sensitive help.  Be sure to look at the technotes
//      in the OLE 2.0 toolkit.
//
//********************************************************************

void CSimpleApp::ContextSensitiveHelp (BOOL fEnterMode)
{
        CStabilize stabilize(this);
        if (m_fCSHMode != fEnterMode)
                {
                m_fCSHMode = fEnterMode;

                // this code "trickles" the context sensitive help via 
shift+f1
                // to the inplace active object.  See the technotes for 
implementation
                // details.
                if (m_lpDoc->m_lpActiveObject)
                        {
                        LPOLEINPLACEOBJECT lpInPlaceObject;
                        m_lpDoc->m_lpActiveObject-
>QueryInterface(IID_IOleInPlaceObject, (LPVOID FAR *)&lpInPlaceObject);
                        lpInPlaceObject->ContextSensitiveHelp(fEnterMode);
                        lpInPlaceObject->Release();
                        }
                }
}


/* OLE2NOTE: forward the WM_QUERYNEWPALETTE message (via
**    SendMessage) to UIActive in-place object if there is one.
**    this gives the UIActive object the opportunity to select
**    and realize its color palette as the FOREGROUND palette.
**    this is optional for in-place containers. if a container
**    prefers to force its color palette as the foreground
**    palette then it should NOT forward the this message. or
**    the container can give the UIActive object priority; if
**    the UIActive object returns 0 from the WM_QUERYNEWPALETTE
**    message (ie. it did not realize its own palette), then
**    the container can realize its palette.
**    (see ContainerDoc_ForwardPaletteChangedMsg for more info)
**
**    (It is a good idea for containers to use the standard
**    palette even if they do not use colors themselves. this
**    will allow embedded object to get a good distribution of
**    colors when they are being drawn by the container)
**
*/

LRESULT CSimpleApp::QueryNewPalette(void)
{
        if (m_hwndUIActiveObj) {
                if (SendMessage(m_hwndUIActiveObj, WM_QUERYNEWPALETTE,
                                (WPARAM)0, (LPARAM)0)) {
                        /* Object selected its palette as foreground palette
*/
                        return (LRESULT)1;
                }
        }

        return wSelectPalette(m_hAppWnd, m_hStdPal, FALSE/*fBackground*/);
}


/* This is just a helper routine */

LRESULT wSelectPalette(HWND hWnd, HPALETTE hPal, BOOL fBackground)
{
        HDC hdc;
        HPALETTE hOldPal;
        UINT iPalChg = 0;

        if (hPal == 0)
                return (LRESULT)0;

        hdc = GetDC(hWnd);
        hOldPal = SelectPalette(hdc, hPal, fBackground);
        iPalChg = RealizePalette(hdc);
        SelectPalette(hdc, hOldPal, TRUE /*fBackground*/);
        ReleaseDC(hWnd, hdc);
        if (iPalChg > 0)
                InvalidateRect(hWnd, NULL, TRUE);

        return (LRESULT)1;
}
```

DOC.CPP   (SIMPCNTR Sample)

```
//***********************************************************************
// File name: DOC.CPP
//
//      Implementation file for CSimpleDoc.
//
// Functions:
//
//      See DOC.H for Class Definition
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//***********************************************************************

#include "pre.h"
#include "iocs.h"
#include "ias.h"
#include "ioipf.h"
#include "ioips.h"
#include "app.h"
#include "site.h"
#include "doc.h"
```

CSimpleDoc::Create   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleDoc::Create
//
// Purpose:
//
//      Creation for the CSimpleDoc Class
//
// Parameters:
//
//      CSimpleApp FAR * lpApp  -   Pointer to the CSimpleApp Class
//
//      LPRECT lpRect           -   Client area rect of "frame" window
//
//      HWND hWnd               -   Window Handle of "frame" window
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      StgCreateDocfile            OLE API
//      CreateWindow                Windows API
//      ShowWindow                  Windows API
//      UpdateWindow                Windows API
//
// Comments:
//
//      This routine was added so that failure could be returned
//      from object creation.
//
//********************************************************************

CSimpleDoc FAR * CSimpleDoc::Create(CSimpleApp FAR *lpApp, LPRECT
lpRect,HWND hWnd)
{
        CSimpleDoc FAR * lpTemp = new CSimpleDoc(lpApp, hWnd);

        if (!lpTemp)
                return NULL;

        // create storage for the doc.
        HRESULT hErr = StgCreateDocfile (NULL, STGM_READWRITE |
STGM_TRANSACTED | STGM_SHARE_EXCLUSIVE,
                                                                         0,
&lpTemp->m_lpStorage);

        if (hErr != NOERROR)
                goto error;
        // create the document Window
        lpTemp->m_hDocWnd = CreateWindow(
                        "SimpCntrDocWClass",
                        NULL,
                        WS_CHILD | WS_CLIPCHILDREN,
                        lpRect->left,
                        lpRect->top,
                        lpRect->right,
                        lpRect->bottom,
                        hWnd,
                        NULL,
                        lpApp->m_hInst,
                        NULL);

        if (!lpTemp->m_hDocWnd)
                goto error;

        ShowWindow(lpTemp->m_hDocWnd, SW_SHOWNORMAL);  // Show the window
        UpdateWindow(lpTemp->m_hDocWnd);               // Sends WM_PAINT
message

        // Ensable InsertObject menu choice
        EnableMenuItem( lpApp->m_hEditMenu, 0, MF_BYPOSITION | MF_ENABLED);

        // we will add one ref count on our document. later in
CSimpleDoc::Close
        // we will release this  ref count. when the document's ref count
goes
        // to 0, the document will be deleted.
        lpTemp->AddRef();

        return (lpTemp);

error:
        delete (lpTemp);
        return NULL;

}
```

CSimpleDoc::Close (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleDoc::Close
//
// Purpose:
//
//      Close CSimpleDoc object.
//      when the document's reference count goes to 0, the document
//      will be destroyed.
//
// Parameters:
//
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      RevokeDragDrop              OLE API
//      CoLockObjectExternal        OLE API
//      OleFlushClipboard           OLE API
//      ShowWindow                  Windows API
//
// Comments:
//
//********************************************************************

void CSimpleDoc::Close(void)
{
        OutputDebugString("In CSimpleDoc::Close\r\n");

        ShowWindow(m_hDocWnd, SW_HIDE);  // Hide the window

        // Close the OLE object in our document
        if (m_lpSite)
                m_lpSite->CloseOleObject();

        // Release the ref count added in CSimpleDoc::Create. this will make
        // the document's ref count go to 0, and the document will be
deleted.
        Release();
}
```

CSimpleDoc::CSimpleDoc   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleDoc::CSimpleDoc
//
// Purpose:
//
//      Constructor for the CSimpleDoc Class
//
// Parameters:
//
//      CSimpleApp FAR * lpApp  -   Pointer to the CSimpleApp Class
//
//      HWND hWnd               -   Window Handle of "frame" window
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      GetMenu                     Windows API
//      GetSubMenu                  Windows API
//
// Comments:
//
//********************************************************************

CSimpleDoc::CSimpleDoc(CSimpleApp FAR * lpApp,HWND hWnd)
{
        OutputDebugString("In CSimpleDoc's Constructor\r\n");
        m_lpApp = lpApp;
        m_lpSite = NULL;
        // set up menu handles
        m_lpApp->m_hMainMenu = GetMenu(hWnd);
        m_lpApp->m_hFileMenu = GetSubMenu(m_lpApp->m_hMainMenu, 0);
        m_lpApp->m_hEditMenu = GetSubMenu(m_lpApp->m_hMainMenu, 1);
        m_lpApp->m_hHelpMenu = GetSubMenu(m_lpApp->m_hMainMenu, 2);
        m_lpApp->m_hCascadeMenu = NULL;

        m_lpActiveObject = NULL;

        // flags
        m_fInPlaceActive = FALSE;
        m_fAddMyUI = FALSE;
        m_fModifiedMenu = FALSE;
}
```

CSimpleDoc::~CSimpleDoc (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleDoc::~CSimpleDoc
//
// Purpose:
//
//      Destructor for CSimpleDoc
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpleSite::Release        SITE.CPP
//      IStorage::Release           OLE API
//
// Comments:
//
//********************************************************************

CSimpleDoc::~CSimpleDoc()
{
        OutputDebugString("In CSimpleDoc's Destructor\r\n");

        // Release all pointers we hold to the OLE object. also release
        // the ref count added in CSimpleSite::Create. this will make
        // the Site's ref count go to 0, and the Site will be deleted.
        if (m_lpSite) {
                m_lpSite->UnloadOleObject();
                m_lpSite->Release();
                m_lpSite = NULL;
        }

        // Release the Storage
        if (m_lpStorage) {
                m_lpStorage->Release();
                m_lpStorage = NULL;
        }

        // if the edit menu was modified, remove the menu item and
        // destroy the popup if it exists
        if (m_fModifiedMenu)
                {
                int nCount = GetMenuItemCount(m_lpApp->m_hEditMenu);
                RemoveMenu(m_lpApp->m_hEditMenu, nCount-1, MF_BYPOSITION);
                if (m_lpApp->m_hCascadeMenu)
                        DestroyMenu(m_lpApp->m_hCascadeMenu);
                }

        DestroyWindow(m_hDocWnd);
}
```

CSimpleDoc::QueryInterface (SIMPCNTR Sample)

```
//***********************************************************************
//
// CSimpleDoc::QueryInterface
//
// Purpose:
//
//      Return a pointer to a requested interface
//
// Parameters:
//
//      REFIID riid         -   ID of interface to be returned
//      LPVOID FAR* ppvObj  -   Location to return the interface
//
// Return Value:
//
//      S_FALSE -   Always
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//      In this implementation, there are no doc level interfaces.
//      In an MDI application, there would be an IOleInPlaceUIWindow
//      associated with the document to provide document level tool
 //     space negotiation.
//
//*********************************************************************

STDMETHODIMP CSimpleDoc::QueryInterface(REFIID riid, LPVOID FAR* ppvObj)
{
        OutputDebugString("In CSimpleDoc::QueryInterface\r\n");

        *ppvObj = NULL;     // must set out pointer parameters to NULL

        // Not a supported interface
        return ResultFromScode(E_NOINTERFACE);
}
```

CSimpleDoc::AddRef   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleDoc::AddRef
//
// Purpose:
//
//      Increments the document reference count
//
// Parameters:
//
//      None
//
// Return Value:
//
//      UINT    -   The current reference count on the document
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpleApp::AddRef          APP.CPP
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpleDoc::AddRef()
{
        OutputDebugString("In CSimpleDoc::AddRef\r\n");
        return SafeAddRef();
}
```

CSimpleDoc::Release   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleDoc::Release
//
// Purpose:
//
//      Decrements the document reference count
//
// Parameters:
//
//      None
//
// Return Value:
//
//      UINT    -   The current reference count on the document
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpleDoc::Release()
{
        OutputDebugString("In CSimpleDoc::Release\r\n");

        return SafeRelease();
}
```

CSimpleDoc::InsertObject   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleDoc::InsertObject
//
// Purpose:
//
//      Inserts a new object to this document
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      CSimpleSite::CSimpleSite    SITE.CPP
//      CSimpleSite::InitObject     SITE.CPP
//      memset                      C Runtime
//      OleUIInsertObject           OUTLUI function
//      CSimpleDoc::DisableInsertObject DOC.CPP
//
// Comments:
//
//      This implementation only allows one object to be inserted
//      into a document.  Once the object has been inserted, then
//      the Insert Object menu choice is greyed out, to prevent
//      the user from inserting another.
//
//********************************************************************

void CSimpleDoc::InsertObject()
{
        CStabilize stabilize(this);
        OLEUIINSERTOBJECT io;
        UINT iret;
  //@@WTK WIN32, UNICODE
        //char szFile[OLEUI_CCHPATHMAX];
        OLECHAR szFile[OLEUI_CCHPATHMAX];

        m_lpSite = CSimpleSite::Create(this);

        // clear the structure
        _fmemset(&io, 0, sizeof(OLEUIINSERTOBJECT));

        // fill the structure
        io.cbStruct = sizeof(OLEUIINSERTOBJECT);
        io.dwFlags = IOF_SELECTCREATENEW |
                                        IOF_DISABLELINK |
IOF_DISABLEDISPLAYASICON |
                                        IOF_CREATENEWOBJECT |
IOF_CREATEFILEOBJECT;
        io.hWndOwner = m_hDocWnd;
  //@@WTK WIN32, UNICODE
        //io.lpszCaption = (LPSTR)"Insert Object";
        io.lpszCaption = OLESTR("Insert Object");
        io.iid = IID_IOleObject;
        io.oleRender = OLERENDER_DRAW;
        io.lpIOleClientSite = &m_lpSite->m_OleClientSite;
        io.lpIStorage = m_lpSite->m_lpObjStorage;
        io.ppvObj = (LPVOID FAR *)&m_lpSite->m_lpOleObject;
        io.lpszFile = szFile;
  //@@WTK WIN32, UNICODE
        //io.cchFile = sizeof(szFile);
        io.cchFile = sizeof(szFile) / sizeof *szFile;
  //@@WTK WIN32, UNICODE
        //_fmemset((LPSTR)szFile, 0, sizeof(szFile));
        _fmemset(szFile, 0, sizeof(szFile));

        // call OUTLUI to do all the hard work
        iret = OleUIInsertObject(&io);

        if (iret == OLEUI_OK)
                {
                m_lpSite->InitObject((BOOL)(io.dwFlags &
IOF_SELECTCREATENEW));
                // disable Insert Object menu item
                DisableInsertObject();
                }
        else
                {
                m_lpSite->Release();
                m_lpSite = NULL;
                m_lpStorage->Revert();
                }

}
```

CSimpleDoc::lResizeDoc (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleDoc::lResizeDoc
//
// Purpose:
//
//      Resizes the document
//
// Parameters:
//
//      LPRECT lpRect   -   The size of the client are of the "frame"
//                          Window.
//
// Return Value:
//
//      NULL
//
// Function Calls:
//      Function                                Location
//
//      IOleInPlaceActiveObject::ResizeBorder   Object
//      MoveWindow                              Windows API
//
// Comments:
//
//********************************************************************

long CSimpleDoc::lResizeDoc(LPRECT lpRect)
{
        CStabilize stabilize(this);
        // if we are InPlace, then call ResizeBorder on the object,
otherwise
        // just move the document window.
        if (m_fInPlaceActive)
                m_lpActiveObject->ResizeBorder(lpRect, &m_lpApp-
>m_OleInPlaceFrame, TRUE);
        else
                MoveWindow(m_hDocWnd, lpRect->left, lpRect->top, lpRect-
>right, lpRect->bottom, TRUE);

        return NULL;
}
```

CSimpleDoc::lAddVerbs   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleDoc::lAddVerbs
//
// Purpose:
//
//      Adds the objects verbs to the edit menu.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      NULL
//
// Function Calls:
//      Function                    Location
//
//      GetMenuItemCount            Windows API
//      OleUIAddVerbMenu            OUTLUI function
//
// Comments:
//
//********************************************************************

long CSimpleDoc::lAddVerbs(void)
{
        CStabilize stabilize(this);
        // m_fModifiedMenu is TRUE if the menu has already been modified
        // once.  Since we only support one obect every time the application
        // is run, then once the menu is modified, it doesn't have
        // to be done again.
        if (m_lpSite && !m_fInPlaceActive  && !m_fModifiedMenu)
                {
                int nCount = GetMenuItemCount(m_lpApp->m_hEditMenu);

                OleUIAddVerbMenu ( m_lpSite->m_lpOleObject,
                                                   NULL,
                                                   m_lpApp->m_hEditMenu,
                                                   nCount + 1,
                                                   IDM_VERB0,
                                                   0,           // no
maximum verb IDM enforced
                                                   FALSE,
                                                   0,
                                                   &m_lpApp-
>m_hCascadeMenu);

                m_fModifiedMenu = TRUE;
                }
        return (NULL);
}
```

CSimpleDoc::PaintDoc   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleDoc::PaintDoc
//
// Purpose:
//
//      Paints the Document
//
// Parameters:
//
//      HDC hDC -   hDC of the document Window
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      CSimpleSite::PaintObj       SITE.CPP
//
// Comments:
//
//********************************************************************

void CSimpleDoc::PaintDoc (HDC hDC)
{
        CStabilize stabilize(this);
        // if we supported multiple objects, then we would enumerate
        // the objects and call paint on each of them from here.

        if (m_lpSite)
                m_lpSite->PaintObj(hDC);

}
```

CSimpleDoc::DisableInsertObject   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleDoc::DisableInsertObject
//
// Purpose:
//
//      Disable the ability to insert a new object in this document.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      EnableMenuItem              Windows API
//
// Comments:
//
//      This implementation only allows one object to be inserted
//      into a document.  Once the object has been inserted, then
//      the Insert Object menu choice is greyed out, to prevent
//      the user from inserting another.
//
//********************************************************************

void CSimpleDoc::DisableInsertObject(void)
{
        // Disable InsertObject menu choice
        EnableMenuItem( m_lpApp->m_hEditMenu, 0, MF_BYPOSITION | MF_DISABLED
| MF_GRAYED);
}
```

IAS.CPP   (SIMPCNTR Sample)

```
//**********************************************************************
// File name: IAS.CPP
//
//      Implementation file of CAdviseSink
//
//
// Functions:
//
//      See IAS.H for Class Definition
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "iocs.h"
#include "ias.h"
#include "ioipf.h"
#include "ioips.h"
#include "app.h"
#include "site.h"
#include "doc.h"
```

CAdviseSink::QueryInterface   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CAdviseSink::QueryInterface
//
// Purpose:
//
//      Returns a pointer to a requested interface.
//
// Parameters:
//
//      REFIID riid         - The requested interface
//
//      LPVOID FAR* ppvObj  - Place to return the interface
//
// Return Value:
//
//      HRESULT from CSimpleSite::QueryInterface
//
// Function Calls:
//      Function                    Location
//
//      CSimpleSite::QueryInterface SITE.CPP
//      OutputDebugString           Windows API
//
// Comments:
//
//      This function simply delegates to the Object class, which is
//      aware of the supported interfaces.
//
//********************************************************************

STDMETHODIMP CAdviseSink::QueryInterface(REFIID riid, LPVOID FAR* ppvObj)
{
        OutputDebugString("In IAS::QueryInterface\r\n");

        // delegate to the document Object
        return m_pSite->QueryInterface(riid, ppvObj);
}
```

CAdviseSink::AddRef   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CAdviseSink::AddRef
//
// Purpose:
//
//      Increments the reference count on this interface
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The current reference count on this interface.
//
// Function Calls:
//      Function                    Location
//
//      CSimpleSite::AddReff        SITE.CPP
//      OutputDebugString           Windows API
//
// Comments:
//
//      This function adds one to the ref count of the interface,
//      and calls then calls CSimpleSite to increment its ref.
//      count.
//
//********************************************************************

STDMETHODIMP_(ULONG) CAdviseSink::AddRef()
{
        OutputDebugString("In IAS::AddRef\r\n");

        // increment the interface reference count (for debugging only)
        ++m_nCount;

        // delegate to the container Site
        return m_pSite->AddRef();
}
```

CAdviseSink::Release   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CAdviseSink::Release
//
// Purpose:
//
//      Decrements the reference count on this interface
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The current reference count on this interface.
//
// Function Calls:
//      Function                    Location
//
//      CSimpleSite::Release        SITE.CPP
//      OutputDebugString           Windows API
//
// Comments:
//
//      This function subtracts one from the ref count of the interface,
//      and calls then calls CSimpleSite to decrement its ref.
//      count.
//
//********************************************************************

STDMETHODIMP_(ULONG) CAdviseSink::Release()
{
        OutputDebugString("In IAS::Release\r\n");

        // decrement the interface reference count (for debugging only)
        m_nCount--;

        // delegate to the container Site
        return m_pSite->Release();
}
```

CAdviseSink::OnDataChange (SIMPCNTR Sample)

```
//**********************************************************************
//
// CAdviseSink::OnDataChange
//
// Purpose:
//
//      Not Implemented (needs to be stubbed out)
//
// Parameters:
//
//      Not Implemented (needs to be stubbed out)
//
// Return Value:
//
//      Not Implemented (needs to be stubbed out)
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Not Implemented (needs to be stubbed out)
//
//********************************************************************

STDMETHODIMP_(void) CAdviseSink::OnDataChange (FORMATETC FAR* pFormatetc,
STGMEDIUM FAR* pStgmed)
{
        OutputDebugString("In IAS::OnDataChange\r\n");
}
```

CAdviseSink::OnViewChange   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CAdviseSink::OnViewChange
//
// Purpose:
//
//      Notifies us that the view has changed and needs to be updated.
//
// Parameters:
//
//      DWORD dwAspect  - Aspect that has changed
//
//      LONG lindex     - Index that has changed
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      InvalidateRect              Windows API
//      IViewObject2::GetExtent     Object
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(void) CAdviseSink::OnViewChange (DWORD dwAspect, LONG lindex)
{
        CStabilize stabilize(m_pSite);
        LPVIEWOBJECT2 lpViewObject2;
        OutputDebugString("In IAS::OnViewChange\r\n");

        // get a pointer to IViewObject2
        HRESULT hErr = m_pSite->m_lpOleObject->QueryInterface(
                        IID_IViewObject2,(LPVOID FAR *)&lpViewObject2);

        if (hErr == NOERROR) {
                // get extent of the object
                // NOTE: this method will never be remoted; it can be called
w/i this async method
                lpViewObject2->GetExtent(DVASPECT_CONTENT, -1 , NULL,
&m_pSite->m_sizel);
                lpViewObject2->Release();
        }

        InvalidateRect(m_pSite->m_lpDoc->m_hDocWnd, NULL, TRUE);
}
```

CAdviseSink::OnRename   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CAdviseSink::OnRename
//
// Purpose:
//
//      Not Implemented (needs to be stubbed out)
//
// Parameters:
//
//      Not Implemented (needs to be stubbed out)
//
// Return Value:
//
//      Not Implemented (needs to be stubbed out)
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Not Implemented (needs to be stubbed out)
//
//********************************************************************

STDMETHODIMP_(void) CAdviseSink::OnRename (LPMONIKER pmk)
{
        OutputDebugString("In IAS::OnRename\r\n");
}
```

CAdviseSink::OnSave   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CAdviseSink::OnSave
//
// Purpose:
//
//      Not Implemented (needs to be stubbed out)
//
// Parameters:
//
//      Not Implemented (needs to be stubbed out)
//
// Return Value:
//
//      Not Implemented (needs to be stubbed out)
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Not Implemented (needs to be stubbed out)
//
//********************************************************************

STDMETHODIMP_(void) CAdviseSink::OnSave ()
{
        OutputDebugString("In IAS::OnSave\r\n");
}
```

CAdviseSink::OnClose   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CAdviseSink::OnClose
//
// Purpose:
//
//      Not Implemented (needs to be stubbed out)
//
// Parameters:
//
//      Not Implemented (needs to be stubbed out)
//
// Return Value:
//
//      Not Implemented (needs to be stubbed out)
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Not Implemented (needs to be stubbed out)
//
//********************************************************************

STDMETHODIMP_(void) CAdviseSink::OnClose()
{
        OutputDebugString("In IAS::OnClose\r\n");
}
```

IOCS.CPP   (SIMPCNTR Sample)

```
//**********************************************************************
// File name: IOCS.CPP
//
//      Implementation file for COleClientSite
//
// Functions:
//
//      See IOCS.H for class definition
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "iocs.h"
#include "ias.h"
#include "ioipf.h"
#include "ioips.h"
#include "app.h"
#include "site.h"
#include "doc.h"
```

COleClientSite::QueryInterface   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleClientSite::QueryInterface
//
// Purpose:
//
//      Used for interface negotiation at this interface
//
// Parameters:
//
//      REFIID riid         -   A reference to the interface that is
//                              being queried.
//
//      LPVOID FAR* ppvObj  -   An out parameter to return a pointer to
//                              the interface.
//
// Return Value:
//
//      S_OK                -   The interface is supported.
//      E_NOINTERFACE       -   The interface is not supported
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpleSite::QueryInterface SITE.CPP
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleClientSite::QueryInterface(REFIID riid, LPVOID FAR* ppvObj)
{
        OutputDebugString("In IOCS::QueryInterface\r\n");

        // delegate to the container Site
        return m_pSite->QueryInterface(riid, ppvObj);
}
```

CSimpleApp::AddRef (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::AddRef
//
// Purpose:
//
//      Adds to the reference count at the interface level.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the interface
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) COleClientSite::AddRef()
{
        OutputDebugString("In IOCS::AddRef\r\n");

        // increment the interface reference count (for debugging only)
        ++m_nCount;

        // delegate to the container Site
        return m_pSite->AddRef();
}
```

CSimpleApp::Release   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::Release
//
// Purpose:
//
//      Decrements the reference count at this level
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the interface.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) COleClientSite::Release()
{
        OutputDebugString("In IOCS::Release\r\n");

        // decrement the interface reference count (for debugging only)
        --m_nCount;

        // delegate to the container Site
        return m_pSite->Release();
}
```

COleClientSite::SaveObject (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleClientSite::SaveObject
//
// Purpose:
//
//      Called by the object when it wants to be saved to persistant
//      storage
//
// Parameters:
//
//      None
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                            Location
//
//      OutputDebugString                   Windows API
//      IOleObject::QueryInterface          Object
//      IPersistStorage::SaveCompleted      Object
//      IPersistStorage::Release            Object
//      OleSave                             OLE API
//      ResultFromScode                     OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleClientSite::SaveObject()
{
        CStabilize stabilize(m_pSite);
        LPPERSISTSTORAGE lpPS;
        SCODE sc = E_FAIL;

        OutputDebugString("In IOCS::SaveObject\r\n");

        // get a pointer to IPersistStorage
        HRESULT hErr = m_pSite->m_lpOleObject-
>QueryInterface(IID_IPersistStorage, (LPVOID FAR *)&lpPS);

        // save the object
        if (hErr == NOERROR)
                {
                sc = GetScode( OleSave(lpPS, m_pSite->m_lpObjStorage,
TRUE) );
                lpPS->SaveCompleted(NULL);
                lpPS->Release();
                }
        return ResultFromScode(sc);
}
```

COleClientSite::GetMoniker   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleClientSite::GetMoniker
//
// Purpose:
//
//      Not Implemented
//
// Parameters:
//
//      Not Implemented
//
// Return Value:
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      This function is not implemented because we don't support
//      linking.
//
//********************************************************************

STDMETHODIMP COleClientSite::GetMoniker(DWORD dwAssign, DWORD
dwWhichMoniker, LPMONIKER FAR* ppmk)
{
        OutputDebugString("In IOCS::GetMoniker\r\n");

        // need to null the out pointer
        *ppmk = NULL;

        return ResultFromScode(E_NOTIMPL);
}
```

COleClientSite::GetContainer   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleClientSite::GetContainer
//
// Purpose:
//
//      Not Implemented
//
// Parameters:
//
//      Not Implemented
//
// Return Value:
//
//      Not Implemented
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Not Implemented
//
//********************************************************************

STDMETHODIMP COleClientSite::GetContainer(LPOLECONTAINER FAR* ppContainer)
{
        OutputDebugString("In IOCS::GetContainer\r\n");

        // NULL the out pointer
        *ppContainer = NULL;

        return ResultFromScode(E_NOTIMPL);
}
```

COleClientSite::ShowObject (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleClientSite::ShowObject
//
// Purpose:
//
//      Not Implemented
//
// Parameters:
//
//      Not Implemented
//
// Return Value:
//
//      Not Implemented
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      This function is not implemented because we don't support
//      linking.
//
//********************************************************************

STDMETHODIMP COleClientSite::ShowObject()
{
        OutputDebugString("In IOCS::ShowObject\r\n");
        return NOERROR;
}
```

COleClientSite::OnShowWindow   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleClientSite::OnShowWindow
//
// Purpose:
//
//      Object calls this method when it is opening/closing non-InPlace
//      Window
//
// Parameters:
//
//      BOOL fShow  - TRUE if Window is opening, FALSE if closing
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      InvalidateRect              Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleClientSite::OnShowWindow(BOOL fShow)
{
        CStabilize stabilize(m_pSite);
        OutputDebugString("In IOCS::OnShowWindow\r\n");
        m_pSite->m_fObjectOpen = fShow;
        InvalidateRect(m_pSite->m_lpDoc->m_hDocWnd, NULL, TRUE);

        // if object window is closing, then bring container window to top
        if (! fShow) {
                BringWindowToTop(m_pSite->m_lpDoc->m_hDocWnd);
                SetFocus(m_pSite->m_lpDoc->m_hDocWnd);
        }
        return ResultFromScode(S_OK);
}
```

COleClientSite::RequestNewObjectLayout   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleClientSite::RequestNewObjectLayout
//
// Purpose:
//
//      Not Implemented
//
// Parameters:
//
//      Not Implemented
//
// Return Value:
//
//      Not Implemented
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Not Implemented
//
//********************************************************************

STDMETHODIMP COleClientSite::RequestNewObjectLayout()
{
        OutputDebugString("In IOCS::RequestNewObjectLayout\r\n");
        return ResultFromScode(E_NOTIMPL);
}
```

IOIPF.CPP (SIMPCNTR Sample)

```
//**********************************************************************
// File name: IOIPF.CPP
//
//      Implementation file for COleInPlaceFrame
//
// Functions:
//
//      See IOIPF.H for class definition
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "iocs.h"
#include "ias.h"
#include "ioipf.h"
#include "ioips.h"
#include "app.h"
#include "site.h"
#include "doc.h"
```

CSimpleApp::QueryInterface   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::QueryInterface
//
// Purpose:
//
//      Used for interface negotiation at the Interface level.
//
// Parameters:
//
//      REFIID riid         -   A reference to the interface that is
//                              being queried.
//
//      LPVOID FAR* ppvObj  -   An out parameter to return a pointer to
//                              the interface.
//
// Return Value:
//
//      S_OK    -   The interface is supported.
//      S_FALSE -   The interface is not supported
//
// Function Calls:
//      Function                      Location
//
//      OutputDebugString             Windows API
//      CSimpleApp::QueryInterface  APP.CPP
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleInPlaceFrame::QueryInterface(REFIID riid, LPVOID FAR*
ppvObj)
{
        OutputDebugString("In IOIPF::QueryInterface\r\n");

// delegate to the document Object
        return m_pApp->QueryInterface(riid, ppvObj);
}
```

CSimpleApp::AddRef   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::AddRef
//
// Purpose:
//
//      Adds to the reference count at the interface level.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the interface.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) COleInPlaceFrame::AddRef()
{
        OutputDebugString("In IOIPF::AddRef\r\n");

// delegate to the document Object
        m_pApp->AddRef();
// increment the interface reference count
        return ++m_nCount;
}
```

CSimpleApp::Release (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleApp::Release
//
// Purpose:
//
//      Decrements the reference count at this level
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the interface.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpleApp::Release         APP.CPP
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) COleInPlaceFrame::Release()
{
        OutputDebugString("In IOIPF::Release\r\n");

// delegate to the document object
        m_pApp->Release();

// decrement the interface reference count
        return --m_nCount;
}
```

COleInPlaceFrame::GetWindow (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceFrame::GetWindow
//
// Purpose:
//
//      Returns the frame window handle
//
// Parameters:
//
//      HWND FAR* lphwnd    - Location to return the window handle
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleInPlaceFrame::GetWindow (HWND FAR* lphwnd)
{
        OutputDebugString("In IOIPF::GetWindow\r\n");
        CStabilize stabilize(m_pApp);
        *lphwnd = m_pApp->m_hAppWnd;
        return ResultFromScode(S_OK);
}
```

COleInPlaceFrame::ContextSensitiveHelp (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceFrame::ContextSensitiveHelp
//
// Purpose:
//
//      Used in implementing Context sensitive help
//
// Parameters:
//
//      BOOL fEnterMode -   TRUE if starting Context Sensitive help mode
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//      Be sure to read the technotes in the OLE toolkit.
//
//********************************************************************

STDMETHODIMP COleInPlaceFrame::ContextSensitiveHelp (BOOL fEnterMode)
{
        OutputDebugString("In IOIPF::ContextSensitiveHelp\r\n");
        CStabilize stabilize(m_pApp);

        m_pApp->m_fMenuMode = fEnterMode;

        return ResultFromScode(S_OK);
```

COleInPlaceFrame::GetBorder   (SIMPCNTR Sample)}

```
//**********************************************************************
//
// COleInPlaceFrame::GetBorder
//
// Purpose:
//
//      Returns the outermost border that frame adornments can be attached
//      during InPlace Activation.
//
// Parameters:
//
//      LPRECT lprectBorder - return parameter to contain the outermost
//                            rect for frame adornments
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      GetClientRect               Windows API
//      CopyRect                    Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleInPlaceFrame::GetBorder (LPRECT lprectBorder)
{
        RECT rect;

        OutputDebugString("In IOIPF::GetBorder\r\n");

        // get the rect for the entire frame.
        GetClientRect(m_pApp->m_hAppWnd, &rect);

        CopyRect(lprectBorder, &rect);

        return ResultFromScode(S_OK);
}
```

COleInPlaceFrame::RequestBorderSpace (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceFrame::RequestBorderSpace
//
// Purpose:
//
//      Approves/Denies requests for border space during InPlace
//      negotiation.
//
// Parameters:
//
//      LPCBORDERWIDTHS lpborderwidths  - The width in pixels needed on
//                                        each side of the frame.
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//      This implementation doesn't care about how much border space
//      is used.  It always returns S_OK.
//
//********************************************************************

STDMETHODIMP COleInPlaceFrame::RequestBorderSpace (LPCBORDERWIDTHS
lpborderwidths)
{
        OutputDebugString("In IOIPF::RequestBorderSpace\r\n");

        // always approve the request
        return ResultFromScode(S_OK);
}
```

COleInPlaceFrame::SetBorderSpace   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceFrame::SetBorderSpace
//
// Purpose:
//
//      The object calls this method when it is actually going to
//      start using the border space.
//
// Parameters:
//
//      LPCBORDERWIDTHS lpborderwidths  - Border space actually being used
//                                        by the object
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                         Location
//
//      CSimpleApp::AddFrameLevelTools  APP.CPP
//      OutputDebugString                Windows API
//      GetClientRect                    Windows API
//      MoveWindow                       Windows API
//      ResultFromScode                  Windows API
//
// Comments:
//
//      This routine could be a little smarter and check to see if
//      the object is requesting the entire client area of the
//      window.
//
//********************************************************************

STDMETHODIMP COleInPlaceFrame::SetBorderSpace (LPCBORDERWIDTHS
lpborderwidths)
{

        OutputDebugString("In IOIPF::SetBorderSpace\r\n");
        CStabilize stabilize(m_pApp);

        if (lpborderwidths == NULL)
                m_pApp->AddFrameLevelTools();
        else
                {
                RECT rect;

                GetClientRect(m_pApp->m_hAppWnd, &rect);

                MoveWindow( m_pApp->m_lpDoc->m_hDocWnd,
                                        rect.left + lpborderwidths->left,
                                        rect.top + lpborderwidths->top,
                                        rect.right - lpborderwidths->right -
lpborderwidths->left,
                                        rect.bottom - lpborderwidths->bottom
- lpborderwidths->top,
                                        TRUE);
                }
        return ResultFromScode(S_OK);
}
```

COleInPlaceFrame::SetActiveObject   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceFrame::SetActiveObject
//
// Purpose:
//
//
// Parameters:
//
//      LPOLEINPLACEACTIVEOBJECT lpActiveObject     -   Pointer to the
//                                                      objects
//
IOleInPlaceActiveObject
//                                                      interface
//
//@@WTK WIN32, UNICODE
//      //LPCSTR lpszObjName                          -   Name of the object
//      LPCOLESTR lpszObjName                          -   Name of the
object
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                            Location
//
//      OutputDebugString                   Windows API
//      IOleInPlaceActiveObject::AddRef     Object
//      IOleInPlaceActiveObject::Release    Object
//      ResultFromScode                     OLE API
//
// Comments:
//
//********************************************************************

//@@WTK WIN32, UNICODE
//STDMETHODIMP COleInPlaceFrame::SetActiveObject (LPOLEINPLACEACTIVEOBJECT
lpActiveObject,LPCSTR lpszObjName)
STDMETHODIMP COleInPlaceFrame::SetActiveObject (
LPOLEINPLACEACTIVEOBJECT lpActiveObject,
LPCOLESTR lpszObjName)
{

        OutputDebugString("In IOIPF::SetActiveObject\r\n");
        CStabilize stabilize(m_pApp);

        // AddRef() it and save it...
        if (lpActiveObject)
                {
                lpActiveObject->AddRef();
                lpActiveObject->GetWindow(&m_pApp->m_hwndUIActiveObj);
                if (m_pApp->m_hwndUIActiveObj)
                        SendMessage(m_pApp->m_hwndUIActiveObj,
WM_QUERYNEWPALETTE, 0, 0L);
                }
        else
                {
                if (m_pApp->m_lpDoc->m_lpActiveObject)
                        m_pApp->m_lpDoc->m_lpActiveObject->Release();
                m_pApp->m_hwndUIActiveObj = NULL;
                }

        // in an MDI app, this method really shouldn't be called,
        // this method associated with the doc is called instead.

        m_pApp->m_lpDoc->m_lpActiveObject = lpActiveObject;
        // should set window title here

        return ResultFromScode(S_OK);
}
```

COleInPlaceFrame::InsertMenus (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceFrame::InsertMenus
//
// Purpose:
//
//      Inserts the container menu into the combined menu
//
// Parameters:
//
//      HMENU hmenuShared                   -   Menu Handle to be set.
//      LPOLEMENUGROUPWIDTHS lpMenuWidths   -   Width of menus
//
// Return Value:
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      AppendMenu                  Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleInPlaceFrame::InsertMenus (HMENU hmenuShared,
LPOLEMENUGROUPWIDTHS lpMenuWidths)
{
        OutputDebugString("In IOIPF::InsertMenus\r\n");
        CStabilize stabilize(m_pApp);

  //@@WTK WIN32, UNICODE
        //AppendMenu(hmenuShared, MF_BYPOSITION | MF_POPUP, m_pApp-
>m_hFileMenu, "&File");
        //AppendMenu(hmenuShared, MF_BYPOSITION | MF_POPUP, m_pApp-
>m_hHelpMenu, "&Other");
        AppendMenu(hmenuShared, MF_BYPOSITION | MF_POPUP, (UINT)m_pApp-
>m_hFileMenu, "&File");
        AppendMenu(hmenuShared, MF_BYPOSITION | MF_POPUP, (UINT)m_pApp-
>m_hHelpMenu, "&Other");

        lpMenuWidths->width[0] = 1;
        lpMenuWidths->width[2] = 0;
        lpMenuWidths->width[4] = 1;

        return ResultFromScode(S_OK);
}
```

COleInPlaceFrame::SetMenu (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceFrame::SetMenu
//
// Purpose:
//
//      Sets the application menu to the combined menu
//
// Parameters:
//
//      HMENU hmenuShared       - The combined menu
//
//      HOLEMENU holemenu       - Used by OLE
//
//      HWND hwndActiveObject   - Used by OLE
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      SetMenu                     Windows API
//      OleSetMenuDescriptor        OLE API
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleInPlaceFrame::SetMenu (HMENU hmenuShared, HOLEMENU
holemenu, HWND hwndActiveObject)
{

        OutputDebugString("In IOIPF::SetMenu\r\n");
        CStabilize stabilize(m_pApp);

        HMENU hMenu = m_pApp->m_hMainMenu;

        if (holemenu)
                hMenu = hmenuShared;

        // call the windows api, not this method
        ::SetMenu (m_pApp->m_hAppWnd, hMenu);

        OleSetMenuDescriptor(holemenu, m_pApp->m_hAppWnd, hwndActiveObject,
this, m_pApp->m_lpDoc->m_lpActiveObject);

        return ResultFromScode(S_OK);
}
```

COleInPlaceFrame::RemoveMenus   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceFrame::RemoveMenus
//
// Purpose:
//
//      Removes the container menus from the combined menu
//
// Parameters:
//
//      HMENU hmenuShared   - Handle to the combined menu.
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      GetMenuItemCount            Windows API
//      RemoveMenu                  Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleInPlaceFrame::RemoveMenus (HMENU hmenuShared)
{
        int retval;

        OutputDebugString("In IOIPF::RemoveMenus\r\n");

        while ((retval = GetMenuItemCount(hmenuShared)) && (retval != -1))
                RemoveMenu(hmenuShared, 0, MF_BYPOSITION);

        return ResultFromScode(S_OK);
}
```

COleInPlaceFrame::SetStatusText   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceFrame::SetStatusText
//
// Purpose:
//
//      Not Implemented
//
// Parameters:
//
//      Not Implemented
//
// Return Value:
//
//      Not Implemented
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      This function is not implemented due to the fact
//      that this application does not have a status bar.
//
//********************************************************************

//@@WTK WIN32, UNICODE
//STDMETHODIMP COleInPlaceFrame::SetStatusText (LPCSTR lpszStatusText)
STDMETHODIMP COleInPlaceFrame::SetStatusText (LPCOLESTR lpszStatusText)
{
        OutputDebugString("In IOIPF::SetStatusText\r\n");
        return ResultFromScode(E_FAIL);
}
```

COleInPlaceFrame::EnableModeless   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceFrame::EnableModeless
//
// Purpose:
//
//      Enables/Disables container modeless dialogs
//
// Parameters:
//
//      BOOL fEnable    - Enable/Disable
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      There are no modeless dialogs in this application, so the
//      implementation of this method is trivial.
//
//********************************************************************

STDMETHODIMP COleInPlaceFrame::EnableModeless (BOOL fEnable)
{
        OutputDebugString("In IOIPF::EnableModeless\r\n");
        return ResultFromScode(S_OK);
}
```

COleInPlaceFrame::TranslateAccelerator   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceFrame::TranslateAccelerator
//
// Purpose:
//
//      Not Implemented
//
// Parameters:
//
//      Not Implemented
//
// Return Value:
//
//      Not Implemented
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Not Implemented
//
//********************************************************************

STDMETHODIMP COleInPlaceFrame::TranslateAccelerator (LPMSG lpmsg, WORD wID)
{
        OutputDebugString("In IOIPF::TranslateAccelerator\r\n");
        return ResultFromScode(S_FALSE);
}
```

IOIPS.CPP   (SIMPCNTR Sample)

```
//**********************************************************************
// File name: IOIPS.CPP
//
//      Implementation file for COleInPlaceSite
//
// Functions:
//
//      See IOIPS.H for class Definition
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "iocs.h"
#include "ias.h"
#include "ioipf.h"
#include "ioips.h"
#include "app.h"
#include "site.h"
#include "doc.h"
```

CSimpleSite::COleInPlaceSite::QueryInterface   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleSite::COleInPlaceSite::QueryInterface
//
// Purpose:
//
//      Used for interface negotiation at the interface level.
//
// Parameters:
//
//      REFIID riid         -   A reference to the interface that is
//                              being queried.
//
//      LPVOID FAR* ppvObj  -   An out parameter to return a pointer to
//                              the interface.
//
// Return Value:
//
//      S_OK    -   The interface is supported.
//      S_FALSE -   The interface is not supported
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpleSite::QueryInterface SITE.CPP
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleInPlaceSite::QueryInterface(REFIID riid, LPVOID FAR*
ppvObj)
{
        OutputDebugString("In IOIPS::QueryInterface\r\n");
        CStabilize stabilize(m_pSite);

        // delegate to the container Site
        return m_pSite->QueryInterface(riid, ppvObj);
}
```

CSimpleSite::COleInPlaceSite::AddRef   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleSite::COleInPlaceSite::AddRef
//
// Purpose:
//
//      Adds to the reference count at the interface level.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the interface.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) COleInPlaceSite::AddRef()
{
        OutputDebugString("In IOIPS::AddRef\r\n");

        // increment the interface reference count (for debugging only)
        ++m_nCount;

        // delegate to the container Site
        return m_pSite->AddRef();
}
```

CSimpleSite::COleInPlaceSite::Release   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleSite::COleInPlaceSite::Release
//
// Purpose:
//
//      Decrements the reference count at this level
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the interface.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpleSite::Release        SITE.CPP
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) COleInPlaceSite::Release()
{
        OutputDebugString("In IOIPS::Release\r\n");
        // decrement the interface reference count (for debugging only)
        m_nCount--;

        // delegate to the container Site
        return m_pSite->Release();
}
```

COleInPlaceSite::GetWindow   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceSite::GetWindow
//
// Purpose:
//
//      Returns the Window Handle of the client site
//
// Parameters:
//
//      HWND FAR* lphwnd    - place to return the handle
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleInPlaceSite::GetWindow (HWND FAR* lphwnd)
{
        OutputDebugString("In IOIPS::GetWindow\r\n");
        CStabilize stabilize(m_pSite);

        // return the handle to our editing window.
        *lphwnd = m_pSite->m_lpDoc->m_hDocWnd;

        return ResultFromScode(S_OK);
}
```

COleInPlaceSite::ContextSensitiveHelp (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceSite::ContextSensitiveHelp
//
// Purpose:
//
//
// Parameters:
//
//      BOOL fEnterMode - TRUE for entering Context Sensitive help mode
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//      Be sure to read the technotes included with the OLE toolkit.
//
//********************************************************************

STDMETHODIMP COleInPlaceSite::ContextSensitiveHelp (BOOL fEnterMode)
{
        OutputDebugString("In IOIPS::ContextSensitiveHelp\r\n");

        if (m_pSite->m_lpDoc->m_lpApp->m_fCSHMode != fEnterMode)
                m_pSite->m_lpDoc->m_lpApp->m_fCSHMode = fEnterMode;

        return ResultFromScode(S_OK);
}
```

COleInPlaceSite::CanInPlaceActivate (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceSite::CanInPlaceActivate
//
// Purpose:
//
//      Object calls to find out if the container can InPlace activate
//
// Parameters:
//
//      None
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleInPlaceSite::CanInPlaceActivate ()
{
        OutputDebugString("In IOIPS::CanInPlaceActivate\r\n");

        // return S_OK to indicate we can in-place activate
        return ResultFromScode(S_OK);
}
```

COleInPlaceSite::OnInPlaceActivate   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceSite::OnInPlaceActivate
//
// Purpose:
//
//      Called by the object on InPlace Activation
//
// Parameters:
//
//      None
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleInPlaceSite::OnInPlaceActivate ()
{
        HRESULT hrErr;
        OutputDebugString("In IOIPS::OnInPlaceActivate\r\n");
        CStabilize stabilize(m_pSite);

        hrErr = m_pSite->m_lpOleObject->QueryInterface(
                        IID_IOleInPlaceObject, (LPVOID FAR *)&m_pSite-
>m_lpInPlaceObject);
        if (hrErr != NOERROR)
                return ResultFromScode(E_FAIL);

        // return S_OK to indicate we can in-place activate.
        return ResultFromScode(S_OK);
}
```

COleInPlaceSite::OnUIActivate (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceSite::OnUIActivate
//
// Purpose:
//
//      Object calls this method when it displays it's UI.
//
// Parameters:
//
//      None.
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleInPlaceSite::OnUIActivate ()
{
        OutputDebugString("In IOIPS::OnUIActivate\r\n");

        m_pSite->m_lpDoc->m_fAddMyUI=FALSE;
        m_pSite->m_lpDoc->m_fInPlaceActive = TRUE;
        m_pSite->m_fInPlaceActive = TRUE;

        m_pSite->m_lpInPlaceObject->GetWindow((HWND FAR*)&m_pSite-
>m_hwndIPObj);

        // return S_OK to continue in-place activation
        return ResultFromScode(S_OK);
}
```

COleInPlaceSite::GetWindowContext (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceSite::GetWindowContext
//
// Purpose:
//
//      Called by the object to get information for InPlace Negotiation.
//
// Parameters:
//
//      LPOLEINPLACEFRAME FAR* lplpFrame    - Location to return a pointer
//                                            to IOleInPlaceFrame.
//
//      LPOLEINPLACEUIWINDOW FAR* lplpDoc   - Location to return a pointer
//                                            to IOleInPlaceUIWindow.
//
//      LPRECT lprcPosRect                  - The rect that the object
//                                            occupies
//
//      LPRECT lprcClipRect                 - The clipping rect
//
//      LPOLEINPLACEFRAMEINFO lpFrameInfo   - Pointer to FRAMEINFO
//
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      COleInPlaceFrame::AddRef    IOIPF.CPP
//      CSimpleSite::GetObjRect     SITE.CPP
//      OutputDebugString           Windows API
//      SetMapMode                  Windows API
//      GetDC                       Windows API
//      ReleaseDC                   Windows API
//      CopyRect                    Windows API
//      GetClientRect               Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleInPlaceSite::GetWindowContext (LPOLEINPLACEFRAME FAR*
lplpFrame,

LPOLEINPLACEUIWINDOW FAR* lplpDoc,
                                                           LPRECT
lprcPosRect,
                                                           LPRECT
lprcClipRect,

LPOLEINPLACEFRAMEINFO lpFrameInfo)
{
        RECT rect;
        CStabilize stabilize(m_pSite);

        OutputDebugString("In IOIPS::GetWindowContext\r\n");

        // the frame is associated with the application object.
        // need to AddRef() it...
        m_pSite->m_lpDoc->m_lpApp->m_OleInPlaceFrame.AddRef();
        *lplpFrame = &m_pSite->m_lpDoc->m_lpApp->m_OleInPlaceFrame;
        *lplpDoc = NULL;  // must be NULL, cause we're SDI.

        // get the size of the object in pixels
        m_pSite->GetObjRect(&rect);

        // Copy this to the passed buffer
        CopyRect(lprcPosRect, &rect);

        // fill the clipping region
        GetClientRect(m_pSite->m_lpDoc->m_hDocWnd, &rect);
        CopyRect(lprcClipRect, &rect);

        // fill the FRAMEINFO
        lpFrameInfo->fMDIApp = FALSE;
        lpFrameInfo->hwndFrame = m_pSite->m_lpDoc->m_lpApp->m_hAppWnd;
        lpFrameInfo->haccel = NULL;
        lpFrameInfo->cAccelEntries = 0;

        return ResultFromScode(S_OK);
}
```

COleInPlaceSite::Scroll   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceSite::Scroll
//
// Purpose:
//
//      Not Implemented
//
// Parameters:
//
//      Not Implemented
//
// Return Value:
//
//      Not Implemented
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Not Implemented
//
//********************************************************************

STDMETHODIMP COleInPlaceSite::Scroll (SIZE scrollExtent)
{
        OutputDebugString("In IOIPS::Scroll\r\n");
        return ResultFromScode(E_FAIL);
}
```

COleInPlaceSite::OnUIDeactivate   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceSite::OnUIDeactivate
//
// Purpose:
//
//      Called by the object when its UI goes away
//
// Parameters:
//
//       BOOL fUndoable
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpleAPP::AddFrameLevelUI APP.CPP
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleInPlaceSite::OnUIDeactivate (BOOL fUndoable)
{
        OutputDebugString("In IOIPS::OnUIDeactivate\r\n");
        CStabilize stabilize(m_pSite);

        // need to clear this flag first
        m_pSite->m_lpDoc->m_fInPlaceActive = FALSE;
        m_pSite->m_fInPlaceActive = FALSE;

        m_pSite->m_lpDoc->m_lpApp->QueryNewPalette();
        m_pSite->m_lpDoc->m_lpApp->AddFrameLevelUI();
        return ResultFromScode(S_OK);
}
```

COleInPlaceSite::OnInPlaceDeactivate (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceSite::OnInPlaceDeactivate
//
// Purpose:
//
//      Called when the inplace session is over
//
// Parameters:
//
//      None
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleInPlaceSite::OnInPlaceDeactivate ()
{
        OutputDebugString("In IOIPS::OnInPlaceDeactivate\r\n");

        if (m_pSite->m_lpInPlaceObject) {
                m_pSite->m_lpInPlaceObject->Release();
                m_pSite->m_lpInPlaceObject = NULL;
        }
        return ResultFromScode(S_OK);
}
```

COleInPlaceSite::DiscardUndoState   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceSite::DiscardUndoState
//
// Purpose:
//
//      Not Implemented
//
// Parameters:
//
//      Not Implemented
//
// Return Value:
//
//      Not Implemented
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Not Implemented
//
//********************************************************************

STDMETHODIMP COleInPlaceSite::DiscardUndoState ()
{
        OutputDebugString("In IOIPS::DiscardUndoState\r\n");
        return ResultFromScode(E_FAIL);
}
```

COleInPlaceSite::DeactivateAndUndo   (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceSite::DeactivateAndUndo
//
// Purpose:
//
//      Not Implemented
//
// Parameters:
//
//      Not Implemented
//
// Return Value:
//
//      Not Implemented
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Not Implemented
//
//********************************************************************

STDMETHODIMP COleInPlaceSite::DeactivateAndUndo ()
{
        OutputDebugString("In IOIPS::DeactivateAndUndo\r\n");
        return ResultFromScode(E_FAIL);
}
```

COleInPlaceSite::OnPosRectChange (SIMPCNTR Sample)

```
//**********************************************************************
//
// COleInPlaceSite::OnPosRectChange
//
// Purpose:
//
//      The object calls this method when it's size changes during an
//      InPlace Session
//
// Parameters:
//
//      LPCRECT lprcPosRect -   The new object rect
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                            Location
//
//      OutputDebugString                   Windows API
//      GetClientRect                       Windows API
//      IOleObject::GetExtent               Object
//      IOleObject::QueryInterface          Object
//      IOleInPlaceObject::SetObjectRects   Object
//      IOleInPlaceObject::Release          Object
//      ResultFromScode                     OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleInPlaceSite::OnPosRectChange (LPCRECT lprcPosRect)
{
        OutputDebugString("In IOIPS::OnPosRectChange\r\n");
        CStabilize stabilize(m_pSite);

        // update the size in the document object
        // NOTE: here we must call IOleObject::GetExtent to get actual
extents
        //       of the running object. IViewObject2::GetExtent returns the
        //       last cached extents.
        m_pSite->m_lpOleObject->GetExtent(DVASPECT_CONTENT, &m_pSite-
>m_sizel);
        RECT rect;
        GetClientRect(m_pSite->m_lpDoc->m_hDocWnd, &rect);

        // tell the object its new size
        m_pSite->m_lpInPlaceObject->SetObjectRects(lprcPosRect, &rect);

        return ResultFromScode(S_OK);
}
```

SIMPCNTR.CPP (SIMPCNTR Sample)

```
//**********************************************************************
// File name: Simple.cpp
//
//      Main source file for the Simple OLE 2.0 object container
//
// Functions:
//
//      WinMain         - Program entry point
//      MainWndProc     - Processes messages for the frame window
//      About           - Processes messages for the about dialog
//      DocWndProc      - Processes messages for the doc window
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "iocs.h"
#include "ias.h"
#include "ioipf.h"
#include "ioips.h"
#include "app.h"
#include "site.h"
#include "doc.h"

// This line is needed for the debug utilities in OLE2UI
extern "C" {
        OLEDBGDATA_MAIN("SIMPCNTR")
}

CSimpleApp FAR * lpCSimpleApp;
```

WinMain (SIMPCNTR Sample)

```
//**********************************************************************
//
// WinMain
//
// Purpose:
//
//      Program entry point
//
// Parameters:
//
//      HANDLE hInstance        - Instance handle for this instance
//
//      HANDLE hPrevInstance    - Instance handle for the last instance
//
//      LPSTR lpCmdLine         - Pointer to the command line
//
//      int nCmdShow            - Window State
//
// Return Value:
//
//      msg.wParam
//
// Function Calls:
//      Function                        Location
//
//      CSimpleApp::CSimpleApp          APP.CPP
//      CSimpleApp::fInitApplication    APP.CPP
//      CSimpleApp::fInitInstance       APP.CPP
//      CSimpleApp::HandleAccelerators  APP.CPP
//      CSimpleApp::~CSimpleApp         APP.CPP
//      OleUIInitialize                 OLE2UI
//      OleUIUninitialize               OLE2UI
//      GetMessage                      Windows API
//      TranslateMessage                Windows API
//      DispatchMessage                 Windows API
//
// Comments:
//
//********************************************************************

int PASCAL WinMain(HANDLE hInstance,HANDLE hPrevInstance,LPSTR lpCmdLine,int
nCmdShow)

{
        MSG msg;

        // needed for LRPC to work properly...
        SetMessageQueue(96);

        lpCSimpleApp = new CSimpleApp;
        // we will add one ref count on our App. later when we want to
destroy
        // the App object we will release this  ref count. when the App's
ref
        // count goes to 0, it will be deleted.
        lpCSimpleApp->AddRef();

        // app initialization
        if (!hPrevInstance)
                if (!lpCSimpleApp->fInitApplication(hInstance))
                        return (FALSE);

        // instance initialization
        if (!lpCSimpleApp->fInitInstance(hInstance, nCmdShow))
                return (FALSE);

        /* Initialization required for OLE 2 UI library.  This call is
        **    needed ONLY if we are using the static link version of the UI
        **    library. If we are using the DLL version, we should NOT call
        **    this function in our application.
        */
#if 0
        if (!OleUIInitialize(hInstance, hPrevInstance))
                {
                OleDbgOut("Could not initialize OLEUI library\n");
                return FALSE;
                }
#endif
        // message loop
        while (GetMessage(&msg, NULL, NULL, NULL))
                if (!lpCSimpleApp->HandleAccelerators(&msg))
                        {
                        TranslateMessage(&msg);    /* Translates virtual key
codes */
                        DispatchMessage(&msg);     /* Dispatches message to
window */
                        }

        // De-initialization for UI libraries.  Just like OleUIInitialize,
this
        // funciton is needed ONLY if we are using the static link version
of the
        // OLE UI library.
#if 0
        OleUIUninitialize();
#endif

        // Release the ref count added on the App above. this will make
        // the App's ref count go to 0, and the App object will be deleted.
        lpCSimpleApp->Release();

        return (msg.wParam);          /* Returns the value from
PostQuitMessage */
}
```

MainWndProc (SIMPCNTR Sample)

```
//**********************************************************************
//
// MainWndProc
//
// Purpose:
//
//      Processes messages for the frame window
//
// Parameters:
//
//      HWND hWnd       - Window handle for frame window
//
//      UINT message    - Message value
//
//      WPARAM wParam   - Message info
//
//      LPARAM lParam   - Message info
//
// Return Value:
//
//      long
//
// Function Calls:
//      Function                        Location
//
//      CSimpleApp::lCommandHandler     APP.CPP
//      CSimpleApp::DestroyDocs         APP.CPP
//      CSimpleApp::lCreateDoc          APP.CPP
//      CSimpleApp::lSizeHandler        APP.CPP
//      CSimpleDoc::lAddVerbs           DOC.CPP
//      PostQuitMessage                 Windows API
//      DefWindowProc                   Windows API
//
// Comments:
//
//********************************************************************

//@@WTK WIN32, UNICODE
//long FAR PASCAL _export MainWndProc(HWND hWnd,UINT message,WPARAM
wParam,LPARAM lParam)
long FAR PASCAL EXPORT MainWndProc(HWND hWnd,UINT message,WPARAM
wParam,LPARAM lParam)

{

        switch (message)
                {
                case WM_COMMAND:           // message: command from
application menu
                        return lpCSimpleApp->lCommandHandler(hWnd, message,
wParam, lParam);
                        break;
                case WM_CREATE:
                        return lpCSimpleApp->lCreateDoc(hWnd, message,
wParam, lParam);
                        break;

                case WM_DESTROY:                  // message: window being
destroyed
                        lpCSimpleApp->DestroyDocs();  // need to destroy the
doc...
                        PostQuitMessage(0);
                        break;

                case WM_INITMENUPOPUP:
                        // is this the edit menu?
                        // Message packing is the same for WIN16 and 32
                        if ( LOWORD(lParam) == 1)
                                return lpCSimpleApp->m_lpDoc->lAddVerbs();

                        break;

                // this code is needed for 256 color objects to work
properly.
                case WM_QUERYNEWPALETTE:
                        if (! lpCSimpleApp->m_fAppActive)
                                return 0L;

                        return lpCSimpleApp->QueryNewPalette();


                case WM_PALETTECHANGED:
                {
                        HWND hWndPalChg = (HWND) wParam;

                        if (hWnd != hWndPalChg)
                                wSelectPalette(hWnd, lpCSimpleApp-
>m_hStdPal, TRUE/*fBackground*/);

                        /* OLE2NOTE: always forward the WM_PALETTECHANGED
message (via
                        **    SendMessage) to any in-place objects that
currently have
                        **    their window visible. this gives these objects
the chance
                        **    to select their palettes. this is
                        **    REQUIRED by all in-place containers
independent of
                        **    whether they use color palettes themselves--
their objects
                        **    may use color palettes.
                        **    (see ContainerDoc_ForwardPaletteChangedMsg for
more info)
                        */
                        if (lpCSimpleApp->m_lpDoc && lpCSimpleApp->m_lpDoc-
>m_lpSite &&
                                lpCSimpleApp->m_lpDoc->m_lpSite-
>m_hwndIPObj)
                                SendMessage(lpCSimpleApp->m_lpDoc->m_lpSite-
>m_hwndIPObj,
                                                WM_PALETTECHANGED, wParam,
lParam);

                        return 0L;
                }

                case WM_ACTIVATEAPP:
                        if ((lpCSimpleApp->m_fAppActive = (BOOL)wParam) ==
TRUE)
                                lpCSimpleApp->QueryNewPalette();

                        if (lpCSimpleApp->m_lpDoc->m_lpActiveObject) {
                                lpCSimpleApp->m_lpDoc->m_lpActiveObject-
>OnFrameWindowActivate(
                                                (BOOL)wParam);
                        }
                        break;


                case WM_SIZE:
                        return lpCSimpleApp->lSizeHandler(hWnd, message,
wParam, lParam);

                default:                          // Passes it on if
unproccessed
                        return (DefWindowProc(hWnd, message, wParam,
lParam));
                }
                return (NULL);
}
```

About (SIMPCNTR Sample)

```
//**********************************************************************
//
// About
//
// Purpose:
//
//      Processes dialog box messages
//
// Parameters:
//
//      HWND hWnd       - Window handle for dialog box
//
//      UINT message    - Message value
//
//      WPARAM wParam   - Message info
//
//      LPARAM lParam   - Message info
//
// Return Value:
//
// Function Calls:
//      Function                    Location
//
//      EndDialog                   Windows API
//
// Comments:
//
//********************************************************************

//@@WTK WIN32, UNICODE
//BOOL FAR PASCAL _export About(HWND hDlg,unsigned message,WORD wParam,LONG
lParam)
BOOL FAR PASCAL EXPORT About(HWND hDlg,UINT message,WPARAM wParam,LPARAM
lParam)

{
        switch (message) {
        case WM_INITDIALOG:                /* message: initialize dialog box
*/
                return (TRUE);

        case WM_COMMAND:                      /* message: received a command
*/
                //@@WTK WIN32, UNICODE
                if (LOWORD(wParam) == IDOK                /* "OK" box
selected?          */
                || LOWORD(wParam) == IDCANCEL) {      /* System menu close
command? */
                        EndDialog(hDlg, TRUE);        /* Exits the dialog
box        */
                        return (TRUE);
                }
                break;
        }
        return (FALSE);                                    /* Didn't process a
message    */
}
```

DocWndProc   (SIMPCNTR Sample)

```
//**********************************************************************
//
// DocWndProc
//
// Purpose:
//
//      Processes dialog box messages
//
// Parameters:
//
//      HWND hWnd       - Window handle for doc window
//
//      UINT message    - Message value
//
//      WPARAM wParam   - Message info
//
//      LPARAM lParam   - Message info
//
// Return Value:
//
// Function Calls:
//      Function                            Location
//
//      CSimpleApp::PaintApp                APP.CPP
//      BeginPaint                          Windows API
//      EndPaint                            Windows API
//      DefWindowProc                       Windows API
//      IOleObject::QueryInterface          Object
//      IOleInPlaceObject::UIDeactivate     Object
//      IOleObject::DoVerb                  Object
//      IOleInPlaceObject::Release          Object
//
// Comments:
//
//********************************************************************

//@@WTK WIN32, UNICODE
//long FAR PASCAL _export DocWndProc(HWND hWnd,UINT message,WPARAM
wParam,LPARAM lParam)
long FAR PASCAL EXPORT DocWndProc(HWND hWnd,UINT message,WPARAM
wParam,LPARAM lParam)
{
        HDC hDC;
        PAINTSTRUCT ps;

        switch (message) {
                case WM_PAINT:

                        hDC = BeginPaint(hWnd, &ps);

                        if (lpCSimpleApp)
                                lpCSimpleApp->PaintApp (hDC);
                        EndPaint(hWnd, &ps);
                        break;


                case WM_LBUTTONDBLCLK:
                        {
                        POINT pt;

                        pt.x = (int)(short)LOWORD (lParam );
                        pt.y = (int)(short)HIWORD (lParam );

                        if (lpCSimpleApp->m_lpDoc->m_lpSite &&
                                lpCSimpleApp->m_lpDoc->m_lpSite-
>m_lpOleObject)
                                {
                                RECT rect;

                                lpCSimpleApp->m_lpDoc->m_lpSite-
>GetObjRect(&rect);

                                if ( PtInRect(&rect, pt) )
                                        {
                                        // Execute object's default verb
                                        lpCSimpleApp->m_lpDoc->m_lpSite-
>m_lpOleObject->DoVerb(
                                                        OLEIVERB_PRIMARY,
(LPMSG)&message,
                                                        &lpCSimpleApp-
>m_lpDoc->m_lpSite->m_OleClientSite,
                                                        -1, hWnd, &rect);
                                        }
                                }
                        }
                        break;

        // no code is added to WM_LBUTTONDOWN for context sensitive help,
because
        // this app does not do context sensitive help.
        case WM_LBUTTONDOWN:

                LPOLEINPLACEOBJECT lpObject;

                if (lpCSimpleApp->m_lpDoc->m_fInPlaceActive)
                        {
                        lpCSimpleApp->m_lpDoc->m_lpSite->m_lpOleObject-
>QueryInterface(IID_IOleInPlaceObject, (LPVOID FAR *)&lpObject);
                        lpObject->UIDeactivate();

                        // this code is needed because we don't support
inside out.
                        RECT rect;
                        lpCSimpleApp->m_lpDoc->m_lpSite->GetObjRect(&rect);
                        lpCSimpleApp->m_lpDoc->m_lpSite->m_lpOleObject-
>DoVerb(OLEIVERB_HIDE, (LPMSG)&message, &lpCSimpleApp->m_lpDoc->m_lpSite-
>m_OleClientSite, -1, hWnd, &rect);

                        lpObject->Release();
                        }

                break;

                default:                          /* Passes it on if
unproccessed */
                        return (DefWindowProc(hWnd, message, wParam,
lParam));
        }
        return (NULL);
}
```

SITE.CPP (SIMPCNTR Sample)

```
//**********************************************************************
// File name: SITE.CPP
//
//      Implementation file for CSimpleSite
//
// Functions:
//
//      See SITE.H for class definition
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "iocs.h"
#include "ias.h"
#include "ioipf.h"
#include "ioips.h"
#include "app.h"
#include "site.h"
#include "doc.h"
```

CSimpleSite::Create (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleSite::Create
//
// Purpose:
//
//      Creation routine for CSimpleSite
//
// Parameters:
//
//      CSimpleDoc FAR *lpDoc   - Pointer to CSimpleDoc
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      IStorage::CreateStorage     OLE API
//      assert                      C Runtime
//
// Comments:
//
//********************************************************************

CSimpleSite FAR * CSimpleSite::Create(CSimpleDoc FAR *lpDoc)
{
        CSimpleSite FAR * lpTemp = new CSimpleSite(lpDoc);

        if (!lpTemp)
                return NULL;

        // create a sub-storage for the object
  //@@WTK WIN32, UNICODE
        //HRESULT hErr = lpDoc->m_lpStorage->CreateStorage( "Object",
        HRESULT hErr = lpDoc->m_lpStorage->CreateStorage( OLESTR("Object"),
                                STGM_READWRITE | STGM_TRANSACTED |
STGM_SHARE_EXCLUSIVE,
                                0,
                                0,
                                &lpTemp->m_lpObjStorage);

        assert(hErr == NOERROR);

        if (hErr != NOERROR)
                {
                delete lpTemp;
                return NULL;
                }
        // we will add one ref count on our Site. later when we want to 
destroy
        // the Site object we will release this  ref count. when the Site's 
ref
        // count goes to 0, it will be deleted.
        lpTemp->AddRef();

        return lpTemp;
}
```

CSimpleSite::CSimpleSite   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleSite::CSimpleSite
//
// Purpose:
//
//      Constructor for CSimpleSite
//
// Parameters:
//
//      CSimpleDoc FAR *lpDoc   - Pointer to CSimpleDoc
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
// Comments:
//
//********************************************************************
#pragma warning(disable : 4355)  // turn off this warning.  This warning
                                                                // tells us
that we are passing this in
                                                                // an
initializer, before "this" is through
                                                                //
initializing.  This is ok, because
                                                                // we just
store the ptr in the other
                                                                //
constructors

CSimpleSite::CSimpleSite (CSimpleDoc FAR *lpDoc) : m_OleClientSite(this),

m_AdviseSink(this),

m_OleInPlaceSite(this)
#pragma warning (default : 4355)  // Turn the warning back on
{
        // remember the pointer to the doc
        m_lpDoc = lpDoc;

        // clear the reference count

        m_dwDrawAspect = DVASPECT_CONTENT;
        m_lpOleObject = NULL;
        m_lpInPlaceObject = NULL;
        m_hwndIPObj = NULL;
        m_fInPlaceActive = FALSE;
        m_fObjectOpen = FALSE;
}
```

CSimpleSite::~CSimpleSite (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleSite::~CSimpleSite
//
// Purpose:
//
//      Destructor for CSimpleSite
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                                Location
//
//      OutputDebugString                       Windows API
//      IOleObject::Release                     Object
//      IStorage::Release                       OLE API
//
// Comments:
//
//********************************************************************

CSimpleSite::~CSimpleSite ()
{
        OutputDebugString ("In CSimpleSite's Destructor \r\n");

        if (m_lpOleObject)
           m_lpOleObject->Release();

        if (m_lpObjStorage)
           m_lpObjStorage->Release();
}
```

CSimpleSite::CloseOleObject (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleSite::CloseOleObject
//
// Purpose:
//
//      Call IOleObject::Close on the object of the CSimpleSite
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                                Location
//
//      OutputDebugString                       Windows API
//      IOleObject::QueryInterface              Object
//      IOleObject::Close                       Object
//      IOleInPlaceObject::UIDeactivate         Object
//      IOleInPlaceObject::InPlaceDeactivate    Object
//      IOleInPlaceObject::Release              Object
//
// Comments:
//
//********************************************************************

void CSimpleSite::CloseOleObject (void)
{
        CStabilize stabilize(this);
        LPOLEINPLACEOBJECT lpObject;
        LPVIEWOBJECT lpViewObject = NULL;

        OutputDebugString ("In CSimpleSite::CloseOleObject \r\n");

        if (m_lpOleObject)
           {
           if (m_fInPlaceActive)
                   {
                   m_lpOleObject->QueryInterface(IID_IOleInPlaceObject,
(LPVOID FAR *)&lpObject);
                   lpObject->UIDeactivate();
                   // don't need to worry about inside-out because the
object
                   // is going away.
                   lpObject->InPlaceDeactivate();
                   lpObject->Release();
                   }
```

```
              m_lpOleObject->Close(OLECLOSE_NOSAVE);
              }
}
```

CSimpleSite::UnloadOleObject   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleSite::UnloadOleObject
//
// Purpose:
//
//      Close and release all pointers to the object of the CSimpleSite
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                                Location
//
//      OutputDebugString                       Windows API
//      CSimpleSite::CloseOleObject             SITE.CPP
//      IOleObject::QueryInterface              Object
//      IViewObject::SetAdvise                  Object
//      IViewObject::Release                    Object
//      IStorage::Release                       OLE API
//
// Comments:
//
//********************************************************************

void CSimpleSite::UnloadOleObject (void)
{
        CStabilize stabilize(this);
        OutputDebugString ("In CSimpleSite::UnloadOleObject \r\n");

        if (m_lpOleObject)
           {
           LPVIEWOBJECT lpViewObject;
           CloseOleObject();    // ensure object is closed; NOP if already
closed

           m_lpOleObject->QueryInterface(IID_IViewObject, (LPVOID FAR
*)&lpViewObject);

           if (lpViewObject)
                   {
                   // Remove the view advise
                   lpViewObject->SetAdvise(m_dwDrawAspect, 0, NULL);
                   lpViewObject->Release();
                   }

           m_lpOleObject->Release();
```

```
                m_lpOleObject = NULL;
                }
}
```

CSimpleSite::QueryInterface   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleSite::QueryInterface
//
// Purpose:
//
//      Used for interface negotiation of the container Site.
//
// Parameters:
//
//      REFIID riid         -   A reference to the interface that is
//                              being queried.
//
//      LPVOID FAR* ppvObj  -   An out parameter to return a pointer to
//                              the interface.
//
// Return Value:
//
//      S_OK    -   The interface is supported.
//      S_FALSE -   The interface is not supported
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      IsEqualIID                  OLE API
//      ResultFromScode             OLE API
//      CSimpleSite::AddRef          OBJ.CPP
//      COleClientSite::AddRef      IOCS.CPP
//      CAdviseSink::AddRef         IAS.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP CSimpleSite::QueryInterface(REFIID riid, LPVOID FAR* ppvObj)
{
        CStabilize stabilize(this);
        OutputDebugString("In CSimpleSite::QueryInterface\r\n");

        *ppvObj = NULL;     // must set out pointer parameters to NULL

        if ( riid == IID_IUnknown)
                {
                AddRef();
                *ppvObj = this;
                return ResultFromScode(S_OK);
                }

        if ( riid == IID_IOleClientSite)
                {
```

```
                m_OleClientSite.AddRef();
                *ppvObj = &m_OleClientSite;
                return ResultFromScode(S_OK);
                }

        if ( riid == IID_IAdviseSink)
                {
                m_AdviseSink.AddRef();
                *ppvObj = &m_AdviseSink;
                return ResultFromScode(S_OK);
                }

        if ( riid == IID_IOleInPlaceSite)
                {
                m_OleInPlaceSite.AddRef();
                *ppvObj = &m_OleInPlaceSite;
                return ResultFromScode(S_OK);
                }

        // Not a supported interface
        return ResultFromScode(E_NOINTERFACE);
}
```

CSimpleSite::AddRef   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleSite::AddRef
//
// Purpose:
//
//      Increments the reference count of the container Site.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the site.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpleSite::AddRef()
{
        OutputDebugString("In CSimpleSite::AddRef\r\n");

        return SafeAddRef();
}
```

CSimpleSite::Release   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleSite::Release
//
// Purpose:
//
//      Decrements the reference count of the container Site
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the Site.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpleSite::Release()
{
        OutputDebugString("In CSimpleSite::Release\r\n");

        return(SafeRelease());
}
```

CSimpleSite::InitObject (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleSite::InitObject
//
// Purpose:
//
//      Used to initialize a newly create object (can't be done in the
//      constructor).
//
// Parameters:
//
//      BOOL fCreateNew -   TRUE if insert NEW object
//                          FALSE if create object FROM FILE
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                        Location
//
//      IOleObject::SetHostNames        Object
//      IOleObject::QueryInterface      Object
//      IViewObject2::GetExtent         Object
//      IOleObject::DoVerb              Object
//      IViewObject::SetAdvise          Object
//      IViewObject::Release            Object
//      GetClientRect                   Windows API
//      OleSetContainedObject           OLE API
//
// Comments:
//
//********************************************************************

void CSimpleSite::InitObject(BOOL fCreateNew)
{
        CStabilize stabilize(this);
        LPVIEWOBJECT2 lpViewObject2;
        RECT rect;

        // Set a View Advise
        m_lpOleObject->QueryInterface(IID_IViewObject2,(LPVOID FAR
*)&lpViewObject2);
        lpViewObject2->SetAdvise(m_dwDrawAspect, ADVF_PRIMEFIRST,
&m_AdviseSink);

        // get the initial size of the object
        lpViewObject2->GetExtent(m_dwDrawAspect, -1 /*lindex*/, NULL
/*ptd*/, &m_sizel);
        GetObjRect(&rect);  // get the rectangle of the object in pixels
        lpViewObject2->Release();
        // give the object the name of the container app/document
  //@@WTK WIN32, UNICODE
        //m_lpOleObject->SetHostNames("Simple Application", "Simple OLE 2.0
In-Place Container");
        m_lpOleObject->SetHostNames(OLESTR("Simple Application"),
                        OLESTR("Simple OLE 2.0 In-Place Container"));

        // inform object handler/DLL object that it is used in the embedding
container's context
        OleSetContainedObject(m_lpOleObject, TRUE);

        if (fCreateNew) {
           // force new object to save to guarantee valid object in our
storage.
           // OLE 1.0 objects may close w/o saving. this is NOT necessary if
the
           // object is created FROM FILE; its data in storage is already
valid.
           m_OleClientSite.SaveObject();

           // we only want to DoVerb(SHOW) if this is an InsertNew object.
           // we should NOT DoVerb(SHOW) if the object is created FromFile.
           m_lpOleObject->DoVerb(
                           OLEIVERB_SHOW,
                           NULL,
                           &m_OleClientSite,
                           -1,
                           m_lpDoc->m_hDocWnd,
                           &rect);
        }
}
```

CSimpleSite::PaintObj (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleSite::PaintObj
//
// Purpose:
//
//      Paints the object
//
// Parameters:
//
//      HDC hDC     - Device context of the document window
//
// Return Value:
//
// Function Calls:
//      Function                        Location
//
//      IOleObject::QueryInterface      Object
//      IViewObject::GetColorSet        Object
//      IViewObject::Release            Object
//      SetMapMode                      Windows API
//      LPtoDP                          Windows API
//      CreateHatchBrush                Windows API
//      SelectObject                    Windows API
//      DeleteObject                    Windows API
//      CreatePalette                   Windows API
//      SelectPalette                   Windows API
//      RealizePalette                  Windows API
//      OleStdFree                      OUTLUI Function
//      OleDraw                         OLE API
//
// Comments:
//
//********************************************************************

void CSimpleSite::PaintObj(HDC hDC)
{
RECT rect;
        CStabilize stabilize(this);

        // need to check to make sure there is a valid object
        // available.  This is needed if there is a paint msg
        // between the time that CSimpleSite is instantiated
        // and OleUIInsertObject returns.
        if (!m_lpOleObject)
                return;

        // convert it to pixels
        GetObjRect(&rect);

        LPLOGPALETTE pColorSet = NULL;
        LPVIEWOBJECT lpView = NULL;
        // get a pointer to IViewObject
        m_lpOleObject->QueryInterface(IID_IViewObject,(LPVOID FAR *)
&lpView);

        // if the QI succeeds, get the LOGPALETTE for the object
        if (lpView)
                lpView->GetColorSet(m_dwDrawAspect, -1, NULL, NULL, NULL,
&pColorSet);

        HPALETTE hPal=NULL;
        HPALETTE hOldPal=NULL;

        // if a LOGPALETTE was returned (not guarateed), create the palette
and
        // realize it.  NOTE: A smarter application would want to get the
LOGPALETTE
        // for each of its visible objects, and try to create a palette that
        // satisfies all of the visible objects.  ALSO: OleStdFree() is use
to
        // free the returned LOGPALETTE.
        if ((pColorSet))
                {
                //@@WTK WIN32, UNICODE
                //hPal = CreatePalette((const LPLOGPALETTE) pColorSet);
                hPal = CreatePalette( pColorSet);
                hOldPal = SelectPalette(hDC, hPal, FALSE);
                RealizePalette(hDC);
                OleStdFree(pColorSet);
                }

        // draw the object
        OleDraw(m_lpOleObject, m_dwDrawAspect, hDC, &rect);

        // if the object is open, draw a hatch rect.
        if (m_fObjectOpen)
                {
                HBRUSH hBrush = CreateHatchBrush ( HS_BDIAGONAL,
RGB(0,0,0) );
                HBRUSH hOldBrush = SelectObject (hDC, hBrush);
                SetROP2(hDC, R2_MASKPEN);
                Rectangle (hDC, rect.left, rect.top, rect.right,
rect.bottom);
                SelectObject(hDC, hOldBrush);
                DeleteObject(hBrush);
                }

        // if we created a palette, restore the old one, and destroy
        // the object.
        if (hPal)
                {
                SelectPalette(hDC,hOldPal,FALSE);
                DeleteObject(hPal);
                }
        // if a view pointer was successfully returned, it needs to be
released.
        if (lpView)
                lpView->Release();
}
```

CSimpleSite::GetObjRect   (SIMPCNTR Sample)

```
//**********************************************************************
//
// CSimpleSite::GetObjRect
//
// Purpose:
//
//      Retrieves the rect of the object in pixels
//
// Parameters:
//
//      LPRECT lpRect - Rect structure filled with object's rect in pixels
//
// Return Value:
//
// Function Calls:
//      Function                        Location
//
//      XformWidthInHimetricToPixels    OUTLUI Function
//      XformHeightInHimetricToPixels   OUTLUI Function
//
// Comments:
//
//********************************************************************

void CSimpleSite::GetObjRect(LPRECT lpRect)
{
        // convert it to pixels
        lpRect->left = lpRect->top = 0;
        lpRect->right = XformWidthInHimetricToPixels(NULL,(int)m_sizel.cx);
        lpRect->bottom = XformHeightInHimetricToPixels(NULL,
(int)m_sizel.cy);
}
```

SIMPDND

Simple Drag and Drop Sample

This sample demonstrates an implementation of OLE 2 drag and drop. For information on compiling and building the sample, see the makefile in this directory.

MAKEFILE (SIMPDND Sample)

```
!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

all: simpdnd.exe

simpdnd.exe:  app.obj         \
            doc.obj         \
            dxferobj.obj    \
            ias.obj         \
            ids.obj         \
            idt.obj         \
            iocs.obj        \
            simpdnd.obj     \
            simpdnd.res     \
            site.obj
  $(link) $(linkdebug) $(guiflags) -machine:$(CPU) -out:$*.exe $** $
(olelibs) ole2ui.lib

.cpp.obj:
    $(cc) $(cdebug) $(cflags) $(cvars) $*.cpp

simpdnd.res: simpdnd.rc simpdnd.h
    $(rc) -r -I..\ole2ui -I..\ole2ui\res\usa -I..\ole2ui\res\static
simpdnd.rc
```

APP.H (SIMPDND Sample)

```
//**********************************************************************
// File name: app.h
//
//      Definition of CSimpleApp
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#if !defined( _APP_H_)
#define _APP_H_

#include <ole2.h>
#include <stablize.h>

class CSimpleDoc;

class CSimpleApp : public IUnknown, public CSafeRefCount
{
public:

        HWND m_hAppWnd;         // main window handle
        HINSTANCE m_hInst;      // application instance
        CSimpleDoc FAR * m_lpDoc;   // pointer to document object
        BOOL m_fInitialized;    // OLE initialization flag
        HMENU           m_hMainMenu;
        HMENU           m_hFileMenu;
        HMENU           m_hEditMenu;
        HMENU           m_hHelpMenu;
        HMENU           m_hCascadeMenu;     // OLE object's verb


        // Drag/Drop related fields
        int m_nDragDelay;       // time delay (in msec) before drag should
start
        int m_nDragMinDist;     // min. distance (radius) before drag should
start
        int m_nScrollDelay;     // time delay (in msec) before scroll should
start
        int m_nScrollInset;     // Border inset distance to start drag
scroll
        int m_nScrollInterval;  // scroll interval time (in msec)

        CSimpleApp();           // Constructor
        ~CSimpleApp();          // Destructor

        // IUnknown Interfaces
        STDMETHODIMP QueryInterface(REFIID riid, LPVOID FAR* ppvObj);
        STDMETHODIMP_(ULONG) AddRef();
        STDMETHODIMP_(ULONG) Release();

        // Initialization methods
        BOOL fInitApplication (HANDLE hInstance);
        BOOL fInitInstance (HANDLE hInstance, int nCmdShow);

        // Message handling methods

        long lCommandHandler (HWND hWnd,UINT message,WPARAM wParam,LPARAM
lParam);
        long lSizeHandler (HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam);
        long lCreateDoc (HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam);
        BOOL HandleAccelerators (LPMSG lpMsg);
        void PaintApp(HDC hDC);
        void DestroyDocs();
};

#endif  // _APP_H_
```

DOC.H   (SIMPDND Sample)

```
//**********************************************************************
// File name: doc.h
//
//      Definition of CSimpleDoc
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#if !defined( _DOC_H_ )
#define _DOC_H_

#include "idt.h"
#include "ids.h"
#include <stablize.h>

class CSimpleSite;
class CSimpleApp;

class CSimpleDoc : public IUnknown, public CSafeRefCount
{
public:
        LPSTORAGE       m_lpStorage;        // IStorage* pointer for Doc
        BOOL            m_fModifiedMenu;    // is object's verb menu on menu

        // Drag/Drop related fields
        BOOL            m_fRegDragDrop;     // is doc registered as drop
target?
        BOOL            m_fLocalDrag;       // is doc source of the drag
        BOOL            m_fLocalDrop;       // was doc target of the drop
        BOOL            m_fCanDropCopy;     // is Drag/Drop copy/move
possible?
        BOOL            m_fCanDropLink;     // is Drag/Drop link possible?
        BOOL            m_fDragLeave;       // has drag left
        BOOL            m_fPendingDrag;     // LButtonDown--possible drag
pending
        POINT           m_ptButDown;        // LButtonDown coordinates

        CSimpleSite FAR * m_lpSite;
        CSimpleApp FAR * m_lpApp;

        HWND m_hDocWnd;

        CDropTarget m_DropTarget;
        CDropSource m_DropSource;

        static CSimpleDoc FAR* Create(CSimpleApp FAR *lpApp, LPRECT lpRect,
                        HWND hWnd);

        void Close(void);

        CSimpleDoc();
        CSimpleDoc(CSimpleApp FAR *lpApp, HWND hWnd);
        ~CSimpleDoc();

        // IUnknown Interface
        STDMETHODIMP QueryInterface(REFIID riid, LPVOID FAR* ppvObj);
        STDMETHODIMP_(ULONG) AddRef();
        STDMETHODIMP_(ULONG) Release();

        void InsertObject(void);
        void DisableInsertObject(void);
        long lResizeDoc(LPRECT lpRect);
        long lAddVerbs(void);
        void PaintDoc(HDC hDC);

        // Drag/Drop and clipboard support methods
        void CopyObjectToClip(void);
        BOOL QueryDrag(POINT pt);
        DWORD DoDragDrop(void);
        void Scroll(DWORD dwScrollDir) { /*...scroll Doc here...*/ }
};

#endif  // _DOC_H_
```

DXFEROBJ.H   (SIMPDND Sample)

```
//**********************************************************************
// File name: dxferobj.h
//
//      Definition of CDataXferObj
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#if !defined( _DATAXFEROBJ_H_)
#define _DATAXFEROBJ_H_

class CSimpleSite;

interface CDataObject;

class CDataXferObj : public IDataObject
{
private:
     int m_nCount;                       // reference count
     SIZEL m_sizel;
     POINTL m_pointl;
     LPSTORAGE m_lpObjStorage;
     LPOLEOBJECT m_lpOleObject;

     // construction/destruction
     CDataXferObj();
     ~CDataXferObj();

public:
     STDMETHODIMP QueryInterface (REFIID riid, LPVOID FAR* ppvObj);
     STDMETHODIMP_(ULONG) AddRef ();
     STDMETHODIMP_(ULONG) Release ();

     STDMETHODIMP DAdvise  ( FORMATETC FAR* pFormatetc, DWORD advf,
     LPADVISESINK pAdvSink, DWORD FAR* pdwConnection)
           { return ResultFromScode(OLE_E_ADVISENOTSUPPORTED); }
     STDMETHODIMP DUnadvise  ( DWORD dwConnection)
           { return ResultFromScode(OLE_E_ADVISENOTSUPPORTED); }
     STDMETHODIMP EnumDAdvise  ( LPENUMSTATDATA FAR* ppenumAdvise)
           { return ResultFromScode(OLE_E_ADVISENOTSUPPORTED); }
     STDMETHODIMP EnumFormatEtc  ( DWORD dwDirection,
                 LPENUMFORMATETC FAR* ppenumFormatEtc);
                 STDMETHODIMP GetCanonicalFormatEtc  ( LPFORMATETC
pformatetc,
                 LPFORMATETC pformatetcOut)
           { pformatetcOut->ptd = NULL; return ResultFromScode(E_NOTIMPL); }
     STDMETHODIMP GetData  (LPFORMATETC pformatetcIn, LPSTGMEDIUM pmedium );
     STDMETHODIMP GetDataHere  (LPFORMATETC pformatetc, LPSTGMEDIUM
pmedium);
     STDMETHODIMP QueryGetData  (LPFORMATETC pformatetc );
     STDMETHODIMP SetData  (LPFORMATETC pformatetc, STGMEDIUM FAR * pmedium,
                 BOOL fRelease)
           { return ResultFromScode(E_NOTIMPL); }

     static CDataXferObj FAR* Create(CSimpleSite FAR* lpSite,
                 POINTL FAR* pPointl);

};
#endif  // _DATAXFEROBJ_H_
```

IAS.H (SIMPDND Sample)

```
//**********************************************************************
// File name: IAS.H
//
//      Definition of CAdviseSink
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************
#if !defined( _IAS_H_ )
#define _IAS_H_

#include <assert.h>

class CSimpleSite;

interface CAdviseSink : public IAdviseSink
{
     int m_nCount;
     CSimpleSite FAR * m_pSite;

     CAdviseSink(CSimpleSite FAR * pSite) {
           OutputDebugString("In IAS's constructor\r\n");
           m_pSite = pSite;
           m_nCount = 0;
           };

     ~CAdviseSink() {
           OutputDebugString("In IAS's destructor\r\n");
           assert(m_nCount == 0);
           } ;

     STDMETHODIMP QueryInterface (REFIID riid, LPVOID FAR* ppv);
     STDMETHODIMP_(ULONG) AddRef ();
     STDMETHODIMP_(ULONG) Release ();

     // *** IAdviseSink methods ***
     STDMETHODIMP_(void) OnDataChange (FORMATETC FAR* pFormatetc, STGMEDIUM
FAR* pStgmed);
     STDMETHODIMP_(void) OnViewChange (DWORD dwAspect, LONG lindex);
     STDMETHODIMP_(void) OnRename (LPMONIKER pmk);
     STDMETHODIMP_(void) OnSave ();
     STDMETHODIMP_(void) OnClose ();
};


#endif
```

IDS.H (SIMPDND Sample)

```
//*********************************************************************
// File name: ids.h
//
//      Definition of CDropSource
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//*********************************************************************
#if !defined( _IDS_H_ )
#define _IDS_H_

#include <assert.h>

class CSimpleDoc;

interface CDropSource : public IDropSource
{
     int m_nCount;
     CSimpleDoc FAR * m_pDoc;

     CDropSource(CSimpleDoc FAR * pDoc) {
           OutputDebugString("In IDS's constructor\r\n");
           m_pDoc = pDoc;
           m_nCount = 0;
           };

     ~CDropSource() {
           OutputDebugString("In IDS's destructor\r\n");
           } ;

     STDMETHODIMP QueryInterface (REFIID riid, LPVOID FAR* ppv);
     STDMETHODIMP_(ULONG) AddRef ();
     STDMETHODIMP_(ULONG) Release ();

        // *** IDropSource methods ***
     STDMETHODIMP QueryContinueDrag (BOOL fEscapePressed, DWORD
grfKeyState);
     STDMETHODIMP GiveFeedback (DWORD dwEffect);

private:

};


#endif
```

IDT.H (SIMPDND Sample)

```
//*********************************************************************
// File name: idt.h
//
//      Definition of CDropTarget
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//*********************************************************************
#if !defined( _IDT_H_ )
#define _IDT_H_

#include <assert.h>

class CSimpleDoc;

/* Flags to control direction for drag scrolling */
typedef enum tagSCROLLDIR {
     SCROLLDIR_NULL          = 0,
     SCROLLDIR_UP            = 1,
     SCROLLDIR_DOWN          = 2,
     SCROLLDIR_RIGHT         = 3,
     SCROLLDIR_LEFT          = 4
} SCROLLDIR;

interface CDropTarget : public IDropTarget
{
     int   m_nCount;                 // reference count
     CSimpleDoc FAR * m_pDoc;
     BOOL  m_fCanDropCopy;
     BOOL  m_fCanDropLink;
     DWORD m_dwSrcAspect;
     RECT  m_rcDragRect;
     POINT m_ptLast;
     BOOL  m_fDragFeedbackDrawn;
     DWORD m_dwTimeEnterScrollArea;  // time of entering scroll border
region
     DWORD m_dwLastScrollDir;        // current dir for drag scroll
     DWORD m_dwNextScrollTime;       // time for next scroll

     CDropTarget(CSimpleDoc FAR * pDoc) {
           OutputDebugString("In IDT's constructor\r\n");
           m_pDoc = pDoc;
           m_nCount = 0;
           m_fCanDropCopy = FALSE;
           m_fCanDropLink = FALSE;
           m_fDragFeedbackDrawn = FALSE;
           m_dwTimeEnterScrollArea = 0L;
           m_dwNextScrollTime = 0L;
           m_dwLastScrollDir = SCROLLDIR_NULL;
           };

     ~CDropTarget() {
           OutputDebugString("In IDT's destructor\r\n");
```

```
            assert(m_nCount == 0);
            } ;

     STDMETHODIMP QueryInterface (REFIID riid, LPVOID FAR* ppv);
     STDMETHODIMP_(ULONG) AddRef ();
     STDMETHODIMP_(ULONG) Release ();

        // *** IDropTarget methods ***
     STDMETHODIMP DragEnter (LPDATAOBJECT pDataObj, DWORD grfKeyState,
                 POINTL pt, LPDWORD pdwEffect);
     STDMETHODIMP DragOver  (DWORD grfKeyState, POINTL pt, LPDWORD
pdwEffect);
     STDMETHODIMP DragLeave ();
     STDMETHODIMP Drop (LPDATAOBJECT pDataObj, DWORD grfKeyState, POINTL pt,
                 LPDWORD pdwEffect);

private:
     // Drag/Drop support methods
     BOOL QueryDrop (DWORD grfKeyState, POINTL pointl, BOOL fDragScroll,
                 LPDWORD lpdwEffect);
     BOOL DoDragScroll( POINTL pointl );
     void InitDragFeedback(LPDATAOBJECT pDataObj, POINTL pointl);
     void DrawDragFeedback( POINTL pointl );
     void UndrawDragFeedback( void );
};

#endif  // _IDT_H_
```

IOCS.H   (SIMPDND Sample)

```
//**********************************************************************
// File name: IOCS.H
//
//      Definition of COleClientSite
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************
#if !defined( _IOCS_H_ )
#define _IOCS_H_

#include <assert.h>

class CSimpleSite;

interface COleClientSite : public IOleClientSite
{
     int m_nCount;
     CSimpleSite FAR * m_pSite;

     COleClientSite(CSimpleSite FAR * pSite) {
           OutputDebugString("In IOCS's constructor\r\n");
           m_pSite = pSite;
           m_nCount = 0;
           }

     ~COleClientSite() {
           OutputDebugString("In IOCS's destructor\r\n");
           assert(m_nCount == 0);
           }

     STDMETHODIMP QueryInterface(REFIID riid, LPVOID FAR* ppvObj);
     STDMETHODIMP_(ULONG) AddRef();
     STDMETHODIMP_(ULONG) Release();

     // *** IOleClientSite methods ***
     STDMETHODIMP SaveObject();
     STDMETHODIMP GetMoniker(DWORD dwAssign, DWORD dwWhichMoniker, LPMONIKER
FAR* ppmk);
     STDMETHODIMP GetContainer(LPOLECONTAINER FAR* ppContainer);
     STDMETHODIMP ShowObject();
     STDMETHODIMP OnShowWindow(BOOL fShow);
     STDMETHODIMP RequestNewObjectLayout();
};

#endif
```

PRE.H (SIMPDND Sample)

```
//**********************************************************************
// File name: pre.h
//
//      Used for precompiled headers
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#if !defined( _PRE_H_)
#define _PRE_H_

#include <windows.h>
#include <ole2.h>
#include <ole2ui.h>
#include <assert.h>
#include <string.h>
#include "simpdnd.h"
#include "resource.h"
#include <ole2ver.h>


#endif
```

RESOURCE.H   (SIMPDND Sample)

```
//{{NO_DEPENDENCIES}}
// App Studio generated include file.
// Used by SIMPDND.RC
//
#define IDM_OPEN                        102
#define IDC_DRAGPENDING                 102
#define IDM_SAVE                        103
#define IDM_SAVEAS                      104
#define IDM_PRINT                       105
#define IDM_EXIT                        106
#define IDM_UNDO                        107
#define IDM_CUT                         108
#define IDM_COPY                        109
#define IDM_PASTE                       110
#define ID_EDIT_INSERTOBJECT            111
#define IDM_INSERTOBJECT                111
#define IDM_NEW                         112

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS

#define _APS_NEXT_RESOURCE_VALUE        103
#define _APS_NEXT_COMMAND_VALUE         113
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

SIMPDND.H (SIMPDND Sample)

```
//**********************************************************************
// File name: simpdnd.h
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************
#define IDM_ABOUT 100
#define IDM_INSERT  101
#define IDM_VERB0 1000

int PASCAL WinMain(HANDLE hInstance,HANDLE hPrevInstance,LPSTR lpCmdLine,int
nCmdShow);
BOOL InitApplication(HANDLE hInstance);
BOOL InitInstance(HANDLE hInstance, int nCmdShow);
//@@WTK WIN32, UNICODE
//long FAR PASCAL _export MainWndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam);
long FAR PASCAL EXPORT MainWndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam);
//@@WTK WIN32, UNICODE
//long FAR PASCAL _export DocWndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam);
long FAR PASCAL EXPORT DocWndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam);
//@@WTK WIN32, UNICODE
//BOOL FAR PASCAL _export About(HWND hDlg, unsigned message, WORD wParam,
LONG lParam);
BOOL FAR PASCAL EXPORT About(HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam);
```

SITE.H (SIMPDND Sample)

```
//**********************************************************************
// File name: SITE.H
//
//      Definition of CSimpleSite
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************
#if !defined( _SITE_H_ )
#define _SITE_H_

#include <ole2.h>
#include "ias.h"
#include "iocs.h"

class CSimpleDoc;

class CSimpleSite : public IUnknown
{
public:
     int m_nCount;
     LPOLEOBJECT m_lpOleObject;
     DWORD m_dwDrawAspect;
     SIZEL m_sizel;
     BOOL m_fObjectOpen;
     LPSTORAGE m_lpObjStorage;

     CAdviseSink m_AdviseSink;
     COleClientSite m_OleClientSite;

     CSimpleDoc FAR * m_lpDoc;

     // IUnknown Interfaces
     STDMETHODIMP QueryInterface(REFIID riid, LPVOID FAR* ppvObj);
     STDMETHODIMP_(ULONG) AddRef();
     STDMETHODIMP_(ULONG) Release();

     void InitObject(BOOL fCreateNew);
     static CSimpleSite FAR * Create(CSimpleDoc FAR *lpDoc);
     CSimpleSite(CSimpleDoc FAR *lpDoc);
     ~CSimpleSite();
     void PaintObj(HDC hDC);
     void GetObjRect(LPRECT lpRect);
     void CloseOleObject(void);
     void UnloadOleObject(void);
};

#endif  // _SITE_H_
```

APP.CPP (SIMPDND Sample)

```
//**********************************************************************
// File name: app.cpp
//
//    Implementation file for the CSimpleApp Class
//
// Functions:
//
//    See app.h for a list of member functions.
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "iocs.h"
#include "ias.h"
#include "app.h"
#include "site.h"
#include "doc.h"
```

CSimpleApp::CSimpleApp (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleApp::CSimpleApp()
//
// Purpose:
//
//      Constructor for CSimpleApp
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      SetRectEmpty                Windows API
//
// Comments:
//
//********************************************************************
CSimpleApp::CSimpleApp()
{
        OutputDebugString("In CSimpleApp's Constructor \r\n");

        // clear members
        m_hAppWnd = NULL;
        m_hInst = NULL;
        m_lpDoc = NULL;

        // clear flags
        m_fInitialized = FALSE;
}
```

CSimpleApp::~CSimpleApp   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleApp::~CSimpleApp()
//
// Purpose:
//
//      Destructor for CSimpleApp Class.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      OleUninitialize             OLE API
//
// Comments:
//
//********************************************************************

CSimpleApp::~CSimpleApp()
{
        OutputDebugString("In CSimpleApp's Destructor\r\n");

        // need to uninit the library...
        if (m_fInitialized)
                OleUninitialize();
}
```

CSimpleApp::DestroyDocs (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleApp::DestroyDocs()
//
// Purpose:
//
//      Destroys all of the open documents in the application (Only one
//      since this is an SDI app, but could easily be modified to
//      support MDI).
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
// Comments:
//
//********************************************************************

void CSimpleApp::DestroyDocs()
{
        CStabilize stabilize(this);
        m_lpDoc->Close();   // we have only 1 document
}
```

CSimpleApp::QueryInterface (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleApp::QueryInterface
//
// Purpose:
//
//      Used for interface negotiation at the Frame level.
//
// Parameters:
//
//      REFIID riid         -   A reference to the interface that is
//                              being queried.
//
//      LPVOID FAR* ppvObj  -   An out parameter to return a pointer to
//                              the interface.
//
// Return Value:
//
//      S_OK    -   The interface is supported.
//      S_FALSE -   The interface is not supported
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP CSimpleApp::QueryInterface(REFIID riid, LPVOID FAR* ppvObj)
{
        OutputDebugString("In CSimpleApp::QueryInterface\r\n");

        *ppvObj = NULL;     // must set out pointer parameters to NULL

        // Not a supported interface
        return ResultFromScode(E_NOINTERFACE);
}
```

CSimpleApp::AddRef (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleApp::AddRef
//
// Purpose:
//
//      Adds to the reference count at the Application level.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the application.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Due to the reference counting model that is used in this
//      implementation, this reference count is the sum of the
//      reference counts on all interfaces of all objects open
//      in the application.
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpleApp::AddRef()
{
        OutputDebugString("In CSimpleApp::AddRef\r\n");
        return SafeAddRef();
}
```

CSimpleApp::Release   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleApp::Release
//
// Purpose:
//
//      Decrements the reference count at this level
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the application.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpleApp::Release()
{
        OutputDebugString("In CSimpleApp::Release\r\n");

        return SafeRelease();
}
```

CSimpleApp::fInitApplication   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleApp::fInitApplication
//
// Purpose:
//
//      Initializes the application
//
// Parameters:
//
//      HANDLE hInstance    -   Instance handle of the application.
//
// Return Value:
//
//      TRUE    -   Application was successfully initialized.
//      FALSE   -   Application could not be initialized
//
// Function Calls:
//      Function                    Location
//
//      LoadIcon                    Windows API
//      LoadCursor                  Windows API
//      GetStockObject              Windows API
//      RegisterClass               Windows API
//
// Comments:
//
//********************************************************************

BOOL CSimpleApp::fInitApplication(HANDLE hInstance)
{
        WNDCLASS  wc;
        CStabilize stabilize(this);

        // Fill in window class structure with parameters that describe the
        // main window.

        wc.style = NULL;                    // Class style(s).
        wc.lpfnWndProc = MainWndProc;       // Function to retrieve messages
for

// windows of this class.
        wc.cbClsExtra = 0;                  // No per-class extra data.
        wc.cbWndExtra = 0;                  // No per-window extra data.
        wc.hInstance = hInstance;           // Application that owns the
class.
        wc.hIcon = LoadIcon(hInstance,"SimpDnd");
        wc.hCursor = LoadCursor(NULL, IDC_ARROW);
        wc.hbrBackground = GetStockObject(WHITE_BRUSH);
        wc.lpszMenuName =  "SIMPLEMENU";    // Name of menu resource in .RC
file.
```

```
        wc.lpszClassName = "SimpDndAppWClass";  // Name used in CreateWindow
call.

        if (!RegisterClass(&wc))
                return FALSE;

        wc.style = CS_DBLCLKS;              // Class style(s). allow
DBLCLK's
        wc.lpfnWndProc = DocWndProc;        // Function to retrieve messages
for

// windows of this class.
        wc.cbClsExtra = 0;                  // No per-class extra data.
        wc.cbWndExtra = 0;                  // No per-window extra data.
        wc.hInstance = hInstance;           // Application that owns the
class.
        wc.hIcon = NULL;
        wc.hCursor = LoadCursor(NULL, IDC_ARROW);
        wc.hbrBackground = GetStockObject(WHITE_BRUSH);
        wc.lpszMenuName =  NULL;
        wc.lpszClassName = "SimpDndDocWClass"; // Name used in CreateWindow
call.

        // Register the window class and return success/failure code.

        return (RegisterClass(&wc));
}
```

CSimpleApp::fInitInstance (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleApp::fInitInstance
//
// Purpose:
//
//      Instance initialization.
//
// Parameters:
//
//      HANDLE hInstance    -   App. Instance Handle.
//
//      int nCmdShow        -   Show parameter from WinMain
//
// Return Value:
//
//      TRUE    -   Initialization Successful
//      FALSE   -   Initialization Failed.
//
//
// Function Calls:
//      Function                    Location
//
//      CreateWindow                Windows API
//      ShowWindow                  Windows API
//      UpdateWindow                Windows API
//      OleBuildVersion             OLE API
//      OleInitialize               OLE API
//
// Comments:
//
//      Note that successful Initalization of the OLE libraries
//      is remembered so the UnInit is only called if needed.
//
//********************************************************************

BOOL CSimpleApp::fInitInstance (HANDLE hInstance, int nCmdShow)
{
        CStabilize stabilize(this);
        DWORD dwVer = OleBuildVersion();
        LPMALLOC lpMalloc = NULL;

        // check to see if we are compatible with this version of the
libraries
        if (HIWORD(dwVer) != rmm || LOWORD(dwVer) < rup) {
#ifdef _DEBUG
                OutputDebugString("WARNING: Incompatible OLE library
version\r\n");
#else
                return FALSE;
#endif
        }
```

```
#if defined( _DEBUG )
        /* OLE2NOTE: Use a special debug allocator to help track down
        **    memory leaks.
        */
        OleStdCreateDbAlloc(0, &lpMalloc);
#endif

        if (SUCCEEDED(OleInitialize(lpMalloc)))
        {
            m_fInitialized = TRUE;
        }
        else
        {
            // Replacing the allocator may not be legal.
            // Try again using the default allocator.
            if (SUCCEEDED(OleInitialize(NULL)))
            {
                m_fInitialized = TRUE;
            }
        }

#if defined( _DEBUG )
        /* OLE2NOTE: release the special debug allocator so that only OLE is
        **    holding on to it. later when OleUninitialize is called, then
        **    the debug allocator object will be destroyed. when the debug
        **    allocator object is destoyed, it will report (to the Output
        **    Debug Terminal) whether there are any memory leaks.
        */
        if (lpMalloc) lpMalloc->Release();
#endif

        m_hInst = hInstance;

        // Create the "application" windows
        m_hAppWnd = CreateWindow ("SimpDndAppWClass",
                                                          "Simple OLE 2.0
Drag/Drop Container",

WS_OVERLAPPEDWINDOW,
                                                          CW_USEDEFAULT,
                                                          CW_USEDEFAULT,
                                                          CW_USEDEFAULT,
                                                          CW_USEDEFAULT,
                                                          NULL,
                                                          NULL,
                                                          hInstance,
                                                          NULL);

        if (!m_hAppWnd)
                return FALSE;

        // delay before dragging should start, in milliseconds
        m_nDragDelay = GetProfileInt(
                        "windows",
                        "DragDelay",
                        DD_DEFDRAGDELAY
        );

        // minimum distance (radius) before drag should start, in pixels
        m_nDragMinDist = GetProfileInt(
                        "windows",
                        "DragMinDist",
                        DD_DEFDRAGMINDIST
        );

        // delay before scrolling, in milliseconds
        m_nScrollDelay = GetProfileInt(
                        "windows",
                        "DragScrollDelay",
                        DD_DEFSCROLLDELAY
        );

        // inset-width of the hot zone, in pixels
        m_nScrollInset = GetProfileInt(
                        "windows",
                        "DragScrollInset",
                        DD_DEFSCROLLINSET
        );

        // scroll interval, in milliseconds
        m_nScrollInterval = GetProfileInt(
                        "windows",
                        "DragScrollInterval",
                        DD_DEFSCROLLINTERVAL
        );

        ShowWindow (m_hAppWnd, nCmdShow);
        UpdateWindow (m_hAppWnd);

        return m_fInitialized;
}
```

CSimpleApp::lCommandHandler   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleApp::lCommandHandler
//
// Purpose:
//
//      Handles the processing of WM_COMMAND.
//
// Parameters:
//
//      HWND hWnd       -   Handle to the application Window
//
//      UINT message    -   message (always WM_COMMAND)
//
//      WPARAM wParam   -   Same as passed to the WndProc
//
//      LPARAM lParam   -   Same as passed to the WndProc
//
// Return Value:
//
//      NULL
//
// Function Calls:
//      Function                                    Location
//
//      IOleObject::DoVerb                          Object
//      GetClientRect                               Windows API
//      MessageBox                                  Windows API
//      DialogBox                                   Windows API
//      MakeProcInstance                            Windows API
//      FreeProcInstance                            Windows API
//      SendMessage                                 Windows API
//      DefWindowProc                               Windows API
//      CSimpleDoc::InsertObject                    DOC.CPP
//
// Comments:
//
//********************************************************************

long CSimpleApp::lCommandHandler (HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
        RECT rect;
        CStabilize stabilize(this);

        // see if the command is a verb selections
        //@@WTK WIN32, UNICODE
        //if (wParam >= IDM_VERB0)
        if (LOWORD(wParam) >= IDM_VERB0)
                {
                // get the rectangle of the object
                m_lpDoc->m_lpSite->GetObjRect(&rect);
                m_lpDoc->m_lpSite->m_lpOleObject->DoVerb(
                                LOWORD(wParam) - IDM_VERB0, NULL,
                                &m_lpDoc->m_lpSite->m_OleClientSite, -1,
                                m_lpDoc->m_hDocWnd, &rect);
                }
        else
                {
                //@@WTK WIN32, UNICODE
                //switch (wParam) {
                switch (LOWORD(wParam)) {
                        // bring up the About box
                        case IDM_ABOUT:
                                {
                                FARPROC lpProcAbout =
MakeProcInstance((FARPROC)About, m_hInst);

                                DialogBox(m_hInst,               // current
instance
                                        "AboutBox",                 //
resource to use
                                        m_hAppWnd,                  //
parent handle
                                        lpProcAbout);               //
About() instance address

                                FreeProcInstance(lpProcAbout);
                                break;
                                }

                        // bring up the InsertObject Dialog
                        case IDM_INSERTOBJECT:
                                m_lpDoc->InsertObject();
                                break;

                        // Copy the object to the Clipboard
                        case IDM_COPY:
                                m_lpDoc->CopyObjectToClip();
                                break;

                        // exit the application
                        case IDM_EXIT:
                                SendMessage(hWnd, WM_SYSCOMMAND, SC_CLOSE,
0L);

                                break;

                        case IDM_NEW:
                                m_lpDoc->Close();
                                m_lpDoc = NULL;
                                lCreateDoc(hWnd, 0, 0, 0);
                                break;

                        default:
                                return (DefWindowProc(hWnd, message, wParam,
lParam));
```

```
                        }   // end of switch
                }  // end of else
        return NULL;
}
```

CSimpleApp::lSizeHandler   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleApp::lSizeHandler
//
// Purpose:
//
//      Handles the WM_SIZE message
//
// Parameters:
//
//      HWND hWnd       -   Handle to the application Window
//
//      UINT message    -   message (always WM_SIZE)
//
//      WPARAM wParam   -   Same as passed to the WndProc
//
//      LPARAM lParam   -   Same as passed to the WndProc
//
// Return Value:
//
//      LONG    -   returned from the "document" resizing
//
// Function Calls:
//      Function                    Location
//
//      GetClientRect               Windows API
//      CSimpleDoc::lResizeDoc      DOC.CPP
//
// Comments:
//
//********************************************************************

long CSimpleApp::lSizeHandler (HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
        RECT rect;

        CStabilize stabilize(this);
        GetClientRect(m_hAppWnd, &rect);
        return m_lpDoc->lResizeDoc(&rect);
}
```

CSimpleApp::lCreateDoc (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleApp::lCreateDoc
//
// Purpose:
//
//      Handles the creation of a document.
//
// Parameters:
//
//      HWND hWnd       -   Handle to the application Window
//
//      UINT message    -   message (always WM_CREATE)
//
//      WPARAM wParam   -   Same as passed to the WndProc
//
//      LPARAM lParam   -   Same as passed to the WndProc
//
// Return Value:
//
//      NULL
//
// Function Calls:
//      Function                    Location
//
//      GetClientRect               Windows API
//      CSimpleDoc::CSimpleDoc      DOC.CPP
//
// Comments:
//
//********************************************************************

long CSimpleApp::lCreateDoc (HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
        RECT rect;

        CStabilize stabilize(this);
        GetClientRect(hWnd, &rect);

        m_lpDoc = CSimpleDoc::Create(this, &rect, hWnd);

        return NULL;
}
```

CSimpleApp::HandleAccelerators   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleApp::HandleAccelerators
//
// Purpose:
//
//      To properly handle accelerators in the Message Loop
//
// Parameters:
//
//      LPMSG lpMsg -   A pointer to the message structure.
//
// Return Value:
//
//      TRUE    -   The accelerator was handled
//      FALSE   -   The accelerator was not handled
//
// Function Calls:
//      Function                                        Location
//
// Comments:
//
//********************************************************************

BOOL CSimpleApp::HandleAccelerators(LPMSG lpMsg)
{
        BOOL retval = FALSE;

        // we do not have any accelerators

        return retval;
}
```

CSimpleApp::PaintApp (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleApp::PaintApp
//
// Purpose:
//
//      Handles the painting of the doc window.
//
//
// Parameters:
//
//      HDC hDC -   hDC to the Doc Window.
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      CSimpleDoc::PaintDoc        DOC.CPP
//
// Comments:
//
//      This is an app level function in case we want to do palette
//      management.
//
//********************************************************************

void CSimpleApp::PaintApp (HDC hDC)
{
        CStabilize stabilize(this);
        // at this level, we could enumerate through all of the
        // visible objects in the application, so that a palette
        // that best fits all of the objects can be built.

        // This app is designed to take on the same palette
        // functionality that was provided in OLE 1.0, the palette
        // of the last object drawn is realized.  Since we only
        // support one object at a time, it shouldn't be a big
        // deal.

        // if we supported multiple documents, we would enumerate
        // through each of the open documents and call paint.

        if (m_lpDoc)
                m_lpDoc->PaintDoc(hDC);

}
```

DOC.CPP   (SIMPDND Sample)

```
//**********************************************************************
// File name: DOC.CPP
//
//      Implementation file for CSimpleDoc.
//
// Functions:
//
//      See DOC.H for Class Definition
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "iocs.h"
#include "ias.h"
#include "app.h"
#include "site.h"
#include "doc.h"
#include "idt.h"
#include "dxferobj.h"
```

CSimpleDoc::Create   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleDoc::Create
//
// Purpose:
//
//      Creation for the CSimpleDoc Class
//
// Parameters:
//
//      CSimpleApp FAR * lpApp  -   Pointer to the CSimpleApp Class
//
//      LPRECT lpRect           -   Client area rect of "frame" window
//
//      HWND hWnd               -   Window Handle of "frame" window
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      StgCreateDocfile            OLE API
//      RegisterDragDrop            OLE API
//      CoLockObjectExternal        OLE API
//      CreateWindow                Windows API
//      ShowWindow                  Windows API
//      UpdateWindow                Windows API
//
// Comments:
//
//      This routine was added so that failure could be returned
//      from object creation.
//
//********************************************************************

CSimpleDoc FAR * CSimpleDoc::Create(CSimpleApp FAR *lpApp, LPRECT
lpRect,HWND hWnd)
{
        CSimpleDoc FAR * lpTemp = new CSimpleDoc(lpApp, hWnd);

        if (!lpTemp)
                return NULL;

        // create storage for the doc.
        HRESULT hErr = StgCreateDocfile (
                NULL,       // generate temp name
                STGM_READWRITE | STGM_TRANSACTED | STGM_SHARE_EXCLUSIVE,
                0, &lpTemp->m_lpStorage);

        if (hErr != NOERROR)
```

```
                goto error;

        // create the document Window
        lpTemp->m_hDocWnd = CreateWindow(
                        "SimpDndDocWClass",
                        NULL,
                        WS_CHILD | WS_CLIPCHILDREN,
                        lpRect->left,
                        lpRect->top,
                        lpRect->right,
                        lpRect->bottom,
                        hWnd,
                        NULL,
                        lpApp->m_hInst,
                        NULL);

        if (!lpTemp->m_hDocWnd)
                goto error;

        ShowWindow(lpTemp->m_hDocWnd, SW_SHOWNORMAL);  // Show the window
        UpdateWindow(lpTemp->m_hDocWnd);               // Sends WM_PAINT
message

        // Ensable InsertObject menu choice
        EnableMenuItem( lpApp->m_hEditMenu, 1, MF_BYPOSITION | MF_ENABLED);
        // Disable Copy menu choice
        EnableMenuItem( lpApp->m_hEditMenu, 0, MF_BYPOSITION | MF_DISABLED |
MF_GRAYED);

        // It is *REQUIRED* to hold a strong LOCK on the object that is
        // registered as drop target. this call will result in at least one
        // ref count held on our document. later in CSimpleDoc::Close we
will
        // unlock this lock which will make our document's ref count go to
0.
        // when the document's ref count goes to 0, it will be deleted.
        CoLockObjectExternal (&lpTemp->m_DropTarget, TRUE, 0);

        // Register our window as a DropTarget
        RegisterDragDrop(lpTemp->m_hDocWnd, &lpTemp->m_DropTarget);
        lpTemp->m_fRegDragDrop = TRUE;

        return (lpTemp);

error:
        delete (lpTemp);
        return NULL;

}
```

CSimpleDoc::Close   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleDoc::Close
//
// Purpose:
//
//      Close CSimpleDoc object.
//      when the document's reference count goes to 0, the document
//      will be destroyed.
//
// Parameters:
//
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      RevokeDragDrop              OLE API
//      CoLockObjectExternal        OLE API
//      OleFlushClipboard           OLE API
//      ShowWindow                  Windows API
//
// Comments:
//
//********************************************************************

void CSimpleDoc::Close(void)
{
        OutputDebugString("In CSimpleDoc::Close\r\n");
        CStabilize stabilize(this);
        ShowWindow(m_hDocWnd, SW_HIDE);  // Hide the window

        // Remove our data transfer object from clipboard if it is there.
        //  this will leave HGLOBAL based data behind on the clipboard
        //  including OLE 1.0 compatibility formats.
        OleFlushClipboard();

        // Revoke our window as a DropTarget
        if (m_fRegDragDrop) {
                RevokeDragDrop(m_hDocWnd);
                m_fRegDragDrop = FALSE;
        }

        // Close the OLE object in our document
        if (m_lpSite)
                m_lpSite->CloseOleObject();

        // Unlock the lock added in CSimpleDoc::Create. this will make
        // the document's ref count go to 0, and the document will be
deleted.
        CoLockObjectExternal (&m_DropTarget, FALSE, TRUE);
}
```

CSimpleDoc::CSimpleDoc   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleDoc::CSimpleDoc
//
// Purpose:
//
//      Constructor for the CSimpleDoc Class
//
// Parameters:
//
//      CSimpleApp FAR * lpApp  -   Pointer to the CSimpleApp Class
//
//      HWND hWnd               -   Window Handle of "frame" window
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      GetMenu                     Windows API
//      GetSubMenu                  Windows API
//
// Comments:
//
//********************************************************************
#pragma warning(disable : 4355)  // turn off this warning.  This warning
                                                                // tells us
that we are passing this in
                                                                // an
initializer, before "this" is through
                                                                //
initializing.  This is ok, because
                                                                // we just
store the ptr in the other
                                                                //
constructor

CSimpleDoc::CSimpleDoc(CSimpleApp FAR * lpApp,HWND hWnd)
                : m_DropTarget(this), m_DropSource(this)
#pragma warning (default : 4355)  // Turn the warning back on
{
        OutputDebugString("In CSimpleDoc's Constructor\r\n");
        m_lpApp = lpApp;
        m_lpSite = NULL;
        // set up menu handles
        lpApp->m_hMainMenu = GetMenu(hWnd);
        lpApp->m_hFileMenu = GetSubMenu(lpApp->m_hMainMenu, 0);
        lpApp->m_hEditMenu = GetSubMenu(lpApp->m_hMainMenu, 1);
        lpApp->m_hHelpMenu = GetSubMenu(lpApp->m_hMainMenu, 2);
        lpApp->m_hCascadeMenu = NULL;
        m_fModifiedMenu = FALSE;

        // drag/drop related stuff
        m_fRegDragDrop = FALSE;       // is doc registered as drop target?
        m_fLocalDrag = FALSE;         // is doc source of the drag
        m_fLocalDrop = FALSE;         // was doc target of the drop
        m_fCanDropCopy = FALSE;       // is Drag/Drop copy/move possible?
        m_fCanDropLink = FALSE;       // is Drag/Drop link possible?
        m_fDragLeave = FALSE;         // has drag left
        m_fPendingDrag = FALSE;       // LButtonDown--possible drag pending
        m_ptButDown.x = m_ptButDown.y = 0; // LButtonDown coordinates
}
```

CSimpleDoc::~CSimpleDoc   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleDoc::~CSimpleDoc
//
// Purpose:
//
//      Destructor for CSimpleDoc
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpleSite::Release        SITE.CPP
//      IStorage::Release           OLE API
//
// Comments:
//
//********************************************************************

CSimpleDoc::~CSimpleDoc()
{
        OutputDebugString("In CSimpleDoc's Destructor\r\n");

        // Release all pointers we hold to the OLE object. also release
        // the ref count added in CSimpleSite::Create. this will make
        // the Site's ref count go to 0, and the Site will be deleted.
        if (m_lpSite) {
                m_lpSite->UnloadOleObject();
                m_lpSite->Release();
                m_lpSite = NULL;
        }

        // Release the Storage
        if (m_lpStorage) {
                m_lpStorage->Release();
                m_lpStorage = NULL;
        }

        // if the edit menu was modified, remove the menu item and
        // destroy the popup if it exists
        if (m_fModifiedMenu)
                {
                int nCount = GetMenuItemCount(m_lpApp->m_hEditMenu);
                RemoveMenu(m_lpApp->m_hEditMenu, nCount-1, MF_BYPOSITION);
                if (m_lpApp->m_hCascadeMenu)
                        DestroyMenu(m_lpApp->m_hCascadeMenu);
                }

        DestroyWindow(m_hDocWnd);
}
```

CSimpleDoc::QueryInterface   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleDoc::QueryInterface
//
// Purpose:
//
//      Return a pointer to a requested interface
//
// Parameters:
//
//      REFIID riid         -   ID of interface to be returned
//      LPVOID FAR* ppvObj  -   Location to return the interface
//
// Return Value:
//
//      S_OK                -   Interface supported
//      E_NOINTERFACE       -   Interface NOT supported
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP CSimpleDoc::QueryInterface(REFIID riid, LPVOID FAR* ppvObj)
{
        OutputDebugString("In CSimpleDoc::QueryInterface\r\n");

        *ppvObj = NULL;     // must set out pointer parameters to NULL

        // looking for IUnknown
        if ( riid == IID_IUnknown)
                {
                AddRef();
                *ppvObj = this;
                return ResultFromScode(S_OK);
                }

        // looking for IDropTarget
        if ( riid == IID_IDropTarget)
                {
                m_DropTarget.AddRef();
                *ppvObj=&m_DropTarget;
                return ResultFromScode(S_OK);
                }

        // looking for IDropSource
        if ( riid == IID_IDropSource)
```

```
                {
                m_DropSource.AddRef();
                *ppvObj=&m_DropSource;
                return ResultFromScode(S_OK);
                }

        // Not a supported interface
        return ResultFromScode(E_NOINTERFACE);
}
```

CSimpleDoc::AddRef   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleDoc::AddRef
//
// Purpose:
//
//      Increments the document reference count
//
// Parameters:
//
//      None
//
// Return Value:
//
//      UINT    -   The current reference count on the document
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpleApp::AddRef          APP.CPP
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpleDoc::AddRef()
{
        OutputDebugString("In CSimpleDoc::AddRef\r\n");
        return SafeAddRef();
}
```

CSimpleDoc::Release (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleDoc::Release
//
// Purpose:
//
//      Decrements the document reference count
//
// Parameters:
//
//      None
//
// Return Value:
//
//      UINT    -   The current reference count on the document
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpleDoc::Release()
{
        OutputDebugString("In CSimpleDoc::Release\r\n");

        return SafeRelease();
}
```

CSimpleDoc::InsertObject (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleDoc::InsertObject
//
// Purpose:
//
//      Inserts a new object to this document
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      CSimpleSite::CSimpleSite    SITE.CPP
//      CSimpleSite::InitObject     SITE.CPP
//      memset                      C Runtime
//      OleUIInsertObject           OUTLUI function
//      CSimpleDoc::DisableInsertObject DOC.CPP
//
// Comments:
//
//      This implementation only allows one object to be inserted
//      into a document.  Once the object has been inserted, then
//      the Insert Object menu choice is greyed out, to prevent
//      the user from inserting another.
//
//********************************************************************

void CSimpleDoc::InsertObject()
{
        OLEUIINSERTOBJECT io;
        CStabilize stabilize(this);
        UINT iret;
        //@@WTK WIN32, UNICODE
        //char szFile[OLEUI_CCHPATHMAX];
        OLECHAR szFile[OLEUI_CCHPATHMAX];

        m_lpSite = CSimpleSite::Create(this);

        // clear the structure
        _fmemset(&io, 0, sizeof(OLEUIINSERTOBJECT));

        // fill the structure
        io.cbStruct = sizeof(OLEUIINSERTOBJECT);
        io.dwFlags = IOF_SELECTCREATENEW |
                                        IOF_DISABLELINK |
IOF_DISABLEDISPLAYASICON |
                                        IOF_CREATENEWOBJECT |
IOF_CREATEFILEOBJECT;
        io.hWndOwner = m_hDocWnd;
        io.lpszCaption = OLESTR("Insert Object");
        io.iid = IID_IOleObject;
        io.oleRender = OLERENDER_DRAW;
        io.lpIOleClientSite = &m_lpSite->m_OleClientSite;
        io.lpIStorage = m_lpSite->m_lpObjStorage;
        io.ppvObj = (LPVOID FAR *)&m_lpSite->m_lpOleObject;
        io.lpszFile = szFile;
        //@@WTK WIN32, UNICODE
        //io.cchFile = sizeof(szFile);
        io.cchFile = sizeof(szFile) / sizeof *szFile;
        //@@WTK WIN32, UNICODE
        //_fmemset((LPSTR)szFile, 0, sizeof(szFile));
        _fmemset(szFile, 0, sizeof(szFile));

        // call OUTLUI to do all the hard work
        iret = OleUIInsertObject(&io);

        if (iret == OLEUI_OK)
                {
                m_lpSite->InitObject((BOOL)(io.dwFlags &
IOF_SELECTCREATENEW));
                // disable Insert Object menu item
                DisableInsertObject();
                }
        else
                {
                m_lpSite->Release();
                m_lpSite = NULL;
                m_lpStorage->Revert();
                }

}
```

CSimpleDoc::lResizeDoc (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleDoc::lResizeDoc
//
// Purpose:
//
//      Resizes the document
//
// Parameters:
//
//      LPRECT lpRect   -   The size of the client are of the "frame"
//                          Window.
//
// Return Value:
//
//      NULL
//
// Function Calls:
//      Function                                Location
//
//      MoveWindow                              Windows API
//
// Comments:
//
//********************************************************************

long CSimpleDoc::lResizeDoc(LPRECT lpRect)
{
        MoveWindow(
                        m_hDocWnd,
                        lpRect->left, lpRect->top,
                        lpRect->right, lpRect->bottom, TRUE);

        return NULL;
}
```

CSimpleDoc::lAddVerbs   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleDoc::lAddVerbs
//
// Purpose:
//
//      Adds the objects verbs to the edit menu.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      NULL
//
// Function Calls:
//      Function                    Location
//
//      GetMenuItemCount            Windows API
//      OleUIAddVerbMenu            OUTLUI function
//
// Comments:
//
//********************************************************************

long CSimpleDoc::lAddVerbs(void)
{
        CStabilize stabilize(this);
        // m_fModifiedMenu is TRUE if the menu has already been modified
        // once.  Since we only support one obect every time the application
        // is run, then once the menu is modified, it doesn't have
        // to be done again.
        if (m_lpSite && !m_fModifiedMenu)
                {
                int nCount = GetMenuItemCount(m_lpApp->m_hEditMenu);

                OleUIAddVerbMenu ( m_lpSite->m_lpOleObject,
                                                   NULL,
                                                   m_lpApp->m_hEditMenu,
                                                   nCount + 1,
                                                   IDM_VERB0,
                                                   0,            // no
maximum verb IDM enforced
                                                   FALSE,
                                                   1,
                                                   &m_lpApp-
>m_hCascadeMenu);

                m_fModifiedMenu = TRUE;
                }
        return (NULL);
}
```

CSimpleDoc::PaintDoc   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleDoc::PaintDoc
//
// Purpose:
//
//      Paints the Document
//
// Parameters:
//
//      HDC hDC -   hDC of the document Window
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      CSimpleSite::PaintObj       SITE.CPP
//
// Comments:
//
//********************************************************************

void CSimpleDoc::PaintDoc (HDC hDC)
{
        CStabilize stabilize(this);
        // if we supported multiple objects, then we would enumerate
        // the objects and call paint on each of them from here.

        if (m_lpSite)
                m_lpSite->PaintObj(hDC);

}
```

CSimpleDoc::DisableInsertObject (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleDoc::DisableInsertObject
//
// Purpose:
//
//      Disable the ability to insert a new object in this document.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      RevokeDragDrop              OLE API
//      EnableMenuItem              Windows API
//
// Comments:
//
//      This implementation only allows one object to be inserted
//      into a document.  Once the object has been inserted, then
//      the Insert Object menu choice is greyed out, to prevent
//      the user from inserting another. Also we revoke ourself as
//      a potential drop target.
//
//********************************************************************

void CSimpleDoc::DisableInsertObject(void)
{
        // Disable InsertObject menu choice
        EnableMenuItem( m_lpApp->m_hEditMenu, 1, MF_BYPOSITION | MF_DISABLED
| MF_GRAYED);
        // Enable Copy menu choice
        EnableMenuItem( m_lpApp->m_hEditMenu, 0, MF_BYPOSITION |
MF_ENABLED);

        // We no longer accept dropping of objects
        if (m_fRegDragDrop) {
                RevokeDragDrop(m_hDocWnd);
                m_fRegDragDrop = FALSE;
        }
}
```

CSimpleDoc::CopyObjectToClip   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleDoc::CopyObjectToClip
//
// Purpose:
//
//      Copy the embedded OLE object to the clipboard
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      CDataXferObj::Create        DXFEROBJ.CPP
//      CDataXferObj::QueryInterface DXFEROBJ.CPP
//      OleSetClipboard             OLE API
//
// Comments:
//
//      This implementation only allows one object to be inserted
//      into a document.  Once the object has been inserted, then
//      the Copy menu choice is enabled.
//
//********************************************************************

void CSimpleDoc::CopyObjectToClip(void)
{
        LPDATAOBJECT lpDataObj;

        // Create a data transfer object by cloning the existing OLE object
        CDataXferObj FAR* pDataXferObj =
CDataXferObj::Create(m_lpSite,NULL);
        if (! pDataXferObj) {
                MessageBox(NULL,"Out-of-memory","SimpDnD",MB_SYSTEMMODAL|
MB_ICONHAND);
                return;
        }
        // initially obj is created with 0 refcnt. this QI will make it go
to 1.
        pDataXferObj->QueryInterface(IID_IDataObject, (LPVOID
FAR*)&lpDataObj);

        // put out data transfer object on the clipboard. this API will
AddRef.
        OleSetClipboard(lpDataObj);
        // Give ownership of data transfer object to clipboard
        pDataXferObj->Release();
}
```

DXFEROBJ.CPP   (SIMPDND Sample)

```
//**********************************************************************
// File name: DXFEROBJ.CPP
//
//       Implementation file for CDataXferObj, data transfer object
//     implementation of IDataObject interface.
//
// Functions:
//
//      See DXFEROBJ.H for class definition
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
//@@WTK WIN32, UNICODE
//#include "..\include\enumfetc.h"
#include <assert.h>
#include "dxferobj.h"
#include "site.h"

CLIPFORMAT g_cfEmbeddedObject = RegisterClipboardFormat(CF_EMBEDDEDOBJECT);
CLIPFORMAT g_cfObjectDescriptor
=RegisterClipboardFormat(CF_OBJECTDESCRIPTOR);

// List of formats offered by our data transfer object via EnumFormatEtc
static FORMATETC s_arrGetFmtEtcs[] =
{
     { g_cfEmbeddedObject, NULL, DVASPECT_CONTENT, -1, TYMED_ISTORAGE},
     { g_cfObjectDescriptor, NULL, DVASPECT_CONTENT, -1, TYMED_HGLOBAL},
     { CF_METAFILEPICT, NULL, DVASPECT_CONTENT, -1, TYMED_MFPICT}
};
```

CDataXferObj::Create (SIMPDND Sample)

```
//**********************************************************************
//
// CDataXferObj::Create
//
// Purpose:
//
//      Creation routine for CDataXferObj
//
// Parameters:
//
//      CSimpleSite FAR *lpSite   - Pointer to source CSimpleSite
//                                  this is the container site of the
//                                  source OLE object to be transfered
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      StgCreateDocfile            OLE API
//      assert                      C Runtime
//
// Comments:
//      reference count of CDataXferObj will be 0 on return.
//
//********************************************************************

CDataXferObj FAR * CDataXferObj::Create(
           CSimpleSite FAR *lpSite,
           POINTL FAR* pPointl
)
{
     CDataXferObj FAR * lpTemp = new CDataXferObj();

     if (!lpTemp)
           return NULL;

     // create a sub-storage for the object
     HRESULT hErr = StgCreateDocfile(
                       NULL,
                       STGM_READWRITE | STGM_DIRECT | STGM_SHARE_EXCLUSIVE |
                             STGM_DELETEONRELEASE,
                       0,
                       &lpTemp->m_lpObjStorage);

     assert(hErr == NOERROR);

     if (hErr != NOERROR)
           {
           delete lpTemp;
```

```
            return NULL;
            }

      // Clone the source object
      if (lpSite->m_lpOleObject) {
            // Object is loaded; ask the object to save into the new storage
            LPPERSISTSTORAGE pPersistStorage;

            lpSite->m_lpOleObject->QueryInterface(IID_IPersistStorage,
                        (LPVOID FAR*)&pPersistStorage);
            assert(pPersistStorage);
            OleSave(pPersistStorage, lpTemp->m_lpObjStorage, FALSE);

            // pass NULL so that object application won't forget the real stg
            pPersistStorage->SaveCompleted(NULL);
            pPersistStorage->Release();
      } else {
            // Object not loaded so use cheaper IStorage CopyTo operation
            lpSite->m_lpObjStorage->CopyTo(0, NULL, NULL, lpTemp-
>m_lpObjStorage);
      }

      OleLoad(lpTemp->m_lpObjStorage, IID_IOleObject, NULL,
                  (LPVOID FAR*)&lpTemp->m_lpOleObject);
      assert(lpTemp->m_lpOleObject);

      lpTemp->m_sizel = lpSite->m_sizel;
      if (pPointl)
            lpTemp->m_pointl = *pPointl;
      else
            lpTemp->m_pointl.x = lpTemp->m_pointl.y = 0;
      return lpTemp;
}
```

CDataXferObj::CDataXferObj   (SIMPDND Sample)

```
//**********************************************************************
//
// CDataXferObj::CDataXferObj
//
// Purpose:
//
//      Constructor for CDataXferObj
//
// Parameters:
//
//      CSimpleDoc FAR *lpDoc   - Pointer to CSimpleDoc
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
// Comments:
//
//********************************************************************

CDataXferObj::CDataXferObj (void)
{
     // clear the reference count
     m_nCount = 0;

     m_lpObjStorage = NULL;
     m_lpOleObject = NULL;
     m_sizel.cx = m_sizel.cy = 0;
     m_pointl.x = m_pointl.y = 0;
}
```

CDataXferObj::~CDataXferObj (SIMPDND Sample)

```
//**********************************************************************
//
// CDataXferObj::~CDataXferObj
//
// Purpose:
//
//      Destructor for CDataXferObj
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                                Location
//
//      OutputDebugString                       Windows API
//      IOleObject::Release                     Object
//      IStorage::Release                       OLE API
//
// Comments:
//
//********************************************************************

CDataXferObj::~CDataXferObj ()
{
     OutputDebugString ("In CDataXferObj's Destructor \r\n");

     if (m_lpOleObject)
        {
        m_lpOleObject->Release();
        m_lpOleObject = NULL;

        // Release the storage for this object
        m_lpObjStorage->Release();
        m_lpObjStorage = NULL;
        }
}
```

CDataXferObj::QueryInterface   (SIMPDND Sample)

```
//**********************************************************************
//
// CDataXferObj::QueryInterface
//
// Purpose:
//
//      Used for interface negotiation of the CDataXferObj instance
//
// Parameters:
//
//      REFIID riid         -   A reference to the interface that is
//                              being queried.
//
//      LPVOID FAR* ppvObj  -   An out parameter to return a pointer to
//                              the interface.
//
// Return Value:
//
//      S_OK            -   The interface is supported.
//      E_NOINTERFACE   -   The interface is not supported
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      IsEqualIID                  OLE API
//      ResultFromScode             OLE API
//      CDataXferObj::AddRef        DXFEROBJ.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP CDataXferObj::QueryInterface(REFIID riid, LPVOID FAR* ppvObj)
{
     OutputDebugString("In CDataXferObj::QueryInterface\r\n");

     if ( riid == IID_IUnknown || riid == IID_IDataObject)
           {
           AddRef();
           *ppvObj = this;
           return NOERROR;
           }

     // unknown interface requested
     *ppvObj = NULL;     // must set out pointer parameters to NULL
     return ResultFromScode(E_NOINTERFACE);
}
```

CDataXferObj::AddRef (SIMPDND Sample)

```
//**********************************************************************
//
// CDataXferObj::AddRef
//
// Purpose:
//
//      Increments the reference count of the CDataXferObj instance
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the object
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) CDataXferObj::AddRef()
{
     OutputDebugString("In CDataXferObj::AddRef\r\n");

     return ++m_nCount;
}
```

CDataXferObj::Release   (SIMPDND Sample)

```
//**********************************************************************
//
// CDataXferObj::Release
//
// Purpose:
//
//      Decrements the reference count of the CDataXferObj object
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the object.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) CDataXferObj::Release()
{
     OutputDebugString("In CDataXferObj::Release\r\n");

     if (--m_nCount == 0) {
           delete this;
           return 0;
     }
     return m_nCount;
}


/*********************************************************************
** This IDataObject implementation is used for data transfer.
**
** The following methods are NOT supported for data transfer:
**      IDataObject::SetData    -- return E_NOTIMPL
**      IDataObject::DAdvise    -- return OLE_E_ADVISENOTSUPPORTED
**                 ::DUnadvise
**                 ::EnumDAdvise
**      IDataObject::GetCanonicalFormatEtc -- return E_NOTIMPL
**                      (NOTE: must set pformatetcOut->ptd = NULL)
*********************************************************************/
```

CDataXferObj::QueryGetData   (SIMPDND Sample)

```
//**********************************************************************
//
// CDataXferObj::QueryGetData
//
// Purpose:
//
//      Called to determine if our object supports a particular
//      FORMATETC.
//
// Parameters:
//
//      LPFORMATETC pformatetc  - Pointer to the FORMATETC being queried
for.
//
// Return Value:
//
//      DV_E_FORMATETC    - The FORMATETC is not supported
//      S_OK              - The FORMATETC is supported.
//
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      ResultFromScode             OLE API
//
// Comments:
//      we support the following formats:
//          "Embedded Object"
//          "Object Descriptor"
//          CF_METAFILEPICT
//
//********************************************************************
STDMETHODIMP CDataXferObj::QueryGetData (LPFORMATETC pformatetc)
{
     SCODE sc = DV_E_FORMATETC;

     OutputDebugString("In CDataXferObj::QueryGetData\r\n");

     // check the validity of the formatetc.

     if ( (pformatetc->cfFormat == g_cfEmbeddedObject) &&
           (pformatetc->dwAspect == DVASPECT_CONTENT) &&
           (pformatetc->tymed == TYMED_ISTORAGE) )
          sc = S_OK;

     else if ( (pformatetc->cfFormat == g_cfObjectDescriptor) &&
           (pformatetc->dwAspect == DVASPECT_CONTENT) &&
           (pformatetc->tymed == TYMED_HGLOBAL) )
          sc = S_OK;

     else if ( (pformatetc->cfFormat == CF_METAFILEPICT) &&
            (pformatetc->dwAspect == DVASPECT_CONTENT) &&
            (pformatetc->tymed == TYMED_MFPICT) )
           sc = S_OK;

     return ResultFromScode(sc);
}


STDMETHODIMP CDataXferObj::EnumFormatEtc(
           DWORD dwDirection,
           LPENUMFORMATETC FAR* ppenumFormatEtc
)
{
     SCODE sc = E_NOTIMPL;

     OutputDebugString("In CDataXferObj::EnumFormatEtc\r\n");
     *ppenumFormatEtc = NULL;

     if (dwDirection == DATADIR_GET) {
           *ppenumFormatEtc = OleStdEnumFmtEtc_Create(
                       sizeof(s_arrGetFmtEtcs)/sizeof(s_arrGetFmtEtcs[0]),
                       s_arrGetFmtEtcs);
           if (*ppenumFormatEtc == NULL)
                 sc = E_OUTOFMEMORY;
           else
                 sc = S_OK;
     }
     return ResultFromScode(sc);
}
```

CDataXferObj::GetData   (SIMPDND Sample)

```
//**********************************************************************
//
// CDataXferObj::GetData
//
// Purpose:
//
//      Returns the data in the format specified in pformatetcIn.
//
// Parameters:
//
//      LPFORMATETC pformatetcIn    -   The format requested by the caller
//
//      LPSTGMEDIUM pmedium         -   The medium requested by the caller
//
// Return Value:
//
//      DV_E_FORMATETC    - Format not supported
//      S_OK                - Success
//
// Function Calls:
//      Function                        Location
//
//      OutputDebugString               Windows API
//      OleStdGetOleObjectData          OLE2UI API
//      OleStdGetMetafilePictFromOleObject OLE2UI API
//      ResultFromScode                 OLE API
//
// Comments:
//      we support GetData for the following formats:
//          "Embedded Object"
//          "Object Descriptor"
//          CF_METAFILEPICT
//
//********************************************************************

STDMETHODIMP CDataXferObj::GetData (
           LPFORMATETC pformatetcIn,
           LPSTGMEDIUM pmedium
)
{
     SCODE sc = DV_E_FORMATETC;

     OutputDebugString("In CDataXferObj::GetData\r\n");

     // we must set all out pointer parameters to NULL. */
     pmedium->tymed = TYMED_NULL;
     pmedium->pUnkForRelease = NULL;    // we transfer ownership to caller
     pmedium->hGlobal = NULL;

     // Check the FORMATETC and fill pmedium if valid.
     if ( (pformatetcIn->cfFormat == g_cfEmbeddedObject) &&
            (pformatetcIn->dwAspect == DVASPECT_CONTENT) &&
            (pformatetcIn->tymed == TYMED_ISTORAGE) ) {
            LPPERSISTSTORAGE pPersistStorage;

            /* render CF_EMBEDDEDOBJECT by asking the object to save
            **    into a temporary, DELETEONRELEASE IStorage allocated by
us.
            */
            m_lpOleObject->QueryInterface(
                        IID_IPersistStorage, (LPVOID FAR*)&pPersistStorage);
            assert(pPersistStorage);
            HRESULT hrErr = OleStdGetOleObjectData(
                             pPersistStorage,
                             pformatetcIn,
                             pmedium,
                             FALSE   /* fUseMemory -- (use file-base stg) */
            );
            pPersistStorage->Release();
            sc = GetScode( hrErr );

     } else if ( (pformatetcIn->cfFormat == g_cfObjectDescriptor) &&
            (pformatetcIn->dwAspect == DVASPECT_CONTENT) &&
            (pformatetcIn->tymed == TYMED_HGLOBAL) ) {

            // render CF_OBJECTDESCRIPTOR data
            pmedium->hGlobal = OleStdGetObjectDescriptorDataFromOleObject(
                        m_lpOleObject,
                        //@@WTK WIN32, UNICODE
                        //"Simple OLE 2.0 Container",    // string to
identify source
                        OLESTR("Simple OLE 2.0 Container"),    // string to
identify source
                        DVASPECT_CONTENT,
                        m_pointl,
                        (LPSIZEL)&m_sizel
                 );
            if (! pmedium->hGlobal)
                  sc = E_OUTOFMEMORY;
            else {
                  pmedium->tymed = TYMED_HGLOBAL;
                  sc = S_OK;
            }

     } else if ( (pformatetcIn->cfFormat == CF_METAFILEPICT) &&
            (pformatetcIn->dwAspect == DVASPECT_CONTENT) &&
            (pformatetcIn->tymed == TYMED_MFPICT) ) {

            // render CF_METAFILEPICT by drawing the object into a metafile
DC
            pmedium->hGlobal = OleStdGetMetafilePictFromOleObject(
                        m_lpOleObject, DVASPECT_CONTENT, NULL, pformatetcIn-
>ptd);
            if (! pmedium->hGlobal)
                  sc = E_OUTOFMEMORY;
            else {
                  pmedium->tymed = TYMED_MFPICT;
                  sc = S_OK;
            }
     }

     return ResultFromScode( sc );
}
```

CDataXferObj::GetDataHere   (SIMPDND Sample)

```
//**********************************************************************
//
// CDataXferObj::GetDataHere
//
// Purpose:
//
//      Called to get a data format in a caller supplied location
//
// Parameters:
//
//      LPFORMATETC pformatetc  - FORMATETC requested
//
//      LPSTGMEDIUM pmedium     - Medium to return the data
//
// Return Value:
//
//      DATA_E_FORMATETC    - We don't support the requested format
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      OleStdGetOleObjectData      OLE2UI API
//
// Comments:
//
//********************************************************************

STDMETHODIMP CDataXferObj::GetDataHere (
           LPFORMATETC pformatetc,
           LPSTGMEDIUM pmedium
)
{
     SCODE sc = DV_E_FORMATETC;

     OutputDebugString("In CDataXferObj::GetDataHere\r\n");

     // NOTE: pmedium is an IN parameter. we should NOT set
     //           pmedium->pUnkForRelease to NULL

     // Check the FORMATETC and fill pmedium if valid.
     if ( (pformatetc->cfFormat == g_cfEmbeddedObject) &&
            (pformatetc->dwAspect == DVASPECT_CONTENT) &&
            (pformatetc->tymed == TYMED_ISTORAGE) ) {
            LPPERSISTSTORAGE pPersistStorage;

            /* render CF_EMBEDDEDOBJECT by asking the object to save
            **    into the IStorage allocated by the caller.
            */
            m_lpOleObject->QueryInterface(
                        IID_IPersistStorage, (LPVOID FAR*)&pPersistStorage);
            assert(pPersistStorage);
            HRESULT hrErr = OleStdGetOleObjectData(
                        pPersistStorage, pformatetc, pmedium,0
/*fUseMemory--N/A*/ );
            pPersistStorage->Release();
            sc = GetScode( hrErr );
     }
     return ResultFromScode( sc );
}
```

IAS.CPP (SIMPDND Sample)

```
//**********************************************************************
// File name: IAS.CPP
//
//      Implementation file of CAdviseSink
//
//
// Functions:
//
//      See IAS.H for Class Definition
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "iocs.h"
#include "ias.h"
#include "app.h"
#include "site.h"
#include "doc.h"
```

CAdviseSink::QueryInterface (SIMPDND Sample)

```
//**********************************************************************
//
// CAdviseSink::QueryInterface
//
// Purpose:
//
//      Returns a pointer to a requested interface.
//
// Parameters:
//
//      REFIID riid         - The requested interface
//
//      LPVOID FAR* ppvObj  - Place to return the interface
//
// Return Value:
//
//      HRESULT from CSimpleSite::QueryInterface
//
// Function Calls:
//      Function                    Location
//
//      CSimpleSite::QueryInterface SITE.CPP
//      OutputDebugString           Windows API
//
// Comments:
//
//      This function simply delegates to the Object class, which is
//      aware of the supported interfaces.
//
//********************************************************************

STDMETHODIMP CAdviseSink::QueryInterface(REFIID riid, LPVOID FAR* ppvObj)
{
     OutputDebugString("In IAS::QueryInterface\r\n");

     // delegate to the document Object
     return m_pSite->QueryInterface(riid, ppvObj);
}
```

CAdviseSink::AddRef   (SIMPDND Sample)

```
//**********************************************************************
//
// CAdviseSink::AddRef
//
// Purpose:
//
//      Increments the reference count on this interface
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The current reference count on this interface.
//
// Function Calls:
//      Function                    Location
//
//      CSimpleSite::AddReff        SITE.CPP
//      OutputDebugString           Windows API
//
// Comments:
//
//      This function adds one to the ref count of the interface,
//      and calls then calls CSimpleSite to increment its ref.
//      count.
//
//********************************************************************

STDMETHODIMP_(ULONG) CAdviseSink::AddRef()
{
     OutputDebugString("In IAS::AddRef\r\n");

     // increment the interface reference count (for debugging only)
     ++m_nCount;

     // delegate to the container Site
     return m_pSite->AddRef();
}
```

CAdviseSink::Release   (SIMPDND Sample)

```
//**********************************************************************
//
// CAdviseSink::Release
//
// Purpose:
//
//      Decrements the reference count on this interface
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The current reference count on this interface.
//
// Function Calls:
//      Function                    Location
//
//      CSimpleSite::Release        SITE.CPP
//      OutputDebugString           Windows API
//
// Comments:
//
//      This function subtracts one from the ref count of the interface,
//      and calls then calls CSimpleSite to decrement its ref.
//      count.
//
//********************************************************************

STDMETHODIMP_(ULONG) CAdviseSink::Release()
{
     OutputDebugString("In IAS::Release\r\n");

     // decrement the interface reference count (for debugging only)
     m_nCount--;

     // delegate to the container Site
     return m_pSite->Release();
}
```

CAdviseSink::OnDataChange   (SIMPDND Sample)

```
//**********************************************************************
//
// CAdviseSink::OnDataChange
//
// Purpose:
//
//      Not Implemented (needs to be stubbed out)
//
// Parameters:
//
//      Not Implemented (needs to be stubbed out)
//
// Return Value:
//
//      Not Implemented (needs to be stubbed out)
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Not Implemented (needs to be stubbed out)
//
//********************************************************************

STDMETHODIMP_(void) CAdviseSink::OnDataChange (FORMATETC FAR* pFormatetc,
STGMEDIUM FAR* pStgmed)
{
     OutputDebugString("In IAS::OnDataChange\r\n");
}
```

CAdviseSink::OnViewChange (SIMPDND Sample)

```
//**********************************************************************
//
// CAdviseSink::OnViewChange
//
// Purpose:
//
//      Notifies us that the view has changed and needs to be updated.
//
// Parameters:
//
//      DWORD dwAspect  - Aspect that has changed
//
//      LONG lindex     - Index that has changed
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      InvalidateRect              Windows API
//      IViewObject2::GetExtent     Object
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(void) CAdviseSink::OnViewChange (DWORD dwAspect, LONG lindex)
{
     LPVIEWOBJECT2 lpViewObject2;
     OutputDebugString("In IAS::OnViewChange\r\n");

     // get a pointer to IViewObject2
     HRESULT hErr = m_pSite->m_lpOleObject->QueryInterface(
                 IID_IViewObject2,(LPVOID FAR *)&lpViewObject2);

     if (hErr == NOERROR) {
           // get extent of the object
           // NOTE: this method will never be remoted; it can be called w/i
this async method
           lpViewObject2->GetExtent(DVASPECT_CONTENT, -1 , NULL, &m_pSite-
>m_sizel);
           lpViewObject2->Release();
     }

     InvalidateRect(m_pSite->m_lpDoc->m_hDocWnd, NULL, TRUE);
}
```

CAdviseSink::OnRename   (SIMPDND Sample)

```
//**********************************************************************
//
// CAdviseSink::OnRename
//
// Purpose:
//
//      Not Implemented (needs to be stubbed out)
//
// Parameters:
//
//      Not Implemented (needs to be stubbed out)
//
// Return Value:
//
//      Not Implemented (needs to be stubbed out)
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Not Implemented (needs to be stubbed out)
//
//********************************************************************

STDMETHODIMP_(void) CAdviseSink::OnRename (LPMONIKER pmk)
{
     OutputDebugString("In IAS::OnRename\r\n");
}
```

CAdviseSink::OnSave   (SIMPDND Sample)

```
//***********************************************************************
//
// CAdviseSink::OnSave
//
// Purpose:
//
//      Not Implemented (needs to be stubbed out)
//
// Parameters:
//
//      Not Implemented (needs to be stubbed out)
//
// Return Value:
//
//      Not Implemented (needs to be stubbed out)
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Not Implemented (needs to be stubbed out)
//
//*********************************************************************

STDMETHODIMP_(void) CAdviseSink::OnSave ()
{
     OutputDebugString("In IAS::OnSave\r\n");
}
```

CAdviseSink::OnClose   (SIMPDND Sample)

```
//**********************************************************************
//
// CAdviseSink::OnClose
//
// Purpose:
//
//      Not Implemented (needs to be stubbed out)
//
// Parameters:
//
//      Not Implemented (needs to be stubbed out)
//
// Return Value:
//
//      Not Implemented (needs to be stubbed out)
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Not Implemented (needs to be stubbed out)
//
//********************************************************************

STDMETHODIMP_(void) CAdviseSink::OnClose()
{
     OutputDebugString("In IAS::OnClose\r\n");
}
```

IDS.CPP (SIMPDND Sample)

```
//**********************************************************************
// File name: IDS.CPP
//
//      Implementation file for CDropSource
//
// Functions:
//
//      See IDS.H for class definition
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "doc.h"
#include "site.h"
#include "dxferobj.h"
```

CSimpleDoc::QueryDrag   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleDoc::QueryDrag
//
// Purpose:
//
//      Check to see if Drag operation should be initiated based on the
//      current position of the mouse.
//
// Parameters:
//
//      POINT pt                - position of mouse
//
// Return Value:
//
//      BOOL                    - TRUE if drag should take place,
//                                else FALSE
//
// Function Calls:
//      Function                    Location
//
//      CSimpleSite::GetObjRect     SITE.CPP
//      PtInRect                    Windows API
//
// Comments:
//
//********************************************************************

BOOL CSimpleDoc::QueryDrag(POINT pt)
{
     // if pt is within rect of object, then start drag
     if (m_lpSite)
           {
           RECT rect;
           m_lpSite->GetObjRect(&rect);
           return ( PtInRect(&rect, pt) ? TRUE : FALSE );
           }
     else
           return FALSE;
}
```

CSimpleDoc::DoDragDrop (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleDoc::DoDragDrop
//
// Purpose:
//
//      Actually perform a drag/drop operation with the current
//      selection in the source document.
//
// Parameters:
//
//      none.
//
// Return Value:
//
//      DWORD                    - returns the result effect of the
//                                 drag/drop operation:
//                                      DROPEFFECT_NONE,
//                                      DROPEFFECT_COPY,
//                                      DROPEFFECT_MOVE, or
//                                      DROPEFFECT_LINK
//
// Function Calls:
//      Function                    Location
//
//      CDataXferObj::Create        DXFEROBJ.CPP
//      CDataXferObj::QueryInterface DXFEROBJ.CPP
//      CDataXferObj::Release       DXFEROBJ.CPP
//      DoDragDrop                  OLE API
//      OutputDebugString           Windows API
//
// Comments:
//
//********************************************************************

DWORD CSimpleDoc::DoDragDrop (void)
{
     DWORD       dwEffect     = 0;
     LPDATAOBJECT lpDataObj;

     OutputDebugString("In CSimpleDoc::DoDragDrop\r\n");

     // Create a data transfer object by cloning the existing OLE object
     CDataXferObj FAR* pDataXferObj = CDataXferObj::Create(m_lpSite,NULL);

     if (! pDataXferObj) {
           MessageBox(NULL,"Out-of-memory","SimpDnD",MB_SYSTEMMODAL|
MB_ICONHAND);
           return DROPEFFECT_NONE;
     }
     // initially obj is created with 0 refcnt. this QI will make it go to
1.
     pDataXferObj->QueryInterface(IID_IDataObject, (LPVOID FAR*)&lpDataObj);
     assert(lpDataObj);

     m_fLocalDrop     = FALSE;
     m_fLocalDrag     = TRUE;

     ::DoDragDrop ( lpDataObj,
                        &m_DropSource,
                        DROPEFFECT_COPY,   // we only allow copy
                        &dwEffect
     );

     m_fLocalDrag     = FALSE;

     /* if after the Drag/Drop modal (mouse capture) loop is finished
     **    and a drag MOVE operation was performed, then we must delete
     **    the selection that was dragged.
     */
     if ( (dwEffect & DROPEFFECT_MOVE) != 0 ) {
           // ... delete source object here (we do NOT support MOVE)
     }

     pDataXferObj->Release();    // this should destroy the DataXferObj
     return dwEffect;
}
```

CDropSource::QueryInterface   (SIMPDND Sample)

```
//**********************************************************************
//
// CDropSource::QueryInterface
//
// Purpose:
//
//      Return a pointer to a requested interface
//
// Parameters:
//
//      REFIID riid          -   ID of interface to be returned
//      LPVOID FAR* ppvObj  -   Location to return the interface
//
// Return Value:
//
//      S_OK                -   Interface supported
//      E_NOINTERFACE       -   Interface NOT supported
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpleDoc::QueryInterface  DOC.CPP
//
// Comments:
//
//********************************************************************

STDMETHODIMP CDropSource::QueryInterface(REFIID riid, LPVOID FAR* ppvObj)
{
     OutputDebugString("In IDS::QueryInterface\r\n");

     // delegate to the document
     return m_pDoc->QueryInterface(riid, ppvObj);
}
```

CDropSource::AddRef (SIMPDND Sample)

```
//**********************************************************************
//
// CDropSource::AddRef
//
// Purpose:
//
//      Increments the reference count on this interface
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The current reference count on this interface.
//
// Function Calls:
//      Function                    Location
//
//      CSimpleObj::AddReff         OBJ.CPP
//      OutputDebugString           Windows API
//
// Comments:
//
//      This function adds one to the ref count of the interface,
//      and calls then calls CSimpleDoc to increment its ref.
//      count.
//
//********************************************************************

STDMETHODIMP_(ULONG) CDropSource::AddRef()
{
     OutputDebugString("In IDS::AddRef\r\n");

     // increment the interface reference count (for debugging only)
     ++m_nCount;

     // delegate to the document Object
     return m_pDoc->AddRef();
}
```

CDropSource::Release   (SIMPDND Sample)

```
//**********************************************************************
//
// CDropSource::Release
//
// Purpose:
//
//      Decrements the reference count on this interface
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The current reference count on this interface.
//
// Function Calls:
//      Function                    Location
//
//      CSimpleObj::Release         OBJ.CPP
//      OutputDebugString           Windows API
//
// Comments:
//
//      This function subtracts one from the ref count of the interface,
//      and calls then calls CSimpleDoc to decrement its ref.
//      count.
//
//********************************************************************

STDMETHODIMP_(ULONG) CDropSource::Release()
{
     OutputDebugString("In IDS::Release\r\n");

     // decrement the interface reference count (for debugging only)
     --m_nCount;

     // delegate to the document object
     return m_pDoc->Release();
}
```

CDropSource::QueryContinueDrag (SIMPDND Sample)

```
//**********************************************************************
//
// CDropSource::QueryContinueDrag
//
// Purpose:
//
//      Called to determine if a drop should take place or be canceled.
//
// Parameters:
//
//      BOOL fEscapePressed - TRUE if ESCAPE key has been pressed
//      DWORD grfKeyState   - key state
//
// Return Value:
//
//      DRAGDROP_S_CANCEL   - drag operation should be canceled
//      DRAGDROP_S_DROP     - drop operation should be performed
//      S_OK                - dragging should continue
//
//
// Function Calls:
//      Function                    Location
//
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP CDropSource::QueryContinueDrag (
           BOOL    fEscapePressed,
           DWORD   grfKeyState
)
{
     if (fEscapePressed)
           return ResultFromScode(DRAGDROP_S_CANCEL);
     else if (!(grfKeyState & MK_LBUTTON))
           return ResultFromScode(DRAGDROP_S_DROP);
     else
           return NOERROR;
}
```

CDropSource::GiveFeedback (SIMPDND Sample)

```
//**********************************************************************
//
// CDropSource::GiveFeedback
//
// Purpose:
//
//      Called to set cursor feedback
//
// Parameters:
//
//      DWORD dwEffect      - drop operation to give feedback for
//
// Return Value:
//
//      DRAGDROP_S_USEDEFAULTCURSORS  - tells OLE to use standard cursors
//
// Function Calls:
//      Function                    Location
//
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP CDropSource::GiveFeedback (DWORD dwEffect)
{
     return ResultFromScode(DRAGDROP_S_USEDEFAULTCURSORS);
}
```

IDT.CPP   (SIMPDND Sample)

```
//**********************************************************************
// File name: IDT.CPP
//
//      Implementation file for CDropTarget
//
// Functions:
//
//      See IDT.H for class definition
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "iocs.h"
#include "ias.h"
#include "app.h"
#include "site.h"
#include "doc.h"
#include "idt.h"

extern CLIPFORMAT g_cfObjectDescriptor;
```

CDropTarget::QueryDrop (SIMPDND Sample)

```
//**********************************************************************
//
// CDropTarget::QueryDrop
//
// Purpose:
//
//      Check if the desired drop operation (identified by the given key
//      state) is possible at the current mouse position (pointl).
//
// Parameters:
//
//      DWORD grfKeyState       - current key state
//      POINTL pointl           - position of mouse
//      BOOL fDragScroll        - TRUE if drag scrolling cursor should
//                                be shown.
//      LPDWORD pdwEffect       - (OUT) drag effect that should occur
//
// Return Value:
//
//      BOOL                    - TRUE if drop could take place,
//                                else FALSE
//
// Function Calls:
//      Function                  Location
//
//      OleStdGetDropEffect       OLE2UI API
//
// Comments:
//
//********************************************************************

BOOL CDropTarget::QueryDrop (
     DWORD           grfKeyState,
     POINTL          pointl,
     BOOL            fDragScroll,
     LPDWORD         pdwEffect
)
{
     DWORD      dwScrollEffect = 0L;
     DWORD      dwOKEffects = *pdwEffect;

     /* check if the cursor is in the active scroll area, if so need the
     **    special scroll cursor.
     */
     if (fDragScroll)
          dwScrollEffect = DROPEFFECT_SCROLL;

     /* if we have already determined that the source does NOT have any
     **    acceptable data for us, the return NO-DROP
     */
     if (! m_fCanDropCopy && ! m_fCanDropLink)
          goto dropeffect_none;
     /* OLE2NOTE: determine what type of drop should be performed given
     **    the current modifier key state. we rely on the standard
     **    interpretation of the modifier keys:
     **          no modifier -- DROPEFFECT_MOVE or whatever is allowed by
src
     **          SHIFT       -- DROPEFFECT_MOVE
     **          CTRL        -- DROPEFFECT_COPY
     **          CTRL-SHIFT  -- DROPEFFECT_LINK
     */

     *pdwEffect = OleStdGetDropEffect(grfKeyState);
     if (*pdwEffect == 0) {
           // No modifier keys given. Try in order MOVE, COPY, LINK.
           if ((DROPEFFECT_MOVE & dwOKEffects) && m_fCanDropCopy)
                 *pdwEffect = DROPEFFECT_MOVE;
           else if ((DROPEFFECT_COPY & dwOKEffects) && m_fCanDropCopy)
                 *pdwEffect = DROPEFFECT_COPY;
           else if ((DROPEFFECT_LINK & dwOKEffects) && m_fCanDropLink)
                 *pdwEffect = DROPEFFECT_LINK;
           else
                 goto dropeffect_none;
     } else {
           /* OLE2NOTE: we should check if the drag source application
allows
           **    the desired drop effect.
           */
           if (!(*pdwEffect & dwOKEffects))
                 goto dropeffect_none;

           if ((*pdwEffect == DROPEFFECT_COPY || *pdwEffect ==
DROPEFFECT_MOVE)
                       && ! m_fCanDropCopy)
                 goto dropeffect_none;

           if (*pdwEffect == DROPEFFECT_LINK && ! m_fCanDropLink)
                 goto dropeffect_none;
     }

     *pdwEffect |= dwScrollEffect;
     return TRUE;

dropeffect_none:

     *pdwEffect = DROPEFFECT_NONE;
     return FALSE;
}
```

CDropTarget::QueryDrop (SIMPDND Sample)

```
//**********************************************************************
//
// CDropTarget::QueryDrop
//
// Purpose:
//
//     Check to see if Drag scroll operation should be initiated.
//
// Parameters:
//
//      POINTL pointl           - position of mouse
//
// Return Value:
//
//      BOOL                    - TRUE if scroll cursor should be given
//                                else FALSE
//
// Function Calls:
//      Function                    Location
//
//      ScreenToClient              WINDOWS API
//      GetClientRect               WINDOWS API
//
// Comments:
//     A Drag scroll operation should be initiated when the mouse has
//     remained in the active scroll area (11 pixels frame around border
//     of window) for a specified amount of time (50ms).
//
//********************************************************************

BOOL CDropTarget::DoDragScroll (POINTL pointl)
{
     DWORD dwScrollDir = SCROLLDIR_NULL;
     DWORD dwTime = GetCurrentTime();
     int nScrollInset = m_pDoc->m_lpApp->m_nScrollInset;
     int nScrollDelay = m_pDoc->m_lpApp->m_nScrollDelay;
     int nScrollInterval = m_pDoc->m_lpApp->m_nScrollInterval;
     POINT point;
     RECT rect;

     point.x = (int)pointl.x;
     point.y = (int)pointl.y;

     ScreenToClient( m_pDoc->m_hDocWnd, &point);
     GetClientRect ( m_pDoc->m_hDocWnd, (LPRECT) &rect );

     if (rect.top <= point.y && point.y<=(rect.top+nScrollInset))
           dwScrollDir = SCROLLDIR_UP;
     else if ((rect.bottom-nScrollInset) <= point.y && point.y <=
rect.bottom)
           dwScrollDir = SCROLLDIR_DOWN;
     else if (rect.left <= point.x && point.x <= (rect.left+nScrollInset))
```

```
           dwScrollDir = SCROLLDIR_LEFT;
     else if ((rect.right-nScrollInset) <= point.x && point.x <= rect.right)
           dwScrollDir = SCROLLDIR_RIGHT;

     if (m_dwTimeEnterScrollArea) {

           /* cursor was already in Scroll Area */

           if (! dwScrollDir) {
                 /* cusor moved OUT of scroll area.
                 **      clear "EnterScrollArea" time.
                 */
                 m_dwTimeEnterScrollArea = 0L;
                 m_dwNextScrollTime = 0L;
                 m_dwLastScrollDir = SCROLLDIR_NULL;

           } else if (dwScrollDir != m_dwLastScrollDir) {
                 /* cusor moved into a different direction scroll area.
                 **      reset "EnterScrollArea" time to start a new 50ms
delay.
                 */
                 m_dwTimeEnterScrollArea = dwTime;
                 m_dwNextScrollTime = dwTime + (DWORD)nScrollDelay;
                 m_dwLastScrollDir = dwScrollDir;

           } else if (dwTime && dwTime >= m_dwNextScrollTime) {
                 m_pDoc->Scroll ( dwScrollDir );    // Scroll document now
                 m_dwNextScrollTime = dwTime + (DWORD)nScrollInterval;
           }
     } else {
           if (dwScrollDir) {
                 /* cusor moved INTO a scroll area.
                 **      reset "EnterScrollArea" time to start a new 50ms
delay.
                 */
                 m_dwTimeEnterScrollArea = dwTime;
                 m_dwNextScrollTime = dwTime + (DWORD)nScrollDelay;
                 m_dwLastScrollDir = dwScrollDir;
           }
     }

     return (dwScrollDir ? TRUE : FALSE);
}


// Support functions/macros
#define SetTopLeft(rc, pt)      \
     ((rc)->top = (pt)->y,(rc)->left = (pt)->x)
#define SetBottomRight(rc, pt)      \
     ((rc)->bottom = (pt)->y,(rc)->right = (pt)->x)
#define OffsetPoint(pt, dx, dy)     \
     ((pt)->x += dx, (pt)->y += dy)


/* HighlightRect
```

```
** -------------
**    Invert rectangle on screen. used for drop target feedback.
*/

static int HighlightRect(HWND hwnd, HDC hdc, LPRECT rc)
{
     POINT pt1, pt2;
     int old = SetROP2(hdc, R2_NOT);
     HPEN hpen;
     HGDIOBJ hold;

     pt1.x = rc->left;
     pt1.y = rc->top;
     pt2.x = rc->right;
     pt2.y = rc->bottom;

     ScreenToClient(hwnd, &pt1);
     ScreenToClient(hwnd, &pt2);

     hold = SelectObject(hdc, GetStockObject(HOLLOW_BRUSH));
     hpen = SelectObject(hdc, CreatePen(PS_SOLID, 2,
                                     GetSysColor(COLOR_ACTIVEBORDER)));

     Rectangle(hdc, pt1.x, pt1.y, pt2.x, pt2.y);

     SetROP2(hdc, old);

     hold = SelectObject(hdc, hold);

     hpen = SelectObject(hdc, hpen);

     DeleteObject(hpen);

  return 0;
}
```

CDropTarget::InitDragFeedback (SIMPDND Sample)

```
//**********************************************************************
//
// CDropTarget::InitDragFeedback
//
// Purpose:
//
//      Initialize data used to draw drop target feedback.
//      As feedback we draw a rectangle the size of the object.
//
// Parameters:
//
//      LPDATAOBJECT pDataObj   - IDataObject from Drop source
//      POINTL pointl           - position of mouse
//
// Return Value:
//
//      none.
//
// Function Calls:
//      Function                    Location
//
//      IDataObject::GetData        Object
//      XformSizeInHimetricToPixels OLE2UI Library
//      GlobalLock                  WINDOWS API
//      GlobalUnlock                WINDOWS API
//      ReleaseStgMedium            OLE2 API
//      OffsetPoint                 IDT.CPP
//      SetTopLeft                  IDT.CPP
//      SetBottomRight              IDT.CPP
//
// Comments:
//      In order to known the size of the object before the object
//      is actually dropped, we render CF_OBJECTDESCRIPTOR format.
//      this data format tells us both the size of the object as
//      well as which aspect is the object is displayed as in the
//      source. if the object is currently displayed as DVASPECT_ICON
//      then we want to create the object also as DVASPECT_ICON.
//
//********************************************************************

void CDropTarget::InitDragFeedback(LPDATAOBJECT pDataObj, POINTL pointl)
{
     FORMATETC fmtetc;
     STGMEDIUM stgmed;
     POINT pt;
     int height, width;
     HRESULT hrErr;

     height = width = 100; // some default values
     pt.x = (int)pointl.x;
     pt.y = (int)pointl.y;
     // do a GetData for CF_OBJECTDESCRIPTOR format to get the size of the
     // object as displayed in the source. using this size, initialize the
     // size for the drag feedback rectangle.
     fmtetc.cfFormat = g_cfObjectDescriptor;
     fmtetc.ptd = NULL;
     fmtetc.lindex = -1;
     fmtetc.dwAspect = DVASPECT_CONTENT;
     fmtetc.tymed = TYMED_HGLOBAL;

     hrErr = pDataObj->GetData(&fmtetc, &stgmed);
     if (hrErr == NOERROR) {
           LPOBJECTDESCRIPTOR
pOD=(LPOBJECTDESCRIPTOR)GlobalLock(stgmed.hGlobal);
           if (pOD != NULL) {
                 XformSizeInHimetricToPixels(NULL, &pOD->sizel, &pOD-
>sizel);

                 width = (int)pOD->sizel.cx;
                 height = (int)pOD->sizel.cy;
                 m_dwSrcAspect = pOD->dwDrawAspect;
           }

           GlobalUnlock(stgmed.hGlobal);
           ReleaseStgMedium(&stgmed);
     }

     m_ptLast = pt;
     m_fDragFeedbackDrawn = FALSE;

     OffsetPoint(&pt, -(width/2), -(height/2));
     SetTopLeft(&m_rcDragRect, &pt);

     OffsetPoint(&pt, width, height);
     SetBottomRight(&m_rcDragRect, &pt);
}
```

CDropTarget::UndrawDragFeedback   (SIMPDND Sample)

```
//**********************************************************************
//
// CDropTarget::UndrawDragFeedback
//
// Purpose:
//
//      Erase any drop target feedback.
//      As feedback we draw a rectangle the size of the object.
//
// Parameters:
//
//      none.
//
// Return Value:
//
//      none.
//
// Function Calls:
//      Function                    Location
//
//      GetDC                       WINDOWS API
//      ReleaseDC                   WINDOWS API
//      GlobalUnlock                WINDOWS API
//      HighlightRect               IDT.CPP
//
// Comments:
//      In order to known the size of the object before the object
//      is actually dropped, we render CF_OBJECTDESCRIPTOR format.
//      this data format tells us both the size of the object as
//      well as which aspect is the object is displayed as in the
//      source. if the object is currently displayed as DVASPECT_ICON
//      then we want to create the object also as DVASPECT_ICON.
//
//********************************************************************

void CDropTarget::UndrawDragFeedback( void )
{
     if (m_fDragFeedbackDrawn) {
           m_fDragFeedbackDrawn = FALSE;
           HDC hDC = GetDC(m_pDoc->m_hDocWnd);
           HighlightRect(m_pDoc->m_hDocWnd, hDC, &m_rcDragRect);
           ReleaseDC(m_pDoc->m_hDocWnd, hDC);
     }
}
```

CDropTarget::DrawDragFeedback   (SIMPDND Sample)

```
//**********************************************************************
//
// CDropTarget::DrawDragFeedback
//
// Purpose:
//
//      Compute new position of drop target feedback rectangle and
//      erase old rectangle and draw new rectangle.
//      As feedback we draw a rectangle the size of the object.
//
// Parameters:
//
//      POINTL pointl           - position of mouse
//
// Return Value:
//
//      none.
//
// Function Calls:
//      Function                    Location
//
//      OffsetPoint                 IDT.CPP
//      HighlightRect               IDT.CPP
//      GetDC                       WINDOWS API
//      ReleaseDC                   WINDOWS API
//
// Comments:
//
//********************************************************************

void CDropTarget::DrawDragFeedback( POINTL pointl )
{
     POINT ptDiff;

     ptDiff.x = (int)pointl.x - m_ptLast.x;
     ptDiff.y = (int)pointl.y - m_ptLast.y;

     if (m_fDragFeedbackDrawn && (ptDiff.x == 0 && ptDiff.y == 0))
           return;     // mouse did not move; leave rectangle as drawn

     HDC hDC = GetDC(m_pDoc->m_hDocWnd);
     if (m_fDragFeedbackDrawn) {
           m_fDragFeedbackDrawn = FALSE;
           HighlightRect(m_pDoc->m_hDocWnd, hDC, &m_rcDragRect);
     }

     OffsetRect(&m_rcDragRect, ptDiff.x, ptDiff.y);
     HighlightRect(m_pDoc->m_hDocWnd, hDC, &m_rcDragRect);
     m_fDragFeedbackDrawn = TRUE;
     m_ptLast.x = (int)pointl.x;
     m_ptLast.y = (int)pointl.y;
     ReleaseDC(m_pDoc->m_hDocWnd, hDC);
}
```

CDropTarget::QueryInterface   (SIMPDND Sample)

```
//**********************************************************************
//
// CDropTarget::QueryInterface
//
// Purpose:
//
//      Return a pointer to a requested interface
//
// Parameters:
//
//      REFIID riid         -   ID of interface to be returned
//      LPVOID FAR* ppvObj  -   Location to return the interface
//
// Return Value:
//
//      S_OK                -   Interface supported
//      E_NOINTERFACE       -   Interface NOT supported
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpleDoc::QueryInterface  DOC.CPP
//
// Comments:
//
//********************************************************************

STDMETHODIMP CDropTarget::QueryInterface(REFIID riid, LPVOID FAR* ppvObj)
{
     OutputDebugString("In IDT::QueryInterface\r\n");

     // delegate to the document
     return m_pDoc->QueryInterface(riid, ppvObj);
}
```

CDropTarget::AddRef   (SIMPDND Sample)

```
//**********************************************************************
//
// CDropTarget::AddRef
//
// Purpose:
//
//      Increments the reference count on this interface
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The current reference count on this interface.
//
// Function Calls:
//      Function                    Location
//
//      CSimpleDoc::AddReff         DOC.CPP
//      OutputDebugString           Windows API
//
// Comments:
//
//      This function adds one to the ref count of the interface,
//      and calls then calls CSimpleObj to increment its ref.
//      count.
//
//********************************************************************

STDMETHODIMP_(ULONG) CDropTarget::AddRef()
{
     OutputDebugString("In IDT::AddRef\r\n");

     // increment the interface reference count (for debugging only)
     ++m_nCount;

     // delegate to the document Object
     return m_pDoc->AddRef();
}
```

CDropTarget::Release (SIMPDND Sample)

```
//**********************************************************************
//
// CDropTarget::Release
//
// Purpose:
//
//      Decrements the reference count on this interface
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The current reference count on this interface.
//
// Function Calls:
//      Function                    Location
//
//      CSimpleDoc::Release         DOC.CPP
//      OutputDebugString           Windows API
//
// Comments:
//
//      This function subtracts one from the ref count of the interface,
//      and calls then calls CSimpleDoc to decrement its ref.
//      count.
//
//********************************************************************

STDMETHODIMP_(ULONG) CDropTarget::Release()
{
     OutputDebugString("In IDT::Release\r\n");

     // decrement the interface reference count (for debugging only)
     --m_nCount;

     // delegate to the document object
     return m_pDoc->Release();
}
```

CDropTarget::DragEnter (SIMPDND Sample)

```
//**********************************************************************
//
// CDropTarget::DragEnter
//
// Purpose:
//
//      Called when the mouse first enters our DropTarget window
//
// Parameters:
//
//      LPDATAOBJECT pDataObj   - IDataObject from Drop source
//      DWORD grfKeyState       - current key state
//      POINTL pointl           - position of mouse
//      LPDWORD pdwEffect       - (IN-OUT) drag effect that should occur
//                                ON INPUT, this is dwOKEffects that source
//                                passed to DoDragDrop API.
//                                ON OUTPUT, this is the effect that we
//                                want to take effect (used to determine
//                                cursor feedback).
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      OleQueryCreateFromData      OLE2 API
//      DoDragScroll                IDT.CPP
//      QueryDrop                   IDT.CPP
//      InitDragFeedback            IDT.CPP
//      DrawDragFeedback            IDT.CPP
//      UndrawDragFeedback          IDT.CPP
//
// Comments:
//      Callee should honor the dwEffects as passed in to determine
//      if the caller allows DROPEFFECT_MOVE.
//
//********************************************************************

STDMETHODIMP CDropTarget::DragEnter (LPDATAOBJECT pDataObj, DWORD
grfKeyState, POINTL pointl, LPDWORD pdwEffect)
{
     OutputDebugString("In IDT::DragEnter\r\n");

     /* Determine if the drag source data object offers a data format
     **  that we understand. we accept only creating embedded objects.
     */
     m_fCanDropCopy = ((OleQueryCreateFromData(pDataObj) == NOERROR) ?
                 TRUE : FALSE);
     m_fCanDropLink = FALSE; // linking NOT supported in this simple sample
     if (m_fCanDropCopy || m_fCanDropLink)
           InitDragFeedback(pDataObj, pointl);

     BOOL fDragScroll = DoDragScroll ( pointl );

     if (QueryDrop(grfKeyState,pointl,fDragScroll,pdwEffect))
           DrawDragFeedback( pointl );

     return NOERROR;
}
```

CDropTarget::DragOver (SIMPDND Sample)

```
//**********************************************************************
//
// CDropTarget::DragOver
//
// Purpose:
//
//      Called when the mouse moves, key state changes, or a time
//      interval passes while the mouse is still within our DropTarget
//      window.
//
// Parameters:
//
//      DWORD grfKeyState       - current key state
//      POINTL pointl           - position of mouse
//      LPDWORD pdwEffect       - (IN-OUT) drag effect that should occur
//                                ON INPUT, this is dwOKEffects that source
//                                passed to DoDragDrop API.
//                                ON OUTPUT, this is the effect that we
//                                want to take effect (used to determine
//                                cursor feedback).
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      DoDragScroll                IDT.CPP
//      QueryDrop                   IDT.CPP
//      DrawDragFeedback            IDT.CPP
//      UndrawDragFeedback          IDT.CPP
//
// Comments:
//      Callee should honor the dwEffects as passed in to determine
//      if the caller allows DROPEFFECT_MOVE. OLE pulses the DragOver
//      calls in order that the DropTarget can implement drag scrolling
//
//********************************************************************

STDMETHODIMP CDropTarget::DragOver  (DWORD grfKeyState, POINTL pointl,
LPDWORD pdwEffect)
{
     OutputDebugString("In IDT::DragOver\r\n");

     BOOL fDragScroll = DoDragScroll ( pointl );

     if (QueryDrop(grfKeyState,pointl,fDragScroll,pdwEffect))
          DrawDragFeedback( pointl );
     else
          UndrawDragFeedback();
```

```
      return NOERROR;
}
```

CDropTarget::DragLeave   (SIMPDND Sample)

```
//**********************************************************************
//
// CDropTarget::DragLeave
//
// Purpose:
//
//      Called when the mouse leaves our DropTarget window
//
// Parameters:
//
//      none.
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      UndrawDragFeedback          IDT.CPP
//      ResultFromScode             OLE2 API
//
// Comments:
//
//********************************************************************

STDMETHODIMP CDropTarget::DragLeave ()
{
     OutputDebugString("In IDT::DragLeave\r\n");

     UndrawDragFeedback();

     return ResultFromScode(S_OK);
}
```

CDropTarget::Drop   (SIMPDND Sample)

```
//**********************************************************************
//
// CDropTarget::Drop
//
// Purpose:
//
//      Called when a Drop operation should take place.
//
// Parameters:
//
//      LPDATAOBJECT pDataObj   - IDataObject from Drop source
//      DWORD grfKeyState       - current key state
//      POINTL pointl           - position of mouse
//      LPDWORD pdwEffect       - (IN-OUT) drag effect that should occur
//                                ON INPUT, this is dwOKEffects that source
//                                passed to DoDragDrop API.
//                                ON OUTPUT, this is the effect that we
//                                want to take effect (used to determine
//                                cursor feedback).
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpleSite::Create         SITE.CPP
//      CSimpleSite::InitObject     SITE.CPP
//      OleCreateFromData           OLE2 API
//      DoDragScroll                IDT.CPP
//      QueryDrop                   IDT.CPP
//      InitDragFeedback            IDT.CPP
//      DrawDragFeedback            IDT.CPP
//      UndrawDragFeedback          IDT.CPP
//      GetScode                    OLE2 API
//      ResultFromScode             OLE2 API
//
// Comments:
//      Callee should honor the dwEffects as passed in to determine
//      if the caller allows DROPEFFECT_MOVE.
//
//********************************************************************

STDMETHODIMP CDropTarget::Drop (LPDATAOBJECT pDataObj, DWORD grfKeyState,
POINTL pointl, LPDWORD pdwEffect)
{
     FORMATETC fmtetc;
     SCODE sc = S_OK;

     OutputDebugString("In IDT::Drop\r\n");
     UndrawDragFeedback();

     if (pDataObj && QueryDrop(grfKeyState,pointl,FALSE,pdwEffect))
           {
           m_pDoc->m_lpSite = CSimpleSite::Create(m_pDoc);
           // keep same aspect as drop source
           m_pDoc->m_lpSite->m_dwDrawAspect = m_dwSrcAspect;

           // in order to specify a particular drawing Aspect we must
           // pass a FORMATETC* to OleCreateFromData
           fmtetc.cfFormat = NULL;             // use whatever for drawing
           fmtetc.ptd = NULL;
           fmtetc.lindex = -1;
           fmtetc.dwAspect = m_dwSrcAspect;    // desired drawing aspect
           fmtetc.tymed = TYMED_NULL;

           HRESULT hrErr = OleCreateFromData (
                                         pDataObj,
                                         IID_IOleObject,
                                         OLERENDER_DRAW,
                                         &fmtetc,
                                         &m_pDoc->m_lpSite->m_OleClientSite,
                                         m_pDoc->m_lpSite->m_lpObjStorage,
                                         (LPVOID FAR *)&m_pDoc->m_lpSite-
>m_lpOleObject);

           if (hrErr == NOERROR)
                 {
                 m_pDoc->m_lpSite->InitObject(FALSE /* fCreateNew */);
                 m_pDoc->DisableInsertObject();
                 }
           else
                 sc = GetScode(hrErr);
           }

     return ResultFromScode(sc);
}
```

IOCS.CPP   (SIMPDND Sample)

```
//**********************************************************************
// File name: IOCS.CPP
//
//      Implementation file for COleClientSite
//
// Functions:
//
//      See IOCS.H for class definition
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "iocs.h"
#include "ias.h"
#include "app.h"
#include "site.h"
#include "doc.h"
```

COleClientSite::QueryInterface (SIMPDND Sample)

```
//**********************************************************************
//
// COleClientSite::QueryInterface
//
// Purpose:
//
//      Used for interface negotiation at this interface
//
// Parameters:
//
//      REFIID riid         -   A reference to the interface that is
//                              being queried.
//
//      LPVOID FAR* ppvObj  -   An out parameter to return a pointer to
//                              the interface.
//
// Return Value:
//
//      S_OK                -   The interface is supported.
//      E_NOINTERFACE       -   The interface is not supported
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpleSite::QueryInterface SITE.CPP
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleClientSite::QueryInterface(REFIID riid, LPVOID FAR* ppvObj)
{
     OutputDebugString("In IOCS::QueryInterface\r\n");

     // delegate to the container Site
     return m_pSite->QueryInterface(riid, ppvObj);
}
```

CSimpleApp::AddRef   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleApp::AddRef
//
// Purpose:
//
//      Adds to the reference count at the interface level.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the interface
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) COleClientSite::AddRef()
{
     OutputDebugString("In IOCS::AddRef\r\n");

     // increment the interface reference count (for debugging only)
     ++m_nCount;

     // delegate to the container Site
     return m_pSite->AddRef();
}
```

CSimpleApp::Release   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleApp::Release
//
// Purpose:
//
//      Decrements the reference count at this level
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the interface.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) COleClientSite::Release()
{
     OutputDebugString("In IOCS::Release\r\n");

     // decrement the interface reference count (for debugging only)
     --m_nCount;

     // delegate to the container Site
     return m_pSite->Release();
}
```

COleClientSite::SaveObject (SIMPDND Sample)

```
//**********************************************************************
//
// COleClientSite::SaveObject
//
// Purpose:
//
//      Called by the object when it wants to be saved to persistant
//      storage
//
// Parameters:
//
//      None
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                                Location
//
//      OutputDebugString                       Windows API
//      IOleObject::QueryInterface              Object
//      IPersistStorage::SaveCompleted          Object
//      IPersistStorage::Release                Object
//      OleSave                                 OLE API
//      ResultFromScode                         OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleClientSite::SaveObject()
{
     LPPERSISTSTORAGE lpPS;
     SCODE sc = E_FAIL;

     OutputDebugString("In IOCS::SaveObject\r\n");

     // get a pointer to IPersistStorage
     HRESULT hErr = m_pSite->m_lpOleObject-
>QueryInterface(IID_IPersistStorage, (LPVOID FAR *)&lpPS);

     // save the object
     if (hErr == NOERROR)
            {
            sc = GetScode( OleSave(lpPS, m_pSite->m_lpObjStorage, TRUE) );
            lpPS->SaveCompleted(NULL);
            lpPS->Release();
            }

     return ResultFromScode(sc);
}
```

COleClientSite::GetMoniker   (SIMPDND Sample)

```
//**********************************************************************
//
// COleClientSite::GetMoniker
//
// Purpose:
//
//      Not Implemented
//
// Parameters:
//
//      Not Implemented
//
// Return Value:
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      This function is not implemented because we don't support
//      linking.
//
//********************************************************************

STDMETHODIMP COleClientSite::GetMoniker(DWORD dwAssign, DWORD
dwWhichMoniker, LPMONIKER FAR* ppmk)
{
     OutputDebugString("In IOCS::GetMoniker\r\n");

     // need to null the out pointer
     *ppmk = NULL;

     return ResultFromScode(E_NOTIMPL);
}
```

COleClientSite::GetContainer   (SIMPDND Sample)

```
//**********************************************************************
//
// COleClientSite::GetContainer
//
// Purpose:
//
//      Not Implemented
//
// Parameters:
//
//      Not Implemented
//
// Return Value:
//
//      Not Implemented
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Not Implemented
//
//********************************************************************

STDMETHODIMP COleClientSite::GetContainer(LPOLECONTAINER FAR* ppContainer)
{
     OutputDebugString("In IOCS::GetContainer\r\n");

     // NULL the out pointer
     *ppContainer = NULL;

     return ResultFromScode(E_NOTIMPL);
}
```

COleClientSite::ShowObject (SIMPDND Sample)

```
//**********************************************************************
//
// COleClientSite::ShowObject
//
// Purpose:
//
//      Not Implemented
//
// Parameters:
//
//      Not Implemented
//
// Return Value:
//
//      Not Implemented
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      This function is not implemented because we don't support
//      linking.
//
//********************************************************************

STDMETHODIMP COleClientSite::ShowObject()
{
     OutputDebugString("In IOCS::ShowObject\r\n");
     return NOERROR;
}
```

COleClientSite::OnShowWindow (SIMPDND Sample)

```
//**********************************************************************
//
// COleClientSite::OnShowWindow
//
// Purpose:
//
//      Object calls this method when it is opening/closing non-InPlace
//      Window
//
// Parameters:
//
//      BOOL fShow  - TRUE if Window is opening, FALSE if closing
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      InvalidateRect              Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//********************************************************************

STDMETHODIMP COleClientSite::OnShowWindow(BOOL fShow)
{
     OutputDebugString("In IOCS::OnShowWindow\r\n");
     m_pSite->m_fObjectOpen = fShow;
     InvalidateRect(m_pSite->m_lpDoc->m_hDocWnd, NULL, TRUE);

     // if object window is closing, then bring container window to top
     if (! fShow) {
           BringWindowToTop(m_pSite->m_lpDoc->m_hDocWnd);
           SetFocus(m_pSite->m_lpDoc->m_hDocWnd);
     }
     return ResultFromScode(S_OK);
}
```

COleClientSite::RequestNewObjectLayout (SIMPDND Sample)

```
//********************************************************************
//
// COleClientSite::RequestNewObjectLayout
//
// Purpose:
//
//      Not Implemented
//
// Parameters:
//
//      Not Implemented
//
// Return Value:
//
//      Not Implemented
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Not Implemented
//
//******************************************************************

STDMETHODIMP COleClientSite::RequestNewObjectLayout()
{
     OutputDebugString("In IOCS::RequestNewObjectLayout\r\n");
     return ResultFromScode(E_NOTIMPL);
}
```

SIMPDND.CPP (SIMPDND Sample)

```
//**********************************************************************
// File name: Simple.cpp
//
//      Main source file for the Simple OLE 2.0 object container
//
// Functions:
//
//      WinMain         - Program entry point
//      MainWndProc     - Processes messages for the frame window
//      About           - Processes messages for the about dialog
//      DocWndProc      - Processes messages for the doc window
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "iocs.h"
#include "ias.h"
#include "app.h"
#include "site.h"
#include "doc.h"

// This line is needed for the debug utilities in OLE2UI
extern "C" {
        OLEDBGDATA_MAIN("SIMPDND")
}

CSimpleApp FAR * lpCSimpleApp;
```

WinMain (SIMPDND Sample)

```
//**********************************************************************
//
// WinMain
//
// Purpose:
//
//      Program entry point
//
// Parameters:
//
//      HANDLE hInstance        - Instance handle for this instance
//
//      HANDLE hPrevInstance    - Instance handle for the last instance
//
//      LPSTR lpCmdLine         - Pointer to the command line
//
//      int nCmdShow            - Window State
//
// Return Value:
//
//      msg.wParam
//
// Function Calls:
//      Function                        Location
//
//      CSimpleApp::CSimpleApp          APP.CPP
//      CSimpleApp::fInitApplication    APP.CPP
//      CSimpleApp::fInitInstance       APP.CPP
//      CSimpleApp::HandleAccelerators  APP.CPP
//      CSimpleApp::~CSimpleApp         APP.CPP
//      OleUIInitialize                 OLE2UI
//      OleUIUninitialize               OLE2UI
//      GetMessage                      Windows API
//      TranslateMessage                Windows API
//      DispatchMessage                 Windows API
//
// Comments:
//
//********************************************************************

int PASCAL WinMain(HANDLE hInstance,HANDLE hPrevInstance,LPSTR lpCmdLine,int
nCmdShow)

{
        MSG msg;

        // needed for LRPC to work properly...
        SetMessageQueue(96);

        lpCSimpleApp = new CSimpleApp;
        // we will add one ref count on our App. later when we want to
destroy
        // the App object we will release this  ref count. when the App's
ref
        // count goes to 0, it will be deleted.
        lpCSimpleApp->AddRef();

        // app initialization
        if (!hPrevInstance)
                if (!lpCSimpleApp->fInitApplication(hInstance))
                        return (FALSE);

        // instance initialization
        if (!lpCSimpleApp->fInitInstance(hInstance, nCmdShow))
                return (FALSE);

        /* Initialization required for OLE 2 UI library.  This call is
        **    needed ONLY if we are using the static link version of the UI
        **    library. If we are using the DLL version, we should NOT call
        **    this function in our application.
        */
#if 0
        if (!OleUIInitialize(hInstance, hPrevInstance))
                {
                OleDbgOut("Could not initialize OLEUI library\n");
                return FALSE;
                }
#endif
        // message loop
        while (GetMessage(&msg, NULL, NULL, NULL))
                if (!lpCSimpleApp->HandleAccelerators(&msg))
                        {
                        TranslateMessage(&msg);    /* Translates virtual key
codes */
                        DispatchMessage(&msg);     /* Dispatches message to
window */
                        }

        // De-initialization for UI libraries.  Just like OleUIInitialize,
this
        // funciton is needed ONLY if we are using the static link version
of the
        // OLE UI library.
#if 0
        OleUIUninitialize();
#endif

        // Release the ref count added on the App above. this will make
        // the App's ref count go to 0, and the App object will be deleted.
        lpCSimpleApp->Release();

        return (msg.wParam);          /* Returns the value from
PostQuitMessage */
}
```

MainWndProc   (SIMPDND Sample)

```
//**********************************************************************
//
// MainWndProc
//
// Purpose:
//
//      Processes messages for the frame window
//
// Parameters:
//
//      HWND hWnd       - Window handle for frame window
//
//      UINT message    - Message value
//
//      WPARAM wParam   - Message info
//
//      LPARAM lParam   - Message info
//
// Return Value:
//
//      long
//
// Function Calls:
//      Function                        Location
//
//      CSimpleApp::lCommandHandler     APP.CPP
//      CSimpleApp::DestroyDocs         APP.CPP
//      CSimpleApp::lCreateDoc          APP.CPP
//      CSimpleApp::lSizeHandler        APP.CPP
//      CSimpleDoc::lAddVerbs           DOC.CPP
//      PostQuitMessage                 Windows API
//      DefWindowProc                   Windows API
//
// Comments:
//
//********************************************************************

//@@WTK WIN32, UNICODE
//long FAR PASCAL _export MainWndProc(HWND hWnd,UINT message,WPARAM
wParam,LPARAM lParam)
long FAR PASCAL EXPORT MainWndProc(HWND hWnd,UINT message,WPARAM
wParam,LPARAM lParam)

{

        switch (message)
                {
                case WM_COMMAND:           // message: command from
application menu
                        return lpCSimpleApp->lCommandHandler(hWnd, message,
wParam, lParam);
                        break;
                case WM_CREATE:
                        return lpCSimpleApp->lCreateDoc(hWnd, message,
wParam, lParam);
                        break;

                case WM_DESTROY:                  // message: window being
destroyed
                        lpCSimpleApp->DestroyDocs();  // need to destroy the
doc...
                        PostQuitMessage(0);
                        break;

                case WM_INITMENUPOPUP:
                        // is this the edit menu?
                        if ( LOWORD(lParam) == 1)
                                return lpCSimpleApp->m_lpDoc->lAddVerbs();

                        break;

                case WM_SIZE:
                        return lpCSimpleApp->lSizeHandler(hWnd, message,
wParam, lParam);

                default:                          // Passes it on if
unproccessed
                        return (DefWindowProc(hWnd, message, wParam,
lParam));
                }
                return (NULL);
}
```

About   (SIMPDND Sample)

```
//*********************************************************************
//
// About
//
// Purpose:
//
//      Processes dialog box messages
//
// Parameters:
//
//      HWND hWnd       - Window handle for dialog box
//
//      UINT message    - Message value
//
//      WPARAM wParam   - Message info
//
//      LPARAM lParam   - Message info
//
// Return Value:
//
// Function Calls:
//      Function                    Location
//
//      EndDialog                   Windows API
//
// Comments:
//
//********************************************************************

//@@WTK WIN32, UNICODE
//BOOL FAR PASCAL _export About(HWND hDlg,unsigned message,WORD wParam,LONG
lParam)
BOOL FAR PASCAL EXPORT About(HWND hDlg,UINT message,WPARAM wParam,LPARAM
lParam)

{
        switch (message) {
        case WM_INITDIALOG:                /* message: initialize dialog box
*/
                return (TRUE);

        case WM_COMMAND:                      /* message: received a command
*/
                //@@WTK WIN32, UNICODE
                //if (wParam == IDOK                /* "OK" box selected?
*/
                if (LOWORD(wParam) == IDOK                /* "OK" box
selected?           */
                //@@WTK WIN32, UNICODE
                //|| wParam == IDCANCEL) {      /* System menu close
command? */
```

```
                || LOWORD(wParam) == IDCANCEL) {      /* System menu close
command? */
                EndDialog(hDlg, TRUE);        /* Exits the dialog box
*/
                return (TRUE);
                }
                break;
        }
        return (FALSE);                           /* Didn't process a
message    */
}
```

DocWndProc   (SIMPDND Sample)

```
//**********************************************************************
//
// DocWndProc
//
// Purpose:
//
//      Processes dialog box messages
//
// Parameters:
//
//      HWND hWnd       - Window handle for doc window
//
//      UINT message    - Message value
//
//      WPARAM wParam   - Message info
//
//      LPARAM lParam   - Message info
//
// Return Value:
//
// Function Calls:
//      Function                           Location
//
//      CSimpleApp::PaintApp               APP.CPP
//      BeginPaint                         Windows API
//      EndPaint                           Windows API
//      DefWindowProc                      Windows API
//      IOleObject::QueryInterface         Object
//      IOleObject::DoVerb                 Object
//      CSimpleSite::GetObjRect            SITE.CPP
//      CSimpleDoc::QueryDrag              DOC.CPP
//      CSimpleDoc::DoDragDrop             DOC.CPP
//      SetTimer                           Windows API
//      KillTimer                          Windows API
//      SetCapture                         Windows API
//      ReleaseCapture                     Windows API
//
// Comments:
//
//********************************************************************

//@@WTK WIN32, UNICODE
//long FAR PASCAL _export DocWndProc(HWND hWnd,UINT message,WPARAM
wParam,LPARAM lParam)
long FAR PASCAL EXPORT DocWndProc(HWND hWnd,UINT message,WPARAM
wParam,LPARAM lParam)
{
        HDC hDC;
        PAINTSTRUCT ps;

        switch (message) {
                case WM_PAINT:
                        hDC = BeginPaint(hWnd, &ps);

                        if (lpCSimpleApp)
                                lpCSimpleApp->PaintApp (hDC);

                        EndPaint(hWnd, &ps);
                        break;

                case WM_LBUTTONDBLCLK:
                        {
                        POINT pt;

                        pt.x = (int)(short)LOWORD (lParam );
                        pt.y = (int)(short)HIWORD (lParam );

                        if (lpCSimpleApp->m_lpDoc->m_lpSite &&
                                lpCSimpleApp->m_lpDoc->m_lpSite-
>m_lpOleObject)
                                {
                                RECT rect;

                                lpCSimpleApp->m_lpDoc->m_lpSite-
>GetObjRect(&rect);

                                if ( PtInRect(&rect, pt) )
                                        {
                                        // Execute object's default verb
                                        lpCSimpleApp->m_lpDoc->m_lpSite-
>m_lpOleObject->DoVerb(
                                                        OLEIVERB_PRIMARY,
(LPMSG)&message,
                                                        &lpCSimpleApp-
>m_lpDoc->m_lpSite->m_OleClientSite,
                                                        -1, hWnd, &rect);
                                        }
                                }
                        }
                        break;

                case WM_LBUTTONDOWN:
                        {
                        POINT pt;

                        pt.x = (int)(short)LOWORD (lParam );
                        pt.y = (int)(short)HIWORD (lParam );

                        /* OLE2NOTE: check if this is a button down on the
region
                        **    that is a handle to start a drag operation.
for us,
                        **    this this is any where in the window. we
                        **    do NOT want to start a drag immediately; we
want to
```

```
                        **    wait until the mouse moves a certain
threshold. or a
                        **    certain amount of time has elapsed. if
                        **    LButtonUp comes before the drag is started,
then
                        **    the fPendingDrag state is cleared. we must
capture
                        **    the mouse to ensure the modal state is handled
                        **    properly.
                        */
                        if (lpCSimpleApp->m_lpDoc->QueryDrag(pt) )
                                {
                                lpCSimpleApp->m_lpDoc->m_fPendingDrag =
TRUE;
                                lpCSimpleApp->m_lpDoc->m_ptButDown = pt;
                                SetTimer(hWnd, 1, lpCSimpleApp-
>m_nDragDelay, NULL);
                                SetCapture(hWnd);
                                }
                        break;
                        }

                case WM_LBUTTONUP:

                        if (lpCSimpleApp->m_lpDoc->m_fPendingDrag)
                                {
                                /* ButtonUP came BEFORE distance/time
threshholds were
                                **    exceeded. clear fPendingDrag state.
                                */
                                ReleaseCapture();
                                KillTimer(hWnd, 1);
                                lpCSimpleApp->m_lpDoc->m_fPendingDrag =
FALSE;
                                }
                        break;

                case WM_MOUSEMOVE:
                        {
                        if (lpCSimpleApp->m_lpDoc->m_fPendingDrag)
                                {
                                int  x = (int)(short)LOWORD (lParam );
                                int  y = (int)(short)HIWORD (lParam );
                                POINT pt = lpCSimpleApp->m_lpDoc-
>m_ptButDown;
                                int nDragMinDist = lpCSimpleApp-
>m_nDragMinDist;

                                if (! ( ((pt.x - nDragMinDist) <= x)
                                                && (x <= (pt.x +
nDragMinDist))
                                                && ((pt.y - nDragMinDist) <=
y)
```

```
                                                        && (y <= (pt.y + 
nDragMinDist)) ) )
                                                {
                                                // mouse moved beyond threshhold to 
start drag
                                                ReleaseCapture();
                                                KillTimer(hWnd, 1);
                                                lpCSimpleApp->m_lpDoc-
>m_fPendingDrag = FALSE;

                                                // perform the modal drag/drop 
operation.
                                                lpCSimpleApp->m_lpDoc-
>DoDragDrop( );
                                                }
                                        }
                                break;
                                }

                case WM_TIMER:
                                {
                                // drag time delay threshhold exceeded -- start drag
                                ReleaseCapture();
                                KillTimer(hWnd, 1);
                                lpCSimpleApp->m_lpDoc->m_fPendingDrag = FALSE;

                                // perform the modal drag/drop operation.
                                lpCSimpleApp->m_lpDoc->DoDragDrop( );
                                break;
                                }

                default:                                /* Passes it on if 
unproccessed */
                                return (DefWindowProc(hWnd, message, wParam, 
lParam));
        }
        return (NULL);
}
```

SITE.CPP (SIMPDND Sample)

```
//**********************************************************************
// File name: SITE.CPP
//
//      Implementation file for CSimpleSite
//
// Functions:
//
//      See SITE.H for class definition
//
// Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "iocs.h"
#include "ias.h"
#include "app.h"
#include "site.h"
#include "doc.h"
```

CSimpleSite::Create (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleSite::Create
//
// Purpose:
//
//      Creation routine for CSimpleSite
//
// Parameters:
//
//      CSimpleDoc FAR *lpDoc   - Pointer to CSimpleDoc
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      IStorage::CreateStorage     OLE API
//      assert                      C Runtime
//
// Comments:
//
//********************************************************************

CSimpleSite FAR * CSimpleSite::Create(CSimpleDoc FAR *lpDoc)
{
     CSimpleSite FAR * lpTemp = new CSimpleSite(lpDoc);

     if (!lpTemp)
           return NULL;

     // create a sub-storage for the object
     //@@WTK WIN32, UNICODE
     //HRESULT hErr = lpDoc->m_lpStorage->CreateStorage( "Object",
     HRESULT hErr = lpDoc->m_lpStorage->CreateStorage( OLESTR("Object"),
                       STGM_READWRITE | STGM_TRANSACTED |
STGM_SHARE_EXCLUSIVE,
                       0,
                       0,
                       &lpTemp->m_lpObjStorage);

     assert(hErr == NOERROR);

     if (hErr != NOERROR)
           {
           delete lpTemp;
           return NULL;
           }

     // we will add one ref count on our Site. later when we want to destroy
     // the Site object we will release this  ref count. when the Site's ref
     // count goes to 0, it will be deleted.
     lpTemp->AddRef();

     return lpTemp;
}
```

CSimpleSite::CSimpleSite (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleSite::CSimpleSite
//
// Purpose:
//
//      Constructor for CSimpleSite
//
// Parameters:
//
//      CSimpleDoc FAR *lpDoc   - Pointer to CSimpleDoc
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
// Comments:
//
//********************************************************************
#pragma warning(disable : 4355)  // turn off this warning.  This warning
                                               // tells us that we are
passing this in
                                               // an initializer, before
"this" is through
                                               // initializing.  This is ok,
because
                                               // we just store the ptr in
the other
                                               // constructors

CSimpleSite::CSimpleSite (CSimpleDoc FAR *lpDoc) : m_OleClientSite(this),

m_AdviseSink(this)
#pragma warning (default : 4355)  // Turn the warning back on
{
     // remember the pointer to the doc
     m_lpDoc = lpDoc;

     // clear the reference count
     m_nCount = 0;

     m_dwDrawAspect = DVASPECT_CONTENT;
     m_lpOleObject = NULL;
     m_fObjectOpen = FALSE;
}
```

CSimpleSite::~CSimpleSite   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleSite::~CSimpleSite
//
// Purpose:
//
//      Destructor for CSimpleSite
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                                Location
//
//      OutputDebugString                       Windows API
//      IOleObject::Release                     Object
//      IStorage::Release                       OLE API
//
// Comments:
//
//********************************************************************

CSimpleSite::~CSimpleSite ()
{
     OutputDebugString ("In CSimpleSite's Destructor \r\n");

     if (m_lpOleObject)
        m_lpOleObject->Release();

     if (m_lpObjStorage)
        m_lpObjStorage->Release();
}
```

CSimpleSite::CloseOleObject (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleSite::CloseOleObject
//
// Purpose:
//
//      Call IOleObject::Close on the object of the CSimpleSite
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                                Location
//
//      OutputDebugString                       Windows API
//      IOleObject::Close                       Object
//
// Comments:
//
//********************************************************************

void CSimpleSite::CloseOleObject (void)
{
     LPVIEWOBJECT lpViewObject = NULL;

     OutputDebugString ("In CSimpleSite::CloseOleObject \r\n");

     if (m_lpOleObject)
        {
        m_lpOleObject->Close(OLECLOSE_NOSAVE);
        }
}
```

CSimpleSite::UnloadOleObject (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleSite::UnloadOleObject
//
// Purpose:
//
//      Close and release all pointers to the object of the CSimpleSite
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                                Location
//
//      OutputDebugString                       Windows API
//      CSimpleSite::CloseOleObject             SITE.CPP
//      IOleObject::QueryInterface              Object
//      IViewObject::SetAdvise                  Object
//      IViewObject::Release                    Object
//      IStorage::Release                       OLE API
//
// Comments:
//
//********************************************************************

void CSimpleSite::UnloadOleObject (void)
{
     OutputDebugString ("In CSimpleSite::UnloadOleObject \r\n");

     if (m_lpOleObject)
        {
        LPVIEWOBJECT lpViewObject;
        CloseOleObject();    // ensure object is closed; NOP if already
closed

        m_lpOleObject->QueryInterface(IID_IViewObject, (LPVOID FAR
*)&lpViewObject);

        if (lpViewObject)
              {
              // Remove the view advise
              lpViewObject->SetAdvise(m_dwDrawAspect, 0, NULL);
              lpViewObject->Release();
              }

        m_lpOleObject->Release();
        m_lpOleObject = NULL;
        }
}
```

CSimpleSite::QueryInterface   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleSite::QueryInterface
//
// Purpose:
//
//      Used for interface negotiation of the container Site.
//
// Parameters:
//
//      REFIID riid         -   A reference to the interface that is
//                              being queried.
//
//      LPVOID FAR* ppvObj  -   An out parameter to return a pointer to
//                              the interface.
//
// Return Value:
//
//      S_OK    -   The interface is supported.
//      S_FALSE -   The interface is not supported
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      IsEqualIID                  OLE API
//      ResultFromScode             OLE API
//      CSimpleSite::AddRef          OBJ.CPP
//      COleClientSite::AddRef      IOCS.CPP
//      CAdviseSink::AddRef         IAS.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP CSimpleSite::QueryInterface(REFIID riid, LPVOID FAR* ppvObj)
{
     OutputDebugString("In CSimpleSite::QueryInterface\r\n");

     *ppvObj = NULL;     // must set out pointer parameters to NULL

     if ( riid == IID_IUnknown)
           {
           AddRef();
           *ppvObj = this;
           return ResultFromScode(S_OK);
           }

     if ( riid == IID_IOleClientSite)
           {
           m_OleClientSite.AddRef();
```

```
            *ppvObj = &m_OleClientSite;
            return ResultFromScode(S_OK);
            }

      if ( riid == IID_IAdviseSink)
            {
            m_AdviseSink.AddRef();
            *ppvObj = &m_AdviseSink;
            return ResultFromScode(S_OK);
            }

      // Not a supported interface
      return ResultFromScode(E_NOINTERFACE);
}
```

CSimpleSite::AddRef   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleSite::AddRef
//
// Purpose:
//
//      Increments the reference count of the container Site.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the site.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//*********************************************************************

STDMETHODIMP_(ULONG) CSimpleSite::AddRef()
{
     OutputDebugString("In CSimpleSite::AddRef\r\n");

     return ++m_nCount;
}
```

CSimpleSite::Release   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleSite::Release
//
// Purpose:
//
//      Decrements the reference count of the container Site
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the Site.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpleSite::Release()
{
     OutputDebugString("In CSimpleSite::Release\r\n");

     if (--m_nCount == 0) {
           delete this;
           return 0;
     }
     return m_nCount;
}
```

CSimpleSite::InitObject   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleSite::InitObject
//
// Purpose:
//
//      Used to initialize a newly create object (can't be done in the
//      constructor).
//
// Parameters:
//
//      BOOL fCreateNew -   TRUE if insert NEW object
//                          FALSE if create object FROM FILE
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                        Location
//
//      IOleObject::SetHostNames        Object
//      IOleObject::QueryInterface      Object
//      IViewObject2::GetExtent         Object
//      IOleObject::DoVerb              Object
//      IViewObject::SetAdvise          Object
//      IViewObject::Release            Object
//      GetClientRect                   Windows API
//      OleSetContainedObject           OLE API
//
// Comments:
//
//********************************************************************

void CSimpleSite::InitObject(BOOL fCreateNew)
{
     LPVIEWOBJECT2 lpViewObject2;
     RECT rect;

     // Set a View Advise
     m_lpOleObject->QueryInterface(IID_IViewObject2,(LPVOID FAR
*)&lpViewObject2);
     lpViewObject2->SetAdvise(m_dwDrawAspect, ADVF_PRIMEFIRST,
&m_AdviseSink);

     // get the initial size of the object
     lpViewObject2->GetExtent(m_dwDrawAspect, -1 /*lindex*/, NULL /*ptd*/,
&m_sizel);
     GetObjRect(&rect);  // get the rectangle of the object in pixels
     lpViewObject2->Release();

     // give the object the name of the container app/document
     //@@WTK WIN32, UNICODE
     //m_lpOleObject->SetHostNames("Simple Application", "Simple OLE 2.0
Drag/Drop Container");
     m_lpOleObject->SetHostNames(OLESTR("Simple Application"),
OLESTR("Simple OLE 2.0 Drag/Drop Container"));

     // inform object handler/DLL object that it is used in the embedding
container's context
     OleSetContainedObject(m_lpOleObject, TRUE);

     if (fCreateNew) {
        // force new object to save to guarantee valid object in our
storage.
        // OLE 1.0 objects may close w/o saving. this is NOT necessary if
the
        // object is created FROM FILE; its data in storage is already
valid.
        m_OleClientSite.SaveObject();

        // we only want to DoVerb(SHOW) if this is an InsertNew object.
        // we should NOT DoVerb(SHOW) if the object is created FromFile.
        m_lpOleObject->DoVerb(
                    OLEIVERB_SHOW,
                    NULL,
                    &m_OleClientSite,
                    -1,
                    m_lpDoc->m_hDocWnd,
                    &rect);
     }
}
```

CSimpleSite::PaintObj (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleSite::PaintObj
//
// Purpose:
//
//      Paints the object
//
// Parameters:
//
//      HDC hDC     - Device context of the document window
//
// Return Value:
//
// Function Calls:
//      Function                        Location
//
//      IOleObject::QueryInterface      Object
//      IViewObject::GetColorSet        Object
//      IViewObject::Release            Object
//      SetMapMode                      Windows API
//      LPtoDP                          Windows API
//      CreateHatchBrush                Windows API
//      SelectObject                    Windows API
//      DeleteObject                    Windows API
//      CreatePalette                   Windows API
//      SelectPalette                   Windows API
//      RealizePalette                  Windows API
//      OleStdFree                      OUTLUI Function
//      OleDraw                         OLE API
//
// Comments:
//
//********************************************************************

void CSimpleSite::PaintObj(HDC hDC)
{
RECT rect;

     // need to check to make sure there is a valid object
     // available.  This is needed if there is a paint msg
     // between the time that CSimpleSite is instantiated
     // and OleUIInsertObject returns.
     if (!m_lpOleObject)
           return;

     // convert it to pixels
     GetObjRect(&rect);

     LPLOGPALETTE pColorSet = NULL;
     LPVIEWOBJECT lpView = NULL;
     // get a pointer to IViewObject
     m_lpOleObject->QueryInterface(IID_IViewObject,(LPVOID FAR *) &lpView);

     // if the QI succeeds, get the LOGPALETTE for the object
     if (lpView)
           lpView->GetColorSet(m_dwDrawAspect, -1, NULL, NULL, NULL,
&pColorSet);

     HPALETTE hPal=NULL;
     HPALETTE hOldPal=NULL;

     // if a LOGPALETTE was returned (not guarateed), create the palette and
     // realize it.  NOTE: A smarter application would want to get the
LOGPALETTE
     // for each of its visible objects, and try to create a palette that
     // satisfies all of the visible objects.  ALSO: OleStdFree() is use to
     // free the returned LOGPALETTE.
     if ((pColorSet))
           {
           hPal = CreatePalette((const LPLOGPALETTE) pColorSet);
           hOldPal = SelectPalette(hDC, hPal, FALSE);
           RealizePalette(hDC);
           OleStdFree(pColorSet);
           }

     // draw the object
     OleDraw(m_lpOleObject, m_dwDrawAspect, hDC, &rect);

     // if the object is open, draw a hatch rect.
     if (m_fObjectOpen)
           {
           HBRUSH hBrush = CreateHatchBrush ( HS_BDIAGONAL, RGB(0,0,0) );
           HBRUSH hOldBrush = SelectObject (hDC, hBrush);
           SetROP2(hDC, R2_MASKPEN);
           Rectangle (hDC, rect.left, rect.top, rect.right, rect.bottom);
           SelectObject(hDC, hOldBrush);
           DeleteObject(hBrush);
           }

     // if we created a palette, restore the old one, and destroy
     // the object.
     if (hPal)
           {
           SelectPalette(hDC,hOldPal,FALSE);
           DeleteObject(hPal);
           }

     // if a view pointer was successfully returned, it needs to be
released.
     if (lpView)
           lpView->Release();
}
```

CSimpleSite::GetObjRect   (SIMPDND Sample)

```
//**********************************************************************
//
// CSimpleSite::GetObjRect
//
// Purpose:
//
//      Retrieves the rect of the object in pixels
//
// Parameters:
//
//      LPRECT lpRect - Rect structure filled with object's rect in pixels
//
// Return Value:
//
// Function Calls:
//      Function                        Location
//
//      XformWidthInHimetricToPixels    OUTLUI Function
//      XformHeightInHimetricToPixels   OUTLUI Function
//
// Comments:
//
//********************************************************************

void CSimpleSite::GetObjRect(LPRECT lpRect)
{
     // convert it to pixels
     lpRect->left = lpRect->top = 0;
     lpRect->right = XformWidthInHimetricToPixels(NULL,(int)m_sizel.cx);
     lpRect->bottom = XformHeightInHimetricToPixels(NULL,(int)m_sizel.cy);
}
```

SIMPSVR

Simpsvr -------- This sample is the simplest OLE 2.0 object that can be written and still support the visual editing feature. The object that this server supports is a colored square with a black border.

See the MAKEFILE for compilation instructions.

MAKEFILE   (SIMPSVR Sample)

```
!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

all: simpsvr.exe

simpsvr.exe:  app.obj         \
            doc.obj         \
            icf.obj         \
            ido.obj        \
            iec.obj        \
            ioipao.obj        \
            ioipo.obj        \
            ioo.obj        \
            ips.obj        \
            obj.obj         \
            simpsvr.obj     \
            simpsvr.res
  $(link) $(linkdebug) $(guiflags) -machine:$(CPU) -out:$*.exe $** $
(olelibs) ole2ui.lib

.cpp.obj:
    $(cc) $(cdebug) $(cflags) $(cvars) $*.cpp

simpsvr.res: simpsvr.rc simpsvr.h
    $(rc) -r -I..\ole2ui -I..\ole2ui\res\usa -I..\ole2ui\res\static
simpsvr.rc
```

APP.H (SIMPSVR Sample)

```
//**********************************************************************
// File name: app.h
//
//      Definition of CSimpSvrApp
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#if !defined( _APP_H_)
#define _APP_H_
#include <stablize.h>

class CSimpSvrDoc;
interface CClassFactory;

class CSimpSvrApp : public IUnknown, public CSafeRefCount
{
private:

        HWND m_hAppWnd;             // main window handle
        HINSTANCE m_hInst;          // application instance
        BOOL m_fStartByOle;         // TRUE if app started by OLE
        DWORD m_dwRegisterClass;    // returned by RegisterClassFactory

        HMENU m_hMainMenu;
        HMENU m_hColorMenu;
        HMENU m_hHelpMenu;


        LPOLEOBJECT m_OleObject;    // pointer to "dummy" object


        CSimpSvrDoc FAR * m_lpDoc;   // pointer to document object
        BOOL m_fInitialized;         // OLE initialization flag

        RECT nullRect;               // used in inplace negotiation

public:
        // IUnknown Interfaces
        STDMETHODIMP QueryInterface(REFIID riid, LPVOID FAR* ppvObj);
        STDMETHODIMP_(ULONG) AddRef();
        STDMETHODIMP_(ULONG) Release();

        // Initialization methods

        CSimpSvrApp();           // Constructor
        ~CSimpSvrApp();          // Destructor


        BOOL fInitApplication (HANDLE hInstance);
        BOOL fInitInstance (HANDLE hInstance, int nCmdShow, CClassFactory
FAR * lpClassFactory);
```

```
        // Message handling methods

        long lCommandHandler (HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam);
        long lSizeHandler (HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam);
        long lCreateDoc (HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam);
        void PaintApp(HDC hDC);

        //  Utility functions
        void ParseCmdLine(LPSTR lpCmdLine);
        void SetStatusText();
        BOOL IsInPlaceActive();
        void ShowAppWnd(int nCmdShow=SW_SHOWNORMAL);
        void HideAppWnd();


        // member variable access
        inline HWND GethAppWnd() { return m_hAppWnd; };
        inline HINSTANCE GethInst() { return m_hInst; };
        inline BOOL IsStartedByOle() { return m_fStartByOle; };
        inline BOOL IsInitialized() { return m_fInitialized; };
        inline DWORD GetRegisterClass() { return m_dwRegisterClass; };
        inline CSimpSvrDoc FAR * GetDoc() { return m_lpDoc; };
        inline void ClearDoc() { m_lpDoc = NULL; };
        inline LPOLEOBJECT GetOleObject() { return m_OleObject; };
                inline HMENU GetMainMenu() { return m_hMainMenu; };
        inline HMENU GetColorMenu() { return m_hColorMenu; };
        inline HMENU GetHelpMenu() { return m_hHelpMenu; } ;


        friend interface CClassFactory;  // make the contained class a
friend
};

#endif
```

DOC.H   (SIMPSVR Sample)

```
//***********************************************************************
// File name: doc.h
//
//      Definition of CSimpSvrDoc
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//***********************************************************************

#if !defined( _DOC_H_ )
#define _DOC_H_
#include <stablize.h>

class CSimpSvrApp;
class CSimpSvrObj;

class CSimpSvrDoc : IUnknown, public CSafeRefCount
{
private:
        CSimpSvrApp FAR * m_lpApp;
        CSimpSvrObj FAR * m_lpObj;

        HWND m_hDocWnd;
        HWND m_hHatchWnd;
        BOOL m_fClosing;

public:
        static CSimpSvrDoc FAR * Create(CSimpSvrApp FAR *lpApp, LPRECT
lpRect,HWND hWnd);

        CSimpSvrDoc();
        CSimpSvrDoc(CSimpSvrApp FAR *lpApp, HWND hWnd);
        ~CSimpSvrDoc();

// IUnknown Interfaces
        STDMETHODIMP QueryInterface(REFIID riid, LPVOID FAR* ppvObj);
        STDMETHODIMP_(ULONG) AddRef();
        STDMETHODIMP_(ULONG) Release();

        long lResizeDoc(LPRECT lpRect);
        long lAddVerbs();

        BOOL Load(LPSTR lpszFileName);
        void PaintDoc(HDC hDC);
        void lButtonDown(WPARAM wParam,LPARAM lParam);

        HRESULT CreateObject(REFIID riid, LPVOID FAR *ppvObject);

        void Close();
        void SetStatusText();
        void ShowDocWnd();
        void ShowHatchWnd();
        void CSimpSvrDoc::HideDocWnd();
        void CSimpSvrDoc::HideHatchWnd();

// member access
        inline HWND GethDocWnd() { return m_hDocWnd; };
        inline HWND GethHatchWnd() { return m_hHatchWnd; };
        inline HWND GethAppWnd() { return m_lpApp->GethAppWnd(); };
        inline CSimpSvrApp FAR * GetApp() { return m_lpApp; };
        inline CSimpSvrObj FAR * GetObj() { return m_lpObj; };
        inline void ClearObj() { m_lpObj = NULL; };

};

#endif
```

ICF.H   (SIMPSVR Sample)

```
//**********************************************************************
// File name: icf.h
//
//      Definition of CClassFactory
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#if !defined( _ICF_H_)
#define _ICF_H_

class CSimpSvrApp;

interface CClassFactory :  IClassFactory
{
private:
     int m_nCount;               // reference count
     CSimpSvrApp FAR * m_lpApp;

public:
     CClassFactory::CClassFactory(CSimpSvrApp FAR * lpApp)
           {
           OutputDebugString("In CClassFactory's Constructor\r\n");
           m_lpApp = lpApp;
           m_nCount = 0;
           };
     CClassFactory::~CClassFactory()
        {
        OutputDebugString("In CClassFactory's Destructor\r\n");
        };

     // IUnknown Methods

     STDMETHODIMP QueryInterface (REFIID riid, LPVOID FAR* ppvObj);
     STDMETHODIMP_(ULONG) AddRef ();
     STDMETHODIMP_(ULONG) Release ();

     STDMETHODIMP CreateInstance (LPUNKNOWN pUnkOuter,
                                          REFIID riid,
                                          LPVOID FAR* ppvObject);
     STDMETHODIMP LockServer ( BOOL fLock);

};

#endif
```

IDO.H (SIMPSVR Sample)

```
//**********************************************************************
// File name: ido.h
//
//      Definition of CDataObject
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************
#if !defined( _IDO_H_)
#define _IDO_H_


#include <ole2.h>
#include "obj.h"

class CSimpSvrObj;

interface CDataObject : public IDataObject
{
private:
     CSimpSvrObj FAR * m_lpObj;
     int m_nCount;

public:
     CDataObject::CDataObject(CSimpSvrObj FAR * lpSimpSvrObj)
           {
           m_lpObj = lpSimpSvrObj;
           m_nCount = 0;
           };

     CDataObject::~CDataObject() {};

     STDMETHODIMP QueryInterface (REFIID riid, LPVOID FAR* ppvObj);
     STDMETHODIMP_(ULONG) AddRef ();
     STDMETHODIMP_(ULONG) Release ();

     STDMETHODIMP DAdvise  ( FORMATETC FAR* pFormatetc, DWORD advf,
                            LPADVISESINK pAdvSink, DWORD FAR*
pdwConnection);
     STDMETHODIMP DUnadvise  ( DWORD dwConnection);
     STDMETHODIMP EnumDAdvise  ( LPENUMSTATDATA FAR* ppenumAdvise);
     STDMETHODIMP EnumFormatEtc  ( DWORD dwDirection,
                                                LPENUMFORMATETC FAR*
ppenumFormatEtc);
     STDMETHODIMP GetCanonicalFormatEtc  ( LPFORMATETC pformatetc,
                                                            LPFORMATETC
pformatetcOut);
     STDMETHODIMP GetData  ( LPFORMATETC pformatetcIn, LPSTGMEDIUM
pmedium );
     STDMETHODIMP GetDataHere  ( LPFORMATETC pformatetc, LPSTGMEDIUM pmedium
);
     STDMETHODIMP QueryGetData  ( LPFORMATETC pformatetc );
     STDMETHODIMP SetData  ( LPFORMATETC pformatetc, STGMEDIUM FAR *
pmedium,
                                                BOOL fRelease);


};

#endif
```

IEC.H (SIMPSVR Sample)

```
//**********************************************************************
// File name: iec.h
//
//      Definition of CExternalConnection
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************
#if !defined( _IEC_H_)
#define _IEC_H_


#include <ole2.h>
#include "obj.h"

class CSimpSvrObj;

interface CExternalConnection : public IExternalConnection
{
private:
     CSimpSvrObj FAR * m_lpObj;  // Ptr to object
     int m_nCount;               // Ref count
     DWORD m_dwStrong;           // Connection Count

public:
     CExternalConnection::CExternalConnection(CSimpSvrObj FAR *
lpSimpSvrObj)
           {
           m_lpObj = lpSimpSvrObj;
           m_nCount = 0;
           m_dwStrong = 0;
           };

     CExternalConnection::~CExternalConnection() {};

     // *** IUnknown methods ***
     STDMETHODIMP QueryInterface (REFIID riid, LPVOID FAR* ppvObj);
     STDMETHODIMP_(ULONG) AddRef ();
     STDMETHODIMP_(ULONG) Release ();

     // *** IExternalConnection methods ***
     STDMETHODIMP_(DWORD) AddConnection (DWORD extconn, DWORD reserved);
     STDMETHODIMP_(DWORD) ReleaseConnection (DWORD extconn, DWORD reserved,
BOOL fLastReleaseCloses);
};

#endif
```

IOIPAO.H (SIMPSVR Sample)

```
//**********************************************************************
// File name: IOIPAO.H
//
//      Definition of COleInPlaceActiveObject
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#if !defined( _IOIPAO_H_)
#define _IOIPAO_H_


#include <ole2.h>
#include "obj.h"

class CSimpSvrObj;

interface COleInPlaceActiveObject : public IOleInPlaceActiveObject
{
private:
     CSimpSvrObj FAR * m_lpObj;
     int m_nCount;

public:
     COleInPlaceActiveObject::COleInPlaceActiveObject(CSimpSvrObj FAR *
lpSimpSvrObj)
           {
           m_lpObj = lpSimpSvrObj;     // set up the back ptr
           m_nCount = 0;            // clear the ref count.
           };
     COleInPlaceActiveObject::~COleInPlaceActiveObject() {};   // destructor

// IUnknown Methods

     STDMETHODIMP QueryInterface (REFIID riid, LPVOID FAR* ppvObj);
     STDMETHODIMP_(ULONG) AddRef ();
     STDMETHODIMP_(ULONG) Release ();

     STDMETHODIMP OnDocWindowActivate  ( BOOL fActivate) ;
     STDMETHODIMP OnFrameWindowActivate  ( BOOL fActivate) ;
     STDMETHODIMP GetWindow  ( HWND FAR* lphwnd);
     STDMETHODIMP ContextSensitiveHelp  ( BOOL fEnterMode);
     STDMETHODIMP TranslateAccelerator  ( LPMSG lpmsg);
     STDMETHODIMP ResizeBorder  ( LPCRECT lprectBorder,
                                                LPOLEINPLACEUIWINDOW
lpUIWindow,
                                                BOOL fFrameWindow);
     STDMETHODIMP EnableModeless  ( BOOL fEnable);

};

#endif
```

IOIPO.H (SIMPSVR Sample)

```
//**********************************************************************
// File name: ioipo.h
//
//      Definition of COleInPlaceObject
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#if !defined( _IOIPO_H_)
#define _IOIPO_H_


#include <ole2.h>
#include "obj.h"

class CSimpSvrObj;

interface COleInPlaceObject : public IOleInPlaceObject
{
private:
     CSimpSvrObj FAR * m_lpObj;
     int m_nCount;

public:
     COleInPlaceObject::COleInPlaceObject(CSimpSvrObj FAR * lpSimpSvrObj)
           {
           m_lpObj = lpSimpSvrObj;
           m_nCount = 0;
           };
     COleInPlaceObject::~COleInPlaceObject() {};

//  IUnknown Methods
     STDMETHODIMP QueryInterface (REFIID riid, LPVOID FAR* ppvObj);
     STDMETHODIMP_(ULONG) AddRef ();
     STDMETHODIMP_(ULONG) Release ();

     STDMETHODIMP InPlaceDeactivate  ();
     STDMETHODIMP UIDeactivate  () ;
     STDMETHODIMP SetObjectRects  ( LPCRECT lprcPosRect, LPCRECT
lprcClipRect);
     STDMETHODIMP GetWindow  ( HWND FAR* lphwnd) ;
     STDMETHODIMP ContextSensitiveHelp  ( BOOL fEnterMode);
     STDMETHODIMP ReactivateAndUndo  ();
};

#endif
```

IOO.H (SIMPSVR Sample)

```
//**********************************************************************
// File name: ioo.h
//
//      Definition of COleObject
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************
#if !defined( _IOO_H_)
#define _IOO_H_


#include <ole2.h>
#include "obj.h"

class CSimpSvrObj;

interface COleObject : public IOleObject
{
private:
     CSimpSvrObj FAR * m_lpObj;
     int m_nCount;
     BOOL m_fOpen;

public:
     COleObject::COleObject(CSimpSvrObj FAR * lpSimpSvrObj)
           {
           m_lpObj = lpSimpSvrObj;
           m_nCount = 0;
           m_fOpen = FALSE;
           };
     COleObject::~COleObject()
           {
           };
     STDMETHODIMP QueryInterface (REFIID riid, LPVOID FAR* ppvObj);
     STDMETHODIMP_(ULONG) AddRef ();
     STDMETHODIMP_(ULONG) Release ();

     STDMETHODIMP SetClientSite (LPOLECLIENTSITE pClientSite);
     STDMETHODIMP Advise (LPADVISESINK pAdvSink, DWORD FAR* pdwConnection);
     //@@WTK WIN32, UNICODE
     //STDMETHODIMP SetHostNames  ( LPCSTR szContainerApp, LPCSTR
szContainerObj);
     STDMETHODIMP SetHostNames  ( LPCOLESTR szContainerApp, LPCOLESTR
szContainerObj);
     STDMETHODIMP DoVerb  (  LONG iVerb,
                                         LPMSG lpmsg,
                                         LPOLECLIENTSITE pActiveSite,
                                         LONG lindex,
                                         HWND hwndParent,
                                         LPCRECT lprcPosRect);
     STDMETHODIMP GetExtent  ( DWORD dwDrawAspect, LPSIZEL lpsizel);
     STDMETHODIMP Update  () ;
     STDMETHODIMP Close  ( DWORD dwSaveOption) ;
     STDMETHODIMP Unadvise ( DWORD dwConnection);
     STDMETHODIMP EnumVerbs  ( LPENUMOLEVERB FAR* ppenumOleVerb) ;
     STDMETHODIMP GetClientSite  ( LPOLECLIENTSITE FAR* ppClientSite);
     STDMETHODIMP SetMoniker  ( DWORD dwWhichMoniker, LPMONIKER pmk);
     STDMETHODIMP GetMoniker  ( DWORD dwAssign, DWORD dwWhichMoniker,
                                            LPMONIKER FAR* ppmk);
     STDMETHODIMP InitFromData  ( LPDATAOBJECT pDataObject,
                                                BOOL fCreation,
                                                DWORD dwReserved);
     STDMETHODIMP GetClipboardData  ( DWORD dwReserved,
                                                      LPDATAOBJECT FAR*
ppDataObject);
     STDMETHODIMP IsUpToDate  ();
     STDMETHODIMP GetUserClassID  ( CLSID FAR* pClsid);
     //@@WTK WIN32, UNICODE
     //STDMETHODIMP GetUserType  ( DWORD dwFormOfType, LPSTR FAR*
pszUserType);
     STDMETHODIMP GetUserType  ( DWORD dwFormOfType, LPOLESTR FAR*
pszUserType);
     STDMETHODIMP SetExtent  ( DWORD dwDrawAspect, LPSIZEL lpsizel);
     STDMETHODIMP EnumAdvise  ( LPENUMSTATDATA FAR* ppenumAdvise);
     STDMETHODIMP GetMiscStatus  ( DWORD dwAspect, DWORD FAR* pdwStatus);
     STDMETHODIMP SetColorScheme  ( LPLOGPALETTE lpLogpal);

     void OpenEdit(LPOLECLIENTSITE pActiveSite);

};

#endif
```

IPS.H (SIMPSVR Sample)

```
//**********************************************************************
// File name: ips.h
//
//      Definition of CPersistStorage
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#if !defined( _IPS_H_)
#define _IPS_H_


#include <ole2.h>
//@@WTK WIN32, UNICODE
//#include <storage.h>
#include "obj.h"

class CSimpSvrObj;

interface CPersistStorage : IPersistStorage
{
private:
     CSimpSvrObj FAR * m_lpObj;
     int m_nCount;
     BOOL m_fSameAsLoad;

public:
     CPersistStorage::CPersistStorage(CSimpSvrObj FAR * lpSimpSvrObj)
           {
           m_lpObj = lpSimpSvrObj;
           m_nCount = 0;
           };
     CPersistStorage::~CPersistStorage() {};

     STDMETHODIMP QueryInterface (REFIID riid, LPVOID FAR* ppvObj);
     STDMETHODIMP_(ULONG) AddRef ();
     STDMETHODIMP_(ULONG) Release ();

     STDMETHODIMP InitNew (LPSTORAGE pStg);
     STDMETHODIMP GetClassID  ( LPCLSID lpClassID) ;
     STDMETHODIMP Save  ( LPSTORAGE pStgSave, BOOL fSameAsLoad) ;
     STDMETHODIMP SaveCompleted  ( LPSTORAGE pStgNew);
     STDMETHODIMP Load  ( LPSTORAGE pStg);
     STDMETHODIMP IsDirty  ();
     STDMETHODIMP HandsOffStorage  ();

     void ReleaseStreamsAndStorage();
     void OpenStreams(LPSTORAGE lpStg);
     void CreateStreams(LPSTORAGE lpStg);
     void CreateStreams(LPSTORAGE lpStg, LPSTREAM FAR *lpTempColor, LPSTREAM
FAR *lpTempSize);
};

#endif
```

OBJ.H   (SIMPSVR Sample)

```
//**********************************************************************
// File name: obj.h
//
//      Definition of CSimpSvrObj
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#if !defined( _OBJ_H_)
#define _OBJ_H_

#include "ioipao.h"
#include "ioipo.h"
#include "ioo.h"
#include "ips.h"
#include "ido.h"
#include "iec.h"

class CSimpSvrDoc;
interface COleObject;
interface CPersistStorage;
interface CDataObject;
interface COleInPlaceActiveObject;
interface COleInPlaceObject;
interface CExternalConnection;

class CSimpSvrObj : public IUnknown
{
private:
     CSimpSvrDoc FAR * m_lpDoc;      // Back pointer
     int m_nCount;                   // reference count
     BOOL m_fInPlaceActive;          // Used during InPlace Negotiation
     BOOL m_fInPlaceVisible;         // "  "  "  "   "   "   "   "   "
     BOOL m_fUIActive;               // "  "  "  "   "   "   "   "   "
     HMENU m_hmenuShared;            // "  "  "  "   "   "   "   "   "
     HOLEMENU m_hOleMenu;            // "  "  "  "   "   "   "   "   "
     RECT m_posRect;                 // "  "  "  "   "   "   "   "   "
     OLEINPLACEFRAMEINFO m_FrameInfo;
     BOOL m_fSaveWithSameAsLoad;
     BOOL m_fNoScribbleMode;

     DWORD m_dwRegister;             // Registered in ROT

     int m_red, m_green, m_blue;     // current color
     POINT m_size;                   // current size
     int m_xOffset;
     int m_yOffset;
     float m_scale;

     HWND m_hWndParent;              // parent window handle

     // interfaces used
     LPSTORAGE m_lpStorage;
     LPSTREAM m_lpColorStm, m_lpSizeStm;
     LPOLECLIENTSITE m_lpOleClientSite;           // IOleClientSite
     LPOLEADVISEHOLDER m_lpOleAdviseHolder;       // IOleAdviseHolder
     LPDATAADVISEHOLDER m_lpDataAdviseHolder;     // IDataAdviseHolder
     LPOLEINPLACEFRAME m_lpFrame;                 // IOleInPlaceFrame
     LPOLEINPLACEUIWINDOW m_lpCntrDoc;            // IOleInPlaceUIWindow
     LPOLEINPLACESITE m_lpIPSite;                 // IOleInPlaceSite

     // interface implemented
     COleObject m_OleObject;                              // IOleObject
     CPersistStorage m_PersistStorage;                    // IPersistStorage
     CDataObject m_DataObject;                            // IDataObject
     COleInPlaceActiveObject m_OleInPlaceActiveObject;    //
IOleInPlaceActiveObject
     COleInPlaceObject m_OleInPlaceObject;                //
IOleInPlaceObject
     CExternalConnection m_ExternalConnection;

public:
     STDMETHODIMP QueryInterface (REFIID riid, LPVOID FAR* ppvObj);
     STDMETHODIMP_(ULONG) AddRef ();
     STDMETHODIMP_(ULONG) Release ();

// construction/destruction
     CSimpSvrObj(CSimpSvrDoc FAR * lpSimpSvrDoc);
     ~CSimpSvrObj();

// utility functions
     void Draw(HDC hDC, BOOL fMetaDC = TRUE);
     void PaintObj(HDC hDC);
     void lButtonDown(WPARAM wParam,LPARAM lParam);
     HANDLE GetMetaFilePict();
     void SaveToStorage (LPSTORAGE lpStg, BOOL fSameAsLoad);
     void LoadFromStorage ();

// visual editing helper functions
     BOOL DoInPlaceActivate (LONG lVerb);
     void AssembleMenus();
     void AddFrameLevelUI();
     void DoInPlaceHide();
     void DisassembleMenus();
     void SendOnDataChange();
     void DeactivateUI();

// member variable access
     inline BOOL IsInPlaceActive() { return m_fInPlaceActive; };
     inline BOOL IsInPlaceVisible() { return m_fInPlaceVisible; };
     inline BOOL IsUIActive() { return m_fUIActive; };
     inline HWND GetParent() { return m_hWndParent; };
     inline LPSTORAGE GetStorage() { return m_lpStorage; };
     inline LPOLECLIENTSITE GetOleClientSite() { return
m_lpOleClientSite; };
     inline LPDATAADVISEHOLDER GetDataAdviseHolder() { return
m_lpDataAdviseHolder; };
```

```
     inline LPOLEADVISEHOLDER GetOleAdviseHolder() { return
m_lpOleAdviseHolder; };
     inline LPOLEINPLACEFRAME GetInPlaceFrame() { return m_lpFrame; };
     inline LPOLEINPLACEUIWINDOW GetUIWindow() { return m_lpCntrDoc; };
     inline LPOLEINPLACESITE GetInPlaceSite() { return m_lpIPSite; };
     inline COleObject FAR * GetOleObject() { return &m_OleObject; };
     inline CPersistStorage FAR * GetPersistStorage() { return
&m_PersistStorage; };
     inline CDataObject FAR * GetDataObject() { return &m_DataObject; };
     inline COleInPlaceActiveObject FAR * GetOleInPlaceActiveObject()
{ return &m_OleInPlaceActiveObject; };
     inline COleInPlaceObject FAR * GetOleInPlaceObject() { return
&m_OleInPlaceObject; };
     inline void ClearOleClientSite() { m_lpOleClientSite = NULL; };
     inline void ClearDataAdviseHolder() { m_lpDataAdviseHolder = NULL; };
     inline void ClearOleAdviseHolder() { m_lpOleAdviseHolder = NULL; };
     inline LPRECT GetPosRect() { return &m_posRect; };
     inline LPPOINT GetSize() { return &m_size; };
     inline LPOLEINPLACEFRAMEINFO GetFrameInfo() {return &m_FrameInfo;};
     inline DWORD GetRotRegister() { return m_dwRegister; };



     // member manipulation
     inline void SetColor (int nRed, int nGreen, int nBlue)
          { m_red = nRed; m_green = nGreen; m_blue = nBlue; };

     inline void RotateColor()
          { m_red+=10; m_green+=10; m_blue+=10;};


// all of the interface implementations should be friends of this
// class
friend interface COleObject;
friend interface CPersistStorage;
friend interface CDataObject;
friend interface COleInPlaceActiveObject;
friend interface COleInPlaceObject;
friend interface CExternalConnection;

};
#endif
```

PRE.H (SIMPSVR Sample)

```
//**********************************************************************
// File name: pre.h
//
//      Used for precompiled headers
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#if !defined( _PRE_H_)
#define _PRE_H_

#include <windows.h>
#include <ole2.h>
#include <ole2ui.h>
#include <assert.h>
#include <string.h>
#include "simpsvr.h"
#include "resource.h"
#include <ole2ver.h>


#endif
```

RESOURCE.H (SIMPSVR Sample)

```
//{{NO_DEPENDENCIES}}
// App Studio generated include file.
// Used by SIMPSVR.RC
//
#define IDM_OPEN                        102
#define IDM_SAVE                        103
#define IDM_SAVEAS                      104
#define IDM_PRINT                       105
#define IDM_EXIT                        106
#define IDM_UNDO                        107
#define IDM_CUT                         108
#define IDM_COPY                        109
#define IDM_PASTE                       110
#define ID_EDIT_INSERTOBJECT            111
#define IDM_INSERTOBJECT                111
#define IDM_NEW                         112
#define IDM_RED                         113
#define IDM_GREEN                       114
#define IDM_BLUE                        115
#define IDM_ROTATE                      116

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS

#define _APS_NEXT_RESOURCE_VALUE        102
#define _APS_NEXT_COMMAND_VALUE         117
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

SIMPSVR.H (SIMPSVR Sample)

```
//**********************************************************************
// File name: simpsvr.h
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************
#define IDM_ABOUT 100
#define IDM_INSERT  101
#define IDM_VERB0 1000


int PASCAL WinMain(HANDLE hInstance,HANDLE hPrevInstance,LPSTR lpCmdLine,int
nCmdShow);
BOOL InitApplication(HANDLE hInstance);
BOOL InitInstance(HANDLE hInstance, int nCmdShow);
//@@WTK WIN32, UNICODE _export EXPORT
long FAR PASCAL EXPORT MainWndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam);
long FAR PASCAL EXPORT DocWndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam);
//@@WTK WIN32, UNICODE
//BOOL FAR PASCAL export About(HWND hDlg, unsigned message, WORD wParam,
LONG lParam);
BOOL FAR PASCAL EXPORT About(HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam);
```

APP.CPP (SIMPSVR Sample)

```
//**********************************************************************
// File name: app.cpp
//
//    Implementation file for the CSimpSvrApp Class
//
// Functions:
//
//    See app.h for a list of member functions.
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "obj.h"
#include "app.h"
#include "doc.h"
#include "icf.h"

#include "initguid.h"
DEFINE_GUID(GUID_SIMPLE, 0xBCF6D4A0, 0xBE8C, 0x1068, 0xB6, 0xD4, 0x00, 0xDD,
0x01, 0x0C, 0x05, 0x09);
```

CSimpSvrApp::CSimpSvrApp (SIMPSVR Sample)

```
//***********************************************************************
//
// CSimpSvrApp::CSimpSvrApp()
//
// Purpose:
//
//      Constructor for CSimpSvrApp
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      SetRectEmpty                Windows API
//
// Comments:
//
//
//*********************************************************************

CSimpSvrApp::CSimpSvrApp()
{
        OutputDebugString("In CSimpSvrApp's Constructor \r\n");

        // clear members
        m_hAppWnd = NULL;
        m_hInst = NULL;
        m_lpDoc = NULL;

        // clear flags
        m_fInitialized = FALSE;

        // used for inplace
        SetRectEmpty(&nullRect);
}
```

CSimpSvrApp::~CSimpSvrApp   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrApp::~CSimpSvrApp()
//
// Purpose:
//
//      Destructor for CSimpSvrApp Class.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      DestroyWindow               Windows API
//      CSimpSvrApp::IsInitialized  APP.H
//      OleUninitialize             OLE API
//
// Comments:
//
//********************************************************************

CSimpSvrApp::~CSimpSvrApp()
{
        OutputDebugString("In CSimpSvrApp's Destructor\r\n");

        // need to uninit the library...
        if (IsInitialized())
                OleUninitialize();

        DestroyWindow(m_hAppWnd);
}
```

CSimpSvrApp::QueryInterface   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrApp::QueryInterface
//
// Purpose:
//
//      Used for interface negotiation at the Application level.
//
// Parameters:
//
//      REFIID riid         -   A reference to the interface that is
//                              being queried.
//
//      LPVOID FAR* ppvObj  -   An out parameter to return a pointer to
//                              the interface.
//
// Return Value:
//
//      S_OK          -   The interface is supported.
//      E_NOINTERFACE -   The interface is not supported
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      ResultFromScode             OLE API
//      IUnknown::AddRef            APP.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP CSimpSvrApp::QueryInterface(REFIID riid, LPVOID FAR* ppvObj)
{
        OutputDebugString("In CSimpSvrApp::QueryInterface\r\n");

        SCODE sc = S_OK;

        if (riid == IID_IUnknown)
                *ppvObj = this;
        else
                {
                *ppvObj = NULL;
                sc = E_NOINTERFACE;
                }

        if (*ppvObj)
                ((LPUNKNOWN)*ppvObj)->AddRef();

        // asking for something we don't understand at this level.
        return ResultFromScode(sc);
}
```

CSimpSvrApp::AddRef   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrApp::AddRef
//
// Purpose:
//
//      Adds to the reference count at the Application level.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the application.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Due to the reference counting model that is used in this
//      implementation, this reference count is the sum of the
//      reference counts on all interfaces of all objects open
//      in the application.
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpSvrApp::AddRef()
{
        OutputDebugString("In CSimpSvrApp::AddRef\r\n");
        return SafeAddRef();
}
```

CSimpSvrApp::Release   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrApp::Release
//
// Purpose:
//
//      Decrements the reference count at this level
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the application.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Due to the reference counting model that is used in this
//      implementation, this reference count is the sum of the
//      reference counts on all interfaces of all objects open
//      in the application.
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpSvrApp::Release()
{
        OutputDebugString("In CSimpSvrApp::Release\r\n");

        return SafeRelease();
}
```

CSimpSvrApp::fInitApplication (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrApp::fInitApplication
//
// Purpose:
//
//      Initializes the application
//
// Parameters:
//
//      HANDLE hInstance    -   Instance handle of the application.
//
// Return Value:
//
//      TRUE    -   Application was successfully initialized.
//      FALSE   -   Application could not be initialized
//
// Function Calls:
//      Function                    Location
//
//      LoadIcon                    Windows API
//      LoadCursor                  Windows API
//      GetStockObject              Windows API
//      RegisterClass               Windows API
//      RegisterHatchWindowClass    OUTLUI.DLL
//
// Comments:
//
//********************************************************************

BOOL CSimpSvrApp::fInitApplication(HANDLE hInstance)
{
        CStabilize stabilize(this);
        WNDCLASS  wc;

        // Fill in window class structure with parameters that describe the
        // main window.

        wc.style = NULL;                    // Class style(s).
        wc.lpfnWndProc = MainWndProc;       // Function to retrieve messages
for

// windows of this class.
        wc.cbClsExtra = 0;                  // No per-class extra data.
        wc.cbWndExtra = 0;                  // No per-window extra data.
        wc.hInstance = hInstance;           // Application that owns the
class.
        wc.hIcon = LoadIcon(hInstance, "SimpSvr");
        wc.hCursor = LoadCursor(NULL, IDC_ARROW);
        wc.hbrBackground = GetStockObject(WHITE_BRUSH);
        wc.lpszMenuName =  "SimpSvrMENU";    // Name of menu resource in .RC
file.
```

```
        wc.lpszClassName = "SimpSvrWClass";  // Name used in call to
CreateWindow.

        if (!RegisterClass(&wc))
                return FALSE;

        wc.style = CS_VREDRAW | CS_HREDRAW;                     // Class
style(s).
        wc.lpfnWndProc = DocWndProc;        // Function to retrieve messages
for

// windows of this class.
        wc.cbClsExtra = 0;                  // No per-class extra data.
        wc.cbWndExtra = 0;                  // No per-window extra data.
        wc.hInstance = hInstance;           // Application that owns the
class.
        wc.hIcon = NULL;
        wc.hCursor = LoadCursor(NULL, IDC_ARROW);
        wc.hbrBackground = GetStockObject(WHITE_BRUSH);
        wc.lpszMenuName =  NULL;
        wc.lpszClassName = "DocWClass";     // Name used in call to
CreateWindow.

        // Register the window class and return success/failure code.

        if (!RegisterClass(&wc))
                return FALSE;

        return (RegisterHatchWindowClass(hInstance));
}
```

CSimpSvrApp::fInitInstance (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrApp::fInitInstance
//
// Purpose:
//
//      Instance initialization.
//
// Parameters:
//
//      HANDLE hInstance    -   App. Instance Handle.
//
//      int nCmdShow        -   Show parameter from WinMain
//
// Return Value:
//
//      TRUE    -   Initialization Successful
//      FALSE   -   Initialization Failed.
//
//
// Function Calls:
//      Function                    Location
//
//      CreateWindow                Windows API
//      InvalidateRect              Windows API
//      ShowWindow                  Windows API
//      UpdateWindow                Windows API
//      CoRegisterClassObject       OLE API
//      OleBuildVersion             OLE API
//      OleInitialize               OLE API
//      CSimpSvrDoc::CreateObject   DOC.CPP
//
// Comments:
//
//      Note that successful Initalization of the OLE libraries
//      is remembered so the UnInit is only called if needed.
//
//********************************************************************

BOOL CSimpSvrApp::fInitInstance (HANDLE hInstance, int nCmdShow,
CClassFactory FAR * lpClassFactory)
{
        CStabilize stabilize(this);
        m_hInst = hInstance;

                DWORD dwVer = OleBuildVersion();

        // check to see if we are compatible with this version of the
libraries
        if (HIWORD(dwVer) != rmm || LOWORD(dwVer) < rup)
                OutputDebugString("*** WARNING:  Not compatible with current
libs ***\r\n");
```

```
        // initialize the libraries
        if (OleInitialize(NULL) == NOERROR)
                m_fInitialized = TRUE;


        // Create the "application" windows
        m_hAppWnd = CreateWindow ("SimpSvrWClass",
                                                          "Simple OLE 2.0
Server",

WS_OVERLAPPEDWINDOW,
                                                          CW_USEDEFAULT,
                                                          CW_USEDEFAULT,
                                                          CW_USEDEFAULT,
                                                          CW_USEDEFAULT,
                                                          NULL,
                                                          NULL,
                                                          hInstance,
                                                          NULL);

        if (!m_hAppWnd)
                return FALSE;

        // if not started by OLE, then show the Window, and create a "fake"
object, else
        // Register a pointer to IClassFactory so that OLE can instruct us
to make an
        // object at the appropriate time.
        if (!m_fStartByOle)
                {
                ShowAppWnd(nCmdShow);
                m_lpDoc->CreateObject(IID_IOleObject, (LPVOID FAR
*)&m_OleObject);
                InvalidateRect(m_lpDoc->GethDocWnd(), NULL, TRUE);
                }
        else
                {
                lpClassFactory = new CClassFactory(this);

                // shouldn't pass an API an object with a zero ref count
                lpClassFactory->AddRef();

                CoRegisterClassObject(GUID_SIMPLE,(IUnknown FAR
*)lpClassFactory, CLSCTX_LOCAL_SERVER, REGCLS_SINGLEUSE,
&m_dwRegisterClass);

                // remove artificial Ref. count
                lpClassFactory->Release();
                }

        m_hMainMenu = GetMenu(m_hAppWnd);
        m_hColorMenu = GetSubMenu(m_hMainMenu, 1);
        m_hHelpMenu = GetSubMenu(m_hMainMenu, 2);
        return m_fInitialized;
}
```

CSimpSvrApp::lCommandHandler (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrApp::lCommandHandler
//
// Purpose:
//
//      Handles the processing of WM_COMMAND.
//
// Parameters:
//
//      HWND hWnd       -   Handle to the application Window
//
//      UINT message    -   message (always WM_COMMAND)
//
//      WPARAM wParam   -   Same as passed to the WndProc
//
//      LPARAM lParam   -   Same as passed to the WndProc
//
// Return Value:
//
//      NULL
//
// Function Calls:
//      Function                                    Location
//
//      GetClientRect                               Windows API
//      MessageBox                                  Windows API
//      DialogBox                                   Windows API
//      MakeProcInstance                            Windows API
//      FreeProcInstance                            Windows API
//      SendMessage                                 Windows API
//      DefWindowProc                               Windows API
//      InvalidateRect                              Windows API
//      CSimpSvrDoc::InsertObject                   DOC.CPP
//      CSimpSvrObj::SetColor                       OBJ.CPP
//      CSimpSvrObj::RotateColor                    OBJ.CPP
//
// Comments:
//
//********************************************************************

long CSimpSvrApp::lCommandHandler (HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
        CStabilize stabilize(this);
        //@@WTK WIN32, UNICODE
        //switch (wParam) {
        switch (LOWORD(wParam)) {
                // bring up the About box
                case IDM_ABOUT:
                        {
```

```
                        FARPROC lpProcAbout =
MakeProcInstance((FARPROC)About, m_hInst);

                        DialogBox(m_hInst,               // current instance
                                        "AboutBox",                   //
resource to use
                                        m_hAppWnd,                    //
parent handle
                                        lpProcAbout);                 //
About() instance address

                        FreeProcInstance(lpProcAbout);
                        break;
                        }

                // exit the application
                case IDM_EXIT:
                        SendMessage(hWnd, WM_SYSCOMMAND, SC_CLOSE, 0L);
                        break;

                case IDM_RED:
                        m_lpDoc->GetObj()->SetColor (128, 0, 0);
                        InvalidateRect(m_lpDoc->GethDocWnd(), NULL, TRUE);
                        break;

                case IDM_GREEN:
                        m_lpDoc->GetObj()->SetColor (0,128, 0);
                        InvalidateRect(m_lpDoc->GethDocWnd(), NULL, TRUE);
                        break;

                case IDM_BLUE:
                        m_lpDoc->GetObj()->SetColor (0, 0, 128);
                        InvalidateRect(m_lpDoc->GethDocWnd(), NULL, TRUE);
                        break;

                case IDM_ROTATE:
                        m_lpDoc->GetObj()->RotateColor();
                        InvalidateRect(m_lpDoc->GethDocWnd(), NULL, TRUE);
                        break;

                default:
                        return (DefWindowProc(hWnd, message, wParam,
lParam));
                   }   // end of switch
        return NULL;
}
```

CSimpSvrApp::lSizeHandler   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrApp::lSizeHandler
//
// Purpose:
//
//      Handles the WM_SIZE message
//
// Parameters:
//
//      HWND hWnd       -   Handle to the application Window
//
//      UINT message    -   message (always WM_SIZE)
//
//      WPARAM wParam   -   Same as passed to the WndProc
//
//      LPARAM lParam   -   Same as passed to the WndProc
//
// Return Value:
//
//      LONG    -   returned from the "document" resizing
//
// Function Calls:
//      Function                    Location
//
//      GetClientRect               Windows API
//      CSimpSvrDoc::lResizeDoc      DOC.CPP
//
// Comments:
//
//********************************************************************

long CSimpSvrApp::lSizeHandler (HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
        RECT rect;
        CStabilize stabilize(this);

        GetClientRect(m_hAppWnd, &rect);
        return m_lpDoc->lResizeDoc(&rect);
}
```

CSimpSvrApp::lCreateDoc (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrApp::lCreateDoc
//                                                         d
// Purpose:
//
//      Handles the creation of a document.
//
// Parameters:
//
//      HWND hWnd       -   Handle to the application Window
//
//      UINT message    -   message (always WM_CREATE)
//
//      WPARAM wParam   -   Same as passed to the WndProc
//
//      LPARAM lParam   -   Same as passed to the WndProc
//
// Return Value:
//
//      NULL
//
// Function Calls:
//      Function                    Location
//
//      GetClientRect               Windows API
//      CSimpSvrDoc::Create         DOC.CPP
//
// Comments:
//
//********************************************************************

long CSimpSvrApp::lCreateDoc (HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
        RECT rect;
        CStabilize stabilize(this);

        GetClientRect(hWnd, &rect);

        m_lpDoc = CSimpSvrDoc::Create(this, &rect, hWnd);

        return NULL;
}
```

CSimpSvrApp::PaintApp   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrApp::PaintApp
//
// Purpose:
//
//      Handles the painting of the doc window.
//
//
// Parameters:
//
//      HDC hDC -   hDC to the Doc Window.
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      CSimpSvrDoc::PaintDoc        DOC.CPP
//
// Comments:
//
//
//********************************************************************

void CSimpSvrApp::PaintApp (HDC hDC)
{
        CStabilize stabilize(this);

        // if we supported multiple documents, we would enumerate
        // through each of the open documents and call paint.

        if (m_lpDoc)
                m_lpDoc->PaintDoc(hDC);

}
```

CSimpSvrApp::ParseCmdLine   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrApp::ParseCmdLine
//
// Purpose:
//
//      Determines if the app was started by OLE
//
//
// Parameters:
//
//      LPSTR lpCmdLine -   Pointer to the command line
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      lstrlen                     Windows API
//      lstrcmp                     Windows API
//
//
// Comments:
//
//      Parses the command line looking for the -Embedding or /Embedding
//      flag.
//
//********************************************************************

void CSimpSvrApp::ParseCmdLine(LPSTR lpCmdLine)
{
        char szTemp[255];

        m_fStartByOle = TRUE;

        ::ParseCmdLine (lpCmdLine, &m_fStartByOle, szTemp);
}
```

CSimpSvrApp::SetStatusText (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrApp::SetStatusText
//
// Purpose:
//
//      Blanks out the text in the status bar
//
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      CSimpSvrDoc::SetStatusText  DOC.CPP
//
//
// Comments:
//
//
//********************************************************************

void CSimpSvrApp::SetStatusText()
{
        CStabilize stabilize(this);
        m_lpDoc->SetStatusText();
}
```

CSimpSvrApp::IsInPlaceActive (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrApp::IsInPlaceActive
//
// Purpose:
//
//      Safely determines from the app level if currently inplace active.
//
//
// Parameters:
//
//      None
//
// Return Value:
//
//      TRUE    - Inplace active
//      FALSE   - Not Inplace active
//
// Function Calls:
//      Function                    Location
//
//      CSimpSvrDoc::GetObject      OBJ.H
//      CSimpSvrObj:IsInPlaceActive OBJ.H
//
//
// Comments:
//
//
//********************************************************************

BOOL CSimpSvrApp::IsInPlaceActive()
{
        CStabilize stabilize(this);
        BOOL retval = FALSE;

        if (m_lpDoc)
                if (m_lpDoc->GetObj())
                        retval = m_lpDoc->GetObj()->IsInPlaceActive();

        return retval;
}
```

CSimpSvrApp::ShowAppWnd   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrApp::ShowAppWnd
//
// Purpose:
//
//      Shows the Application Window
//
// Parameters:
//
//      int nCmdShow    - Window State
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                        Location
//
//      ShowWindow                      Windows API
//      UpdateWindow                    Windows API
//      CoLockObjectExternal            OLE API
//
// Comments:
//
//********************************************************************

void CSimpSvrApp::ShowAppWnd(int nCmdShow)
{
        CoLockObjectExternal(this, TRUE, FALSE);
        ShowWindow (m_hAppWnd, nCmdShow);
        UpdateWindow (m_hAppWnd);
}
```

CSimpSvrApp::ShowAppWnd (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrApp::ShowAppWnd
//
// Purpose:
//
//      Hides the Application Window
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                        Location
//
//      ShowWindow                      Windows API
//      CoLockObjectExternal            OLE API
//
// Comments:
//
//********************************************************************

void CSimpSvrApp::HideAppWnd()
{
        CoLockObjectExternal(this, FALSE, TRUE);
        ShowWindow (m_hAppWnd, SW_HIDE);
}
```

DOC.CPP (SIMPSVR Sample)

```
//**********************************************************************
// File name: DOC.CPP
//
//      Implementation file for CSimpSvrDoc.
//
// Functions:
//
//      See DOC.H for Class Definition
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "obj.h"
#include "app.h"
#include "doc.h"
```

CSimpSvrDoc::Create   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrDoc::Create
//
// Purpose:
//
//      Creation for the CSimpSvrDoc Class
//
// Parameters:
//
//      CSimpSvrApp FAR * lpApp  -   Pointer to the CSimpSvrApp Class
//
//      LPRECT lpRect           -   Client area rect of "frame" window
//
//      HWND hWnd               -   Window Handle of "frame" window
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      StgCreateDocfile            OLE API
//      CreateWindow                Windows API
//      ShowWindow                  Windows API
//      UpdateWindow                Windows API
//      CSimpSvrDoc::CSimpSvrDoc    DOC.CPP
//      CreateHatchWindow           OLE2UI
//
// Comments:
//
//      This routine was added so that failure could be returned
//      from object creation.
//
//********************************************************************

CSimpSvrDoc FAR * CSimpSvrDoc::Create(CSimpSvrApp FAR *lpApp, LPRECT
lpRect,HWND hWnd)
{
        CSimpSvrDoc FAR * lpTemp = new CSimpSvrDoc(lpApp, hWnd);

        if (!lpTemp)
                return NULL;

        // create the document Window
        lpTemp->m_hDocWnd = CreateWindow(
                        "DocWClass",
                        NULL,
                        WS_CHILD | WS_CLIPSIBLINGS,
                        lpRect->left,
                        lpRect->top,
                        lpRect->right,
                        lpRect->bottom,
                        hWnd,
                        NULL,
                        lpApp->GethInst(),
                        NULL);

        if (!lpTemp->m_hDocWnd)
                goto error;

        lpTemp->ShowDocWnd();

        lpTemp->m_hHatchWnd = CreateHatchWindow( lpTemp->m_hDocWnd, lpApp-
>GethInst());

        lpTemp->HideHatchWnd();

        return (lpTemp);

error:
        delete (lpTemp);
        return NULL;

}
```

CSimpSvrDoc::CSimpSvrDoc (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrDoc::CSimpSvrDoc
//
// Purpose:
//
//      Constructor for the CSimpSvrDoc Class
//
// Parameters:
//
//      CSimpSvrApp FAR * lpApp  -   Pointer to the CSimpSvrApp Class
//
//      HWND hWnd               -   Window Handle of "frame" window
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      GetMenu                     Windows API
//      GetSubMenu                  Windows API
//
// Comments:
//
//********************************************************************

CSimpSvrDoc::CSimpSvrDoc(CSimpSvrApp FAR * lpApp,HWND hWnd)
{
        OutputDebugString("In CSimpSvrDoc's Constructor\r\n");
        m_lpApp = lpApp;
        m_lpObj = NULL;
        m_fClosing = FALSE;

        // set up menu handles


}
```

CSimpSvrDoc::~CSimpSvrDoc   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrDoc::~CSimpSvrDoc
//
// Purpose:
//
//      Destructor for CSimpSvrDoc
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      DestroyWindow               Windows API
//      CSimpSvrApp::ClearDoc       APP.CPP
//
// Comments:
//
//********************************************************************

CSimpSvrDoc::~CSimpSvrDoc()
{
        OutputDebugString("In CSimpSvrDoc's Destructor\r\n");
        m_lpApp->ClearDoc();
        DestroyWindow(m_hHatchWnd);
}
```

CSimpSvrDoc::QueryInterface (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrDoc::QueryInterface
//
// Purpose:
//
//      Return a pointer to a requested interface
//
// Parameters:
//
//      REFIID riid         -   ID of interface to be returned
//      LPVOID FAR* ppvObj  -   Location to return the interface
//
// Return Value:
//
//      E_NOINTERFACE -   Always
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpSvrApp::QueryInterface APP.CPP
//
// Comments:
//
//      Since the document could contain multiple objects, all
//      interfaces except those associated with the document should
//      be returned.  In this implementation, there are no doc level
//      interfaces.
//
//********************************************************************

STDMETHODIMP CSimpSvrDoc::QueryInterface(REFIID riid, LPVOID FAR* ppvObj)
{
        OutputDebugString("In CSimpSvrDoc::QueryInterface\r\n");

        SCODE sc = E_NOINTERFACE;

        if ( (riid == IID_IUnknown) )
           {
           AddRef();
           *ppvObj = this;
           sc = S_OK;
           }

        return ResultFromScode(sc);
}
```

CSimpSvrDoc::AddRef (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrDoc::AddRef
//
// Purpose:
//
//      Increments the document level reference count
//
// Parameters:
//
//      None
//
// Return Value:
//
//      UINT    -   The current reference count on the document
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpSvrApp::AddRef         APP.CPP
//
// Comments:
//
//      The reference count at this level reflects the total ref.
//      count of all interfaces on all objects contained within
//      this document.  Note that it also "trickles up" the
//      ref count to the app level.
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpSvrDoc::AddRef()
{
        OutputDebugString("In CSimpSvrDoc::AddRef\r\n");
        // AddRef the app, but return the doc count
        m_lpApp->AddRef();

        return SafeAddRef();
}
```

CSimpSvrDoc::Release   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrDoc::Release
//
// Purpose:
//
//      Decrements the document level reference count
//
// Parameters:
//
//      None
//
// Return Value:
//
//      UINT    -   The current reference count on the document
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpSvrApp::Release         APP.CPP
//
// Comments:
//
//      The reference count at this level reflects the total ref.
//      count of all interfaces on all objects contained within
//      this document.  Note that it also "trickles up" the
//      ref count to the app level.
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpSvrDoc::Release()
{
        OutputDebugString("In CSimpSvrDoc::Release\r\n");
        // Release the app, but return the app count
        m_lpApp->Release();

        return SafeRelease();
}
```

CSimpSvrDoc::lResizeDoc   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrDoc::lResizeDoc
//
// Purpose:
//
//      Resizes the document
//
// Parameters:
//
//      LPRECT lpRect   -   The size of the client are of the "frame"
//                          Window.
//
// Return Value:
//
//      NULL
//
// Function Calls:
//      Function                                Location
//
//      MoveWindow                              Windows API
//
// Comments:
//
//********************************************************************

long CSimpSvrDoc::lResizeDoc(LPRECT lpRect)
{
        MoveWindow(m_hDocWnd, lpRect->left, lpRect->top, lpRect->right,
lpRect->bottom, TRUE);
        return NULL;
}
```

CSimpSvrDoc::PaintDoc   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrDoc::PaintDoc
//
// Purpose:
//
//      Paints the Document
//
// Parameters:
//
//      HDC hDC -   hDC of the document Window
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      CSimpSvrObj::Draw                   OBJ.CPP
//      CSimpSvrObj::GetDataAdviseHolder    OBJ.H
//      CSimpSvrObj::GetDataObject          OBJ.H
//      CSimpAppObj::IsStartedByOle         APP.CPP
//      IDataAdviseHolder::SendOnDataChange OLE API
//
// Comments:
//
//********************************************************************

void CSimpSvrDoc::PaintDoc (HDC hDC)
{
        // i_f the object hasn't been created yet, then don't draw
        if (m_lpObj)
                m_lpObj->Draw(hDC,FALSE);
        else
                return;

        CStabilize stabilize(this);

        // Sending a data change every time we paint, but only if we
        // were started by OLE
        if (m_lpApp->IsStartedByOle())
                        m_lpObj->SendOnDataChange( );
}
```

CSimpSvrDoc::CreateObject   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrDoc::CreateObject
//
// Purpose:
//
//
// Parameters:
//
//
// Return Value:
//
//      NOERROR if the function succeeds, otherwise E_FAIL
//
// Function Calls:
//      Function                    Location
//
//      CSimpSvrObj::CSimpSvrObj    OBJ.CPP
//      CSimpSvrOjb::QueryInterface OBJ.CPP
//
// Comments:
//
//********************************************************************

HRESULT CSimpSvrDoc::CreateObject(REFIID riid, LPVOID FAR *ppvObject)
{
        SCODE sc = E_FAIL;

        m_lpObj = new CSimpSvrObj(this);

        if (m_lpObj)
                {
                m_lpObj->QueryInterface(riid, ppvObject);
                sc = S_OK;
                }

        return ResultFromScode(sc);
}
```

CSimpSvrDoc::Close   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrDoc::Close
//
// Purpose:
//
//      Closes the object
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                                Location
//
//      OutputDebugString                       Windows API
//      CSimpSvrObj::AddRef                     OBJ.CPP
//      CSimpSvrObj::Release                    OBJ.CPP
//      CSimpSvrObj::IsInPlaceActive            OBJ.H
//      CSimpSvrObj::GetOleInPlaceObject        OBJ.H
//      CSimpSvrObj::ClearOleClientSite         OBJ.H
//      CSimpSvrObj::GetDataAdviseHolder        OBJ.H
//      CSimpSvrObj::GetOleClientSite           OBJ.H
//      CSimpSvrObj::ClearDataAdviseHolder      OBJ.H
//      CSimpSvrObj::GetOleAdviseHolder         OBJ.H
//      CSimpSvrObj::ClearOleAdviseHolder       OBJ.H
//      IOleInPlaceObject::InPlaceDeactivate    Container
//      IOleClientSite::SaveObject              Container
//      IOleClientSite::OnShowWindow            Container
//      IOleClientSite::Release                 Container
//      IDataAdviseHolder::SendOnDataChange     OLE
//      IDataAdviseHolder::Release              OLE
//      IOleAdviseHolder::SendOnClose           OLE
//      IOleAdviseHolder::Release               OLE
//
// Comments:
//
//********************************************************************

void CSimpSvrDoc::Close()
{
        OutputDebugString("In CSimpSvrDoc::Close() \r\n");
        CStabilize stabilize(this);

        m_lpObj->AddRef(); // hold object alive

        if (m_fClosing)
                return;
        m_fClosing = TRUE;

        // if the object is currently inplace active, then deactivate
        if (m_lpObj->IsInPlaceActive())
                m_lpObj->GetOleInPlaceObject()->InPlaceDeactivate();

        // unregister from the ROT...
        if (m_lpObj->GetRotRegister())
                {
                LPRUNNINGOBJECTTABLE lpRot;

                if (GetRunningObjectTable (0, &lpRot) == NOERROR )
                        {
                        lpRot->Revoke(m_lpObj->GetRotRegister());
                        lpRot->Release();
                        }
                }

        // if we have a clientsite, instruct it to save the object
        if (m_lpObj->GetOleClientSite())
                {
                m_lpObj->GetOleClientSite()->SaveObject();
                m_lpObj->GetOleClientSite()->OnShowWindow(FALSE);
                }

        // Do a final SendOnDataChange for those containers that have
specified the
        // ADF_DATAONSTOP flag.
        if (m_lpObj->GetDataAdviseHolder())
                {
                m_lpObj->GetDataAdviseHolder()->SendOnDataChange( m_lpObj-
>GetDataObject(), 0, ADVF_DATAONSTOP);
                m_lpObj->GetDataAdviseHolder()->Release();
                m_lpObj->ClearDataAdviseHolder();
                }


        // Tell the container that we are shutting down.
        if (m_lpObj->GetOleAdviseHolder())
                {
                m_lpObj->GetOleAdviseHolder()->SendOnClose();
                m_lpObj->GetOleAdviseHolder()->Release();
                m_lpObj->ClearOleAdviseHolder();
                }

        if (m_lpObj->GetOleClientSite())
                {
                m_lpObj->GetOleClientSite()->Release();
                m_lpObj->ClearOleClientSite();
                }

        // release our streams and storage
        m_lpObj->GetPersistStorage()->ReleaseStreamsAndStorage();
        // Disconnect the object.  NOTE: This call should not do anything
        // unless the container has cause a GP Fault or some other problem
        // has occured...
        OutputDebugString("*** Before CoDisconnectObject *** \r\n");
        CoDisconnectObject((LPUNKNOWN)m_lpObj, 0);
        OutputDebugString("*** After CoDisconnectObject *** \r\n");

        m_lpObj->Release(); // let object close

}
```

CSimpSvrDoc::SetStatusText (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrDoc::SetStatusText
//
// Purpose:
//
//      Sets the Container's status bar text
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                        Location
//
//      CSimpSvrObj::IsInPlaceActive    OBJ.CPP
//      IOleInPlaceFrame::SetStatusText Container
//
// Comments:
//
//      Even though there is no status line in this sample, this
//      method must be called on WM_MENUSELECT to clear the last
//      message in the status line.
//
//********************************************************************

void CSimpSvrDoc::SetStatusText()
{
        CStabilize stabilize(this);
        if (m_lpObj->IsInPlaceActive())
                //@@WTK WIN32, UNICODE
                //m_lpObj->GetInPlaceFrame()->SetStatusText("\0");
                m_lpObj->GetInPlaceFrame()->SetStatusText(OLESTR("\0"));
}
```

CSimpSvrDoc::ShowDocWnd   (SIMPSVR Sample)

```
//***********************************************************************
//
// CSimpSvrDoc::ShowDocWnd
//
// Purpose:
//
//      Shows the Document Window
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                        Location
//
//      ShowWindow                      Windows API
//      UpdateWindow                    Windows API
//
// Comments:
//
//*********************************************************************

void CSimpSvrDoc::ShowDocWnd()
{
        ShowWindow(m_hDocWnd, SW_SHOWNORMAL);  // Show the window
        UpdateWindow(m_hDocWnd);               // Sends WM_PAINT message
}
```

CSimpSvrDoc::ShowHatchWnd   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrDoc::ShowHatchWnd
//
// Purpose:
//
//      Shows the hatch Window
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                        Location
//
//      ShowWindow                      Windows API
//
// Comments:
//
//********************************************************************

void CSimpSvrDoc::ShowHatchWnd()
{
        ShowWindow(m_hHatchWnd, SW_SHOW);
}
```

CSimpSvrDoc::HideDocWnd (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrDoc::HideDocWnd
//
// Purpose:
//
//      Hides the DocumentWindow
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                        Location
//
//      ShowWindow                      Windows API
//
// Comments:
//
//********************************************************************

void CSimpSvrDoc::HideDocWnd()
{
        ShowWindow(m_hDocWnd, SW_HIDE);
}
```

CSimpSvrDoc::HideHatchWnd (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrDoc::HideHatchWnd
//
// Purpose:
//
//      Hides the Hatch Window
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                        Location
//
//      ShowWindow                      Windows API
//
// Comments:
//
//********************************************************************

void CSimpSvrDoc::HideHatchWnd()
{
        ShowWindow(m_hHatchWnd, SW_HIDE);
}
```

ICF.CPP (SIMPSVR Sample)

```
//**********************************************************************
// File name: ICF.CPP
//
//    Implementation file for the CClassFactory Class
//
// Functions:
//
//    See icf.h for a list of member functions.
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "app.h"
#include "doc.h"
#include "icf.h"
```

CClassFactory::QueryInterface (SIMPSVR Sample)

```
//**********************************************************************
//
// CClassFactory::QueryInterface
//
// Purpose:
//
//
// Parameters:
//
//      REFIID riid         -   Interface being queried for.
//
//      LPVOID FAR *ppvObj  -   Out pointer for the interface.
//
// Return Value:
//
//      S_OK            - Success
//      E_NOINTERFACE   - Failure
//
// Function Calls:
//      Function                    Location
//
//      CSimpSvrApp::QueryInterface APP.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP CClassFactory::QueryInterface  ( REFIID riid, LPVOID FAR*
ppvObj)
{
     OutputDebugString("In CClassFactory::QueryInterface\r\n");

     SCODE sc = S_OK;

     if ( (riid == IID_IUnknown) ||
            (riid == IID_IClassFactory) )
           *ppvObj = this;
     else
           {
           *ppvObj = NULL;
           sc = E_NOINTERFACE;
           }

     if (*ppvObj)
           ((LPUNKNOWN)*ppvObj)->AddRef();

     // pass it on to the Application object
     return ResultFromScode(sc);
};
```

CClassFactory::AddRef (SIMPSVR Sample)

```
//***********************************************************************
//
// CClassFactory::AddRef
//
// Purpose:
//
//      Increments the reference count on CClassFactory and the application
//      object.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The Reference count on CClassFactory
//
// Function Calls:
//      Function                    Location
//
//      OuputDebugString            Windows API
//
// Comments:
//
//
//*********************************************************************


STDMETHODIMP_(ULONG) CClassFactory::AddRef ()
{
     OutputDebugString("In CClassFactory::AddRef\r\n");

     return ++m_nCount;
};
```

CClassFactory::Release   (SIMPSVR Sample)

```
//**********************************************************************
//
// CClassFactory::Release
//
// Purpose:
//
//      Decrements the reference count of CClassFactory and the
//      application object.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The new reference count
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//
//********************************************************************


STDMETHODIMP_(ULONG) CClassFactory::Release ()
{
     OutputDebugString("In CClassFactory::Release\r\n");

     if (--m_nCount == 0) {
           delete this;
        return 0;
    }

     return m_nCount;
};
```

CClassFactory::CreateInstance (SIMPSVR Sample)

```
//**********************************************************************
//
// CClassFactory::CreateInstance
//
// Purpose:
//
//      Instantiates a new OLE object
//
// Parameters:
//
//      LPUNKNOWN pUnkOuter     - Pointer to the controlling unknown
//
//      REFIID riid             - The interface type to fill in ppvObject
//
//      LPVOID FAR* ppvObject   - Out pointer for the object
//
// Return Value:
//
//      S_OK                    - Creation was successful
//      CLASS_E_NOAGGREGATION   - Tried to be created as part of an
aggregate
//
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpSvrDoc::CreateObject   DOC.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP CClassFactory::CreateInstance ( LPUNKNOWN pUnkOuter,
                                           REFIID riid,
                                           LPVOID FAR* ppvObject)
{
     HRESULT hErr;

     OutputDebugString("In CClassFactory::CreateInstance\r\n");

     // need to NULL the out parameter
     *ppvObject = NULL;

     // we don't support aggregation...
     if (pUnkOuter)
           {
           hErr = ResultFromScode(CLASS_E_NOAGGREGATION);
           goto error;
           }
     hErr = m_lpApp->m_lpDoc->CreateObject(riid, ppvObject);

error:
     return hErr;
};
```

CClassFactory::LockServer   (SIMPSVR Sample)

```
//**********************************************************************
//
// CClassFactory::LockServer
//
// Purpose:
//
//
// Parameters:
//
//      BOOL fLock      - TRUE to lock the server, FALSE to unlock it
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CoLockObjectExternal        OLE API
//      ResultFromScode             OLE API
//
// Comments:
//
//
//********************************************************************


STDMETHODIMP CClassFactory::LockServer ( BOOL fLock)
{
     OutputDebugString("In CClassFactory::LockServer\r\n");
     CoLockObjectExternal(m_lpApp, fLock, TRUE);

     return ResultFromScode( S_OK);
};
```

IDO.CPP (SIMPSVR Sample)

```
//**********************************************************************
// File name: IDO.CPP
//
//    Implementation file for the CDataObject Class
//
// Functions:
//
//    See ido.h for a list of member functions.
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "obj.h"
#include "ido.h"
#include "app.h"
#include "doc.h"
```

CDataObject::QueryInterface   (SIMPSVR Sample)

```
//**********************************************************************
//
// CDataObject::QueryInterface
//
// Purpose:
//
//
// Parameters:
//
//      REFIID riid         -   Interface being queried for.
//
//      LPVOID FAR *ppvObj  -   Out pointer for the interface.
//
// Return Value:
//
//      S_OK            - Success
//      E_NOINTERFACE   - Failure
//
// Function Calls:
//      Function                    Location
//
//      CSimpSvrObj::QueryInterface OBJ.CPP
//
// Comments:
//
//
//********************************************************************


STDMETHODIMP CDataObject::QueryInterface ( REFIID riid, LPVOID FAR* ppvObj)
{
     OutputDebugString("In CDataObject::QueryInterface\r\n");

     return m_lpObj->QueryInterface(riid, ppvObj);
};
```

CDataObject::AddRef   (SIMPSVR Sample)

```
//**********************************************************************
//
// CDataObject::AddRef
//
// Purpose:
//
//      Increments the reference count on CClassFactory and the application
//      object.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The Reference count on CDataObject
//
// Function Calls:
//      Function                    Location
//
//      OuputDebugString            Windows API
//      CSimpSvrObj::AddRef         OBJ.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP_(ULONG) CDataObject::AddRef ()
{
     OutputDebugString("In CDataObject::AddRef\r\n");
     ++m_nCount;
     return m_lpObj->AddRef();
};
```

CDataObject::Release (SIMPSVR Sample)

```
//**********************************************************************
//
// CDataObject::Release
//
// Purpose:
//
//      Decrements the reference count of CDataObject
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The new reference count
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpSvrObj::Release        OBJ.CPP
//
// Comments:
//
//
//********************************************************************


STDMETHODIMP_(ULONG) CDataObject::Release ()
{
     OutputDebugString("In CDataObject::Release\r\n");
     --m_nCount;
     return m_lpObj->Release();
};
```

CDataObject::QueryGetData (SIMPSVR Sample)

```
//**********************************************************************
//
// CDataObject::QueryGetData
//
// Purpose:
//
//      Called to determine if our object supports a particular
//      FORMATETC.
//
// Parameters:
//
//      LPFORMATETC pformatetc  - Pointer to the FORMATETC being queried
for.
//
// Return Value:
//
//      DATA_E_FORMATETC    - The FORMATETC is not supported
//      S_OK                - The FORMATETC is supported.
//
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      ResultFromScode             OLE API
//
// Comments:
//
//
//********************************************************************


STDMETHODIMP CDataObject::QueryGetData  ( LPFORMATETC pformatetc )
{
     SCODE sc = DATA_E_FORMATETC;

     OutputDebugString("In CDataObject::QueryGetData\r\n");

     // check the validity of the formatetc.
     if ( (pformatetc->cfFormat == CF_METAFILEPICT)  &&
            (pformatetc->dwAspect == DVASPECT_CONTENT) &&
            (pformatetc->tymed == TYMED_MFPICT) )
           sc = S_OK;

     return ResultFromScode(sc);
};
```

CDataObject::DAdvise (SIMPSVR Sample)

```
//**********************************************************************
//
// CDataObject::DAdvise
//
// Purpose:
//
//      Called by the container when it would like to be notified of
//      changes in the object data.
//
// Parameters:
//
//      FORMATETC FAR* pFormatetc   - The format the container is interested
in.
//
//      DWORD advf                  - The type of advise to be set up.
//
//      LPADVISESINK pAdvSink       - Pointer to the containers IAdviseSink
//
//      DWORD FAR* pdwConnection    - Out parameter to return a unique
connection id.
//
// Return Value:
//
//      passed on from IDataAdviseHolder
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CreateDataAdviseHolder      OLE API
//      IDataAdviseHolder::Advise   OLE API
//
// Comments:
//
//
//********************************************************************


STDMETHODIMP CDataObject::DAdvise  ( FORMATETC FAR* pFormatetc, DWORD advf,
                                                     LPADVISESINK pAdvSink,
DWORD FAR* pdwConnection)
{
     OutputDebugString("In CDataObject::DAdvise\r\n");

     // if no DataAdviseHolder has been created, then create one.
     if (!m_lpObj->m_lpDataAdviseHolder)
           CreateDataAdviseHolder(&m_lpObj->m_lpDataAdviseHolder);

     // pass on to the DataAdviseHolder
     return m_lpObj->m_lpDataAdviseHolder->Advise( this, pFormatetc, advf,

pAdvSink, pdwConnection);
}
```

CDataObject::GetData   (SIMPSVR Sample)

```
//**********************************************************************
//
// CDataObject::GetData
//
// Purpose:
//
//      Returns the data in the format specified in pformatetcIn.
//
// Parameters:
//
//      LPFORMATETC pformatetcIn    -   The format requested by the caller
//
//      LPSTGMEDIUM pmedium         -   The medium requested by the caller
//
// Return Value:
//
//      DATA_E_FORMATETC    - Format not supported
//      S_OK                - Success
//
// Function Calls:
//      Function                        Location
//
//      OutputDebugString               Windows API
//      CSimpSvrObj::GetMetaFilePict()  OBJ.CPP
//      ResultFromScode                 OLE API
//
// Comments:
//
//
//********************************************************************


STDMETHODIMP CDataObject::GetData  ( LPFORMATETC pformatetcIn, LPSTGMEDIUM
pmedium )
{
     SCODE sc = DATA_E_FORMATETC;

     OutputDebugString("In CDataObject::GetData\r\n");

     // Check to the FORMATETC and fill pmedium if valid.
     if ( (pformatetcIn->cfFormat == CF_METAFILEPICT)  &&
           (pformatetcIn->dwAspect == DVASPECT_CONTENT) &&
           (pformatetcIn->tymed & TYMED_MFPICT) )
          {
          HANDLE hmfPict = m_lpObj->GetMetaFilePict();
          pmedium->tymed = TYMED_MFPICT;
          pmedium->hGlobal = hmfPict;
          pmedium->pUnkForRelease = NULL;
          sc = S_OK;
          }

     return ResultFromScode( sc );
};
```

CDataObject::DUnadvise   (SIMPSVR Sample)

```
//**********************************************************************
//
// CDataObject::DUnadvise
//
// Purpose:
//
//      Breaks down an Advise connection.
//
// Parameters:
//
//      DWORD dwConnection  - Advise connection ID.
//
// Return Value:
//
//      Returned from the DataAdviseHolder.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      IDataAdviseHolder::Unadvise OLE
//
// Comments:
//
//
//********************************************************************


STDMETHODIMP CDataObject::DUnadvise  ( DWORD dwConnection)
{
     OutputDebugString("In CDataObject::DUnadvise\r\n");

     return m_lpObj->m_lpDataAdviseHolder->Unadvise(dwConnection);
};
```

CDataObject::GetDataHere   (SIMPSVR Sample)

```
//**********************************************************************
//
// CDataObject::GetDataHere
//
// Purpose:
//
//      Called to get a data format in a caller supplied location
//
// Parameters:
//
//      LPFORMATETC pformatetc  - FORMATETC requested
//
//      LPSTGMEDIUM pmedium     - Medium to return the data
//
// Return Value:
//
//      DATA_E_FORMATETC    - We don't support the requested format
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      In this simple implementation, we don't really support this
//      method, we just always return DATA_E_FORMATETC.
//
//********************************************************************


STDMETHODIMP CDataObject::GetDataHere  ( LPFORMATETC pformatetc,
                                                            LPSTGMEDIUM
pmedium )
{
     OutputDebugString("In CDataObject::GetDataHere\r\n");
     return ResultFromScode( DATA_E_FORMATETC);
};
```

CDataObject::GetCanonicalFormatEtc   (SIMPSVR Sample)

```
//**********************************************************************
//
// CDataObject::GetCanonicalFormatEtc
//
// Purpose:
//
//      Returns a FORMATETC that is equivalent to the one passed in.
//
// Parameters:
//
//      LPFORMATETC pformatetc      - FORMATETC to be tested.
//
//      LPFORMATETC pformatetcOut   - Out ptr for returned FORMATETC.
//
// Return Value:
//
//      DATA_S_SAMEFORMATETC    - Use the same formatetc as was passed.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CoGetMalloc                 OLE API
//      IMalloc::Alloc              OLE
//      IMalloc::Release            OLE
//      _fmemcpy                    C run-time
//
// Comments:
//
//
//********************************************************************


STDMETHODIMP CDataObject::GetCanonicalFormatEtc  ( LPFORMATETC pformatetc,

LPFORMATETC pformatetcOut)
{
     HRESULT hresult;
     OutputDebugString("In CDataObject::GetCanonicalFormatEtc\r\n");

     if (!pformatetcOut)
           return ResultFromScode(E_INVALIDARG);

     /* OLE2NOTE: we must make sure to set all out parameters to NULL. */
     pformatetcOut->ptd = NULL;

     if (!pformatetc)
           return ResultFromScode(E_INVALIDARG);

     // OLE2NOTE: we must validate that the format requested is supported
     if ((hresult = QueryGetData(pformatetc)) != NOERROR)
           return hresult;
     /* OLE2NOTE: an app that is insensitive to target device (as
     **    SimpSvr is) should fill in the lpformatOut parameter
     **    but NULL out the "ptd" field; it should return NOERROR if the
     **    input formatetc->ptd what non-NULL. this tells the caller
     **    that it is NOT necessary to maintain a separate screen
     **    rendering and printer rendering. if should return
     **    DATA_S_SAMEFORMATETC if the input and output formatetc's are
     **    identical.
     */

     *pformatetcOut = *pformatetc;
     if (pformatetc->ptd == NULL)
           return ResultFromScode(DATA_S_SAMEFORMATETC);
     else
           {
           pformatetcOut->ptd = NULL;
           return NOERROR;
           }
};
```

CDataObject::SetData   (SIMPSVR Sample)

```
//**********************************************************************
//
// CDataObject::SetData
//
// Purpose:
//
//      Called to set the data for the object.
//
// Parameters:
//
//      LPFORMATETC pformatetc      - the format of the data being passed
//
//      STGMEDIUM FAR * pmedium     - the location of the data.
//
//      BOOL fRelease               - Defines the ownership of the medium
//
// Return Value:
//
//      DATA_E_FORMATETC    - Not a valid FORMATETC for this object
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      This simple object does not support having its data set, so an
//      error value is always returned.
//
//********************************************************************


STDMETHODIMP CDataObject::SetData  ( LPFORMATETC pformatetc, STGMEDIUM FAR *
pmedium,
                                                      BOOL fRelease)
{
     OutputDebugString("In CDataObject::SetData\r\n");
     return ResultFromScode( DATA_E_FORMATETC );
};
```

CDataObject::EnumFormatEtc (SIMPSVR Sample)

```
//**********************************************************************
//
// CDataObject::EnumFormatEtc
//
// Purpose:
//
//      Enumerates the formats supported by this object.
//
// Parameters:
//
//      DWORD dwDirection                       - Order of enumeration.
//
//      LPENUMFORMATETC FAR* ppenumFormatEtc    - Place to return a pointer
//                                                to the enumerator.
//
// Return Value:
//
//      OLE_S_USEREG    - Indicates that OLE should consult the REG DB
//                        to enumerate the formats.
//
// Function Calls:
//      Function                      Location
//
//      OutputDebugString             Windows API
//
// Comments:
//
//
//********************************************************************


STDMETHODIMP CDataObject::EnumFormatEtc  ( DWORD dwDirection,

LPENUMFORMATETC FAR* ppenumFormatEtc)
{
     OutputDebugString("In CDataObject::EnumFormatEtc\r\n");
     // need to NULL the out parameter
     *ppenumFormatEtc = NULL;
     return ResultFromScode( OLE_S_USEREG );
};
```

CDataObject::EnumDAdvise (SIMPSVR Sample)

```
//**********************************************************************
//
// CDataObject::EnumDAdvise
//
// Purpose:
//
//      Returns an enumerator that enumerates all of the advises
//      set up on this data object.
//
// Parameters:
//
//      LPENUMSTATDATA FAR* ppenumAdvise    - An out ptr in which to
//                                            return the enumerator.
//
// Return Value:
//
//      Passed back from IDataAdviseHolder::EnumAdvise
//
// Function Calls:
//      Function                        Location
//
//      OutputDebugString               Windows API
//      IDAtaAdviseHolder::EnumAdvise   OLE
//
// Comments:
//
//      This just delegates to the DataAdviseHolder.
//
//********************************************************************


STDMETHODIMP CDataObject::EnumDAdvise  ( LPENUMSTATDATA FAR* ppenumAdvise)
{
     OutputDebugString("In CDataObject::EnumDAdvise\r\n");
     // need to NULL the out parameter
     *ppenumAdvise = NULL;

     return m_lpObj->m_lpDataAdviseHolder->EnumAdvise(ppenumAdvise);
};
```

IEC.CPP   (SIMPSVR Sample)

```
//**********************************************************************
// File name: IEC.CPP
//
//    Implementation file for the CExternalConnection Class
//
// Functions:
//
//    See iec.h for a list of member functions.
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "obj.h"
#include "iec.h"
#include "app.h"
#include "doc.h"
```

CExternalConnection::QueryInterface (SIMPSVR Sample)

```
//**********************************************************************
//
// CExternalConnection::QueryInterface
//
// Purpose:
//
//
// Parameters:
//
//      REFIID riid         -   Interface being queried for.
//
//      LPVOID FAR *ppvObj  -   Out pointer for the interface.
//
// Return Value:
//
//      S_OK            - Success
//      E_NOINTERFACE   - Failure
//
// Function Calls:
//      Function                    Location
//
//      CSimpSvrObj::QueryInterface OBJ.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP CExternalConnection::QueryInterface (REFIID riid, LPVOID FAR*
ppvObj)
{
     OutputDebugString("In CExternalConnection::QueryInterface\r\n");

     return m_lpObj->QueryInterface(riid, ppvObj);
}
```

CExternalConnection::AddRef   (SIMPSVR Sample)

```
//**********************************************************************
//
// CExternalConnection::AddRef
//
// Purpose:
//
//      Increments the reference count on CExternalConnection and the
"object"
//      object.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The Reference count on the Object.
//
// Function Calls:
//      Function                    Location
//
//      OuputDebugString            Windows API
//      CSimpSvrObj::AddRef         OBJ.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP_(ULONG) CExternalConnection::AddRef ()
{
     OutputDebugString("In CExternalConnection::AddRef\r\n");
     ++m_nCount;
     return m_lpObj->AddRef();
};
```

CExternalConnection::Release   (SIMPSVR Sample)

```
//**********************************************************************
//
// CExternalConnection::Release
//
// Purpose:
//
//      Decrements the reference count of COleObject and the
//      "object" object.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The new reference count
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpSvrObj::Release        OBJ.CPP
//
// Comments:
//
//
//********************************************************************


STDMETHODIMP_(ULONG) CExternalConnection::Release ()
{
     OutputDebugString("In CExternalConnection::Release\r\n");
     --m_nCount;
     return m_lpObj->Release();
};
```

CExternalConnection::AddConnection   (SIMPSVR Sample)

```
//**********************************************************************
//
// CExternalConnection::AddConnection
//
// Purpose:
//
//      Called when another connection is made to the object.
//
// Parameters:
//
//      DWORD extconn   -   Type of connection
//
//      DWORD reserved  -   Reserved
//
// Return Value:
//
//      Strong connection count
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP_(DWORD) CExternalConnection::AddConnection (DWORD extconn,
DWORD reserved)
{
     OutputDebugString("In CExternalConnection::AddConnection\r\n");

     if (extconn & EXTCONN_STRONG)
           return ++m_dwStrong;

     return 0;
}
```

CExternalConnection::ReleaseConnection (SIMPSVR Sample)

```
//**********************************************************************
//
// CExternalConnection::ReleaseConnection
//
// Purpose:
//
//      Called when a connection to the object is released.
//
// Parameters:
//
//      DWORD extconn               - Type of Connection
//
//      DWORD reserved              - Reserved
//
//      BOOL fLastReleaseCloses     - Close flag
//
// Return Value:
//
//      The new reference count
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      COleObject::Close           IOO.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP_(DWORD) CExternalConnection::ReleaseConnection (DWORD extconn,
DWORD reserved, BOOL fLastReleaseCloses)
{
     OutputDebugString("In CExternalConnection::ReleaseConnection\r\n");

     if (extconn & EXTCONN_STRONG)
            {
            DWORD dwSave = --m_dwStrong;

            if (!m_dwStrong && fLastReleaseCloses)
                  m_lpObj->m_OleObject.Close(OLECLOSE_SAVEIFDIRTY);

            return dwSave;
            }
     return 0;
}
```

IOIPAO.CPP (SIMPSVR Sample)

```
//**********************************************************************
// File name: IOIPAO.CPP
//
//    Implementation file for the CClassFactory Class
//
// Functions:
//
//    See ioipao.h for a list of member functions.
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "obj.h"
#include "ioipao.h"
#include "app.h"
#include "doc.h"
```

COleInPlaceActiveObject::QueryInterface (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceActiveObject::QueryInterface
//
// Purpose:
//
//
// Parameters:
//
//      REFIID riid         -   Interface being queried for.
//
//      LPVOID FAR *ppvObj  -   Out pointer for the interface.
//
// Return Value:
//
//      S_OK            - Success
//      E_NOINTERFACE   - Failure
//
// Function Calls:
//      Function                    Location
//
//      CSimpSvrObj::QueryInterface OBJ.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleInPlaceActiveObject::QueryInterface ( REFIID riid, LPVOID
FAR* ppvObj)
{
     OutputDebugString("In COleInPlaceActiveObject::QueryInterface\r\n");
     // need to NULL the out parameter
     return m_lpObj->QueryInterface(riid, ppvObj);
}
```

COleInPlaceActiveObject::AddRef   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceActiveObject::AddRef
//
// Purpose:
//
//      Increments the reference count on COleInPlaceActiveObject and the
//      "object" object.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The Reference count on the "object" object.
//
// Function Calls:
//      Function                    Location
//
//      OuputDebugString            Windows API
//      CSimpSvrObj::AddRef         OBJ.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP_(ULONG) COleInPlaceActiveObject::AddRef ()
{
     OutputDebugString("In COleInPlaceActiveObject::AddRef\r\n");

     ++m_nCount;

     return m_lpObj->AddRef();
}
```

COleInPlaceActiveObject::Release   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceActiveObject::Release
//
// Purpose:
//
//      Decrements the reference count of COleInPlaceActiveObject.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The new reference count
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpSvrObj::Release        OBJ.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP_(ULONG) COleInPlaceActiveObject::Release ()
{
     OutputDebugString("In COleInPlaceActiveObject::Release\r\n");

     --m_nCount;

     return m_lpObj->Release();
}
```

COleInPlaceActiveObject::OnDocWindowActivate   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceActiveObject::OnDocWindowActivate
//
// Purpose:
//
//      Called when the doc window (in an MDI App) is (de)activated.
//
// Parameters:
//
//      BOOL fActivate  - TRUE if activating, FALSE if deactivating
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                            Location
//
//      OutputDebugString                   Windows API
//      IOleInPlaceFrame::SetActiveObject   Container
//      CSimpSvrObject::AddFrameLevelUI     OBJ.CPP
//
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleInPlaceActiveObject::OnDocWindowActivate  ( BOOL
fActivate )
{
     OutputDebugString("In
COleInPlaceActiveObject::OnDocWindowActivate\r\n");

     // Activating?
     if (fActivate)
           m_lpObj->AddFrameLevelUI();

     // No frame level tools to remove...

     return ResultFromScode(S_OK);
};
```

COleInPlaceActiveObject::OnFrameWindowActivate   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceActiveObject::OnFrameWindowActivate
//
// Purpose:
//
//      Called when the Frame window is (de)activating
//
// Parameters:
//
//      BOOL fActivate  - TRUE if activating, FALSE if Deactivating
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      SetFocus                    Windows API
//
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleInPlaceActiveObject::OnFrameWindowActivate  ( BOOL
fActivate)
{
     OutputDebugString("In
COleInPlaceActiveObject::OnFrameWindowActivate\r\n");

     // set the focus to the object window if we are activating.
/*    if (fActivate)
           SetFocus(m_lpObj->m_lpDoc->GethDocWnd()); */

     return ResultFromScode( S_OK );
};
```

COleInPlaceActiveObject::GetWindow (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceActiveObject::GetWindow
//
// Purpose:
//
//      Gets the objects Window Handle.
//
// Parameters:
//
//      HWND FAR* lphwnd    - Location to return the window handle.
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpSvrDoc::GethDocWnd     DOC.H
//
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleInPlaceActiveObject::GetWindow  ( HWND FAR* lphwnd)
{
     OutputDebugString("In COleInPlaceActiveObject::GetWindow\r\n");
     // need to NULL the out parameter
     *lphwnd = m_lpObj->m_lpDoc->GethDocWnd();
     return ResultFromScode( S_OK );
};
```

COleInPlaceActiveObject::ContextSensitiveHelp   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceActiveObject::ContextSensitiveHelp
//
// Purpose:
//
//      Used to implement Context Sensitive help
//
// Parameters:
//
//      None
//
// Return Value:
//
//      E_NOTIMPL
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
//
// Comments:
//
//      See TECHNOTES.WRI include with the OLE SDK for proper
//      implementation of this function.
//
//********************************************************************

STDMETHODIMP COleInPlaceActiveObject::ContextSensitiveHelp  ( BOOL
fEnterMode )
{
     OutputDebugString("In
COleInPlaceActiveObject::ContextSensitiveHelp\r\n");
     return ResultFromScode( E_NOTIMPL);
};
```

COleInPlaceActiveObject::TranslateAccelerator (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceActiveObject::TranslateAccelerator
//
// Purpose:
//
//      Used for translating accelerators in .DLL objects.
//
// Parameters:
//
//      LPMSG lpmsg - Pointer to a message
//
// Return Value:
//
//      S_FALSE
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
//
// Comments:
//
//      This method should never be called since we are implemented
//      in an executable.
//
//********************************************************************

STDMETHODIMP COleInPlaceActiveObject::TranslateAccelerator  ( LPMSG lpmsg)
{
     OutputDebugString("In
COleInPlaceActiveObject::TranslateAccelerator\r\n");
     // no accelerator table, return FALSE
     return ResultFromScode( S_FALSE );
};
```

COleInPlaceActiveObject::ResizeBorder   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceActiveObject::ResizeBorder
//
// Purpose:
//
//      Called when the border changes size.
//
// Parameters:
//
//      LPCRECT lprectBorder                - New Border
//
//      LPOLEINPLACEUIWINDOW lpUIWindow     - Pointer to UIWindow
//
//      BOOL fFrameWindow                   - True if lpUIWindow is the
//                                            frame window.
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
//
// Comments:
//
//      Need to call SetBorderSpace again...
//
//********************************************************************

STDMETHODIMP COleInPlaceActiveObject::ResizeBorder  ( LPCRECT lprectBorder,

       LPOLEINPLACEUIWINDOW lpUIWindow,

       BOOL fFrameWindow)
{
     OutputDebugString("In COleInPlaceActiveObject::ResizeBorder\r\n");

     // should always have an inplace frame...
     m_lpObj->GetInPlaceFrame()->SetBorderSpace(NULL);

     // There will only be a UIWindow if in an MDI container
     if (m_lpObj->GetUIWindow())
           m_lpObj->GetUIWindow()->SetBorderSpace(NULL);

     return ResultFromScode( S_OK );
};
```

COleInPlaceActiveObject::EnableModeless   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceActiveObject::EnableModeless
//
// Purpose:
//
//      Called to enable/disable modeless dialogs.
//
// Parameters:
//
//      BOOL fEnable    - TRUE to enable, FALSE to disable
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
//
// Comments:
//
//      Called by the container when a model dialog box is added/removed
//      from the screen.  The appropriate action for a server application
//      is to disable/enable any modeless dialogs currently being displayed.
//      Since this application doesn't display any modeless dialogs,
//      this method is essentially ignored.
//
//********************************************************************

STDMETHODIMP COleInPlaceActiveObject::EnableModeless  ( BOOL fEnable)
{
     OutputDebugString("In COleInPlaceActiveObject::EnableModeless\r\n");
     return ResultFromScode( S_OK );
};
```

IOIPO.CPP   (SIMPSVR Sample)

```
//**********************************************************************
// File name: IOIPO.CPP
//
//    Implementation file for the CClassFactory Class
//
// Functions:
//
//    See ioipo.h for a list of member functions.
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "obj.h"
#include "ioipo.h"
#include "app.h"
#include "doc.h"
#include "math.h"
```

COleInPlaceObject::QueryInterface (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceObject::QueryInterface
//
// Purpose:
//
//
// Parameters:
//
//      REFIID riid         -   Interface being queried for.
//
//      LPVOID FAR *ppvObj  -   Out pointer for the interface.
//
// Return Value:
//
//      S_OK            - Success
//      E_NOINTERFACE   - Failure
//
// Function Calls:
//      Function                    Location
//
//      CSimpSvrObj::QueryInterface OBJ.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleInPlaceObject::QueryInterface ( REFIID riid, LPVOID FAR*
ppvObj)
{
     OutputDebugString("In COleInPlaceObject::QueryInterface\r\n");
     // need to NULL the out parameter
     *ppvObj = NULL;
     return m_lpObj->QueryInterface(riid, ppvObj);
}
```

COleInPlaceObject::AddRef (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceObject::AddRef
//
// Purpose:
//
//      Increments the reference count on COleInPlaceObject and the "object"
//      object.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The Reference count on the Object
//
// Function Calls:
//      Function                    Location
//
//      OuputDebugString            Windows API
//      CSimpSvrObj::AddRef         OBJ.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP_(ULONG) COleInPlaceObject::AddRef ()
{
     OutputDebugString("In COleInPlaceObject::AddRef\r\n");
     ++m_nCount;
     return m_lpObj->AddRef();
}
```

COleInPlaceObject::Release (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceObject::Release
//
// Purpose:
//
//      Decrements the reference count of COleInPlaceObject and the
//      object.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The new reference count
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpSvrObj::Release        OBJ.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP_(ULONG) COleInPlaceObject::Release ()
{
     OutputDebugString("In COleInPlaceObject::Release\r\n");
     --m_nCount;
     return m_lpObj->Release();
}
```

COleInPlaceObject::InPlaceDeactivate (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceObject::InPlaceDeactivate
//
// Purpose:
//
//      Called to deactivat the object
//
// Parameters:
//
//      None
//
// Return Value:
//
//
//
// Function Calls:
//      Function                                Location
//
//      OutputDebugString                       Windows API
//      IOleClientSite::QueryInterface          Container
//      IOleInPlaceSite::OnInPlaceDeactivate    Container
//      IOleInPlaceSite::Release                Container
//
// Comments:
//
//
//********************************************************************


STDMETHODIMP COleInPlaceObject::InPlaceDeactivate()
{
      OutputDebugString("In COleInPlaceObject::InPlaceDeactivate\r\n");

      // if not inplace active, return NOERROR
      if (!m_lpObj->m_fInPlaceActive)
            return NOERROR;

      // clear inplace flag
      m_lpObj->m_fInPlaceActive = FALSE;

      // deactivate the UI
      m_lpObj->DeactivateUI();
      m_lpObj->DoInPlaceHide();

      // tell the container that we are deactivating.
      if (m_lpObj->m_lpIPSite)
            {
            m_lpObj->m_lpIPSite->OnInPlaceDeactivate();
            m_lpObj->m_lpIPSite->Release();
            m_lpObj->m_lpIPSite =NULL;
            }
```

```
     return ResultFromScode(S_OK);
}
```

COleInPlaceObject::UIDeactivate (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceObject::UIDeactivate
//
// Purpose:
//
//      Instructs us to remove our UI.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      NOERROR
//
// Function Calls:
//      Function                               Location
//
//      OutputDebugString                      Windows API
//      IOleInPlaceUIWindow::SetActiveObject    Container
//      IOleInPlaceFrame::SetActiveObject       Container
//      IOleClientSite::QueryInterface          Container
//      IOleInPlaceSite::OnUIDeactivate         Container
//      IOleInPlaceSite::Release                Container
//      CSimpSvrObj::DoInPlaceHide              OBJ.H
//      IDataAdviseHolder::SendOnDataChange     OLE
//
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleInPlaceObject::UIDeactivate()
{
     OutputDebugString("In COleInPlaceObject::UIDeactivate\r\n");

     m_lpObj->DeactivateUI();

     return ResultFromScode (S_OK);
}
```

COleInPlaceObject::SetObjectRects (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceObject::SetObjectRects
//
// Purpose:
//
//      Called when the container clipping region or the object position
//      changes.
//
// Parameters:
//
//      LPCRECT lprcPosRect     - New Position Rect.
//
//      LPCRECT lprcClipRect    - New Clipping Rect.
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      IntersectRect               Windows API
//      OffsetRect                  Windows API
//      CopyRect                    Windows API
//      MoveWindow                  Windows API
//      CSimpSvrDoc::GethHatchWnd   DOC.H
//      CSimpSvrDoc::gethDocWnd     DOC.h
//      SetHatchWindowSize          OLE2UI
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleInPlaceObject::SetObjectRects  ( LPCRECT lprcPosRect,
LPCRECT lprcClipRect)
{
     OutputDebugString("In COleInPlaceObject::SetObjectRects\r\n");

     RECT resRect;
     POINT pt;

     // Get the intersection of the clipping rect and the position rect.
     IntersectRect(&resRect, lprcPosRect, lprcClipRect);

     m_lpObj->m_xOffset = abs (resRect.left - lprcPosRect->left);
     m_lpObj->m_yOffset = abs (resRect.top - lprcPosRect->top);

     m_lpObj->m_scale = (float)(lprcPosRect->right -
lprcPosRect->left)/m_lpObj->m_size.x;
```

```
     if (m_lpObj->m_scale == 0)
           m_lpObj->m_scale = (float) 1.0;

     char szBuffer[255];
     wsprintf(szBuffer,"New Scale %3d\r\n",m_lpObj->m_scale);
     OutputDebugString(szBuffer);

     // Adjust the size of the Hatch Window.
     SetHatchWindowSize(m_lpObj->m_lpDoc->GethHatchWnd(),(LPRECT)
lprcPosRect, (LPRECT) lprcClipRect, &pt);

     // offset the rect
     OffsetRect(&resRect, pt.x, pt.y);

     CopyRect(&m_lpObj->m_posRect, lprcPosRect);

     // Move the actual object window
     MoveWindow(m_lpObj->m_lpDoc->GethDocWnd(),
                          resRect.left,
                          resRect.top,
                          resRect.right - resRect.left,
                          resRect.bottom - resRect.top,
                          TRUE);


     return ResultFromScode( S_OK );
};
```

COleInPlaceObject::GetWindow (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceObject::GetWindow
//
// Purpose:
//
//      Returns the Window handle of the inplace object
//
// Parameters:
//
//      HWND FAR* lphwnd    - Out pointer in which to return the window
//                            Handle.
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpleDoc::GethDocWnd      DOC.H
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleInPlaceObject::GetWindow  ( HWND FAR* lphwnd)
{
     OutputDebugString("In COleInPlaceObject::GetWindow\r\n");
     *lphwnd = m_lpObj->m_lpDoc->GethDocWnd();

     return ResultFromScode( S_OK );
};
```

COleInPlaceObject::ContextSensitiveHelp   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceObject::ContextSensitiveHelp
//
// Purpose:
//
//      Used in performing Context Sensitive Help
//
// Parameters:
//
//      BOOL fEnterMode     - Flag to determine if enter or exiting
//                            Context Sensitive Help.
//
// Return Value:
//
//      E_NOTIMPL
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      This function is not implemented due to the fact that it is
//      beyond the scope of a simple object.  All *real* applications
//      are going to want to implement this function, otherwise any
//      container that supports context sensitive help will not work
//      properly while the object is in place.
//
//      See TECHNOTES.WRI include with the OLE SDK for details on
//      Implementing this method.
//
//********************************************************************

STDMETHODIMP COleInPlaceObject::ContextSensitiveHelp  ( BOOL fEnterMode)
{
     OutputDebugString("In COleInPlaceObject::ContextSensitiveHelp\r\n");
     return ResultFromScode( E_NOTIMPL);
};
```

COleInPlaceObject::ReactivateAndUndo   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleInPlaceObject::ReactivateAndUndo
//
// Purpose:
//
//      Called when the container wants to undo the last edit made in
//      the object.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      INPLACE_E_NOTUNDOABLE
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Since this server does not support undo, the value
//      INPLACE_E_NOTUNDOABLE is always returned.
//
//********************************************************************

STDMETHODIMP COleInPlaceObject::ReactivateAndUndo  ()
{
     OutputDebugString("In COleInPlaceObject::ReactivateAndUndo\r\n");
     return ResultFromScode( INPLACE_E_NOTUNDOABLE );
};
```

IOO.CPP (SIMPSVR Sample)

```
//**********************************************************************
// File name: IOO.CPP
//
//    Implementation file for the COleObject Class
//
// Functions:
//
//    See ioo.h for a list of member functions.
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "obj.h"
#include "ioo.h"
#include "app.h"
#include "doc.h"

#define VERB_OPEN 1
```

COleObject::QueryInterface (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::QueryInterface
//
// Purpose:
//
//
// Parameters:
//
//      REFIID riid         -   Interface being queried for.
//
//      LPVOID FAR *ppvObj  -   Out pointer for the interface.
//
// Return Value:
//
//      S_OK            - Success
//      E_NOINTERFACE   - Failure
//
// Function Calls:
//      Function                    Location
//
//      CSimpSvrObj::QueryInterface OBJ.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleObject::QueryInterface ( REFIID riid, LPVOID FAR* ppvObj)
{
     OutputDebugString("In COleObject::QueryInterface\r\n");
     return m_lpObj->QueryInterface(riid, ppvObj);
}
```

COleObject::AddRef   (SIMPSVR Sample)

```
//***********************************************************************
//
// COleObject::AddRef
//
// Purpose:
//
//      Increments the reference count on COleObject and the "object"
//      object.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The Reference count on the Object.
//
// Function Calls:
//      Function                    Location
//
//      OuputDebugString            Windows API
//      CSimpSvrObj::AddRef         OBJ.CPP
//
// Comments:
//
//
//*********************************************************************

STDMETHODIMP_(ULONG) COleObject::AddRef ()
{
     OutputDebugString("In COleObject::AddRef\r\n");
     ++m_nCount;
     return m_lpObj->AddRef();
}
```

COleObject::Release   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::Release
//
// Purpose:
//
//      Decrements the reference count of COleObject and the
//      "object" object.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The new reference count
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpSvrObj::Release        OBJ.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP_(ULONG) COleObject::Release ()
{
     OutputDebugString("In COleObject::Release\r\n");
     --m_nCount;
     return m_lpObj->Release();
}
```

COleObject::SetClientSite   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::SetClientSite
//
// Purpose:
//
//      Called to notify the object of it's client site.
//
// Parameters:
//
//      LPOLECLIENTSITE pClientSite     - ptr to new client site
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      IOleClientSite::Release     Container
//      IOleClientSite::AddRef      Container
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleObject::SetClientSite  ( LPOLECLIENTSITE pClientSite)
{
     OutputDebugString("In COleObject::SetClientSite\r\n");

     // if we already have a client site, release it.
     if (m_lpObj->m_lpOleClientSite)
           {
           m_lpObj->m_lpOleClientSite->Release();
           m_lpObj->m_lpOleClientSite = NULL;
           }

     // store copy of the client site.
     m_lpObj->m_lpOleClientSite = pClientSite;

     // AddRef it so it doesn't go away.
     if (m_lpObj->m_lpOleClientSite)
           m_lpObj->m_lpOleClientSite->AddRef();

     return ResultFromScode(S_OK);
}
```

COleObject::Advise   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::Advise
//
// Purpose:
//
//      Called to set up an advise on the OLE object.
//
// Parameters:
//
//      LPADVISESINK pAdvSink       - ptr to the Advise Sink for
notification
//
//      DWORD FAR* pdwConnection    - place to return the connection ID.
//
// Return Value:
//
//      Passed back from IOleAdviseHolder::Advise.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CreateOleAdviseHolder       OLE API
//      IOleAdviseHolder::Advise    OLE
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleObject::Advise ( LPADVISESINK pAdvSink, DWORD FAR*
pdwConnection)
{
     OutputDebugString("In COleObject::Advise\r\n");

     // if we haven't made an OleAdviseHolder yet, make one.
     if (!m_lpObj->m_lpOleAdviseHolder)
           CreateOleAdviseHolder(&m_lpObj->m_lpOleAdviseHolder);

     // pass this call onto the OleAdviseHolder.
     return m_lpObj->m_lpOleAdviseHolder->Advise(pAdvSink, pdwConnection);
}
```

COleObject::SetHostNames (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::SetHostNames
//
// Purpose:
//
//      Called to pass strings for Window titles.
//
// Parameters:
//
//      LPCSTR szContainerApp   -   ptr to string describing Container App
//
//      LPCSTR szContainerObj   -   ptr to string describing Object
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      This routine is called so that the server application can
//      set the window title appropriately.
//
//********************************************************************

//@@WTK WIN32, UNICODE
//STDMETHODIMP COleObject::SetHostNames  ( LPCSTR szContainerApp, LPCSTR
szContainerObj)
STDMETHODIMP COleObject::SetHostNames  ( LPCOLESTR szContainerApp, LPCOLESTR
szContainerObj)
{
     OutputDebugString("In COleObject::SetHostNames\r\n");

     return ResultFromScode( S_OK);
};
```

COleObject::DoVerb (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::DoVerb
//
// Purpose:
//
//      Called by the container application to invoke a verb.
//
// Parameters:
//
//      LONG iVerb                  - The value of the verb to be
//                                    invoked.
//
//      LPMSG lpmsg                 - The message that caused the
//                                    verb to be invoked.
//
//      LPOLECLIENTSITE pActiveSite - Ptr to the active client site.
//
//      LONG lindex                 - Used in extended layout
//
//      HWND hwndParent             - This should be the window handle of
//                                    the window in which we are contained.
//                                    This value could be used to "fake"
//                                    inplace activation in a manner similar
//                                    to Video for Windows in OLE 1.0.
//
//      LPCRECT lprcPosRect         - The rectangle that contains the object
//                                    within hwndParent.  Also used to
//                                    "fake" inplace activation.
//
// Return Value:
//
//      OLE_E_NOTINPLACEACTIVE      - Returned if attempted to undo while
not
//                                    inplace active.
//      S_OK
//
// Function Calls:
//      Function                            Location
//
//      OutputDebugString                   Windows API
//      ShowWindow                          Windows API
//      CSimpSvrObj::DoInPlaceActivate      OBJ.CPP
//      CSimpSvrObj::DoInPlaceHide          OBJ.CPP
//      COleObject::OpenEdit                IOO.CPP
//      CSimpSvrDoc::GethDocWnd             DOC.H
//      COleInPlaceObj::InPlaceDeactivate   IOIPO.CPP
//
// Comments:
//
//      Be sure to look at TECHNOTES.WRI included with the OLE
//      SDK for a description of handling the inplace verbs
//      properly.
//
//********************************************************************

STDMETHODIMP COleObject::DoVerb  (  LONG iVerb,
                                                   LPMSG lpmsg,
                                                   LPOLECLIENTSITE
pActiveSite,
                                                   LONG lindex,
                                                   HWND hwndParent,
                                                   LPCRECT lprcPosRect)
{
     OutputDebugString("In COleObject::DoVerb\r\n");

     switch (iVerb)
           {
           case OLEIVERB_SHOW:
           case OLEIVERB_PRIMARY:
                 if (m_fOpen)
                       SetFocus(m_lpObj->m_lpDoc->GethAppWnd());
                 else if (m_lpObj->DoInPlaceActivate(iVerb) == FALSE)
                       OpenEdit(pActiveSite);
                 break;

           case OLEIVERB_UIACTIVATE:
                 if (m_fOpen)
                       return ResultFromScode (E_FAIL);

                 // inplace activate
                 if (!m_lpObj->DoInPlaceActivate(iVerb))
                       return ResultFromScode (E_FAIL);
                 break;

           case OLEIVERB_DISCARDUNDOSTATE:
                 // don't have to worry about this situation as we don't
                 // support an undo state.
                 if (!m_lpObj->m_fInPlaceActive)
                       return ResultFromScode(OLE_E_NOT_INPLACEACTIVE);
                 break;

           case OLEIVERB_HIDE:
                 // if inplace active, do an "inplace" hide, otherwise
                 // just hide the app window.
                 if (m_lpObj->m_fInPlaceActive)
                       {
                       m_lpObj->DeactivateUI();
                       m_lpObj->DoInPlaceHide();
                       }
                 else
                       m_lpObj->m_lpDoc->GetApp()->HideAppWnd();
                 break;

           case OLEIVERB_OPEN:
           case VERB_OPEN:
                 // if inplace active, deactivate
```

```
                if (m_lpObj->m_fInPlaceActive)
                      m_lpObj->m_OleInPlaceObject.InPlaceDeactivate();

                // open into another window.
                OpenEdit(pActiveSite);
                break;

          default:
                if (iVerb < 0)
                      return ResultFromScode(E_FAIL);
          }

    return ResultFromScode( S_OK);
};
```

COleObject::GetExtent   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::GetExtent
//
// Purpose:
//
//      Returns the extent of the object.
//
// Parameters:
//
//      DWORD dwDrawAspect  - The aspect in which to get the size.
//
//      LPSIZEL lpsizel     - Out ptr to return the size.
//
// Return Value:
//
//
//
// Function Calls:
//      Function                        Location
//
//      OutputDebugString               Windows API
//      XformWidthInPixelsToHimetric    OLE2UI
//      XformHeightInPixelsToHimetric   OLE2UI
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleObject::GetExtent  ( DWORD dwDrawAspect, LPSIZEL lpsizel)
{
     OutputDebugString("In COleObject::GetExtent\r\n");

     SCODE sc = E_FAIL;

     // Only DVASPECT_CONTENT is supported....
     if (dwDrawAspect == DVASPECT_CONTENT)
           {
           sc = S_OK;

           // return the correct size in HIMETRIC...
           lpsizel->cx = XformWidthInPixelsToHimetric(NULL, m_lpObj-
>m_size.x);
           lpsizel->cy = XformHeightInPixelsToHimetric(NULL, m_lpObj-
>m_size.y);
           }

     return ResultFromScode( sc );
};
```

COleObject::Update   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::Update
//
// Purpose:
//
//      Called to get the most up to date data
//
// Parameters:
//
//      None
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                            Location
//
//      OutputDebugString                   Windows API
//      IDataAdviseHolder::SendOnDataChange OLE
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleObject::Update()
{
     OutputDebugString("In COleObject::Update\r\n");

     // force an update
     m_lpObj->SendOnDataChange();

     return ResultFromScode( S_OK );
};
```

COleObject::Close (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::Close
//
// Purpose:
//
//      Called when the OLE object needs to be closed
//
// Parameters:
//
//      DWORD dwSaveOption  - Flags to instruct the server how to prompt
//                            the user.
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpSvrDoc::Close          DOC.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleObject::Close  ( DWORD dwSaveOption)
{
     OutputDebugString("In COleObject::Close\r\n");

     // delegate to the document object.
     m_lpObj->m_lpDoc->Close();

     return ResultFromScode( S_OK );
};
```

COleObject::Unadvise   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::Unadvise
//
// Purpose:
//
//      Breaks down an OLE advise that has been set up on this object.
//
// Parameters:
//
//      DWORD dwConnection  - Connection that needs to be broken down
//
// Return Value:
//
//      Passed back from IOleAdviseHolder::Unadvise
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      IOleAdviseHolder::Unadvise  OLE
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleObject::Unadvise ( DWORD dwConnection)
{
     OutputDebugString("In COleObject::Unadvise\r\n");

     // pass on to OleAdviseHolder.
     return m_lpObj->m_lpOleAdviseHolder->Unadvise(dwConnection);
};
```

COleObject::EnumVerbs (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::EnumVerbs
//
// Purpose:
//
//      Enumerates the verbs associated with this object.
//
// Parameters:
//
//      LPENUMOLEVERB FAR* ppenumOleVerb    - Out ptr in which to return
//                                            the enumerator
//
// Return Value:
//
//      OLE_S_USEREG    - Instructs OLE to use the verbs found in the
//                        REG DB for this server.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      In a .DLL, an application cannot return OLE_S_USEREG.  This is
//      due to the fact that the default object handler is not being
//      used, and the container is really making direct function calls
//      into the server .DLL.
//
//********************************************************************

STDMETHODIMP COleObject::EnumVerbs  ( LPENUMOLEVERB FAR* ppenumOleVerb)
{
     OutputDebugString("In COleObject::EnumVerbs\r\n");

     return ResultFromScode( OLE_S_USEREG );
};
```

COleObject::GetClientSite   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::GetClientSite
//
// Purpose:
//
//      Called to get the current client site of the object.
//
// Parameters:
//
//      LPOLECLIENTSITE FAR* ppClientSite   - Out ptr in which to return the
//                                            client site.
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleObject::GetClientSite  ( LPOLECLIENTSITE FAR* ppClientSite)
{
     OutputDebugString("In COleObject::GetClientSite\r\n");
     *ppClientSite = m_lpObj->m_lpOleClientSite;
     return ResultFromScode( S_OK );
}
```

COleObject::SetMoniker   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::SetMoniker
//
// Purpose:
//
//      Used to set the objects moniker
//
// Parameters:
//
//      DWORD dwWhichMoniker    - Type of moniker being set
//
//      LPMONIKER pmk           - Pointer to the moniker
//
// Return Value:
//
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleObject::SetMoniker  ( DWORD dwWhichMoniker, LPMONIKER pmk)
{
     OutputDebugString("In COleObject::SetMoniker\r\n");

     LPMONIKER lpmk;

     if (! m_lpObj->GetOleClientSite())
          return ResultFromScode (E_FAIL);

     if (m_lpObj->GetOleClientSite()->GetMoniker (OLEGETMONIKER_ONLYIFTHERE,
OLEWHICHMK_OBJFULL, &lpmk) != NOERROR)
          return ResultFromScode (E_FAIL);


     if (m_lpObj->GetOleAdviseHolder())
          m_lpObj->GetOleAdviseHolder()->SendOnRename(lpmk);

     LPRUNNINGOBJECTTABLE lpRot;

     if (GetRunningObjectTable(0, &lpRot) == NOERROR)
          {
          if (m_lpObj->m_dwRegister)
               lpRot->Revoke(m_lpObj->m_dwRegister);

          lpRot->Register(0, m_lpObj, lpmk, &m_lpObj->m_dwRegister);
```

```
            lpRot->Release();
            }


     return ResultFromScode( S_OK );
};
```

COleObject::GetMoniker (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::GetMoniker
//
// Purpose:
//
////
// Parameters:
//
//      DWORD dwAssign          - Assignment for the moniker
//
//      DWORD dwWhichMoniker    - Which moniker to return
//
//      LPMONIKER FAR* ppmk     - An out ptr to return the moniker
//
// Return Value:
//
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleObject::GetMoniker  (  DWORD dwAssign, DWORD
dwWhichMoniker,
                                                           LPMONIKER FAR*
ppmk)
{
     OutputDebugString("In COleObject::GetMoniker\r\n");
     // need to NULL the out parameter
     *ppmk = NULL;

     return m_lpObj->GetOleClientSite()->GetMoniker
(OLEGETMONIKER_ONLYIFTHERE, OLEWHICHMK_OBJFULL, ppmk);
};
```

COleObject::InitFromData   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::InitFromData
//
// Purpose:
//
//      Initialize the object from the passed pDataObject.
//
// Parameters:
//
//      LPDATAOBJECT pDataObject    - Pointer to data transfer object
//                                    to be used in the initialization
//
//      BOOL fCreation              - TRUE if the object is currently being
//                                    created.
//
//      DWORD dwReserved            - Reserved
//
// Return Value:
//
//      S_FALSE
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      We don't support this functionality, so we will always return
//      error.
//
//********************************************************************

STDMETHODIMP COleObject::InitFromData  ( LPDATAOBJECT pDataObject,
                                                            BOOL fCreation,
                                                            DWORD
dwReserved)
{
     OutputDebugString("In COleObject::InitFromData\r\n");

     return ResultFromScode( S_FALSE );
};
```

COleObject::GetClipboardData   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::GetClipboardData
//
// Purpose:
//
//      Returns an IDataObject that is the same as doing an OleSetClipboard
//
// Parameters:
//
//      DWORD dwReserved                - Reserved
//
//      LPDATAOBJECT FAR* ppDataObject  - Out ptr for the Data Object.
//
// Return Value:
//
//      OLE_E_NOTSUPPORTED
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Support of this method is optional.
//
//********************************************************************

STDMETHODIMP COleObject::GetClipboardData  ( DWORD dwReserved,

LPDATAOBJECT FAR* ppDataObject)
{
     OutputDebugString("In COleObject::GetClipboardData\r\n");
     // NULL the out ptr
     *ppDataObject = NULL;
     return ResultFromScode( E_NOTIMPL );
};
```

COleObject::IsUpToDate   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::IsUpToDate
//
// Purpose:
//
//      Determines if an object is up to date
//
// Parameters:
//
//      None
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Our embedded object is always up to date.  This function is
//      particularly useful in linking situations.
//
//********************************************************************

STDMETHODIMP COleObject::IsUpToDate()
{
     OutputDebugString("In COleObject::IsUpToDate\r\n");
     return ResultFromScode( S_OK );
};
```

COleObject::GetUserClassID   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::GetUserClassID
//
// Purpose:
//
//      Returns the applications CLSID
//
// Parameters:
//
//      CLSID FAR* pClsid   - Out ptr to return the CLSID
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CPersistStorage::GetClassID IPS.CPP
//
// Comments:
//
//      This function is just delegated to IPS::GetClassID.
//
//********************************************************************

STDMETHODIMP COleObject::GetUserClassID  ( CLSID FAR* pClsid)
{
     OutputDebugString("In COleObject::GetUserClassID\r\n");

     m_lpObj->m_PersistStorage.GetClassID(pClsid);
     return ResultFromScode( S_OK );
};
```

COleObject::GetUserType (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::GetUserType
//
// Purpose:
//
//      Used to get a user presentable id for this object
//
// Parameters:
//
//      DWORD dwFormOfType      - The ID requested
//
//      LPSTR FAR* pszUserType  - Out ptr to return the string
//
// Return Value:
//
//      OLE_S_USEREG    - Use the reg db to get these entries.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//
//********************************************************************

//@@WTK WIN32, UNICODE
//STDMETHODIMP COleObject::GetUserType  ( DWORD dwFormOfType, LPSTR FAR*
pszUserType)
STDMETHODIMP COleObject::GetUserType  ( DWORD dwFormOfType, LPOLESTR FAR*
pszUserType)
{
     OutputDebugString("In COleObject::GetUserType\r\n");

     return ResultFromScode( OLE_S_USEREG );
};
```

COleObject::SetExtent   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::SetExtent
//
// Purpose:
//
//      Called to set the extent of the object.
//
// Parameters:
//
//      DWORD dwDrawAspect  - Aspect to have its size set
//
//      LPSIZEL lpsizel     - New size of the object.
//
// Return Value:
//
//      E_NOTIMPL   - This function is not curently implemented.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleObject::SetExtent  ( DWORD dwDrawAspect, LPSIZEL lpsizel)
{
     OutputDebugString("In COleObject::SetExtent\r\n");
     return ResultFromScode( E_NOTIMPL);
};
```

COleObject::EnumAdvise   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::EnumAdvise
//
// Purpose:
//
//      Returns an enumerate which enumerates the outstanding advises
//      associated with this OLE object.
//
// Parameters:
//
//      LPENUMSTATDATA FAR* ppenumAdvise - Out ptr in which to return
//                                         the enumerator.
//
// Return Value:
//
//      Passed on from IOleAdviseHolder::EnumAdvise.
//
// Function Calls:
//      Function                        Location
//
//      OutputDebugString               Windows API
//      IOleAdviseHolder::EnumAdvise    OLE
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleObject::EnumAdvise  ( LPENUMSTATDATA FAR* ppenumAdvise)
{
     OutputDebugString("In COleObject::EnumAdvise\r\n");
     // need to NULL the out parameter
     *ppenumAdvise = NULL;

     // pass on to the OLE Advise holder.
     return m_lpObj->m_lpOleAdviseHolder->EnumAdvise(ppenumAdvise);
};
```

COleObject::GetMiscStatus (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::GetMiscStatus
//
// Purpose:
//
//      Return status information about the object
//
// Parameters:
//
//      DWORD dwAspect          - Aspect interested in.
//
//      DWORD FAR* pdwStatus    - Out ptr in which to return the bits.
//
// Return Value:
//
//      CO_E_READREGDB  - Tell the library to use the reg DB.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP COleObject::GetMiscStatus  ( DWORD dwAspect, DWORD FAR*
pdwStatus)
{
     OutputDebugString("In COleObject::GetMiscStatus\r\n");
     // need to NULL the out parameter
     *pdwStatus = NULL;
     return ResultFromScode( OLE_S_USEREG );
};
```

COleObject::SetColorScheme (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::SetColorScheme
//
// Purpose:
//
//      Used to set the palette for the object to use.
//
// Parameters:
//
//      LPLOGPALETTE lpLogpal   - Pointer to the LOGPALETTE to be used.
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      This server ignores this method.
//
//********************************************************************

STDMETHODIMP COleObject::SetColorScheme  ( LPLOGPALETTE lpLogpal)
{
     OutputDebugString("In COleObject::SetColorScheme\r\n");
     return ResultFromScode( S_OK );
};
```

COleObject::OpenEdit   (SIMPSVR Sample)

```
//**********************************************************************
//
// COleObject::OpenEdit
//
// Purpose:
//
//      Used to Open the object into a seperate window.
//
// Parameters:
//
//      LPOLECLIENTSITE pActiveSite - Pointer to the Active clientsite.
//
// Return Value:
//
//      None.
//
// Function Calls:
//      Function                        Location
//
//      IOleClientSite::OnShowWindow    Container
//      ShowWindow                      Windows API
//      UpdateWindow                    Windows API
//      OutputDebugString               Windows API
//      CSimpSvrDoc::GethAppWnd         DOC.H
//      CSimpSvrDoc::GethHatchWnd       DOC.H
//
// Comments:
//
//
//********************************************************************

void COleObject::OpenEdit(LPOLECLIENTSITE pActiveSite)
{
   if (m_lpObj->GetOleClientSite())
        m_lpObj->GetOleClientSite()->ShowObject();


     m_fOpen = TRUE;

     // tell the site we are opening so the object can be hatched out.
     if (m_lpObj->GetOleClientSite())
           m_lpObj->GetOleClientSite()->OnShowWindow(TRUE);


     m_lpObj->m_lpDoc->ShowDocWnd();

     m_lpObj->m_lpDoc->HideHatchWnd();

     // Show app window.
     m_lpObj->m_lpDoc->GetApp()->ShowAppWnd();

     SetFocus(m_lpObj->m_lpDoc->GethAppWnd());
}
```

IPS.CPP (SIMPSVR Sample)

```
//**********************************************************************
// File name: ips.cpp
//
//    Implementation file for the CSimpSvrApp Class
//
// Functions:
//
//    See ips.h for a list of member functions.
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "obj.h"
#include "ips.h"
#include "app.h"
#include "doc.h"

//@@WTK WIN32, UNICODE
//#include <storage.h>

DEFINE_GUID(GUID_SIMPLE, 0xBCF6D4A0, 0xBE8C, 0x1068, 0xB6, 0xD4, 0x00, 0xDD,
0x01, 0x0C, 0x05, 0x09);
```

CPersistStorage::QueryInterface (SIMPSVR Sample)

```
//**********************************************************************
//
// CPersistStorage::QueryInterface
//
// Purpose:
//
//
// Parameters:
//
//      REFIID riid         -   Interface being queried for.
//
//      LPVOID FAR *ppvObj  -   Out pointer for the interface.
//
// Return Value:
//
//      S_OK            - Success
//      E_NOINTERFACE   - Failure
//
// Function Calls:
//      Function                    Location
//
//      CSimpSvrObj::QueryInterface OBJ.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP CPersistStorage::QueryInterface ( REFIID riid, LPVOID FAR*
ppvObj)
{
     OutputDebugString("In CPersistStorage::QueryInterface\r\n");
     // need to NULL the out parameter
     return m_lpObj->QueryInterface(riid, ppvObj);
};
```

CPersistStorage::AddRef   (SIMPSVR Sample)

```
//***********************************************************************
//
// CPersistStorage::AddRef
//
// Purpose:
//
//      Increments the reference count on CPersistStorage and the "object"
//      object.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The Reference count on the Object.
//
// Function Calls:
//      Function                    Location
//
//      OuputDebugString            Windows API
//      CSimpSvrObj::AddRef         OBJ.CPP
//
// Comments:
//
//
//*********************************************************************

STDMETHODIMP_(ULONG) CPersistStorage::AddRef ()
{
     OutputDebugString("In CPersistStorage::AddRef\r\n");
     ++m_nCount;
     return m_lpObj->AddRef();
};
```

CPersistStorage::Release (SIMPSVR Sample)

```
//**********************************************************************
//
// CPersistStorage::Release
//
// Purpose:
//
//      Decrements the reference count of CPersistStorage and the
//      "object" object.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      The new reference count
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CSimpSvrObj::Release        OBJ.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP_(ULONG) CPersistStorage::Release ()
{
     OutputDebugString("In CPersistStorage::Release\r\n");
     --m_nCount;
     return m_lpObj->Release();
};
```

CPersistStorage::InitNew (SIMPSVR Sample)

```
//**********************************************************************
//
// CPersistStorage::InitNew
//
// Purpose:
//
//      Used to give a new OLE object a ptr to its storage.
//
// Parameters:
//
//      LPSTORAGE pStg  - Pointer to the storage
//
// Return Value:
//
//
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      IStorage::Release           OLE
//      IStorage::AddRef            OLE
//
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP CPersistStorage::InitNew (LPSTORAGE pStg)
{
     OutputDebugString("In CPersistStorage::InitNew\r\n");

     // release any streams and storages that may be open
     ReleaseStreamsAndStorage();

     m_lpObj->m_lpStorage = pStg;

     // AddRef the new Storage
     if (m_lpObj->m_lpStorage)
           m_lpObj->m_lpStorage->AddRef();

     CreateStreams(m_lpObj->m_lpStorage);

     return ResultFromScode(S_OK);
}
```

CPersistStorage::GetClassID (SIMPSVR Sample)

```
//**********************************************************************
//
// CPersistStorage::GetClassID
//
// Purpose:
//
//      Returns the CLSID of this object.
//
// Parameters:
//
//      LPCLSID lpClassID   - Out ptr in which to return the CLSID
//
// Return Value:
//
//       S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP CPersistStorage::GetClassID  ( LPCLSID lpClassID)
{
     OutputDebugString("In CPersistStorage::GetClassID\r\n");

     *lpClassID = GUID_SIMPLE;

     return ResultFromScode( S_OK );
};
```

CPersistStorage::Save (SIMPSVR Sample)

```
//**********************************************************************
//
// CPersistStorage::Save
//
// Purpose:
//
//      Instructs the object to save itself into the storage.
//
// Parameters:
//
//      LPSTORAGE pStgSave  - Storage in which the object should be saved
//
//      BOOL fSameAsLoad    - TRUE if pStgSave is the same as the storage
//                            that the object was originally created with.
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CPersistStorage::InitNew    IPS.CPP
//      CSimpSvrObj::SaveToStorage  OBJ.CPP
//
//
// Comments:
//
//      A real app will want better error checking in this method.
//
//********************************************************************

STDMETHODIMP CPersistStorage::Save  ( LPSTORAGE pStgSave, BOOL fSameAsLoad)
{
     OutputDebugString("In CPersistStorage::Save\r\n");

     // save the data
     m_lpObj->SaveToStorage (pStgSave, fSameAsLoad);

     m_lpObj->m_fSaveWithSameAsLoad = fSameAsLoad;
     m_lpObj->m_fNoScribbleMode = TRUE;

     return ResultFromScode( S_OK );
};
```

CPersistStorage::SaveCompleted   (SIMPSVR Sample)

```
//**********************************************************************
//
// CPersistStorage::SaveCompleted
//
// Purpose:
//
//      Called when the container is finished saving the object
//
// Parameters:
//
//      LPSTORAGE pStgNew   - ptr to the new storage
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP CPersistStorage::SaveCompleted  ( LPSTORAGE pStgNew)
{
     OutputDebugString("In CPersistStorage::SaveCompleted\r\n");

     if (pStgNew)
           {
           ReleaseStreamsAndStorage();
           m_lpObj->m_lpStorage = pStgNew;
           m_lpObj->m_lpStorage->AddRef();
           OpenStreams(pStgNew);
           }


     /* OLE2NOTE: it is only legal to perform a Save or SaveAs operation
     **    on an embedded object. if the document is a file-based document
     **    then we can not be changed to a IStorage-base object.
     **
     **      fSameAsLoad   lpStgNew     Type of Save     Send OnSave
     **    ---------------------------------------------------------
     **         TRUE        NULL        SAVE             YES
     **         TRUE        ! NULL      SAVE *           YES
     **         FALSE       ! NULL      SAVE AS          YES
     **         FALSE       NULL        SAVE COPY AS     NO
     **
     **    * this is a strange case that is possible. it is inefficient
     **    for the caller; it would be better to pass lpStgNew==NULL for
     **    the Save operation.
     */

     if ( pStgNew || m_lpObj->m_fSaveWithSameAsLoad)
           {
           if (m_lpObj->m_fNoScribbleMode)
                 m_lpObj->GetOleAdviseHolder()->SendOnSave();  // normally
would clear a

             // dirty bit
           m_lpObj->m_fSaveWithSameAsLoad = FALSE;
           }

     m_lpObj->m_fNoScribbleMode = FALSE;

     return ResultFromScode( S_OK );
};
```

CPersistStorage::Load   (SIMPSVR Sample)

```
//**********************************************************************
//
// CPersistStorage::Load
//
// Purpose:
//
//      Instructs the object to be loaded from storage.
//
// Parameters:
//
//      LPSTORAGE pStg  - Ptr to the storage in which to be loaded
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                        Location
//
//      OutputDebugString               Windows API
//      CSimpSvrObj::LoadFromStorage    OBJ.CPP
//
//
// Comments:
//
//      A real app will want better error checking in this method.
//
//********************************************************************

STDMETHODIMP CPersistStorage::Load  ( LPSTORAGE pStg)
{
     OutputDebugString("In CPersistStorage::Load\r\n");

     // remember the storage
     if (m_lpObj->m_lpStorage)
           {
           m_lpObj->m_lpStorage->Release();
           m_lpObj->m_lpStorage = NULL;
           }

     m_lpObj->m_lpStorage = pStg;

     m_lpObj->m_lpStorage->AddRef();

     OpenStreams(m_lpObj->m_lpStorage);

     m_lpObj->LoadFromStorage();


     return ResultFromScode( S_OK );
};
```

CPersistStorage::IsDirty   (SIMPSVR Sample)

```
//**********************************************************************
//
// CPersistStorage::IsDirty
//
// Purpose:
//
//      Returns whether or not the object is dirty w/respect to its
//      Storage
//
// Parameters:
//
//      None
//
// Return Value:
//
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
//
// Comments:
//
//      This sample does not implement this function, although a
//      real application should.
//
//********************************************************************

STDMETHODIMP CPersistStorage::IsDirty()
{
     OutputDebugString("In CPersistStorage::IsDirty\r\n");
     return ResultFromScode( S_OK );
};
```

CPersistStorage::HandsOffStorage (SIMPSVR Sample)

```
//**********************************************************************
//
// CPersistStorage::HandsOffStorage
//
// Purpose:
//
//      Forces the object to release its handle to its storage.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      IStorage::Release           OLE
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP CPersistStorage::HandsOffStorage  ()
{
     OutputDebugString("In CPersistStorage::HandsOffStorage\r\n");

     ReleaseStreamsAndStorage();

     return ResultFromScode( S_OK );
};
```

CPersistStorage::CreateStreams (SIMPSVR Sample)

```
//**********************************************************************
//
// CPersistStorage::CreateStreams
//
// Purpose:
//
//      Creates the streams that are held open for the object's lifetime.
//
// Parameters:
//
//      LPSTORAGE lpStg -   Storage in which to create the streams
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      IStorage::Release           OLE
//      IStream::Release            OLE
//      IStorage::CreateStream      OLE
//
// Comments:
//
//
//********************************************************************

void CPersistStorage::CreateStreams(LPSTORAGE lpStg)
{
     if (m_lpObj->m_lpColorStm)
           m_lpObj->m_lpColorStm->Release();

     if (m_lpObj->m_lpSizeStm)
           m_lpObj->m_lpSizeStm->Release();

           // create a stream to save the colors
     //@@WTK WIN32, UNICODE
     //lpStg->CreateStream ( "RGB",
     lpStg->CreateStream ( OLESTR("RGB"),
                                       STGM_READWRITE | STGM_SHARE_EXCLUSIVE
| STGM_CREATE,
                                       0,
                                       0,
                                       &m_lpObj->m_lpColorStm);

     // create a stream to save the size
     //@@WTK WIN32, UNICODE
     //lpStg->CreateStream ( "size",
     lpStg->CreateStream ( OLESTR("size"),
                                       STGM_READWRITE | STGM_SHARE_EXCLUSIVE
| STGM_CREATE,
                                       0,
                                       0,
                                       &m_lpObj->m_lpSizeStm);
}
```

CPersistStorage::OpenStreams (SIMPSVR Sample)

```
//**********************************************************************
//
// CPersistStorage::OpenStreams
//
// Purpose:
//
//      Opens the streams in a storage.
//
// Parameters:
//
//      LPSTORAGE lpStg -   Storage in which to open the streams.
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      IStorage::Release           OLE
//
// Comments:
//
//
//********************************************************************

void CPersistStorage::OpenStreams(LPSTORAGE lpStg)
{
     if (m_lpObj->m_lpColorStm)
           m_lpObj->m_lpColorStm->Release();

     if (m_lpObj->m_lpSizeStm)
           m_lpObj->m_lpSizeStm->Release();

           // open the color stream
     //@@WTK WIN32, UNICODE
     //lpStg->OpenStream ( "RGB",
     lpStg->OpenStream ( OLESTR("RGB"),
                                      0,
                                      STGM_READWRITE | STGM_SHARE_EXCLUSIVE,
                                      0,
                                      &m_lpObj->m_lpColorStm);

     // open the color stream
     //@@WTK WIN32, UNICODE
     //lpStg->OpenStream ( "size",
     lpStg->OpenStream ( OLESTR("size"),
                                      0,
                                      STGM_READWRITE | STGM_SHARE_EXCLUSIVE,
                                      0,
                                      &m_lpObj->m_lpSizeStm);
}
```

CPersistStorage::ReleaseStreamsAndStorage   (SIMPSVR Sample)

```
//***********************************************************************
//
// CPersistStorage::ReleaseStreamsAndStorage
//
// Purpose:
//
//      Releases the stream and storage ptrs
//
// Parameters:
//
//      None
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      IStorage::Release           OLE
//
// Comments:
//
//
//*********************************************************************

void CPersistStorage::ReleaseStreamsAndStorage()
{
     if (m_lpObj->m_lpColorStm)
           {
           m_lpObj->m_lpColorStm->Release();
           m_lpObj->m_lpColorStm = NULL;
           }

     if (m_lpObj->m_lpSizeStm)
           {
           m_lpObj->m_lpSizeStm->Release();
           m_lpObj->m_lpSizeStm = NULL;
           }

     if (m_lpObj->m_lpStorage)
           {
           m_lpObj->m_lpStorage->Release();
           m_lpObj->m_lpStorage = NULL;
           }
}
```

CPersistStorage::CreateStreams (SIMPSVR Sample)

```
//**********************************************************************
//
// CPersistStorage::CreateStreams
//
// Purpose:
//
//      Creates temporary streams in a storage.
//
// Parameters:
//
//      LPSTORAGE lpStg                 - Pointer to the storage
//
//      LPSTREAM FAR* lplpTempColor     - Color Stream
//
//      LPSTREAM FAR* lplpTempSize      - Size Stream
//
// Return Value:
//
//      S_OK
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      IStorage::Release           OLE
//
// Comments:
//
//
//********************************************************************

void CPersistStorage::CreateStreams(LPSTORAGE lpStg, LPSTREAM FAR*
lplpTempColor,LPSTREAM FAR* lplpTempSize)
{
           // create a stream to save the colors
      //@@WTK WIN32, UNICODE
      //lpStg->CreateStream ( "RGB",
      lpStg->CreateStream ( OLESTR("RGB"),
                                      STGM_READWRITE | STGM_SHARE_EXCLUSIVE
| STGM_CREATE,
                                      0,
                                      0,
                                      lplpTempColor);

     // create a stream to save the size
     //@@WTK WIN32, UNICODE
     //lpStg->CreateStream ( "size",
     lpStg->CreateStream ( OLESTR("size"),
                                      STGM_READWRITE | STGM_SHARE_EXCLUSIVE
| STGM_CREATE,
                                      0,
                                      0,
                                                            lplpTempSize);

    }
```

OBJ.CPP   (SIMPSVR Sample)

```
//**********************************************************************
// File name: obj.cpp
//
//    Implementation file for the CSimpSvrApp Class
//
// Functions:
//
//    See obj.h for a list of member functions.
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "obj.h"
#include "ioo.h"
#include "ido.h"
#include "ips.h"
#include "icf.h"
#include "ioipao.h"
#include "ioipo.h"
#include "app.h"
#include "doc.h"
```

CSimpSvrObj::QueryInterface (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrObj::QueryInterface
//
// Purpose:
//
//      Used for interface negotiation at the "Object" level.
//
// Parameters:
//
//      REFIID riid          -   A reference to the interface that is
//                               being queried.
//
//      LPVOID FAR* ppvObj   -   An out parameter to return a pointer to
//                               the interface.
//
// Return Value:
//
//      S_OK           -   The interface is supported.
//      E_NOINTERFACE -   The interface is not supported
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      ResultFromScode             OLE API
//      IUnknown::AddRef            OBJ.CPP, IOO.CPP, IDO.CPP, IPS.CPP
//                                  IOIPO.CPP, IOIPAO.CPP
//
// Comments:
//
//
//********************************************************************

STDMETHODIMP CSimpSvrObj::QueryInterface ( REFIID riid, LPVOID FAR* ppvObj)
{
     OutputDebugString("In CSimpSvrObj::QueryInterface\r\n");

     SCODE sc = S_OK;

     if (riid == IID_IUnknown)
           *ppvObj = this;
     else if (riid == IID_IOleObject)
           *ppvObj = &m_OleObject;
     else if (riid == IID_IDataObject)
           *ppvObj = &m_DataObject;
     else if ( (riid == IID_IPersistStorage) || (riid == IID_IPersist) )
           *ppvObj = &m_PersistStorage;
     else if (riid == IID_IOleInPlaceObject)
           *ppvObj = &m_OleInPlaceObject;
     else if (riid == IID_IOleInPlaceActiveObject)
           *ppvObj = &m_OleInPlaceActiveObject;
     else if (riid == IID_IExternalConnection)
            *ppvObj = &m_ExternalConnection;
     else
           {
           *ppvObj = NULL;
           sc = E_NOINTERFACE;
           }

     if (*ppvObj)
           ((LPUNKNOWN)*ppvObj)->AddRef();

     return ResultFromScode( sc );
};
```

CSimpSvrObj::AddRef   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrObj::AddRef
//
// Purpose:
//
//      Adds to the reference count at the Object level.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the Object.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Due to the reference counting model that is used in this
//      implementation, this reference count is the sum of the
//      reference counts on all interfaces
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpSvrObj::AddRef ()
{
     OutputDebugString("In CSimpSvrObj::AddRef\r\n");

     m_lpDoc->AddRef();

     return ++m_nCount;
};
```

CSimpSvrObj::Release   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrObj::Release
//
// Purpose:
//
//      Decrements the reference count at this level
//
// Parameters:
//
//      None
//
// Return Value:
//
//      ULONG   -   The new reference count of the object.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//
// Comments:
//
//      Due to the reference counting model that is used in this
//      implementation, this reference count is the sum of the
//      reference counts on all interfaces
//
//********************************************************************

STDMETHODIMP_(ULONG) CSimpSvrObj::Release ()
{
     OutputDebugString("In CSimpSvrObj::Release\r\n");

     m_lpDoc->Release();

     if (--m_nCount == 0) {
           delete this;
        return 0;
    }

     return m_nCount;
};
```

CSimpSvrObj::CSimpSvrObj   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrObj::CSimpSvrObj
//
// Purpose:
//
//      Constructor for CSimpSvrObj
//
// Parameters:
//
//      CSimpSvrDoc FAR * lpSimpSvrDoc - ptr to the doc object
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//
// Comments:
//
//
//********************************************************************
#pragma warning (disable : 4355) // "this" used in base initializer list
warning.  This
                                                // can be disabled because
we are not using "this" in
                                                // the constructor for these
objects, rather we are
                                                // just storing it for
future use...
CSimpSvrObj::CSimpSvrObj(CSimpSvrDoc FAR * lpSimpSvrDoc) :
m_OleObject(this),

              m_DataObject(this),

              m_PersistStorage(this),

              m_OleInPlaceActiveObject(this),

              m_OleInPlaceObject(this),

              m_ExternalConnection(this)
#pragma warning (default : 4355) // Turn the warning back on

{
     m_lpDoc = lpSimpSvrDoc;
     m_nCount = 0;
     m_fInPlaceActive = FALSE;
     m_fInPlaceVisible = FALSE;
     m_fUIActive = FALSE;
```

```
    m_hmenuShared = NULL;
    m_hOleMenu = NULL;

    m_dwRegister = 0;

    m_lpFrame = NULL;
    m_lpCntrDoc = NULL;

    m_lpStorage = NULL;
    m_lpColorStm = NULL;
    m_lpSizeStm = NULL;
    m_lpOleClientSite = NULL;
    m_lpOleAdviseHolder = NULL;
    m_lpDataAdviseHolder = NULL;
    m_lpIPSite = NULL;

    m_red = 128;
    m_green = 0;
    m_blue = 0;

    m_size.x = 100;
    m_size.y = 100;

    m_xOffset = 0;
    m_yOffset = 0;

    m_scale = (float) 1.0;

    m_fSaveWithSameAsLoad = FALSE;
    m_fNoScribbleMode = FALSE;

}
```

CSimpSvrObj::~CSimpSvrObj (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrObj::~CSimpSvrObj
//
// Purpose:
//
//      Destructor for CSimpSvrObj
//
// Parameters:
//
//      None
//
// Return Value:
//
//
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      PostMessage                 Windows API
//      CSimpSvrDoc::GetApp         DOC.H
//      CSimpSvrDoc::GethAppWnd     DOC.H
//      CSimpSvrDoc::ClearObj       DOC.H
//      CSimpSvrApp::IsStartedByOle APP.CPP
//
// Comments:
//
//
//********************************************************************

CSimpSvrObj::~CSimpSvrObj()
{
     OutputDebugString("In CSimpSvrObj's Destructor \r\n");

     // if we were started by ole, post ourselves a close message
     if (m_lpDoc->GetApp()->IsStartedByOle())
           PostMessage(m_lpDoc->GethAppWnd(), WM_SYSCOMMAND, SC_CLOSE, 0L);

     // clear the OBJ ptr in the doc class
     m_lpDoc->ClearObj();

}
```

CSimpSvrObj::Draw   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrObj::Draw
//
// Purpose:
//
//      Draws the object into an arbitrary DC
//
// Parameters:
//
//      HDC hDC - DC to draw into
//
// Return Value:
//
//      NONE
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      CreateBrushIndirect         Windows API
//      SelectObject                Windows API
//      Rectangle                   Windows API
//      DeleteObject                Windows API
//
// Comments:
//
//
//********************************************************************

void CSimpSvrObj::Draw (HDC hDC, BOOL m_fMeta)
{
     LOGBRUSH lb;

     OutputDebugString("In CSimpSvrObj::Draw\r\n");

     char szBuffer[255];
     wsprintf(szBuffer,"Drawing Scale %3d\r\n",m_scale);
     OutputDebugString(szBuffer);

     if (!m_fMeta)
           {
           SetMapMode(hDC, MM_ANISOTROPIC);
           //@@WTK WIN32, UNICODE
           //SetWindowOrg(hDC, (int)(m_xOffset/m_scale),
(int)(m_yOffset/m_scale));
           SetWindowOrgEx(hDC, (int)(m_xOffset/m_scale),
                 (int)(m_yOffset/m_scale), NULL);
           //@@WTK WIN32, UNICODE
           //SetWindowExt(hDC, m_size.x, m_size.y);
           SetWindowExtEx(hDC, m_size.x, m_size.y, NULL);
           //@@WTK WIN32, UNICODE
           //SetViewportExt(hDC, (int)(m_size.x*m_scale), (int)
(m_size.y*m_scale));
           SetViewportExtEx(hDC, (int)(m_size.x*m_scale),
                 (int)(m_size.y*m_scale), NULL);
           }

     // fill out a LOGBRUSH
     lb.lbStyle = BS_SOLID;
     lb.lbColor = RGB(m_red, m_green, m_blue);
     lb.lbHatch = 0;

     // create the brush
     HBRUSH hBrush = CreateBrushIndirect(&lb);

     // select the brush
     HBRUSH hOldBrush = SelectObject(hDC, hBrush);
     HPEN hPen = CreatePen(PS_INSIDEFRAME, 6, RGB(0, 0, 0));

     HPEN hOldPen = SelectObject(hDC, hPen);

     // draw the rectangle
     Rectangle (hDC, 0, 0, m_size.x, m_size.y);

     // restore the pen
     hPen = SelectObject(hDC, hOldPen);

     // free the pen
     DeleteObject(hPen);

     // restore the old brush
     hBrush = SelectObject(hDC, hOldBrush);

     // free the brush
     DeleteObject(hBrush);
}
```

CSimpSvrObj::GetMetaFilePict   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrObj::GetMetaFilePict
//
// Purpose:
//
//      Returns a handle to a metafile representation of the object.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      Handle to the metafile.
//
// Function Calls:
//      Function                        Location
//
//      OutputDebugString               Windows API
//      GlobalAlloc                     Windows API
//      GlobalLock                      Windows API
//      SetWindowOrg                    Windows API
//      SetWindowExt                    Windows API
//      CreateMetaFile                  Windows API
//      CloseMetaFile                   Windows API
//      GlobalUnlock                    Windows API
//      XformWidthInPixelsToHimetric    OLE2UI
//      XformHeightInPixelsToHimetric   OLE2UI
//      CSimpSvrObj::Draw               OBJ.CPP
//
// Comments:
//
//
//********************************************************************

HANDLE CSimpSvrObj::GetMetaFilePict()
{
     HANDLE hMFP;
     METAFILEPICT FAR * lpMFP;
     POINT pt;

     OutputDebugString("In CSimpSvrObj::GetMetaFilePict\r\n");

     // allocate the memory for the METAFILEPICT structure
     hMFP = GlobalAlloc (GMEM_SHARE | GHND, sizeof (METAFILEPICT) );
     lpMFP = (METAFILEPICT FAR*) GlobalLock(hMFP);

     // get the size of the object in HIMETRIC
     pt.x = XformWidthInPixelsToHimetric(NULL, m_size.x);
     pt.y = XformHeightInPixelsToHimetric(NULL, m_size.y);
     // fill out the METAFILEPICT structure
     lpMFP->mm = MM_ANISOTROPIC;
     lpMFP->xExt = pt.x;
     lpMFP->yExt = pt.y;

     // Create the metafile
     HDC hDC = CreateMetaFile(NULL);

     //@@WTK WIN32, UNICODE
     //SetWindowOrg (hDC, 0, 0);
     SetWindowOrgEx (hDC, 0, 0, NULL);
     //@@WTK WIN32, UNICODE
     //SetWindowExt (hDC, m_size.x,
     SetWindowExtEx (hDC, m_size.x,
                                m_size.y, NULL);

     Draw(hDC);

     lpMFP->hMF = CloseMetaFile(hDC);

     // unlock the metafilepict
     GlobalUnlock(hMFP);

     return hMFP;
}
```

CSimpSvrObj::SaveToStorage (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrObj::SaveToStorage
//
// Purpose:
//
//      Saves the object to the passed storage
//
// Parameters:
//
//      LPSTORAGE lpStg - Storage in which to save the object
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      IStorage::CreateStream      OLE
//      IStream::Write              OLE
//      IStream::Release            OLE
//
// Comments:
//
//      A real app will want to do better error checking / returning
//
//********************************************************************

void CSimpSvrObj::SaveToStorage (LPSTORAGE lpStg, BOOL fSameAsLoad)
{
     OutputDebugString("In CSimpSvrObj::SaveToStorage\r\n");

     LPSTREAM lpTempColor, lpTempSize;

     if (!fSameAsLoad)
           m_PersistStorage.CreateStreams( lpStg, &lpTempColor,
&lpTempSize);
     else
           {
           lpTempColor = m_lpColorStm;
           lpTempColor->AddRef();
           lpTempSize = m_lpSizeStm;
           lpTempSize->AddRef();
           }

     ULARGE_INTEGER uli;

     uli.LowPart = 0;
     uli.HighPart = 0;
     lpTempColor->SetSize(uli);
     lpTempSize->SetSize(uli);

     LARGE_INTEGER li;

     li.LowPart = 0;
     li.HighPart = 0;

     lpTempColor->Seek(li, STREAM_SEEK_SET, NULL);
     lpTempSize->Seek(li, STREAM_SEEK_SET, NULL);

     // write the colors to the stream
     lpTempColor->Write(&m_red, sizeof(m_red), NULL);
     lpTempColor->Write(&m_green, sizeof(m_green), NULL);
     lpTempColor->Write(&m_blue, sizeof(m_blue), NULL);

     // write the size to the stream
     lpTempSize->Write(&m_size, sizeof(m_size), NULL);

     lpTempColor->Release();
     lpTempSize->Release();
}
```

CSimpSvrObj::LoadFromStorage   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrObj::LoadFromStorage
//
// Purpose:
//
//      Loads the object from the passed storage
//
// Parameters:
//
//      LPSTORAGE lpStg     - Storage in which to load the object from
//
// Return Value:
//
//      None.
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString           Windows API
//      IStorage::OpenStream        OLE
//      IStream::Read               OLE
//      IStream::Release            OLE
//
// Comments:
//
//
//********************************************************************

void CSimpSvrObj::LoadFromStorage ()
{
     OutputDebugString("In CSimpSvrObj::LoadFromStorage\r\n");

     // Read the colors
     m_lpColorStm->Read(&m_red, sizeof(m_red), NULL);
     m_lpColorStm->Read(&m_green, sizeof(m_green), NULL);
     m_lpColorStm->Read(&m_blue, sizeof(m_blue), NULL);

     // read the size
     m_lpSizeStm->Read(&m_size, sizeof(m_size), NULL);

}
```

CSimpSvrObj::DoInPlaceActivate   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrObj::DoInPlaceActivate
//
// Purpose:
//
//      Does the inplace activation for the object
//
// Parameters:
//
//      LONG lVerb  - Verb that caused this function to be called
//
// Return Value:
//
//      TRUE/FALSE depending on success or failure.
//
// Function Calls:
//      Function                                Location
//
//      IOleClientSite::QueryInterface          Container
//      IOleClientSite::ShowObject              Container
//      IOleInPlaceSite::CanInPlaceActivate     Container
//      IOleInPlaceSite::Release                Container
//      IOleInPlaceSite::OnInPlaceActivate      Container
//      IOleInPlaceSite::GetWindow              Container
//      IOleInPlaceSite::GetWindowContext       Container
//      IOleInPlaceSite::OnUIActivate           Container
//      IOleInPlaceSite::Release                Container
//      IOleInPlaceFrame::SetActiveObject       Container
//      IOleInPlaceUIWindow::SetActiveObject    Container
//      OutputDebugString                       Windows API
//      ShowWindow                              Windows API
//      SetParent                               Windows API
//      IntersectRect                           Windows API
//      OffsetRect                              Windows API
//      MoveWindow                              Windows API
//      CopyRect                                Windows API
//      SetFocus                                Windows API
//      SetHatchWindowSize                      OLE2UI
//      CSimpSvrObj::AssembleMenus              OBJ.CPP
//      CSimpSvrObj::AddFrameLevelUI            OBJ.CPP
//
//
// Comments:
//
//      Be sure to read TECHNOTES.WRI included with the OLE SDK
//      for details on implementing inplace activation.
//
//********************************************************************

BOOL CSimpSvrObj::DoInPlaceActivate (LONG lVerb)
{
     BOOL retval = FALSE;
     RECT posRect, clipRect;


     OutputDebugString("In CSimpSvrObj::DoInPlaceActivate\r\n");

     // if not currently in place active
     if (!m_fInPlaceActive)
           {
           // get the inplace site
           if (m_lpOleClientSite->QueryInterface(IID_IOleInPlaceSite,
                                                                    (LPVOID
FAR *)&m_lpIPSite) != NOERROR)
                 goto error;


           // if the inplace site could not be obtained, or refuses to
inplace
           // activate then goto error.
           if (m_lpIPSite == NULL || m_lpIPSite->CanInPlaceActivate() !=
NOERROR)
                 {
                 if (m_lpIPSite)
                       m_lpIPSite->Release();
                 m_lpIPSite == NULL;
                 goto error;
                 }

           // tell the site that we are activating.
           m_lpIPSite->OnInPlaceActivate();
           m_fInPlaceActive = TRUE;
           }

     // if not currently inplace visibl
     if (!m_fInPlaceVisible)
           {
           m_fInPlaceVisible = TRUE;

           // get the window handle of the site
           m_lpIPSite->GetWindow(&m_hWndParent);

           // get window context from the container
           m_lpIPSite->GetWindowContext ( &m_lpFrame,
                                                       &m_lpCntrDoc,
                                                       &posRect,
                                                       &clipRect,
                                                       &m_FrameInfo);

           // show the hatch window
           m_lpDoc->ShowHatchWnd();

           // Set the parenting
           SetParent (m_lpDoc->GethHatchWnd(), m_hWndParent);
           SetParent (m_lpDoc->GethDocWnd(), m_lpDoc->GethHatchWnd());
```

```
            // tell the client site to show the object
            m_lpOleClientSite->ShowObject();

            RECT resRect;

            // figure out the "real" size of the object
            IntersectRect(&resRect, &posRect, &clipRect);
            CopyRect(&m_posRect, &posRect);

            POINT pt;

            // adjust our hatch window size
            SetHatchWindowSize ( m_lpDoc->GethHatchWnd(),
                                        &resRect,
                                        &posRect,
                                        &pt);


            // calculate the actual object rect inside the hatchwnd.
            OffsetRect (&resRect, pt.x, pt.y);

            // move the object window
            MoveWindow(m_lpDoc->GethDocWnd(),
                            resRect.left,
                            resRect.top,
                            resRect.right - resRect.left,
                            resRect.bottom - resRect.top,
                            FALSE);

            // create the combined window
            AssembleMenus();
            }

      // if not UIActive
      if (!m_fUIActive)
            {
            m_fUIActive = TRUE;

            // tell the inplace site that we are activating
            m_lpIPSite->OnUIActivate();

            // set the focus to our object window
            SetFocus(m_lpDoc->GethDocWnd());

            // set the active object on the frame
            //@@WTK WIN32, UNICODE
            //m_lpFrame->SetActiveObject(&m_OleInPlaceActiveObject, "Simple
OLE 2.0 Server");
            m_lpFrame->SetActiveObject(&m_OleInPlaceActiveObject,
OLESTR("Simple OLE 2.0 Server"));

            // set the active object on the Doc, if available.
            if (m_lpCntrDoc)
                  //@@WTK WIN32, UNICODE
                  //m_lpCntrDoc->SetActiveObject(&m_OleInPlaceActiveObject,
"Simple OLE 2.0 Server");
                  m_lpCntrDoc->SetActiveObject(&m_OleInPlaceActiveObject,
OLESTR("Simple OLE 2.0 Server"));

            // add the frame level UI.
            AddFrameLevelUI();
            }

      retval = TRUE;
error:
      return retval;
}
```

CSimpSvrObj::AssembleMenus   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrObj::AssembleMenus
//
// Purpose:
//
//      Creates the combined menus used during inplace activation.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                        Location
//
//      OutputDebugString               Windows API
//      CreateMenu                      Windows API
//      IOleInPlaceFrame::InsertMenus   Container
//      InsertMenu                      Windows API
//      DestroyMenu                     Windows API
//      OleCreateMenuDescriptor         OLE API
//
// Comments:
//
//
//********************************************************************

void CSimpSvrObj::AssembleMenus()
{
     OutputDebugString("In CSimpSvrObj::AssembleMenus\r\n");
     OLEMENUGROUPWIDTHS menugroupwidths;

     m_hmenuShared = NULL;

     //  Create the menu resource
     m_hmenuShared = CreateMenu();

     // have the contaner insert its menus
     if (m_lpFrame->InsertMenus (m_hmenuShared, &menugroupwidths) ==
NOERROR)
            {
            int nFirstGroup = (int) menugroupwidths.width[0];

            // insert the server menus
            //@@WTK WIN32, UNICODE
            //InsertMenu( m_hmenuShared, nFirstGroup, MF_BYPOSITION |
MF_POPUP, m_lpDoc->GetApp()->GetColorMenu(), "&Color");
            InsertMenu( m_hmenuShared, nFirstGroup, MF_BYPOSITION | MF_POPUP,
                    (UINT)m_lpDoc->GetApp()->GetColorMenu(), "&Color");
            menugroupwidths.width[1] = 1;
            menugroupwidths.width[3] = 0;
            menugroupwidths.width[5] = 0;
            }
      else
            {
            // Destroy the menu resource
            DestroyMenu(m_hmenuShared);
            m_hmenuShared = NULL;
            }

      // tell OLE to create the menu descriptor
      m_hOleMenu = OleCreateMenuDescriptor(m_hmenuShared, &menugroupwidths);
}
```

CSimpSvrObj::AddFrameLevelUI (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrObj::AddFrameLevelUI
//
// Purpose:
//
//      Adds the Frame level user interface
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString                   Windows API
//      IOleInPlaceFrame::SetMenu           Container
//      IOleInPlaceFrame::SetBorderSpace    Container
//      IOleInPlaceUIWindow::SetBorderSpace Container
//      CSimpSvrDoc::GethDocWnd             DOC.H
//
// Comments:
//
//
//********************************************************************

void CSimpSvrObj::AddFrameLevelUI()
{
     OutputDebugString("In CSimpSvrObj::AddFrameLevelUI\r\n");

     // add the combined menu
     m_lpFrame->SetMenu(m_hmenuShared, m_hOleMenu, m_lpDoc->GethDocWnd());

     // do hatched border
     SetParent (m_lpDoc->GethHatchWnd(), m_hWndParent);
     SetParent (m_lpDoc->GethDocWnd(), m_lpDoc->GethHatchWnd());

     // set the border space.  Normally we would negotiate for toolbar
     // space at this point.  Since this server doesn't have a toolbar,
     // this isn't needed...
     if (m_lpFrame)
           m_lpFrame->SetBorderSpace(NULL);

     if (m_lpCntrDoc)
           m_lpCntrDoc->SetBorderSpace(NULL);
}
```

CSimpSvrObj::DoInPlaceHide   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrObj::DoInPlaceHide
//
// Purpose:
//
//      Hides the object while inplace actvie
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                        Location
//
//      OutputDebugString               Windows API
//      SetParent                       Windows API
//      CSimpSvrDoc::GethDocWnd         DOC.H
//      CSimpSvrDoc::GethAppWnd         DOC.H
//      CSimpSvrDoc::GethHatchWnd       DOC.H
//      CSimpSvrObj::DisassembleMenus   OBJ.CPP
//      IOleInPlaceFrame::Release       Container
//      IOleInPlaceUIWindow::Release    Container
//
//
// Comments:
//
//      Be sure to read TECHNOTES.WRI included with the OLE SDK
//      for details on implementing inplace activation.
//
//********************************************************************

void CSimpSvrObj::DoInPlaceHide()
{
     OutputDebugString("In CSimpSvrObj::DoInPlaceHide\r\n");

     // if we aren't inplace visible, then this routine is a NOP,
     if (!m_fInPlaceVisible)
           return;

     m_fInPlaceVisible = FALSE;

     // change the parenting
     SetParent (m_lpDoc->GethDocWnd(), m_lpDoc->GethAppWnd());
     SetParent (m_lpDoc->GethHatchWnd(),m_lpDoc->GethDocWnd());

     // rip down the combined menus
     DisassembleMenus();
     // release the inplace frame
     m_lpFrame->Release();

     m_lpFrame = NULL;  // only holding one ref. to frame.

     // release the UIWindow if it is there.
     if (m_lpCntrDoc)
           m_lpCntrDoc->Release();

     m_lpCntrDoc = NULL;

}
```

CSimpSvrObj::DisassembleMenus (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrObj::DisassembleMenus
//
// Purpose:
//
//      Disassembles the combined menus used in inplace activation
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                    Location
//
//      OutputDebugString               Windows API
//      OleDestroyMenuDescriptor        OLE API
//      RemoveMenu                      Windows API
//      IOleInPlaceFrame::RemoveMenus   Container
//      DestroyMenu                     Windows API
//
// Comments:
//
//      Be sure to read TECHNOTES.WRI included with the OLE SDK
//      for details on implementing inplace activation.
//
//********************************************************************

void CSimpSvrObj::DisassembleMenus()
{
     // destroy the menu descriptor
     OleDestroyMenuDescriptor(m_hOleMenu);

     if (m_hmenuShared)
           {
           // remove the menus that we added
           RemoveMenu( m_hmenuShared, 1, MF_BYPOSITION);

           // have the container remove its menus
           m_lpFrame->RemoveMenus(m_hmenuShared);

           // Destroy the menu resource
           DestroyMenu(m_hmenuShared);

           m_hmenuShared = NULL;
           }
}
```

CSimpSvrObj::SendOnDataChange (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrObj::SendOnDataChange
//
// Purpose:
//
//      Uses the data advise holder to send a data change, then updates
//      the ROT to note the time of change.
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                                Location
//
//      IDataAdviseHolder::SendOnDataChange     OLE API
//      GetRunningObjectTable                   OLE API
//      CoFileTimeNow                           OLE API
//      IRunningObjectTable::NoteChangeTime     OLE API
//
// Comments:
//
//
//********************************************************************

void CSimpSvrObj::SendOnDataChange()
{
     if (m_lpDataAdviseHolder)
           m_lpDataAdviseHolder-
>SendOnDataChange( (LPDATAOBJECT)&m_DataObject, 0, 0);

     LPRUNNINGOBJECTTABLE lpRot;

     GetRunningObjectTable(0, &lpRot);

     if ( lpRot && m_dwRegister)
           {
           FILETIME ft;
           CoFileTimeNow(&ft);

           lpRot->NoteChangeTime(m_dwRegister, &ft);
           lpRot->Release();
           }
}
```

CSimpSvrObj::DeactivateUI   (SIMPSVR Sample)

```
//**********************************************************************
//
// CSimpSvrObj::DeactivateUI
//
// Purpose:
//
//      Breaks down the inplace ui
//
// Parameters:
//
//      None
//
// Return Value:
//
//      None
//
// Function Calls:
//      Function                                Location
//
//      SetParent                               Windows API
//      IOleInPlaceUIWindow::SetActiveObject    Container
//      IOleInPlaceFrame::SetActiveObject       Container
//      IOleInPlaceSite::UIDeactivate           Container
//
// Comments:
//
//
//********************************************************************

void CSimpSvrObj::DeactivateUI()
{
     // if not UI active, or no pointer to IOleInPlaceFrame, then
     // return NOERROR
     if (!(m_fUIActive || m_lpFrame))
           return;
     else
           {
           m_fUIActive = FALSE;

           // remove hatching
           SetParent (m_lpDoc->GethDocWnd(), m_lpDoc->GethAppWnd());
           SetParent (m_lpDoc->GethHatchWnd(),m_lpDoc->GethDocWnd());

           // if in an MDI container, call SetActiveObject on the DOC.
           if (m_lpCntrDoc)
                 m_lpCntrDoc->SetActiveObject(NULL, NULL);

           m_lpFrame->SetActiveObject(NULL, NULL);

           // tell the container that our UI is going away.
           if (m_lpIPSite)
                 m_lpIPSite->OnUIDeactivate(FALSE);
```

```
                    }
}
```

SIMPSVR.CPP   (SIMPSVR Sample)

```
//**********************************************************************
// File name: simpsvr.cpp
//
//      Main source file for the simple OLE 2.0 server
//
// Functions:
//
//      WinMain         - Program entry point
//      MainWndProc     - Processes messages for the frame window
//      About           - Processes messages for the about dialog
//      DocWndProc      - Processes messages for the doc window
//
// Copyright (c) 1993 Microsoft Corporation. All rights reserved.
//**********************************************************************

#include "pre.h"
#include "obj.h"
#include "app.h"
#include "doc.h"
#include "icf.h"

// This line is needed for the debug utilities in OLE2UI
extern "C" {
        OLEDBGDATA_MAIN("SIMPSVR")
}

CSimpSvrApp FAR * lpCSimpSvrApp;
CClassFactory FAR * lpClassFactory;
```

WinMain   (SIMPSVR Sample)

```
//**********************************************************************
//
// WinMain
//
// Purpose:
//
//      Program entry point
//
// Parameters:
//
//      HANDLE hInstance        - Instance handle for this instance
//
//      HANDLE hPrevInstance    - Instance handle for the last instance
//
//      LPSTR lpCmdLine         - Pointer to the command line
//
//      int nCmdShow            - Window State
//
// Return Value:
//
//      msg.wParam
//
// Function Calls:
//      Function                        Location
//
//      CSimpSvrApp::CSimpSvrApp          APP.CPP
//      CSimpSvrApp::fInitApplication    APP.CPP
//      CSimpSvrApp::fInitInstance       APP.CPP
//      CSimpSvrApp::HandleAccelerators  APP.CPP
//      CSimpSvrApp::~CSimpSvrApp         APP.CPP
//      OleUIInitialize                 OLE2UI
//      OleUIUninitialize               OLE2UI
//      GetMessage                      Windows API
//      TranslateMessage                Windows API
//      DispatchMessage                 Windows API
//
// Comments:
//
//********************************************************************

int PASCAL WinMain(HANDLE hInstance,HANDLE hPrevInstance,LPSTR lpCmdLine,int
nCmdShow)

{
        MSG msg;

        // recommended size for OLE apps
        SetMessageQueue(96);

        lpCSimpSvrApp = new CSimpSvrApp;
        lpCSimpSvrApp->AddRef();      // need the app ref. count at 1 to
hold the
                                                                  // app
alive.

        lpCSimpSvrApp->ParseCmdLine(lpCmdLine);

        // app initialization
        if (!hPrevInstance)
                if (!lpCSimpSvrApp->fInitApplication(hInstance))
                        return (FALSE);

        // instance initialization
        if (!lpCSimpSvrApp->fInitInstance(hInstance, nCmdShow,
lpClassFactory))
                return (FALSE);

        /* Initialization required for OLE 2 UI library.  This call is
        **    needed ONLY if we are using the static link version of the UI
        **    library. If we are using the DLL version, we should NOT call
        **    this function in our application.
        */
#if 0
        if (!OleUIInitialize(hInstance, hPrevInstance))
                {
                OleDbgOut("Could not initialize OLEUI library\n");
                return FALSE;
                }
#endif
        // message loop
        while (GetMessage(&msg, NULL, NULL, NULL))
                {
                if (lpCSimpSvrApp->IsInPlaceActive())

                        // Only key messages need to be sent to
OleTranslateAccelerator.  Any other message
                        // would result in an extra FAR call to occur for
that message processing...

                        if ( (msg.message >= WM_KEYFIRST) && (msg.message <=
WM_KEYLAST) )

                                // OleTranslateAccelerator MUST be called,
even though this application does
                                // not have an accelerator table.  This has
to be done in order for the
                                // mneumonics for the top level menu items
to work properly.

                                if ( OleTranslateAccelerator
( lpCSimpSvrApp->GetDoc()->GetObj()->GetInPlaceFrame(),

lpCSimpSvrApp->GetDoc()->GetObj()->GetFrameInfo(),

&msg) == NOERROR)
```

```
                                        continue;

                TranslateMessage(&msg);    /* Translates virtual key codes
*/
                DispatchMessage(&msg);     /* Dispatches message to window
*/
                }

        // De-initialization for UI libraries.  Just like OleUIInitialize,
this
        // funciton is needed ONLY if we are using the static link version
of the
        // OLE UI library.
#if 0
        OleUIUninitialize();
#endif

        return (msg.wParam);           /* Returns the value from
PostQuitMessage */
}
```

MainWndProc   (SIMPSVR Sample)

```
//**********************************************************************
//
// MainWndProc
//
// Purpose:
//
//      Processes messages for the frame window
//
// Parameters:
//
//      HWND hWnd       - Window handle for frame window
//
//      UINT message    - Message value
//
//      WPARAM wParam   - Message info
//
//      LPARAM lParam   - Message info
//
// Return Value:
//
//      long
//
// Function Calls:
//      Function                        Location
//
//      CSimpSvrApp::lCommandHandler     APP.CPP
//      CSimpSvrApp::DestroyDocs         APP.CPP
//      CSimpSvrApp::lCreateDoc          APP.CPP
//      CSimpSvrApp::lSizeHandler        APP.CPP
//      CGameDoc::lAddVerbs            DOC.CPP
//      PostQuitMessage                 Windows API
//      DefWindowProc                   Windows API
//
// Comments:
//
//********************************************************************

//@@WTK WIN32, UNICODE
//long FAR PASCAL _export MainWndProc(HWND hWnd,UINT message,WPARAM
wParam,LPARAM lParam)
long FAR PASCAL EXPORT MainWndProc(HWND hWnd,UINT message,WPARAM
wParam,LPARAM lParam)

{

        switch (message)
                {
                case WM_CLOSE:
                        OutputDebugString("*** In WM_CLOSE *** \r\n");

                        // if there is still a document
                        if (lpCSimpSvrApp->GetDoc())
```

```
                                // if there is still an object within a
document
                                if (lpCSimpSvrApp->GetDoc()->GetObj())   //
this case occurs if there is still

// an outstanding Ref count on the object

// when the app is trying to go away.

// typically this case will occur in

// the "open" editing mode.
                                        //  Close the document
                                        lpCSimpSvrApp->GetDoc()->Close();

                        // hide the app window
                        lpCSimpSvrApp->HideAppWnd();

                        // if we were started by ole, unregister the class
factory, otherwise
                        // remove the ref count on our dummy OLE object
                        if (lpCSimpSvrApp->IsStartedByOle())
                                CoRevokeClassObject(lpCSimpSvrApp-
>GetRegisterClass());
                        else
                                lpCSimpSvrApp->GetOleObject()->Release();

                        lpCSimpSvrApp->Release();  // This should close the
app.

                        break;

                case WM_COMMAND:           // message: command from
application menu
                        return lpCSimpSvrApp->lCommandHandler(hWnd, message,
wParam, lParam);
                        break;

                case WM_CREATE:
                        return lpCSimpSvrApp->lCreateDoc(hWnd, message,
wParam, lParam);
                        break;

                case WM_DESTROY:                  // message: window being
destroyed
                        PostQuitMessage(0);
                        break;

                case WM_SIZE:
                        return lpCSimpSvrApp->lSizeHandler(hWnd, message,
wParam, lParam);

                default:                          // Passes it on if
unproccessed
```

```
                              return (DefWindowProc(hWnd, message, wParam,
lParam));
                    }
                    return (NULL);
}
```

About   (SIMPSVR Sample)

```
//**********************************************************************
//
// About
//
// Purpose:
//
//      Processes dialog box messages
//
// Parameters:
//
//      HWND hWnd       - Window handle for dialog box
//
//      UINT message    - Message value
//
//      WPARAM wParam   - Message info
//
//      LPARAM lParam   - Message info
//
// Return Value:
//
// Function Calls:
//      Function                    Location
//
//      EndDialog                   Windows API
//
// Comments:
//
//********************************************************************

//@@WTK WIN32, UNICODE
//BOOL FAR PASCAL _export About(HWND hDlg,unsigned message,WORD wParam,LONG
lParam)
BOOL FAR PASCAL EXPORT About(HWND hDlg,UINT message,WPARAM wParam,LPARAM
lParam)

{
        switch (message) {
        case WM_INITDIALOG:                /* message: initialize dialog box
*/
                return (TRUE);

        case WM_COMMAND:                      /* message: received a command
*/
                //@@WTK WIN32, UNICODE
                //if (wParam == IDOK                /* "OK" box selected?
*/
                if (LOWORD(wParam) == IDOK                /* "OK" box
selected?          */
                || LOWORD(wParam) == IDCANCEL) {      /* System menu close
command? */
                        EndDialog(hDlg, TRUE);        /* Exits the dialog
box        */
```

```
                        return (TRUE);
                }
                break;
        }
        return (FALSE);                                   /* Didn't process a
message    */
}
```

DocWndProc   (SIMPSVR Sample)

```
//**********************************************************************
//
// DocWndProc
//
// Purpose:
//
//      Processes dialog box messages
//
// Parameters:
//
//      HWND hWnd       - Window handle for doc window
//
//      UINT message    - Message value
//
//      WPARAM wParam   - Message info
//
//      LPARAM lParam   - Message info
//
// Return Value:
//
// Function Calls:
//      Function                            Location
//
//      CSimpSvrApp::PaintApp                APP.CPP
//      BeginPaint                          Windows API
//      EndPaint                            Windows API
//      DefWindowProc                       Windows API
//      IOleObject::QueryInterface          Object
//      IOleInPlaceObject::UIDeactivate     Object
//      IOleObject::DoVerb                  Object
//      IOleInPlaceObject::Release          Object
//
// Comments:
//
//********************************************************************

//@@WTK WIN32, UNICODE
//long FAR PASCAL _export DocWndProc(HWND hWnd,UINT message,WPARAM
wParam,LPARAM lParam)
long FAR PASCAL EXPORT DocWndProc(HWND hWnd,UINT message,WPARAM
wParam,LPARAM lParam)
{
        HDC hDC;
        PAINTSTRUCT ps;

        switch (message) {
                case WM_COMMAND:            // message: command from
application menu
                        return lpCSimpSvrApp->lCommandHandler(hWnd, message,
wParam, lParam);
                        break;
                case WM_PAINT:
                        hDC = BeginPaint(hWnd, &ps);

                        // tell the app class to paint itself
                        if (lpCSimpSvrApp)
                                lpCSimpSvrApp->PaintApp (hDC);

                        EndPaint(hWnd, &ps);
                        break;

                case WM_MENUSELECT:
                        lpCSimpSvrApp->SetStatusText();
                        break;

        default:                          /* Passes it on if unproccessed
*/
                return (DefWindowProc(hWnd, message, wParam, lParam));
        }
        return (NULL);
}
```

SPOLY

---------------------------------- OLE Automation Sample Program: SPoly ----------------------------------

SPoly is a program which draws polygons. The only way to make spoly draw a polygon is to use its programmability interface.

One OLE Automation object is exposed by Spoly:

* spoly.application

Spoly.Application is the object associated with spoly's main window. It controls drawing polygons and clearing the display.

---------------- Program Structure ---------------- SPoly implements IDispatch by hand for spoly.application.

---------------------------------- Methods defined on spoly.application ----------------------------------

Name Description --- Draw() Draw the polygon.

Reset() Delete all points from the polygon.

AddPoint(X, Y) Add a point with coordinates (x,y) to the polygon

EnumPoints() as VT_ENUM Return a collection of the polygon's points

GetXOrigin() as short Get the X origin of the polygon. SetXOrigin(x as short) Set the X origin of the polygon. GetYOrigin() as short Get the Y origin of the polygon. SetYOrigin(y as short) Set the Y origin of the polygon. GetWidth() as short Get and the line width of the polygon.

SetWidth(width as short) Set the line width of the polygon.

-------------------------- Shortcomings of this sample -------------------------- 1. Many items in this sample should be properties. Instead, they are implemented as methods. Anything which behaves like an attribute of the object should be a property.

2. This is not a good example of how to implement a collection.

MAKEFILE   (SPOLY Sample)

```
############################################################################
##
#
#  (c) Copyright Microsoft Corp. 1992-1993 All Rights Reserved
#
#  File:
#
#    makefile - makefile for spoly.exe
#
#  Purpose:
#
#    Builds the OLE 2.0 sample IDispatch server, spoly.exe.
#
#
#  Usage:
#
#     NMAKE                     ; build with defaults
#     or: NMAKE option          ; build with the given option(s)
#     or: NMAKE clean           ; erase all compiled files
#
#     option:
#         dev = [win16 | win32] ; dev=win32 is the default
#         DEBUG=[0|1]           ; DEBUG=1 is the default
#
#  Notes:
#
#    This makefile assumes that the PATH, INCLUDE and LIB environment
#    variables are setup properly.
#
############################################################################
##



##########################################################################
#
# Default Settings
#

!if "$(dev)" == ""
dev = win32
!endif

!if !("$(dev)" == "win16" || "$(dev)" == "win32")
!error Invalid dev option, choose from [win16 | win32]
!endif

!if "$(dev)" == "win16"
TARGET  = WIN16
!endif

!if "$(dev)" == "win32"
```

```
TARGET  = WIN32
!endif

!ifdef NODEBUG
DEBUG = 0
!endif

!if "$(DEBUG)" == "0"
NODEBUG = 1
!endif

!if "$(DEBUG)" == ""
DEBUG = 1
!endif


##########################################################################
#
# WIN16 Settings
#
!if "$(TARGET)" == "WIN16"

CC   = cl
LINK = link

RCFLAGS = -dWIN16
CFLAGS = -c -W3 -AM -GA -GEs -DWIN16
LINKFLAGS = /NOD /NOI /BATCH /ONERROR:NOEXE

LIBS = libw.lib mlibcew.lib

!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS) -Od -Zi -D_DEBUG $(CL)
LINKFLAGS = $(LINKFLAGS) /COD
!else
CFLAGS = $(CFLAGS) -Ox $(CL)
LINKFLAGS = $(LINKFLAGS) /FAR /PACKC
!endif
!endif


##########################################################################
#
# WIN32 Settings
#
!if "$(TARGET)" == "WIN32"

!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

CC = $(cc)
CFLAGS = $(cflags) $(cvarsmt) -DUNICODE $(cdebug)

!ifndef NODEBUG
CFLAGS = $(CFLAGS) -D_DEBUG
!endif
```

```
LINK = $(link)
LINKFLAGS = $(linkdebug) $(guilflags)
RCFLAGS = -DWIN32

!endif


##########################################################################
#
# Application Settings
#

!if "$(TARGET)" == "WIN16"
LIBS = ole2.lib compobj.lib ole2disp.lib ole2nls.lib $(LIBS)
!else
!if "$(TARGET)" == "WIN32"
LIBS = $(olelibsmt)
!endif
!endif

OBJS = \
        winmain.obj     \
        cpoly.obj       \
        cpoint.obj      \
        cenumpt.obj     \
        statbar.obj     \
        clsid.obj       \
        misc.obj


goal : setflags spoly.exe


setflags :
        set CL=$(CFLAGS)

clean :
    if exist *.obj       del *.obj
    if exist spoly.exe del spoly.exe
    if exist spoly.map del spoly.map
    if exist spoly.res del spoly.res
    if exist *.pdb       del *.pdb


##########################################################################
#
# Application Build (WIN16 Specific)
#

!if "$(TARGET)" == "WIN16"

spoly.exe : $(OBJS) spoly.def spoly.res spoly.ico
        link $(LINKFLAGS) @<<
$(OBJS),
$@,,
$(LIBS),
spoly.def
<<
        rc -k -t spoly.res $@
!endif


###########################################################################
#
# Application Build (WIN32 Specific)
#
!if "$(TARGET)" == "WIN32"

spoly.exe : $(OBJS) spoly.def spoly.res spoly.ico
      $(LINK) @<<
        $(LINKFLAGS)
        -out:$@
        -map:$*.map
        $(OBJS)
        spoly.res
        $(LIBS)
<<
!endif


spoly.res : spoly.rc resource.h
        rc $(RCFLAGS) -r -fo$@ spoly.rc

winmain.obj: winmain.cpp hostenv.h resource.h spoly.h cpoint.h cpoly.h
statbar.h
        $(CC) winmain.cpp

cpoint.obj: cpoint.cpp cpoint.h hostenv.h spoly.h statbar.h
        $(CC) cpoint.cpp

cpoly.obj: cpoly.cpp cpoint.h cpoly.h hostenv.h spoly.h statbar.h
        $(CC) cpoly.cpp

clsid.obj: clsid.c clsid.h
        $(CC) clsid.c

cenumpt.obj: cenumpt.cpp cenumpt.h
        $(CC) cenumpt.cpp

statbar.obj: statbar.cpp statbar.h
        $(CC) statbar.cpp

misc.obj: misc.cpp hostenv.h spoly.h
        $(CC) misc.cpp
```

CENUMPT.H (SPOLY Sample)

```
/***
*cenumpt.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  Definition of the CEnumPoint class.
*
*Implementation Notes:
*
*****************************************************************************
*/

class CEnumPoint : public IEnumVARIANT
{
public:
    static HRESULT Create(SAFEARRAY FAR* psa, CEnumPoint FAR* FAR* ppenum);

    /* IUnknown methods */
    STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppv);
    STDMETHOD_(unsigned long, AddRef)(void);
    STDMETHOD_(unsigned long, Release)(void);

    /* IEnumVARIANT methods */
    STDMETHOD(Next)(unsigned long celt, VARIANT FAR* rgvar, unsigned long
FAR* pceltFetched);
    STDMETHOD(Skip)(unsigned long celt);
    STDMETHOD(Reset)(void);
    STDMETHOD(Clone)(IEnumVARIANT FAR* FAR* ppenum);

    CEnumPoint();

private:

    unsigned long m_refs;

    unsigned long m_celts;
    unsigned long m_iCurrent;
    SAFEARRAY FAR* m_psa;
};
```

CENUMPT.CPP (SPOLY Sample)

```
/***
*cenumpt.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*  This module implements the CEnumPoint class.
*
*
*Implementation Notes:
*
****************************************************************************
*/

#include "hostenv.h"
#include "cenumpt.h"


CEnumPoint::CEnumPoint()
{
    m_refs = 0;

    m_psa = NULL;
    m_celts = 0;
    m_iCurrent = 0;
}


/***
*HRESULT CEnumPoint::Create(SAFEARRAY*, CEnumPoint**)
*Purpose:
*  This routine creates a CPoint enumerator from the given
*  (1 X N) SafeArray of CPoint IDispatch pointers.
*
*Entry:
*  psa = pointer to a SafeArray of VARIANTs
*
*Exit:
*  return value = HRESULT
*
*  *ppenum = pointer to a CPoint enumerator
*
***********************************************************************/
HRESULT
CEnumPoint::Create(SAFEARRAY FAR* psa, CEnumPoint FAR* FAR* ppenum)
{
    long lBound;
    HRESULT hresult;
    CEnumPoint FAR* penum;


    // Verify that the SafeArray is the proper shape.
    //
    if(SafeArrayGetDim(psa) != 1)
      return ResultFromScode(E_INVALIDARG);

    hresult = SafeArrayGetLBound(psa, 1, &lBound);
    if(FAILED(hresult))
      return hresult;

    if(lBound != 0)
      return ResultFromScode(E_INVALIDARG);

    penum = new FAR CEnumPoint();
    if(penum == NULL)
      return ResultFromScode(E_OUTOFMEMORY);
    penum->AddRef();

    hresult = SafeArrayGetUBound(psa, 1, &lBound);
    if(FAILED(hresult))
      goto LError0;

    penum->m_psa = psa;
    penum->m_celts = lBound + 1;

    *ppenum = penum;

    return NOERROR;

LError0:;
    penum->Release();

    return hresult;
}


//---------------------------------------------------------------------
//                         IUnknown methods
//---------------------------------------------------------------------


STDMETHODIMP
CEnumPoint::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{

    if(!IsEqualIID(riid, IID_IUnknown))
      if(!IsEqualIID(riid, IID_IEnumVARIANT)) {
        *ppv = NULL;
        return ResultFromScode(E_NOINTERFACE);
    }

    *ppv = this;
    AddRef();
    return NOERROR;
}
```

```
STDMETHODIMP_(unsigned long)
CEnumPoint::AddRef()
{
    return ++m_refs;
}


STDMETHODIMP_(unsigned long)
CEnumPoint::Release()
{
    if(--m_refs == 0){
      if(m_psa != NULL)
     SafeArrayDestroy(m_psa);
      delete this;
      return 0;
    }
    return m_refs;
}


//---------------------------------------------------------------------
//                        IEnumVARIANT methods
//---------------------------------------------------------------------


/***
*HRESULT CEnumPoint::Next(unsigned long, VARIANT*, unsigned long*)
*Purpose:
*  Attempt to get the next 'celt' items in the enumeration sequence.
*
*Entry:
*  celt = the number of elements to get

*
*Exit:
*  return value = HRESULT
*    S_OK
*    S_FALSE - the end of the sequence was reached
*
*  rgvar = array of the next 'celt' items
*  *pceltFetched = count of the elements actually fetched.
*
***********************************************************************/
STDMETHODIMP
CEnumPoint::Next(
    unsigned long celt,
    VARIANT FAR* rgvar,
    unsigned long FAR* pceltFetched)
{
    long ix;
    unsigned int i;
    HRESULT hresult;


    for(i = 0; i < celt; ++i)
```

```
      VariantInit(&rgvar[i]);

    for(i = 0; i < celt; ++i){
      if(m_iCurrent == m_celts){
        hresult = ResultFromScode(S_FALSE);
     goto LDone;
      }

      ix = m_iCurrent++;
      hresult = SafeArrayGetElement(m_psa, &ix, &rgvar[i]);
      if(FAILED(hresult))
     goto LError0;
    }

    hresult = NOERROR;

LDone:;
    if (pceltFetched != NULL)
      *pceltFetched = i;

    return hresult;

LError0:;

    for(i = 0; i < celt; ++i)
      VariantClear(&rgvar[i]);

    return hresult;
}


/***
*HRESULT CEnumPoint::Skip(unsigned long)
*Purpose:
*  Attempt to skip over the next 'celt' elements in the enumeration
*  sequence.
*
*Entry:
*  celt = the count of elements to skip
*
*Exit:
*  return value = HRESULT
*    S_OK
*    S_FALSE -  the end of the sequence was reached
*
***********************************************************************/
STDMETHODIMP
CEnumPoint::Skip(unsigned long celt)
{
    m_iCurrent += celt;

    if(m_iCurrent > m_celts)
     m_iCurrent = m_celts;

    return (m_iCurrent == m_celts)
```

```
      ? ResultFromScode(S_FALSE) : NOERROR;
}


/***
*HRESULT CEnumPoint::Reset(void)
*Purpose:
*  Reset the enumeration sequence back to the beginning.
*
*Entry:
*  None
*
*Exit:
*  return value = SHRESULT CODE
*    S_OK
*
***********************************************************************/
STDMETHODIMP
CEnumPoint::Reset()
{
    m_iCurrent = 0;

    return NOERROR;
}


/***
*HRESULT CEnumPoint::Clone(IEnumVARIANT**)
*Purpose:
*  Retrun a CPoint enumerator with exactly the same state as the
*  current one.
*
*Entry:
*  None
*
*Exit:
*  return value = HRESULT
*    S_OK
*    E_OUTOFMEMORY
*
***********************************************************************/
STDMETHODIMP
CEnumPoint::Clone(IEnumVARIANT FAR* FAR* ppenum)
{
    HRESULT hresult;
    SAFEARRAY FAR* psa;
    CEnumPoint FAR* penum;

    hresult = SafeArrayCopy(m_psa, &psa);
    if(FAILED(hresult))
      return hresult;

    hresult = CEnumPoint::Create(psa, &penum);
    if(FAILED(hresult))
      goto LError0;
    // Assert(penum->m_celts == m_celts);

    penum->m_iCurrent = m_iCurrent;

    *ppenum = penum;

    return NOERROR;

LError0:
    SafeArrayDestroy(psa);

    return hresult;
}
```

CLSID.H (SPOLY Sample)

```
/***
*clsid.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This file defines the CLSIDs
*
*Implementation Notes:
*
****************************************************************************
*/

DEFINE_GUID(CLSID_CPoly, 0x00020462, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);
DEFINE_GUID(CLSID_CPoint, 0x00020463, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);

#ifdef INITGUID
# ifdef _MAC
DEFINE_GUID(GUID_NULL, 0L, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);

DEFINE_GUID(IID_IDispatch, 0x00020400L, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);
DEFINE_GUID(IID_IEnumVARIANT, 0x00020404L, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0,
0x46);
DEFINE_GUID(IID_IUnknown, 0x00000000L, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);
DEFINE_GUID(IID_IClassFactory, 0x00000001L, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0,
0x46);
# endif
#endif
```

CLSID.C   (SPOLY Sample)

```
/***
*clsid.c
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This file allocates and initializes the CLSIDs.
*
****************************************************************************
*/

#ifdef _PPCMAC
#pragma data_seg ("_FAR_DATA")
#pragma data_seg ( )
#endif //_PPCMAC

#ifdef _MAC
# include <Types.h>
#ifdef _MSC_VER
# include <Processe.h>
# include <AppleEve.h>
#else //_MSC_VER
# include <Processes.h>
# include <AppleEvents.h>
#endif //_MSC_VER
#else
# include <windows.h>
#endif

#ifndef WIN32
#include <compobj.h>
#endif //!WIN32

// this redefines the DEFINE_GUID() macro to do allocation.
//
#include <initguid.h>

#ifndef INITGUID
# define INITGUID
#endif

// due to the previous header, including this causes the DEFINE_GUID
// definitions in the following header(s) to actually allocate data.
//
#include "clsid.h"
```

CPOINT.H (SPOLY Sample)

```
/***
*cpoint.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  Definition of the CPoint class.
*
*  The CPoint object exposes two properties for programatic access
*  via the IDispatch interface.
*
*  properties:
*    X                 - the 'x' coordinate of the point
*    Y                 - the 'y' coordinate of the point
*
*Implementation Notes:
*
****************************************************************************
*/

#ifndef    CLASS
#ifdef     __TURBOC__
#define CLASS class huge
#else
#define CLASS class FAR
#endif
#endif

class CPoly;

CLASS CPoint : public IDispatch {
    friend class CPoly;

public:
    static CPoint FAR* Create();

    /* IUnknown methods */
    STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppvObj);
    STDMETHOD_(unsigned long, AddRef)(void);
    STDMETHOD_(unsigned long, Release)(void);

    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(unsigned int FAR* pcTypeInfo);

    STDMETHOD(GetTypeInfo)(
      unsigned int iTypeInfo,
      LCID lcid,
      ITypeInfo FAR* FAR* ppTypeInfo);

    STDMETHOD(GetIDsOfNames)(
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      unsigned int cNames,
      LCID lcid,
      DISPID FAR* rgdispid);

    STDMETHOD(Invoke)(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      unsigned short wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      unsigned int FAR* pwArgErr);

    /* Introduced methods */

    virtual short PASCAL GetX(void);
    virtual void  PASCAL SetX(short x);
    virtual short PASCAL GetY(void);
    virtual void  PASCAL SetY(short y);

private:
    CPoint();

    unsigned long m_refs;

    short m_x;
    short m_y;
};


// member DISPIDs
//
enum IDMEMBER_CPOINT {
    IDMEMBER_CPOINT_GETX = 1,
    IDMEMBER_CPOINT_SETX,
    IDMEMBER_CPOINT_GETY,
    IDMEMBER_CPOINT_SETY,
    IDMEMBER_CPOINT_MAX
};

// structure used to link together lists of points
//
struct POINTLINK {
    POINTLINK FAR* next;
    CPoint FAR* ppoint;
};


// The CPoint Class Factory
//
CLASS CPointCF : public IClassFactory
{
public:
    static IClassFactory FAR* Create();

    /* IUnknown methods */
    STDMETHOD(QueryInterface)(REFIID iid, void FAR* FAR* ppv);
    STDMETHOD_(unsigned long, AddRef)(void);
    STDMETHOD_(unsigned long, Release)(void);

    /* IClassFactory methods */
    STDMETHOD(CreateInstance)(
      IUnknown FAR* pUnkOuter, REFIID iid, void FAR* FAR* ppv);
#ifdef _MAC
    STDMETHOD(LockServer)(unsigned long fLock);
#else
    STDMETHOD(LockServer)(BOOL fLock);
#endif

private:
    CPointCF();

    unsigned long m_refs;
};
```

CPOINT.CPP (SPOLY Sample)

```
/***
*cpoint.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*  This module implements the CPoint and CPointCF classes.
*
*  This module is intended as a sample implementation of the IDispatch
*  interface, and its purpose is to demonstrate how an object can
*  expose methods and properties for programatic and cross-process
*  access via the IDispatch interface.
*
*Implementation Notes:
*
*****************************************************************************
*/

#include "spoly.h"
#include "cpoint.h"


CPoint::CPoint()
{
    m_x = 0;
    m_y = 0;
    m_refs = 0;
}

/***
*CPoint::Create(void)
*Purpose:
*  Create an instance of a CPoint object.
*
*Entry:
*  None
*
*Exit:
*  returns a CPoint*, NULL if creation failed.
*
************************************************************************/
CPoint FAR*
CPoint::Create()
{
    CPoint FAR* ppoint;

    if((ppoint = new FAR CPoint()) == NULL)
      return NULL;
    ppoint->AddRef();
    return ppoint;
}
```

```
//---------------------------------------------------------------------
//                      IUnknown Methods
//---------------------------------------------------------------------


STDMETHODIMP
CPoint::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    if(!IsEqualIID(riid, IID_IUnknown))
      if(!IsEqualIID(riid, IID_IDispatch)) {
        *ppv = NULL;
        return ResultFromScode(E_NOINTERFACE);
    }

    *ppv = this;
    AddRef();
    return NOERROR;
}


STDMETHODIMP_(unsigned long)
CPoint::AddRef(void)
{
    return ++m_refs;
}


STDMETHODIMP_(unsigned long)
CPoint::Release(void)
{
    if(--m_refs == 0){
      delete this;
      return 0;
    }
    return m_refs;
}


//---------------------------------------------------------------------
//                      IDispatch methods
//---------------------------------------------------------------------

/*
 * NOTE: Support for the following two methods is not available
 * in this version.
 *
 */

STDMETHODIMP
CPoint::GetTypeInfoCount(unsigned int FAR* pctinfo)
{
    *pctinfo = 0;
    return NOERROR;
}
```

```
STDMETHODIMP
CPoint::GetTypeInfo(unsigned int itinfo, LCID lcid, ITypeInfo FAR* FAR*
pptinfo)
{
    UNUSED(itinfo);
    UNUSED(lcid);
    UNUSED(pptinfo);

    return ResultFromScode(E_NOTIMPL);
}


/***
*HRESULT CPoint::GetIDsOfNames(REFIID, OLECHAR**, unsigned int, LCID, long*)
*Purpose:
*  This method translates the given array of names to a corresponding
*  array of DISPIDs.
*
*  Index 0 of the name array is the member name, and indices 1-N if
*  present represent named parameters on that member.
*
*  The local ID ('lcid') is unused by this naive implementation. A more
*  sophisticated implementation, sensitive to localization and natural
*  language support would use the locale ID to interpret the given names
*  in a correct locale specific context.
*
*Entry:
*  rgszNames = pointer to an array of names
*  cNames = the number of names in the rgszNames array
*  lcid = the callers locale ID
*
*Exit:
*  return value = HRESULT
*  rgid = array of name IDs corresponding to the rgszNames array
*    this array will contain -1 for each entry that is not known.
*
***********************************************************************/
STDMETHODIMP
CPoint::GetIDsOfNames(
    REFIID riid,
    OLECHAR FAR* FAR* rgszNames,
    unsigned int cNames,
    LCID lcid,
    DISPID FAR* rgdispid)
{
static MEMBERDESC rgmdCPoint[] = {
    {OLESTR("GETX"),   IDMEMBER_CPOINT_GETX,   NULL, 0},
    {OLESTR("SETX"),   IDMEMBER_CPOINT_SETX,   NULL, 0},
    {OLESTR("GETY"),   IDMEMBER_CPOINT_GETY,   NULL, 0},
    {OLESTR("SETY"),   IDMEMBER_CPOINT_SETY,   NULL, 0}
};
```

```
    // this object only exposed the "default" interface.
    //
    if(!IsEqualIID(riid, IID_NULL))
      return ResultFromScode(DISP_E_UNKNOWNINTERFACE);

    return SPolyGetIDsOfNames(
      rgmdCPoint, DIM(rgmdCPoint), rgszNames, cNames, lcid, rgdispid);
}


/***
*HRESULT CPoint::Invoke(...)
*Purpose:
*  Dispatch a method or property request for objects of type CPoint.
*
*  see the IDispatch document for more information, and a general
*  description of this method.
*
*Entry:
*  dispidMember = the DISPID of the member being requested
*
*  riid = reference to the interface ID of the interface on this object
*    that the requested member belongs to. IID_NULL means to interpret
*    the member as belonging to the implementation defined "default"
*    or "primary" interface.
*
*  lcid = the caller's locale ID
*
*  wFlags = flags indicating the type of access being requested
*
*  pdispparams = pointer to the DISPPARAMS struct containing the
*    requested members arguments (if any) and its named parameter
*    DISPIDs (if any).
*
*Exit:
*  return value = HRESULT
*   see the IDispatch spec for a description of possible success codes.
*
*  pvarResult = pointer to a caller allocated VARIANT containing
*    the members return value (if any).
*
*  pexcepinfo = caller allocated exception info structure, this will
*    be filled in only if an exception was raised that must be passed
*    up through Invoke to an enclosing handler.
*
*  puArgErr = pointer to a caller allocated UINT, that will contain the
*    index of the offending argument if a DISP_E_TYPEMISMATCH error
*    was returned indicating that one of the arguments was of an
*    incorrect type and/or could not be reasonably coerced to a proper
*    type.
*
***********************************************************************/
STDMETHODIMP
CPoint::Invoke(
    DISPID dispidMember,
    REFIID riid,
    LCID lcid,
    unsigned short wFlags,
    DISPPARAMS FAR* pdispparams,
    VARIANT FAR* pvarResult,
    EXCEPINFO FAR* pexcepinfo,
    unsigned int FAR* puArgErr)
{
    unsigned int uArgErr;
    HRESULT hresult;
    VARIANTARG varg0;
    VARIANT varResultDummy;

    UNUSED(lcid);
    UNUSED(pexcepinfo);

    // make sure the wFlags are legal
    if(wFlags & ~(DISPATCH_METHOD | DISPATCH_PROPERTYGET |
DISPATCH_PROPERTYPUT | DISPATCH_PROPERTYPUTREF))
      return ResultFromScode(E_INVALIDARG);

    // this object only exposes a "default" interface.
    //
    if(!IsEqualIID(riid, IID_NULL))
      return ResultFromScode(DISP_E_UNKNOWNINTERFACE);

    // this makes the following code a bit simpler
    if(puArgErr == NULL)
      puArgErr = &uArgErr;
    if(pvarResult == NULL)
      pvarResult = &varResultDummy;

    VariantInit(&varg0);

    // assume the return type is void, unless we find otherwise.
    VariantInit(pvarResult);

    switch(dispidMember){
    case IDMEMBER_CPOINT_GETX:
      V_VT(pvarResult) = VT_I2;
      V_I2(pvarResult) = GetX();
      break;

    case IDMEMBER_CPOINT_SETX:

      hresult = DispGetParam(pdispparams, 0, VT_I2, &varg0, puArgErr);
      if(hresult != NOERROR)
        return hresult;
      SetX(V_I2(&varg0));
      break;

    case IDMEMBER_CPOINT_GETY:
      V_VT(pvarResult) = VT_I2;
      V_I2(pvarResult) = GetY();
      break;
    case IDMEMBER_CPOINT_SETY:
      hresult = DispGetParam(pdispparams, 0, VT_I2, &varg0, puArgErr);
      if(hresult != NOERROR)
        return hresult;
      SetY(V_I2(&varg0));
      break;

    default:
      return ResultFromScode(DISP_E_MEMBERNOTFOUND);
    }
    return NOERROR;
}

short PASCAL
CPoint::GetX()
{
    return m_x;
}

void PASCAL
CPoint::SetX(short x)
{
    m_x = x;
}

short PASCAL
CPoint::GetY()
{
    return m_y;
}

void PASCAL
CPoint::SetY(short y)
{
    m_y = y;
}


//---------------------------------------------------------------------
//         Implementation of the CPoint Class Factory
//---------------------------------------------------------------------

CPointCF::CPointCF()
{
    m_refs = 0;
}

IClassFactory FAR*
CPointCF::Create()
{
    CPointCF FAR* pCF;

    if((pCF = new FAR CPointCF()) == NULL)
      return NULL;
    pCF->AddRef();
    return pCF;
}

STDMETHODIMP
CPointCF::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{

    if(!IsEqualIID(riid, IID_IUnknown))
      if(!IsEqualIID(riid, IID_IClassFactory)) {
        *ppv = NULL;
        return ResultFromScode(E_NOINTERFACE);
    }

    *ppv = this;
    ++m_refs;
    return NOERROR;
}

STDMETHODIMP_(unsigned long)
CPointCF::AddRef(void)
{
    return ++m_refs;
}

STDMETHODIMP_(unsigned long)
CPointCF::Release(void)
{
    if(--m_refs == 0){
      delete this;
      return 0;
    }
    return m_refs;
}

STDMETHODIMP
CPointCF::CreateInstance(
    IUnknown FAR* punkOuter,
    REFIID riid,
    void FAR* FAR* ppv)
{
    HRESULT hresult;
    CPoint FAR *ppoint;

    UNUSED(punkOuter);

    if((ppoint = CPoint::Create()) == NULL){
      *ppv = NULL;
      return ResultFromScode(E_OUTOFMEMORY);
    }
    hresult = ppoint->QueryInterface(riid, ppv);
    ppoint->Release();
    return hresult;
}
```

```
STDMETHODIMP
#ifdef _MAC
CPointCF::LockServer(unsigned long fLock)
#else
CPointCF::LockServer(BOOL fLock)
#endif
{
    UNUSED(fLock);

    return NOERROR;
}
```

CPOLY.H (SPOLY Sample)

```
/***
*cpoly.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  Definition of the CPoly class.
*
*  The CPoly class defines a number of methods and exposes them for
*  external programmability via IDispatch,
*
*  methods:
*    DRAW        - draw the polygon
*    RESET       - delete all points from the polygon
*
*    ADDPOINT(X, Y)    - add a point with coordinates (x,y) to the polygon
*
*    ENUMPOINTS        - return a collection of the polygon's points
*
*    GETXORIGIN        - get and set the X origin of the polygon
*    SETXORIGIN
*
*    GETYORIGIN        - get and set the Y origin of the polygon
*    SETYORIGIN
*
*    GETWIDTH          - get and set the line width of the polygon
*    SETWIDTH
*
*  UNDONE: update description
*
*Implementation Notes:
*
****************************************************************************
*/


#ifndef    CLASS
# ifdef    __TURBOC__
#  define CLASS class huge
# else
#  define CLASS class FAR
# endif
#endif

class CPoint;

CLASS CPoly : public IDispatch
{
public:
    static CPoly FAR* Create();
```

```
/* IUnknown methods */
STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppvObj);
STDMETHOD_(unsigned long, AddRef)(void);
STDMETHOD_(unsigned long, Release)(void);

/* IDispatch methods */
STDMETHOD(GetTypeInfoCount)(unsigned int FAR* pcTypeInfo);

STDMETHOD(GetTypeInfo)(
  unsigned int iTypeInfo,
  LCID lcid,
  ITypeInfo FAR* FAR* ppTypeInfo);

STDMETHOD(GetIDsOfNames)(
  REFIID riid,
  OLECHAR FAR* FAR* rgszNames,
  unsigned int cNames,
  LCID lcid,
  DISPID FAR* rgdispid);

STDMETHOD(Invoke)(
  DISPID dispidMember,
  REFIID riid,
  LCID lcid,
  unsigned short wFlags,
  DISPPARAMS FAR* pdispparams,
  VARIANT FAR* pvarResult,
  EXCEPINFO FAR* pexcepinfo,
  unsigned int FAR* puArgErr);

/* Introduced methods */

virtual void PASCAL Draw(void);
virtual void PASCAL Reset(void);

// add a point with the given 'x' and 'y' coordinates
virtual HRESULT PASCAL AddPoint(short x, short y);

// return a collection of the polygon's points
virtual HRESULT PASCAL EnumPoints(IEnumVARIANT FAR* FAR* ppenum);

// get/set the polygon's X origin property
virtual short PASCAL GetXOrigin(void);
virtual void  PASCAL SetXOrigin(short x);

// get/set the polygon's Y origin property
virtual short PASCAL GetYOrigin(void);
virtual void  PASCAL SetYOrigin(short y);

virtual short PASCAL GetWidth(void);
virtual void  PASCAL SetWidth(short width);

virtual short PASCAL get_red(void);
virtual void  PASCAL set_red(short red);
```

```
    virtual short PASCAL get_green(void);
    virtual void  PASCAL set_green(short green);

    virtual short PASCAL get_blue(void);
    virtual void  PASCAL set_blue(short blue);

    // Debug method
    virtual void  PASCAL Dump(void);

public:

    // Draw all polygons.
    static void PolyDraw(void);

    // Release all polygons.
    static void PolyTerm(void);

    // Dump all polygons to dbwin.
    static void PolyDump(void);


private:
    CPoly();

    short m_xorg;
    short m_yorg;
    short m_width;

    short m_red;
    short m_green;
    short m_blue;


    unsigned long m_refs;
    unsigned int m_cPoints;

    POINTLINK FAR* m_ppointlink;
    POINTLINK FAR* m_ppointlinkLast;
};

// DISPIDs for the members and properties available via IDispatch.
//
enum IDMEMBER_CPOLY {
    IDMEMBER_CPOLY_DRAW = 1,
    IDMEMBER_CPOLY_RESET,
    IDMEMBER_CPOLY_ADDPOINT,
    IDMEMBER_CPOLY_ENUMPOINTS,
    IDMEMBER_CPOLY_GETXORIGIN,
    IDMEMBER_CPOLY_SETXORIGIN,
    IDMEMBER_CPOLY_GETYORIGIN,
    IDMEMBER_CPOLY_SETYORIGIN,
    IDMEMBER_CPOLY_GETWIDTH,
    IDMEMBER_CPOLY_SETWIDTH,
    IDMEMBER_CPOLY_GETRED,
    IDMEMBER_CPOLY_SETRED,
    IDMEMBER_CPOLY_GETGREEN,
    IDMEMBER_CPOLY_SETGREEN,
    IDMEMBER_CPOLY_GETBLUE,
    IDMEMBER_CPOLY_SETBLUE,
    IDMEMBER_CPOLY_DUMP,
    IDMEMBER_CPOLY_MAX
};

// structure used to link together polygons
//
struct POLYLINK {
    POLYLINK FAR* next;
    CPoly FAR* ppoly;
};


// The CPoly class factory
//
CLASS CPolyCF : public IClassFactory
{
public:
    static IClassFactory FAR* Create();

    /* IUnknown methods */
    STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppv);
    STDMETHOD_(unsigned long, AddRef)(void);
    STDMETHOD_(unsigned long, Release)(void);

    /* IClassFactory methods */
    STDMETHOD(CreateInstance)(
      IUnknown FAR* pUnkOuter, REFIID riid, void FAR* FAR* ppv);
#ifdef _MAC
    STDMETHOD(LockServer)(unsigned long fLock);
#else
    STDMETHOD(LockServer)(BOOL fLock);
#endif

private:
    CPolyCF();

    unsigned long m_refs;
};
```

CPOLY.CPP (SPOLY Sample)

```
/***
*CPoly.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*  This module implements the CPoly and CPolyCF classes.
*
*  This module is intended as a sample implementation of the IDispatch
*  interface, and its purpose is to demonstrate how an object can
*  expose methods and properties for programatic and cross-process
*  access via the IDispatch interface.
*
*Implementation Notes:
*
*****************************************************************************
*/

#include "spoly.h"
#include "cpoint.h"
#include "cpoly.h"
#include "cenumpt.h"

#ifndef _MAC
extern CStatBar FAR* g_psb;
#else
extern "C" WindowPtr g_pwndClient;
#endif

// our global list of polygons.
//
POLYLINK FAR* g_ppolylink = (POLYLINK FAR*)NULL;

// global count of polygons.
//
int g_cPoly = 0;


CPoly::CPoly()
{
    m_refs = 0;
    m_xorg = 0;
    m_yorg = 0;
    m_width = 0;
    m_cPoints = 0;

    m_red   = 0;
    m_green = 0;
    m_blue  = 0;

    m_ppointlink = NULL;
    m_ppointlinkLast = NULL;
```

```
}


/***
*CPoly *CPoly::Create(void)
*Purpose:
*  Create an instance of a CPoly object, and add it to the global
*  list of polygons.
*
*Entry:
*  None
*
*Exit:
*  returns a polygon object, NULL the allocation failed.
*
***********************************************************************/
CPoly FAR*
CPoly::Create()
{
    CPoly FAR* ppoly;
    POLYLINK FAR* ppolylink;

    if((ppolylink = new FAR POLYLINK) == (POLYLINK FAR*)NULL)
      return (CPoly FAR*)NULL;

    if((ppoly = new FAR CPoly()) == (CPoly FAR*)NULL)
      return (CPoly FAR*)NULL;

    ppoly->AddRef();

    // push the new polygon onto the front of the polygon list.
    //
    ++g_cPoly;

    ppolylink->ppoly = ppoly;

    ppolylink->next = g_ppolylink;
    g_ppolylink = ppolylink;

#ifndef _MAC
    SBprintf(g_psb, TSTR("#poly = %d"), g_cPoly);
#endif

    return ppoly;
}


//---------------------------------------------------------------------
//                     IUnknown Methods
//---------------------------------------------------------------------


STDMETHODIMP
CPoly::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    if(IsEqualIID(riid, IID_IUnknown) ||
       IsEqualIID(riid, IID_IDispatch)) {
      *ppv = this;
      AddRef();
      return NOERROR;
    }

    *ppv = NULL;
    return ResultFromScode(E_NOINTERFACE);
}


STDMETHODIMP_(unsigned long)
CPoly::AddRef()
{
    return ++m_refs;
}


STDMETHODIMP_(unsigned long)
CPoly::Release()
{
    POLYLINK FAR* FAR* pppolylink, FAR* ppolylinkDead;

    if(--m_refs == 0){
      Reset(); // release all CPoints

      // remove ourselves from the global list of polygons
      //
      for( pppolylink = &g_ppolylink;
          *pppolylink != NULL;
        pppolylink = &(*pppolylink)->next)
      {
     if((*pppolylink)->ppoly == this){
       ppolylinkDead = *pppolylink;
       *pppolylink = (*pppolylink)->next;
       delete ppolylinkDead;
       break;
     }
      }

      --g_cPoly;

#ifndef _MAC
      SBprintf(g_psb, TSTR("#poly = %d"), g_cPoly);
#endif

      delete this;
      return 0;
    }
    return m_refs;
}
```

```
//---------------------------------------------------------------------
//                      IDispatch Methods
//---------------------------------------------------------------------


/*
 * NOTE: Support for the following two methods is not available
 * in this version.
 *
 */

STDMETHODIMP
CPoly::GetTypeInfoCount(unsigned int FAR* pctinfo)
{
    *pctinfo = 0;
    return NOERROR;
}


STDMETHODIMP
CPoly::GetTypeInfo(unsigned int itinfo, LCID lcid, ITypeInfo FAR* FAR*
pptinfo)
{
    UNUSED(itinfo);
    UNUSED(lcid);
    UNUSED(pptinfo);

    return ResultFromScode(E_NOTIMPL);
}


/***
*HRESULT CPoly::GetIDsOfNames(OLECHAR**, unsigned int, LCID, long*)
*Purpose:
*  This method translates the given array of names to a corresponding
*  array of DISPIDs.
*
*  This method deferrs to a common implementation shared by
*  both the CPoly and CPoint objects. See the description of
*  'SPolyGetIDsOfNames()' in dispimpl.cpp for more information.
*
*Entry:
*  rgszNames = pointer to an array of names
*  cNames = the number of names in the rgszNames array
*  lcid = the callers locale ID
*
*Exit:
*  return value = HRESULT
*  rgdispid = array of DISPIDs corresponding to the rgszNames array
*    this array will contain -1 for each entry that is not known.
*
***********************************************************************/
STDMETHODIMP
CPoly::GetIDsOfNames(
    REFIID riid,
    OLECHAR FAR* FAR* rgszNames,
    unsigned int cNames,
    LCID lcid,
    DISPID FAR* rgdispid)
{
static PARAMDESC rgpdAddPoint[] = {
    {OLESTR("X")}, {OLESTR("Y")}
};
static MEMBERDESC rgmdCPoly[] = {
    {OLESTR("DRAW"),      IDMEMBER_CPOLY_DRAW,       NULL,       0},
    {OLESTR("RESET"),     IDMEMBER_CPOLY_RESET,      NULL,       0},
    {OLESTR("ADDPOINT"),   IDMEMBER_CPOLY_ADDPOINT,  rgpdAddPoint,     2},
    {OLESTR("ENUMPOINTS"), IDMEMBER_CPOLY_ENUMPOINTS,NULL,       0},
    {OLESTR("GETXORIGIN"), IDMEMBER_CPOLY_GETXORIGIN,NULL,       0},
    {OLESTR("SETXORIGIN"), IDMEMBER_CPOLY_SETXORIGIN,NULL,       0},
    {OLESTR("GETYORIGIN"), IDMEMBER_CPOLY_GETYORIGIN,NULL,       0},
    {OLESTR("SETYORIGIN"), IDMEMBER_CPOLY_SETYORIGIN,NULL,       0},
    {OLESTR("GETWIDTH"),   IDMEMBER_CPOLY_GETWIDTH,  NULL,       0},
    {OLESTR("SETWIDTH"),   IDMEMBER_CPOLY_SETWIDTH,  NULL,       0},
    {OLESTR("get_red"),   IDMEMBER_CPOLY_GETRED,     NULL,       0},
    {OLESTR("set_red"),   IDMEMBER_CPOLY_SETRED,     NULL,       0},
    {OLESTR("get_green"),  IDMEMBER_CPOLY_GETGREEN,  NULL,       0},
    {OLESTR("set_green"),  IDMEMBER_CPOLY_SETGREEN,  NULL,       0},
    {OLESTR("get_blue"),   IDMEMBER_CPOLY_GETBLUE,   NULL,       0},
    {OLESTR("set_blue"),   IDMEMBER_CPOLY_SETBLUE,   NULL,       0},
    {OLESTR("DUMP"),      IDMEMBER_CPOLY_DUMP,       NULL,       0}
};

    // this object only exposes a "default" interface.
    //
    if(!IsEqualIID(riid, IID_NULL))
      return ResultFromScode(DISP_E_UNKNOWNINTERFACE);

    return SPolyGetIDsOfNames(
      rgmdCPoly, DIM(rgmdCPoly), rgszNames, cNames, lcid, rgdispid);
}


/***
*HRESULT CPoly::Invoke(...)
*Purpose:
*  Dispatch a method or property request for objects of type CPoly.
*
*  see the IDispatch document for more information, and a general
*  description of this method.
*
*Entry:
*  dispidMember = the DISPID of the member being requested
*
*  riid = reference to the interface ID of the interface on this object
*    that the requested member belongs to. IID_NULL means to interpret
*    the member as belonging to the implementation defined "default"
*    or "primary" interface.
*
```

```
*  lcid = the caller's locale ID
*
*  wFlags = flags indicating the type of access being requested
*
*  pdispparams = pointer to the DISPPARAMS struct containing the
*    requested members arguments (if any) and its named parameter
*    DISPIDs (if any).
*
*Exit:
*  return value = HRESULT
*   see the IDispatch spec for a description of possible success codes.
*
*  pvarResult = pointer to a caller allocated VARIANT containing
*    the members return value (if any).
*
*  pexcepinfo = caller allocated exception info structure, this will
*    be filled in only if an exception was raised that must be passed
*    up through Invoke to an enclosing handler.
*
*  puArgErr = pointer to a caller allocated UINT, that will contain the
*    index of the offending argument if a DISP_E_TYPEMISMATCH error
*    was returned indicating that one of the arguments was of an
*    incorrect type and/or could not be reasonably coerced to a proper
*    type.
*
**********************************************************************/
STDMETHODIMP
CPoly::Invoke(
    DISPID dispidMember,
    REFIID riid,
    LCID lcid,
    unsigned short wFlags,
    DISPPARAMS FAR* pdispparams,
    VARIANT FAR* pvarResult,
    EXCEPINFO FAR* pexcepinfo,
    unsigned int FAR* puArgErr)
{
    HRESULT hresult;
    VARIANTARG varg0, varg1;
    VARIANT varResultDummy;

    UNUSED(lcid);
    UNUSED(pexcepinfo);

    if(wFlags & ~(DISPATCH_METHOD | DISPATCH_PROPERTYGET |
DISPATCH_PROPERTYPUT | DISPATCH_PROPERTYPUTREF))
      return ResultFromScode(E_INVALIDARG);

    // this object only exposes a "default" interface.
    //
    if(!IsEqualIID(riid, IID_NULL))
      return ResultFromScode(DISP_E_UNKNOWNINTERFACE);

    // This makes the following code a bit simpler if the caller
    // happens to be ignoring the return value. Some implementations
    // may choose to deal with this differently.
    //
    if(pvarResult == (VARIANT FAR*)NULL)
      pvarResult = &varResultDummy;

    VariantInit(&varg0);
    VariantInit(&varg1);

    // assume the return type is void, unless we find otherwise.
    VariantInit(pvarResult);

    switch(dispidMember){
    case IDMEMBER_CPOLY_DRAW:
      Draw();
      break;

    case IDMEMBER_CPOLY_RESET:
      Reset();
      break;

    case IDMEMBER_CPOLY_DUMP:
      Dump();
      break;

    case IDMEMBER_CPOLY_ADDPOINT:
      hresult = DispGetParam(pdispparams, 0, VT_I2, &varg0, puArgErr);
      if(hresult != NOERROR)
     return hresult;

      hresult = DispGetParam(pdispparams, 1, VT_I2, &varg1, puArgErr);
      if(hresult != NOERROR)
     return hresult;

      hresult = AddPoint(V_I2(&varg1), V_I2(&varg0));
      if(hresult != NOERROR)
     return hresult;
      break;

    case IDMEMBER_CPOLY_ENUMPOINTS:
      IEnumVARIANT FAR* penum;

      hresult = EnumPoints(&penum);
      if(hresult != NOERROR)
     return hresult;

      V_VT(pvarResult) = VT_UNKNOWN;
      hresult = penum->QueryInterface(
     IID_IUnknown, (void FAR* FAR*)&V_UNKNOWN(pvarResult));
      if(hresult != NOERROR)
     return hresult;
      penum->Release();
      break;

    case IDMEMBER_CPOLY_GETXORIGIN:
      V_VT(pvarResult) = VT_I2;
```

```
    V_I2(pvarResult) = m_xorg;
    break;

  case IDMEMBER_CPOLY_SETXORIGIN:
    hresult = DispGetParam(pdispparams, 0, VT_I2, &varg0, puArgErr);
    if(hresult != NOERROR)
      return hresult;
    m_xorg = V_I2(&varg0);
    break;

  case IDMEMBER_CPOLY_GETYORIGIN:
    V_VT(pvarResult) = VT_I2;
    V_I2(pvarResult) = m_yorg;
    break;

  case IDMEMBER_CPOLY_SETYORIGIN:
    hresult = DispGetParam(pdispparams, 0, VT_I2, &varg0, puArgErr);
    if(hresult != NOERROR)
      return hresult;
    m_yorg = V_I2(&varg0);
    break;

  case IDMEMBER_CPOLY_GETWIDTH:
    V_VT(pvarResult) = VT_I2;
    V_I2(pvarResult) = GetWidth();
    break;

  case IDMEMBER_CPOLY_SETWIDTH:
    hresult = DispGetParam(pdispparams, 0, VT_I2, &varg0, puArgErr);
    if(hresult != NOERROR)
      return hresult;
    SetWidth(V_I2(&varg0));
    break;

  case IDMEMBER_CPOLY_GETRED:
    V_VT(pvarResult) = VT_I2;
    V_I2(pvarResult) = get_red();
    break;

  case IDMEMBER_CPOLY_SETRED:
    hresult = DispGetParam(pdispparams, 0, VT_I2, &varg0, puArgErr);
    if(hresult != NOERROR)
      return hresult;
    set_red(V_I2(&varg0));
    break;

  case IDMEMBER_CPOLY_GETGREEN:
    V_VT(pvarResult) = VT_I2;
    V_I2(pvarResult) = get_green();
    break;

  case IDMEMBER_CPOLY_SETGREEN:
    hresult = DispGetParam(pdispparams, 0, VT_I2, &varg0, puArgErr);
    if(hresult != NOERROR)
```

```
        return hresult;
      set_green(V_I2(&varg0));
      break;

    case IDMEMBER_CPOLY_GETBLUE:
      V_VT(pvarResult) = VT_I2;
      V_I2(pvarResult) = get_blue();
      break;

    case IDMEMBER_CPOLY_SETBLUE:
      hresult = DispGetParam(pdispparams, 0, VT_I2, &varg0, puArgErr);
      if(hresult != NOERROR)
        return hresult;
      set_blue(V_I2(&varg0));
      break;

    default:
      return ResultFromScode(DISP_E_MEMBERNOTFOUND);
    }

    return NOERROR;
}


//---------------------------------------------------------------------
//                     Introduced Methods
//---------------------------------------------------------------------


/***
*void CPoly::Draw(void)
*Purpose:
*  Draw the polygon, using the current x/y origin and line width
*  properties.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
void PASCAL
CPoly::Draw()
{
    short xorg, yorg;
    POINTLINK FAR* ppointlinkFirst, FAR* ppointlink;

    if((ppointlinkFirst = m_ppointlink) == (POINTLINK FAR*)NULL)
      return;

#ifdef _MAC /* { */

    short x, y;
    RGBColor rgb;
    WindowPtr pwndSaved;

    GetPort(&pwndSaved);
    SetPort(g_pwndClient);

    PenNormal();
    PenSize(m_width, m_width);

    rgb.red = m_red;
    rgb.green = m_green;
    rgb.blue = m_blue;
    RGBForeColor(&rgb);

    xorg = m_xorg;
    yorg = m_yorg;

    MoveTo(
      xorg + ppointlinkFirst->ppoint->m_x,
      yorg + ppointlinkFirst->ppoint->m_y);

    for(ppointlink = ppointlinkFirst->next;
     ppointlink != (POINTLINK FAR*)NULL;
     ppointlink = ppointlink->next)
    {
      x = xorg + ppointlink->ppoint->m_x;
      y = yorg + ppointlink->ppoint->m_y;
      LineTo(x, y);
    }

    LineTo(
      xorg + ppointlinkFirst->ppoint->m_x,
      yorg + ppointlinkFirst->ppoint->m_y);

    SetPort(pwndSaved);

#else /* }{ */


    HDC hdc;
    RECT rect;
    HPEN hpen, hpenOld;
extern HWND g_hwndClient;

    GetClientRect(g_hwndClient, &rect);
    xorg = m_xorg + (short) rect.left;
    yorg = m_yorg + (short) rect.top;

    hdc = GetDC(g_hwndClient);
    hpen = CreatePen(PS_SOLID, m_width, RGB(m_red, m_green, m_blue));
    hpenOld = SelectObject(hdc, hpen);

#ifdef WIN32
    MoveToEx(hdc,
      xorg + ppointlinkFirst->ppoint->m_x,
      yorg + ppointlinkFirst->ppoint->m_y, NULL);
```

```
#else
    MoveTo(hdc,
      xorg + ppointlinkFirst->ppoint->m_x,
      yorg + ppointlinkFirst->ppoint->m_y);
#endif

    for(ppointlink = ppointlinkFirst->next;
     ppointlink != (POINTLINK FAR*)NULL;
     ppointlink = ppointlink->next)
    {
      LineTo(hdc,
     xorg + ppointlink->ppoint->m_x,
     yorg + ppointlink->ppoint->m_y);
    }

    LineTo(hdc,
      xorg + ppointlinkFirst->ppoint->m_x,
      yorg + ppointlinkFirst->ppoint->m_y);

    SelectObject(hdc, hpenOld);
    DeleteObject(hpen);

    ReleaseDC(g_hwndClient, hdc);

#endif /* } */
}


/***
*void CPoly::Reset(void)
*Purpose:
*  Release all points referenced by this poly.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
void PASCAL
CPoly::Reset()
{
    POINTLINK FAR* ppointlink, FAR* ppointlinkNext;

    for(ppointlink = m_ppointlink;
     ppointlink != (POINTLINK FAR*)NULL;
     ppointlink = ppointlinkNext)
    {
      ppointlinkNext = ppointlink->next;
      ppointlink->ppoint->Release();
      delete ppointlink;
    }

    m_cPoints = 0;
    m_ppointlink = NULL;
    m_ppointlinkLast = NULL;
}


/***
*HRESULT CPoly::AddPoint(short, short)
*Purpose:
*  Add a CPoint with the given coordinates to the end of our current
*  list of points.
*
*Entry:
*  x,y = the x and y coordinates of the new point.
*
*Exit:
*  return value = HRESULT
*
***********************************************************************/
HRESULT PASCAL
CPoly::AddPoint(short x, short y)
{
    CPoint FAR* ppoint;
    POINTLINK FAR* ppointlink;

    ppoint = CPoint::Create();
    if(ppoint == (CPoint FAR*)NULL)
      return ResultFromScode(E_OUTOFMEMORY);

    ppoint->SetX(x);
    ppoint->SetY(y);

    ppointlink = new FAR POINTLINK;
    if(ppointlink == (POINTLINK FAR*)NULL){
      delete ppoint;
      return ResultFromScode(E_OUTOFMEMORY);
    }

    ppointlink->ppoint = ppoint;
    ppointlink->next = (POINTLINK FAR*)NULL;

    if(m_ppointlinkLast == (POINTLINK FAR*)NULL){
      m_ppointlink = m_ppointlinkLast = ppointlink;
    }else{
      m_ppointlinkLast->next = ppointlink;
      m_ppointlinkLast = ppointlink;
    }

    ++m_cPoints;

    return NOERROR;
}


/***
*HRESULT CPoly::EnumPoints(IEnumVARIANT**);
```

```
*Purpose:
*  Return and enumerator for the points in this polygon.
*
*Entry:
*  None
*
*Exit:
*  return value = HRESULT
*
*  *ppenum = pointer to an IEnumVARIANT for the points in this polygon
*
***********************************************************************/
HRESULT PASCAL
CPoly::EnumPoints(IEnumVARIANT FAR* FAR* ppenum)
{
    unsigned int i;
    VARIANT var;
    HRESULT hresult;
    SAFEARRAY FAR* psa;
    CEnumPoint FAR* penum;
    POINTLINK FAR* ppointlink;
    SAFEARRAYBOUND rgsabound[1];

    rgsabound[0].lLbound = 0;
    rgsabound[0].cElements = m_cPoints;

    psa = SafeArrayCreate(VT_VARIANT, 1, rgsabound);
    if(psa == NULL){
      hresult = ResultFromScode(E_OUTOFMEMORY);
      goto LError0;
    }

    ppointlink = m_ppointlink;
    for(i = 0; i < m_cPoints; ++i){
      long ix[1];

      if(ppointlink == NULL){
        // this indicates an internal consistency error.
     // (this test should probably be an assertion)
        hresult = ResultFromScode(E_FAIL);
        goto LError1;
      }

      V_VT(&var) = VT_DISPATCH;
      hresult = ppointlink->ppoint->QueryInterface(
     IID_IDispatch, (void FAR* FAR*)&V_DISPATCH(&var));
      if(hresult != NOERROR)
        goto LError1;

      ix[0] = i;
      SafeArrayPutElement(psa, ix, &var);

      ppointlink = ppointlink->next;
    }
    hresult = CEnumPoint::Create(psa, &penum);
    if(hresult != NOERROR)
      goto LError1;

    *ppenum = penum;

    return NOERROR;

LError1:;
    // destroy the array if we were not successful creating the enumerator.
    SafeArrayDestroy(psa);

LError0:;
    return hresult;
}

short PASCAL
CPoly::GetXOrigin()
{
    return m_xorg;
}

void PASCAL
CPoly::SetXOrigin(short x)
{
    m_xorg = x;
}

short PASCAL
CPoly::GetYOrigin()
{
    return m_yorg;
}

void PASCAL
CPoly::SetYOrigin(short y)
{
    m_yorg = y;
}

short PASCAL
CPoly::GetWidth()
{
    return m_width;
}

void PASCAL
CPoly::SetWidth(short width)
{
    m_width = width;
}

short PASCAL
CPoly::get_red()
```

```
{
    return m_red;
}

void PASCAL
CPoly::set_red(short red)
{
    m_red = red;
}

short PASCAL
CPoly::get_green()
{
    return m_green;
}

void PASCAL
CPoly::set_green(short green)
{
    m_green = green;
}

short PASCAL
CPoly::get_blue()
{
    return m_blue;
}

void PASCAL
CPoly::set_blue(short blue)
{
    m_blue = blue;
}


/***
*void CPoly::Dump(void)
*Purpose:
*  Output a debug dump of this instance.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
void PASCAL
CPoly::Dump()
{
#ifdef _MAC

    // REVIEW: implement for the mac

#else
```

```
    TCHAR buffer[80];
    POINTLINK FAR* ppointlink;

    wsprintf(buffer, TSTR("CPoly(0x%x) =\n"), (int)this);
    OutputDebugString(buffer);

    wsprintf(buffer,
      TSTR("    xorg = %d, yorg = %d, width = %d, rgb = {%d,%d,%d}\n
points = "),
      m_xorg, m_yorg, m_width,
      get_red(),
      get_green(),
      get_blue());

    OutputDebugString(buffer);

    for(ppointlink = m_ppointlink;
     ppointlink != (POINTLINK FAR*)NULL;
     ppointlink = ppointlink->next)
    {
      wsprintf(buffer, TSTR("{%d,%d}"),
        ppointlink->ppoint->GetX(),
        ppointlink->ppoint->GetY());
      OutputDebugString(buffer);

      wsprintf(buffer, TSTR(" "));
      OutputDebugString(buffer);
    }
    wsprintf(buffer, TSTR("\n"));
    OutputDebugString(buffer);

#endif
}

/***
*void CPoly::PolyDraw(void)
*Purpose:
*  Draw all polygons.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
void
CPoly::PolyDraw()
{
    POLYLINK FAR* polylink;

    for(polylink = g_ppolylink;
     polylink != (POLYLINK FAR*)NULL;
     polylink = polylink->next)
    {
      polylink->ppoly->Draw();
    }
}


/***
*void PolyTerm(void)
*Purpose:
*  Release all polygons.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
void
CPoly::PolyTerm()
{
    POLYLINK FAR* ppolylink;
    POLYLINK FAR* ppolylinkNext;

    for(ppolylink = g_ppolylink;
     ppolylink != (POLYLINK FAR*)NULL;
     ppolylink = ppolylinkNext)
    {
      ppolylinkNext = ppolylink->next;
      ppolylink->ppoly->Release();
      delete ppolylink;
    }
    g_ppolylink = NULL;
}


/***
*void PolyDump(void)
*Purpose:
*  Invoke the debug Dump() method on all polygons were currently
*  holding on to.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
void
CPoly::PolyDump()
{
    POLYLINK FAR* ppolylink;
    if(g_ppolylink == (POLYLINK FAR*)NULL){
#ifndef _MAC
      OutputDebugString(TSTR("\t(none)\n"));
#endif
      return;
    }

    for(ppolylink = g_ppolylink;
     ppolylink != (POLYLINK FAR*)NULL;
     ppolylink = ppolylink->next)
    {
      ppolylink->ppoly->Dump();
    }
}


//----------------------------------------------------------------------
//             Implementation of the CPoly Class Factory
//----------------------------------------------------------------------


CPolyCF::CPolyCF()
{
    m_refs = 0;
}

IClassFactory FAR*
CPolyCF::Create()
{
    CPolyCF FAR* pCF;

    if((pCF = new FAR CPolyCF()) == NULL)
      return NULL;
    pCF->AddRef();
    return pCF;
}

STDMETHODIMP
CPolyCF::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    if(!IsEqualIID(riid, IID_IUnknown)){
      if(!IsEqualIID(riid, IID_IClassFactory)){
     *ppv = NULL;
        return ResultFromScode(E_NOINTERFACE);
      }
    }

    *ppv = this;
    ++m_refs;
    return NOERROR;
}

STDMETHODIMP_(unsigned long)
CPolyCF::AddRef(void)
```

```
{
    return ++m_refs;
}

STDMETHODIMP_(unsigned long)
CPolyCF::Release(void)
{
    if(--m_refs == 0){
      delete this;
      return 0;
    }
    return m_refs;
}

STDMETHODIMP
CPolyCF::CreateInstance(
    IUnknown FAR* punkOuter,
    REFIID iid,
    void FAR* FAR* ppv)
{
    HRESULT hresult;
    CPoly FAR *ppoly;

    UNUSED(punkOuter);

#ifdef _MAC
    if(GetZone() != ApplicZone()){
#ifndef _MSC_VER
#ifndef ConstStr255Param
#define ConstStr255Param StringPtr
#endif
#endif
      DebugStr((ConstStr255Param)"\pZones do not match");
    }

#endif

    if((ppoly = CPoly::Create()) == NULL){
      *ppv = NULL;
      return ResultFromScode(E_OUTOFMEMORY);
    }
    hresult = ppoly->QueryInterface(iid, ppv);
    ppoly->Release();
    return hresult;
}

STDMETHODIMP
#ifdef _MAC
CPolyCF::LockServer(unsigned long fLock)
#else
CPolyCF::LockServer(BOOL fLock)
#endif
{
    UNUSED(fLock);
    return NOERROR;
}
```

HOSTENV.H (SPOLY Sample)

```
/***
*hostenv.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*  Generic host specific includes.
*
*Implementation Notes:
*
*****************************************************************************
*/

#if defined(_MAC)

#if defined(_PPCMAC)
#pragma data_seg("_FAR_DATA")
#pragma data_seg( )
#define MAXLONG 0x7fffffff
#define EventHandlerProcPtr AEEventHandlerUPP
#else //_PPCMAC
#define  GetMenuItemText(mApple,menuItem,daName)
GetItem(mApple,menuItem,daName)
#endif //_PPCMAC

#endif //_MAC

#if defined(_MAC)

#if defined(_MSC_VER)

# include <values.h>
# include <types.h>
# include <string.h>
# include <quickdra.h>
# include <fonts.h>
# include <events.h>
# include <resource.h>
# include <menus.h>
# include <textedit.h>
# include <dialogs.h>
# include <desk.h>
# include <toolutil.h>
# include <memory.h>
# include <files.h>
# include <osutils.h>
# include <osevents.h>
# include <diskinit.h>
# include <packages.h>
# include <traps.h>
# include <AppleEve.h>
```

```
#else //_MSC_VER

# include <values.h>
# include <types.h>
# include <strings.h>
# include <quickdraw.h>
# include <fonts.h>
# include <events.h>
# include <resources.h>
# include <windows.h>
# include <menus.h>
# include <textedit.h>
# include <dialogs.h>
# include <desk.h>
# include <toolutils.h>
# include <memory.h>
# include <files.h>
# include <osutils.h>
# include <osevents.h>
# include <diskinit.h>
# include <packages.h>
# include <traps.h>
# include <AppleEvents.h>

#endif //_MSC_VER

# include <ole2.h>
# include <olenls.h>
# include <dispatch.h>

# define TCHAR                char
# define TSTR(str)            str
# define STRING(str)            (str)
# define WIDESTRING(str)         (str)

#elif defined(WIN32)

# include <windows.h>
# include <ole2.h>
# include <oleauto.h>

# if defined(UNICODE)
    #define TCHAR      WCHAR
    #define TSTR(str)        L##str
    #define STRING(str)        (str)
    #define WIDESTRING(str)  (str)
# else
    #define TCHAR      char
    #define TSTR(str)        str
    #define STRING(str)        AnsiString(str)
    #define WIDESTRING(str)  WideString(str)
    extern "C" char FAR* AnsiString(OLECHAR FAR* strIn);
    extern "C" OLECHAR FAR* WideString(char FAR* strIn);
# endif
```

```
#else /* WIN16 */

# include <windows.h>
# include <ole2.h>
# include <olenls.h>
# include <dispatch.h>

# define TCHAR               char
# define TSTR(str)           str
# define STRING(str)           (str)
# define WIDESTRING(str)        (str)
#endif
```

MISC.CPP   (SPOLY Sample)

```
/***
*misc.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*
*Implementation Notes:
*
*****************************************************************************
*/



#include "spoly.h"
#include "cpoint.h"
#include "cpoly.h"

#include <stdio.h>

#if _MAC
# include <string.h>
# include <ctype.h>
#endif

#ifdef WIN32  // Use native CompareString operation
  #undef CompareString
  #ifdef UNICODE
    #define CompareString CompareStringW
  #else
    // CONSIDER: write a wrapper for CompareStringW if not available
  #endif
#else
  #define CompareString CompareStringA
#endif

unsigned long g_dwPolyCF = 0;
unsigned long g_dwPointCF = 0;

IClassFactory FAR* g_ppolyCF = NULL;
IClassFactory FAR* g_ppointCF = NULL;

#ifdef _MAC
struct regentry{
    char *szKey;
    char *szValue;
} g_rgregentry[] = {

      { "CLSID\\{00020462-0000-0000-C000-000000000046}",
     "OLE Automation SPoly 1.0 Application" }

    , { "CLSID\\{00020462-0000-0000-C000-000000000046}\\LocalServer",
     "SPLy" }

    , { "CLSID\\{00020462-0000-0000-C000-000000000046}\\ProgID",
     "SPoly.Application" }

    , { "CLSID\\{00020462-0000-0000-C000-000000000046}\\InprocHandler",
     "OLE2:Def$DefFSet" }

    , { "SPLy", "{00020462-0000-0000-C000-000000000046}" }

    , { "SPoly.Application\\CLSID",
     "{00020462-0000-0000-C000-000000000046}" }

};

HRESULT
EnsureRegistration()
{
    HKEY hkey;

    if(RegOpenKey(HKEY_CLASSES_ROOT, "SPLy", &hkey) == NOERROR){
      RegCloseKey(hkey);
      return NOERROR;
    }

    for(int i = 0; i < DIM(g_rgregentry); ++i){
      if(RegSetValue(HKEY_CLASSES_ROOT, g_rgregentry[i].szKey, REG_SZ,
g_rgregentry[i].szValue, 0) != ERROR_SUCCESS)
        return ResultFromScode(E_FAIL);
    }

    return NOERROR;
}
#endif


/***
*HRESULT InitOle(void)
*Purpose:
*  Initialize Ole, and register our class factories.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
STDAPI
InitOle()
{
    HRESULT hresult;

    if((hresult = OleInitialize(NULL)) != NOERROR)
      goto LError0;
```

```
#ifdef _MAC
    if((hresult = EnsureRegistration()) != NOERROR)
      goto LError0;
#endif

    // Register the CPoint Class Factory
    //
    if((g_ppointCF = CPointCF::Create()) == NULL){
      hresult = ResultFromScode(E_OUTOFMEMORY);
      goto LError1;
    }

    hresult = CoRegisterClassObject(
      CLSID_CPoint,
      g_ppointCF,
      CLSCTX_LOCAL_SERVER,
      REGCLS_MULTIPLEUSE,
      &g_dwPointCF);
    if(hresult != NOERROR)
      goto LError1;

    // Register the CPoly Class Factory.
    //
    if((g_ppolyCF = CPolyCF::Create()) == NULL){
      hresult = ResultFromScode(E_OUTOFMEMORY);
      goto LError1;
    }

    hresult = CoRegisterClassObject(
      CLSID_CPoly,
      g_ppolyCF,
      CLSCTX_LOCAL_SERVER,
      REGCLS_MULTIPLEUSE,

      &g_dwPolyCF);
    if(hresult != NOERROR)
      goto LError1;

    g_ppolyCF->Release();

    g_ppointCF->Release();

    return NOERROR;


LError1:;
    if(g_ppolyCF != NULL)
      g_ppolyCF->Release();

    if(g_ppointCF != NULL)
      g_ppointCF->Release();

    UninitOle();
```

```
LError0:;
    return hresult;
}

STDAPI
UninitOle()
{
    // Tell Ole to release our class factories.
    //
    if(g_dwPointCF != 0L)
      CoRevokeClassObject(g_dwPointCF);

    if(g_dwPolyCF != 0L)
      CoRevokeClassObject(g_dwPolyCF);

    OleUninitialize();

    return NOERROR;
}

/***
*HRESULT SPolyGetIDsOfNames(MEMBERDESC*, unsigned int, char**, unsigned int,
LCID, long*)
*Purpose:
*  This is the table driven implementation of IDispatch::GetIDsOfNames
*  deferred to by both the CPoint and CPoly objects.
*
*Entry:
*  rgmd = pointer to an array of method descriptors
*  cMethods = number of elements in the array of method descriptors
*  rgszNames = pointer to an array of names
*  cNames = the number of names in the rgszNames array
*  lcid = the callers locale ID
*
*Exit:
*  return value = HRESULT
*  rgdispid = array of name IDs corresponding to the rgszNames array
*    this array will contain -1 for each entry that is not known.
*
***********************************************************************/
STDAPI
SPolyGetIDsOfNames(
    MEMBERDESC FAR* rgmd,
    unsigned int cMethods,
    OLECHAR FAR* FAR* rgszNames,
    unsigned int cNames,
    LCID lcid,
    DISPID FAR* rgdispid)
{
    HRESULT hresult;
    PARAMDESC FAR* ppd;
    MEMBERDESC FAR* pmd;
    unsigned int iName, iTry, cParams;

    hresult = NOERROR;
    // first lookup the member name.
    //
    for(pmd = rgmd;; ++pmd){
      if(pmd == &rgmd[cMethods])
        goto LMemberNotFound;

#if defined(WIN32) && !defined(UNICODE)
      // CompareStringW is not available on Chicago
      if(wcsicmp(rgszNames[0], pmd->szName) == 0)
#else
      if(CompareString(lcid, NORM_IGNORECASE, rgszNames[0], -1, pmd->szName,
-1) == 2)
#endif
      {
     rgdispid[0] = pmd->id;
     break;
      }
    }

    // Lookup the named parameters, if there are any.
    if(cNames > 1){
      cParams = pmd->cParams;
      for(iName = 1; iName < cNames; ++iName){
     for(iTry = 0;; ++iTry){
       if(iTry == cParams){
         hresult = ResultFromScode(DISP_E_UNKNOWNNAME);
         rgdispid[iName] = -1;
         break;
       }
       ppd = &pmd->rgpd[iTry];

#if defined(WIN32) && !defined(UNICODE)
          // CompareStringW is not available on Chicago
          if(wcsicmp(rgszNames[iName], ppd->szName) == 0)
#else
       if(CompareString(lcid, NORM_IGNORECASE, rgszNames[iName], -1, ppd-
>szName, -1) == 2)
#endif
       {
         // The DISPID for a named parameter is defined to be its
         // zero based positional index.  This routine assumes that
         // that PARAMDESC array was declared in correct textual order.
         //
         rgdispid[iName] = iTry;
         break;
       }
        }
      }
    }

    return hresult;

LMemberNotFound:;
    // If the member name is unknown, everything is unknown.
    for(iName = 0; iName < cNames; ++iName)
      rgdispid[iName] = -1;
    return ResultFromScode(DISP_E_UNKNOWNNAME);
}


// disable unicode expansion for assertions
#undef UNICODE

void
Assert(int fCond, char FAR* file, int line, char FAR* message)
{
    char * fmt;
    char buf[128];

    if(fCond)
      return;

    fmt = (message == NULL)
      ? "Assertion failed: %s(%d)"
      : "Assertion failed: %s(%d) '%s'";
    sprintf(buf, fmt, file, line, message);

#ifdef _MAC
    DebugStr(c2pstr(buf));
#else
#ifdef WIN32
    OutputDebugStringA(buf);
#else //WIN32
    OutputDebugString(buf);
#endif //WIN32
    DebugBreak();
#endif
}
```

RESOURCE.H   (SPOLY Sample)

```
/***
*resource.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*
*Implementation Notes:
*
****************************************************************************
*/

#ifdef _MAC

#define kMinSize 500   /* minimum size (in K) */
#define kPrefSize500   /* preferred size (in K) */

#define    rMenuBar    128   /* menu bar */
#define    rAboutAlert 128   /* about alert */
#define    rUserAlert  129   /* error alert */
#define    rWindow           128   /* application's window */
#define rIcon          128

#define    mApple            128   /* Apple menu */
#define    iAbout            1

#define    mFile       129   /* File menu */
#define    iNew        1
#define    iClose            4
#define    iQuit       12

#define    mEdit       130   /* Edit menu */
#define    iUndo       1
#define    iCut        3
#define    iCopy       4
#define    iPaste            5
#define    iClear            6

#define mSpoly         131
#define iTest          1

#define kMinHeap 21 * 1024
#define kMinSpace8 * 1024

#else /* WIN16 || WIN32 */

# define IDM_CLEAR     1
# define IDM_DUMP2
# define IDM_FIRSTCHILD100

#endif
```

SPOLY.DEF (SPOLY Sample)

```
NAME         SPOLY

DESCRIPTION  'IDispatch Polygon Test Server'

EXETYPE      WINDOWS

STUB         'WINSTUB.EXE'

CODE         PRELOAD MOVEABLE DISCARDABLE
DATA         PRELOAD MULTIPLE

HEAPSIZE     4096
STACKSIZE    8192
```

SPOLY.H   (SPOLY Sample)

```
/***
*spoly.h - Application-wide definitions
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*
*Implementation Notes:
*
****************************************************************************
*/

#include "hostenv.h"
#include "resource.h"
#include "clsid.h"

#if defined(_MAC)
# define PASCAL pascal
# define STRSTR strstr
#elif defined(WIN32)
# include "statbar.h"
# define STRSTR strstr
#else /* WIN16 */
# include "statbar.h"
# define STRSTR _fstrstr
#endif

#ifdef _MAC
# define UNUSED(X) ((void)(void*)&(X))
#else
# define UNUSED(X) (X)
#endif

#define DIM(X) (sizeof(X) / sizeof(X[0]))

extern "C" void Assert(int, char FAR*, int, char FAR*);
#define ASSERT(X) Assert(X, __FILE__, __LINE__, NULL)
#define ASSERTSZ(X, MSG) Assert(X, __FILE__, __LINE__, MSG)



// Description of a single named parameter.
//
typedef struct tagPARAMDESC {
    OLECHAR FAR* szName;           // parameter name
} PARAMDESC;

// Description of a single member.
//
typedef struct tagMEMBERDESC {
    OLECHAR FAR* szName;           // member name
    DISPID id;               // member id
    PARAMDESC FAR* rgpd;     // ptr to array of PARAMDESCs
    unsigned int cParams;          // number of parameters
} MEMBERDESC;

STDAPI
SPolyGetIDsOfNames(
    MEMBERDESC FAR* pmd,
    unsigned int cMethods,
    OLECHAR FAR* FAR* rgszNames,
    unsigned int cNames,
    LCID lcid,
    DISPID FAR* rgid);


STDAPI InitOle();
STDAPI UninitOle();
```

SPOLY.RC   (SPOLY Sample)

```
#define NOKERNEL
#define NOGDI
#define NOSOUND
#define NOCOMM
#define NODRIVERS
#include "windows.h"
#include "spoly.h"

SPOLY ICON spoly.ico

SPolyMenu MENU
BEGIN
    MENUITEM     "&Clear"                IDM_CLEAR
    MENUITEM     "&Dump"                       IDM_DUMP
END
```

SPOLY.REG (SPOLY Sample)

```
REGEDIT


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; registration info SPoly.Application (defaults to SPoly.Application.1)

HKEY_CLASSES_ROOT\SPoly.Application = OLE Automation SPoly Application
HKEY_CLASSES_ROOT\SPoly.Application\Clsid = {00020462-0000-0000-C000-
000000000046}



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; registration info SPoly 1.0

; (Application Object)
HKEY_CLASSES_ROOT\SPoly.Application.1 = OLE Automation SPoly 1.0 Application
HKEY_CLASSES_ROOT\SPoly.Application.1\Clsid = {00020462-0000-0000-C000-
000000000046}

HKEY_CLASSES_ROOT\CLSID\{00020462-0000-0000-C000-000000000046} = OLE
Automation SPoly 1.0 Application
HKEY_CLASSES_ROOT\CLSID\{00020462-0000-0000-C000-000000000046}\ProgID =
SPoly.Application.1
HKEY_CLASSES_ROOT\CLSID\{00020462-0000-0000-C000-
000000000046}\VersionIndependentProgID = SPoly.Application
HKEY_CLASSES_ROOT\CLSID\{00020462-0000-0000-C000-000000000046}\LocalServer32
= spoly.exe /Automation

HKEY_CLASSES_ROOT\CLSID\{00020463-0000-0000-C000-000000000046} =
SPoly.SPoint
HKEY_CLASSES_ROOT\CLSID\{00020463-0000-0000-C000-000000000046}\ProgID =
SPoly.Application.1
HKEY_CLASSES_ROOT\CLSID\{00020463-0000-0000-C000-
000000000046}\VersionIndependentProgID = SPoly.Application
HKEY_CLASSES_ROOT\CLSID\{00020463-0000-0000-C000-000000000046}\LocalServer32
= spoly.exe /Automation
```

STATBAR.H   (SPOLY Sample)

```
/***
*statbar.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*
*Implementation Notes:
*  This file requires windows.h and ole2.h
*
*****************************************************************************
*/

class CStatBar : public IUnknown {
public:
    static CStatBar FAR* Create(HANDLE hinst, HWND hwndFrame);

    // IUnknown methods
    //
    STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppv);
    STDMETHOD_(unsigned long, AddRef)(void);
    STDMETHOD_(unsigned long, Release)(void);

    // Introduced methods
    //
    void Show(void);
    inline void Move(void);
    inline void Update(void);

    inline int GetX(void);
    inline void SetX(int x);

    inline int GetY(void);
    inline void SetY(int y);

    inline int GetHeight(void);
    inline void SetHeight(int height);

    inline int GetWidth(void);
    inline void SetWidth(int width);

    //inline HFONT GetFont(void);
    void SetFont(HFONT hfont);

    //char FAR* GetText(void);
    inline void SetText(OLECHAR FAR* sz);

    void WMPaint(void);
    BOOL Register(HANDLE);

private:
    CStatBar();
    ~CStatBar();

    unsigned longm_refs;

    HWND   m_hwnd;                 // the status bar window handle

    int          m_x;              // x coordinate of upper left corner
    int          m_y;              // y coordinate of upper left corner
    int          m_height;
    int          m_width;

    HFONT  m_hfont;
    int          m_dyFont;         // font height
    int          m_dxFont;         // font width

    BSTR   m_bstrMsg;        // the status bar text

    static TCHAR FAR* m_szWndClass;
};

inline void
CStatBar::Move()
{
    MoveWindow(m_hwnd, m_x, m_y, m_width, m_height, TRUE);
}

inline void
CStatBar::Update()
{
    InvalidateRect(m_hwnd, NULL, TRUE);
    UpdateWindow(m_hwnd);
}

inline int
CStatBar::GetX()
{
    return m_x;
}

inline void
CStatBar::SetX(int x)
{
    m_x = x;
}

inline int
CStatBar::GetY(void)
{
    return m_y;
}

inline void
CStatBar::SetY(int y)
{
    m_y = y;
}

inline int
CStatBar::GetHeight(void)
{
    return m_height;
}

inline void
CStatBar::SetHeight(int height)
{
    m_height = height;
}

inline int
CStatBar::GetWidth(void)
{
    return m_width;
}

inline void
CStatBar::SetWidth(int width)
{
    m_width = width;
}

inline void
CStatBar::SetText(OLECHAR FAR* sz)
{
    SysFreeString(m_bstrMsg);
    m_bstrMsg = SysAllocString(sz);
}

extern "C" void
SBprintf(CStatBar FAR* psb, TCHAR FAR* szFmt, ...);
```

STATBAR.CPP (SPOLY Sample)

```
/***
*statbar.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*
*Implementation Notes:
*
*****************************************************************************
*/

#include <stdarg.h>

#include "hostenv.h"
#include "statbar.h"


extern "C" long FAR PASCAL StatBarWndProc(HWND, unsigned int, WPARAM,
LPARAM);


TCHAR FAR* CStatBar::m_szWndClass = TSTR("StatBarWndClass");


CStatBar::CStatBar()
{
    m_refs = 0;

    m_x = 0;
    m_y = 0;
    m_width = 0;
    m_height = 0;

    m_bstrMsg = NULL;

    m_hfont = (HANDLE)0;
}

CStatBar::~CStatBar()
{
    SysFreeString(m_bstrMsg);
}


/***
*PUBLIC CStatBar FAR* CStatBar::Create(HANDLE, HWND)
*
*Purpose:
*
*Entry:
```

```
*
*Exit:
*
***********************************************************************/
CStatBar FAR*
CStatBar::Create(HANDLE hinst, HWND hwndFrame)
{
    CStatBar FAR* psb;

    psb = new FAR CStatBar();
    if(psb == NULL)
      return NULL;
    psb->AddRef();

    if(!psb->Register(hinst))
      goto LFail;

    psb->m_hwnd = CreateWindow(
      CStatBar::m_szWndClass,
      NULL,
      WS_CHILD | WS_CLIPSIBLINGS,
      0, 0, 0, 0,
      hwndFrame,
      0,
      hinst,
      NULL);

    if(!psb->m_hwnd)
      goto LFail;

    // Stash the newly created CStatBar* in the extra bytes of the
    // associated window so we can get at the instance in the message
    // proc.
    //
    // Note: we do not AddRef for this reference. We make sure that the
    // window is destroyed when the refcnt goes to 0.
    //
    SetWindowLong(psb->m_hwnd, 0, (long)psb);

    return psb;

LFail:;
    delete psb;
    return NULL;
}


//---------------------------------------------------------------------
//                     IUnknown Methods
//---------------------------------------------------------------------


STDMETHODIMP
CStatBar::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    if(IsEqualIID(riid,IID_IUnknown)){
      *ppv = this;
      AddRef();
      return NOERROR;
    }
    *ppv = (void FAR*)NULL;
    return ResultFromScode(E_NOINTERFACE);
}


STDMETHODIMP_(unsigned long)
CStatBar::AddRef(void)
{
    return ++m_refs;
}


STDMETHODIMP_(unsigned long)
CStatBar::Release(void)
{
    if(--m_refs == 0){

      // destroy the status bar window.
      //
      SendMessage(m_hwnd, WM_DESTROY, 0, 0L);

      delete this;
      return 0;
    }

    return m_refs;
}



//---------------------------------------------------------------------
//                     Introduced Methods
//---------------------------------------------------------------------


/***
*PRIVATE BOOL CStatBar::Register(HANDLE)
*
*Purpose:
*  Register the status bar window class.
*
*Entry:
*  None
*
*Exit:
*  return value = BOOL, TRUE if successful, FALSE if not.
*
***********************************************************************/
BOOL
CStatBar::Register(HANDLE hinst)
```

```
{
    WNDCLASS  wc;

    // register the class, unless already registered.
    if(GetClassInfo(hinst, m_szWndClass, &wc) == 0){
      wc.style         = 0;
      wc.lpfnWndProc   = StatBarWndProc;
      wc.cbClsExtra    = 0;
      wc.cbWndExtra    = sizeof(CStatBar FAR*);
      wc.hInstance     = hinst;
      wc.hIcon         = 0;
      wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
      wc.hbrBackground = GetStockObject(LTGRAY_BRUSH);
      wc.lpszMenuName  = 0;
      wc.lpszClassName = CStatBar::m_szWndClass;
      if(!RegisterClass(&wc))
        return FALSE;
    }
    return TRUE;
}


/***
*PUBLIC void CStatBar::Show(void)
*
*Purpose:
*  Show the status bar window associated with this CStatBar instance.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
void
CStatBar::Show()
{
    ShowWindow(m_hwnd, SW_SHOW);
}

void
CStatBar::SetFont(HFONT hfont)
{
    HDC hdc;
    TEXTMETRIC tm;
    HFONT hfontOld;

    // compute the character sizes given this new font.
    //
    hdc = GetDC(m_hwnd);
    hfontOld = SelectObject(hdc, hfont);
    GetTextMetrics(hdc, &tm);
    m_dxFont = tm.tmAveCharWidth;
    m_dyFont = tm.tmHeight + tm.tmExternalLeading;
    SelectObject(hdc, hfontOld);
    ReleaseDC(m_hwnd, hdc);

    m_hfont = hfont;
}

/***
*PRIVATE CStatBar::WMPaint(void)
*
*Purpose:
*  This method is responsible for drawing the status bar, and is called
*  in response to a WM_PAINT message.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
void
CStatBar::WMPaint()
{
    HDC hdc;
    RECT rcMsg;
    HRGN hrgn;
    HFONT hfontOld;
    PAINTSTRUCT ps;
    HPEN hpenBlack, hpenWhite, hpenGray, hpenOld;

    hdc = BeginPaint(m_hwnd, &ps);

    // compute the message box rect
    //
    rcMsg.top    = 3;
    rcMsg.bottom= m_height - 3;
    rcMsg.left   = m_dxFont;
    rcMsg.right  = m_width - m_dxFont;

    // prepare the pens
    //
    hpenWhite    = GetStockObject(WHITE_PEN);
    hpenBlack    = GetStockObject(BLACK_PEN);
    hpenGray     = CreatePen(PS_SOLID, 1, GetSysColor(COLOR_BTNSHADOW));

    // draw a top gray line
    //
    hpenOld = SelectObject(hdc, hpenGray);
#if   defined(WIN16)
    MoveTo(hdc, ps.rcPaint.left, 0);
#elif defined(WIN32)
    MoveToEx(hdc, ps.rcPaint.left, 0, NULL);
#endif
    LineTo(hdc, ps.rcPaint.right, 0);
    // draw a white line just under
    //
    SelectObject(hdc, hpenWhite);
#if   defined(WIN16)
    MoveTo(hdc, ps.rcPaint.left, 1);
#elif defined(WIN32)
    MoveToEx(hdc, ps.rcPaint.left, 1, NULL);
#endif
    LineTo(hdc, ps.rcPaint.right, 1);


    // do not overwrite the background color
    //
    SetBkMode(hdc, TRANSPARENT);

    // message area
    //
    SelectObject(hdc, hpenBlack);
#if   defined(WIN16)
    MoveTo(hdc, rcMsg.left,  rcMsg.bottom);
#elif defined(WIN32)
    MoveToEx(hdc, rcMsg.left,  rcMsg.bottom, NULL);
#endif
    LineTo(hdc, rcMsg.left,  rcMsg.top);
    LineTo(hdc, rcMsg.right, rcMsg.top);

    SelectObject(hdc, hpenWhite);
    LineTo(hdc, rcMsg.right, rcMsg.bottom);
    LineTo(hdc, rcMsg.left,  rcMsg.bottom);

    // select the black pen for writing
    //
    SelectObject(hdc, hpenBlack);

    // select the status bar font to write in
    //
    hfontOld = SelectObject(hdc, m_hfont);

    // set the clipping region
    //
    hrgn = CreateRectRgn(
      rcMsg.left, rcMsg.top, rcMsg.right, rcMsg.bottom);

    SelectClipRgn(hdc, hrgn);

    // draw the status message
    //
    TextOut(
      hdc,
      rcMsg.left + (m_dxFont / 2),
      rcMsg.top + ((rcMsg.bottom - rcMsg.top - m_dyFont) / 2),
      STRING(m_bstrMsg), (SysStringLen(m_bstrMsg)));

    // cleanup
    //
    SelectObject(hdc, hpenOld);
    SelectObject(hdc, hfontOld);

    DeleteObject(hrgn);
    DeleteObject(hpenGray);

    EndPaint(m_hwnd, &ps);
}

extern "C" long FAR PASCAL
StatBarWndProc(
    HWND hwnd,
    unsigned int message,
    WPARAM wParam,
    LPARAM lParam)
{
    CStatBar FAR* psb;

    switch(message){
    case WM_SIZE:
      return 0;
    case WM_PAINT:
      psb = (CStatBar FAR*)GetWindowLong(hwnd, 0);
      psb->WMPaint();
      return 0;
    }
    return(DefWindowProc(hwnd, message, wParam, lParam));
}


//---------------------------------------------------------------------
//                    Status Bar Utilities
//---------------------------------------------------------------------

extern "C" void
SBprintf(CStatBar FAR* psb, TCHAR FAR* szFmt, ...)
{
    va_list args;
static TCHAR buf[256];

    va_start(args, szFmt);
    wvsprintf(buf, szFmt, args);
    psb->SetText(WIDESTRING(buf));
    psb->Update();
}
```

WINMAIN.CPP   (SPOLY Sample)

```
/***
*winmain.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This module is the main entry point for the sample IDispatch polygon
*  server, spoly.exe.
*
*  This program is intended to demonstrate an implementation of the
IDispatch
*  interface. Spoly is a very simple app, that implements two simple
objects,
*  CPoly and CPoint and exposes their properties and methods for programatic
*  and cross-process access via IDispatch.
*
*Implementation Notes:
*
*****************************************************************************
*/

#include <stdio.h>
#include <string.h>

#include "spoly.h"
#include "cpoint.h"
#include "cpoly.h"


HANDLE g_hinst = 0;

HWND g_hwndFrame = 0;
HWND g_hwndClient = 0;

TCHAR g_szFrameWndClass[] = TSTR("FrameWClass");

CStatBar FAR* g_psb = NULL;


BOOL InitApplication(HANDLE);
BOOL InitInstance(HANDLE, int);

extern "C" int PASCAL WinMain(HANDLE, HANDLE, LPSTR, int);
extern "C" long FAR PASCAL FrameWndProc(HWND, UINT, WPARAM, LPARAM);


extern "C" int PASCAL
WinMain(
    HANDLE hinst,
    HANDLE hPrevInstance,
    LPSTR lpCmdLine,
    int nCmdShow)
{
    MSG msg;
    int retval;
    HRESULT hresult;

    if(!hPrevInstance)
      if(!InitApplication(hinst))
        return FALSE;

    if((hresult = InitOle()) != NOERROR)
      return FALSE;

    if(!InitInstance(hinst, nCmdShow)){
      retval = FALSE;
      UninitOle();
      goto LExit;
    }

    while(GetMessage(&msg, NULL, NULL, NULL)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }

    CPoly::PolyTerm();

    retval = msg.wParam;

LExit:;

    return retval;
}


BOOL
InitApplication(HANDLE hinst)
{
    WNDCLASS  wc;

    wc.style            = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc      = FrameWndProc;
    wc.cbClsExtra       = 0;
    wc.cbWndExtra       = 0;
    wc.hInstance        = hinst;
    wc.hIcon            = LoadIcon(hinst, TSTR("SPOLY"));
    wc.hCursor          = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground    = (HBRUSH) (COLOR_APPWORKSPACE+1);
    wc.lpszMenuName     = TSTR("SPolyMenu");
    wc.lpszClassName    = g_szFrameWndClass;

    if(!RegisterClass(&wc))
      return FALSE;

    return TRUE;
}
```

```
#ifdef WIN32
#define szAppTitle TSTR("IDispatch Polygon Server (32-bit)")
#else //WIN32
#define szAppTitle TSTR("IDispatch Polygon Server")
#endif //WIN32

BOOL
InitInstance(HANDLE hinst, int nCmdShow)
{
    g_hinst = hinst;

    // Create a main frame window
    //
    g_hwndFrame = CreateWindow(
      g_szFrameWndClass,
      szAppTitle,
      WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
      CW_USEDEFAULT,
      CW_USEDEFAULT,
      CW_USEDEFAULT,
      CW_USEDEFAULT,
      NULL,
      NULL,
      hinst,
      NULL);
    if(!g_hwndFrame)
      return FALSE;

    g_hwndClient = GetWindow(g_hwndFrame, GW_CHILD);
    if(!g_hwndClient)
      return FALSE;

    // create the status bar
    //
    g_psb = CStatBar::Create(g_hinst, g_hwndFrame);
    if(!g_psb)
      return FALSE;

    // initialize and show the status bar
    //
    g_psb->SetHeight(GetSystemMetrics(SM_CYCAPTION) - 1);
    g_psb->SetFont(GetStockObject(SYSTEM_FONT));
    g_psb->SetText(OLESTR(""));
    g_psb->Show();

    ShowWindow(g_hwndFrame, nCmdShow);

    UpdateWindow(g_hwndFrame);

    return TRUE;
}
```

```
void
FrameWndOnCreate(HWND hwnd)
{
    CLIENTCREATESTRUCT ccs;

    ccs.hWindowMenu = NULL;
    ccs.idFirstChild = IDM_FIRSTCHILD;

    g_hwndClient = CreateWindow(
      TSTR("MDICLIENT"),
      0,
      WS_CHILD | WS_CLIPCHILDREN | WS_VISIBLE,
      0, 0, 0, 0,
      hwnd,
      (HMENU) 1,
      g_hinst,
      &ccs);
}


void
FrameWndOnSize(HWND hwnd)
{
    RECT rc;
    int height;

    // Get the client rectangle for the frame window
    GetClientRect(hwnd, &rc);

    height = g_psb->GetHeight();

    // adjust the client win to make room for the status bar.
    //
    MoveWindow(
      g_hwndClient,
      rc.left,
      rc.top,
      rc.right - rc.left,
      rc.bottom - rc.top - height,
      TRUE);

    // move the status bar to the bottom of the newly positioned window.
    //
    g_psb->SetX(rc.left);
    g_psb->SetY(rc.bottom - height),
    g_psb->SetWidth(rc.right - rc.left);
    g_psb->Move();
}


extern "C" long FAR PASCAL
FrameWndProc(
    HWND hwnd,
    UINT message,
    WPARAM wParam,
    LPARAM lParam)
{
    switch(message){
    case WM_COMMAND:
      switch(wParam){
      case IDM_DUMP:
        CPoly::PolyDump();
        return 0;

      case IDM_CLEAR:
        InvalidateRect(g_hwndClient, NULL, TRUE);
        return 0;
      }
      break;

    case WM_CREATE:
      FrameWndOnCreate(hwnd);
      break;

    case WM_SIZE:
      FrameWndOnSize(hwnd);
      return 1;

    case WM_PAINT:
      CPoly::PolyDraw();
      break;

    case WM_CLOSE:
      DestroyWindow(hwnd);
      return 0;

    case WM_DESTROY:
      UninitOle();
      PostQuitMessage(0);
      return 0;
    }
    return DefFrameProc(hwnd, g_hwndClient, message, wParam, lParam);
}


#if defined(WIN32)

extern "C" OLECHAR FAR*
ConvertStrAtoW(char FAR* strIn, OLECHAR FAR* buf, UINT size)
{
  MultiByteToWideChar(CP_ACP, MB_PRECOMPOSED,
                      strIn, -1, buf, size) ;
  return buf;
}

extern "C" OLECHAR FAR*
WideString(char FAR* strIn)
{
  static OLECHAR buf[256];
  return (ConvertStrAtoW(strIn, buf, 256));
}

extern "C" char FAR*
ConvertStrWtoA(OLECHAR FAR* strIn, char FAR* buf, UINT size)
{
  int badConversion = FALSE;

  WideCharToMultiByte(CP_ACP, NULL,
                      strIn, -1,
                      buf, size,
                      NULL, &badConversion);
  return buf;
}


extern "C" char FAR*
AnsiString(OLECHAR FAR* strIn)
{
  static char buf[256];

  return (ConvertStrWtoA(strIn, buf, 256));
}

#endif
```

SPOLY2

----------------------------------- OLE Automation Sample Program: SPoly2 -----------------------------------

SPoly2 is a program which draws polygons. The only way to make spoly2 draw a polygon is to use its programmability interface.

One OLE Automation object is exposed by spoly2:

- spoly2.application

Spoly2.Application is the object associated with spoly2's main window. It controls drawing polygons and clearing the display.

---------------- Program Structure ---------------- SPoly2 implements IDispatch by using INTERFACEDATA, DispGetIDsOfNames and DispInvoke.

----------------------------------- Methods defined on spoly2.application -----------------------------------

Name Description --- Draw() Draw the polygon.

Reset() Delete all points from the polygon.

AddPoint(X, Y) Add a point with coordinates (x,y) to the polygon

EnumPoints() as VT_ENUM Return a collection of the polygon's points

GetXOrigin() as short Get the X origin of the polygon. SetXOrigin(x as short) Set the X origin of the polygon. GetYOrigin() as short Get the Y origin of the polygon. SetYOrigin(y as short) Set the Y origin of the polygon. GetWidth() as short Get and the line width of the polygon.

SetWidth(width as short) Set the line width of the polygon.

-------------------------- Shortcomings of this sample -------------------------- 1. Many items in this sample should be properties. Instead, they are implemented as methods. Anything which behaves like an attribute of the object should be a property.

2. This is not a good example of how to implement a collection.

MAKEFILE   (SPOLY2 Sample)

```
############################################################################
##
#
#  (c) Copyright Microsoft Corp. 1992-1993 All Rights Reserved
#
#  File:
#
#    makefile - makefile for spoly2.exe
#
#  Purpose:
#
#    Builds the OLE 2.0 sample IDispatch server, spoly2.exe.
#
#
#  Usage:
#
#     NMAKE                     ; build with defaults
#     or: NMAKE option          ; build with the given option(s)
#     or: NMAKE clean           ; erase all compiled files
#
#     option:
#         dev = [win16 | win32] ; dev=win32 is the default
#         DEBUG=[0|1]           ; DEBUG=1 is the default
#
#  Notes:
#
#    This makefile assumes that the PATH, INCLUDE and LIB environment
#    variables are setup properly.
#
############################################################################
##



##########################################################################
#
# Default Settings
#

!if "$(dev)" == ""
dev = win32
!endif

!if !("$(dev)" == "win16" || "$(dev)" == "win32")
!error Invalid dev option, choose from [win16 | win32]
!endif

!if "$(dev)" == "win16"
TARGET  = WIN16
!endif

!if "$(dev)" == "win32"
```

```
TARGET  = WIN32
!endif

!ifdef NODEBUG
DEBUG = 0
!endif

!if "$(DEBUG)" == "0"
NODEBUG = 1
!endif

!if "$(DEBUG)" == ""
DEBUG = 1
!endif


###########################################################################
#
# WIN16 Settings
#
!if "$(TARGET)" == "WIN16"

CC   = cl
LINK = link

RCFLAGS = -dWIN16
CFLAGS = -c -W3 -AM -GA -GEs -DWIN16
LINKFLAGS = /NOD /NOI /BATCH /ONERROR:NOEXE

LIBS = libw.lib mlibcew.lib

!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS) -Od -Zi -D_DEBUG $(CL)
LINKFLAGS = $(LINKFLAGS) /COD
!else
CFLAGS = $(CFLAGS) -Ox $(CL)
LINKFLAGS = $(LINKFLAGS) /FAR /PACKC
!endif
!endif


###########################################################################
#
# WIN32 Settings
#
!if "$(TARGET)" == "WIN32"

!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

CC = $(cc)
CFLAGS = $(cflags) $(cvarsmt) -DINC_OLE2 -DUNICODE $(cdebug)

!ifndef NODEBUG
CFLAGS = $(CFLAGS) -D_DEBUG
!endif
```

```
LINK = $(link)
LINKFLAGS = $(linkdebug) $(guilflags)
RCFLAGS = -DWIN32

!endif


###########################################################################
#
# Application Settings
#

!if "$(TARGET)" == "WIN16"
LIBS = ole2.lib compobj.lib ole2disp.lib $(LIBS)
!else
!if "$(TARGET)" == "WIN32"
LIBS = $(olelibsmt)
!endif
!endif

OBJS = \
        winmain.obj     \
        cpoly.obj       \
        cpoint.obj      \
        cenumpt.obj     \
        statbar.obj     \
        clsid.obj       \
        misc.obj        \
        tdata.obj

goal : setflags spoly2.exe


setflags :
        set CL=$(CFLAGS)

clean :
    if exist *.obj      del *.obj
    if exist spoly2.exe del spoly2.exe
    if exist spoly2.map del spoly2.map
    if exist spoly2.res del spoly2.res
    if exist *.pdb       del *.pdb


###########################################################################
#
# Application Build (WIN16 Specific)
#

!if "$(TARGET)" == "WIN16"

spoly2.exe : $(OBJS) spoly2.def spoly2.res spoly2.ico
        link $(LINKFLAGS) @<<
$(OBJS),
$@,,
$(LIBS),
spoly2.def
<<
        rc -k -t spoly2.res $@
!endif


##########################################################################
#
# Application Build (WIN32 Specific)
#
!if "$(TARGET)" == "WIN32"

spoly2.exe : $(OBJS) spoly2.def spoly2.res spoly2.ico
      $(LINK) @<<
        $(LINKFLAGS)
        -out:$@
        -map:$*.map
        $(OBJS)
        spoly2.res
        $(LIBS)
<<
!endif


spoly2.res : spoly2.rc resource.h
        rc $(RCFLAGS) -r -fo$@ spoly2.rc

winmain.obj: winmain.cpp hostenv.h resource.h spoly.h statbar.h
        $(CC) winmain.cpp

cpoint.obj: cpoint.cpp cpoint.h hostenv.h spoly.h statbar.h
        $(CC) cpoint.cpp

cpoly.obj: cpoly.cpp cpoint.h cpoly.h hostenv.h spoly.h statbar.h
        $(CC) cpoly.cpp

clsid.obj: clsid.c clsid.h
        $(CC) clsid.c

cenumpt.obj: cenumpt.cpp cenumpt.h
        $(CC) cenumpt.cpp

statbar.obj: statbar.cpp statbar.h
        $(CC) statbar.cpp

misc.obj: misc.cpp hostenv.h spoly.h
        $(CC) misc.cpp

tdata.obj: tdata.cpp
        $(CC) tdata.cpp
```

CENUMPT.H (SPOLY2 Sample)

```
/***
*cenumpt.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  Definition of the CEnumPoint class.
*
*Implementation Notes:
*
****************************************************************************
*/

class CEnumPoint : public IEnumVARIANT
{
public:
    static HRESULT Create(SAFEARRAY FAR* psa, CEnumPoint FAR* FAR* ppenum);

    /* IUnknown methods */
    STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppv);
    STDMETHOD_(unsigned long, AddRef)(void);
    STDMETHOD_(unsigned long, Release)(void);

    /* IEnumVARIANT methods */
    STDMETHOD(Next)(unsigned long celt, VARIANT FAR* rgvar, unsigned long
FAR* pceltFetched);
    STDMETHOD(Skip)(unsigned long celt);
    STDMETHOD(Reset)(void);
    STDMETHOD(Clone)(IEnumVARIANT FAR* FAR* ppenum);

    CEnumPoint();

private:

    unsigned long m_refs;

    unsigned long m_celts;
    unsigned long m_iCurrent;
    SAFEARRAY FAR* m_psa;
};
```

CENUMPT.CPP   (SPOLY2 Sample)

```
/***
*cenumpt.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*  This module implements the CEnumPoint class.
*
*
*Implementation Notes:
*
*****************************************************************************
*/

#include "hostenv.h"
#include "cenumpt.h"


CEnumPoint::CEnumPoint()
{
    m_refs = 0;

    m_psa = NULL;
    m_celts = 0;
    m_iCurrent = 0;
}


/***
*HRESULT CEnumPoint::Create(SAFEARRAY*, CEnumPoint**)
*Purpose:
*  This routine creates a CPoint enumerator from the given
*  (1 X N) SafeArray of CPoint IDispatch pointers.
*
*Entry:
*  psa = pointer to a SafeArray of VARIANTs
*
*Exit:
*  return value = HRESULT
*
*  *ppenum = pointer to a CPoint enumerator
*
***********************************************************************/
HRESULT
CEnumPoint::Create(SAFEARRAY FAR* psa, CEnumPoint FAR* FAR* ppenum)
{
    long lBound;
    HRESULT hresult;
    CEnumPoint FAR* penum;


    // Verify that the SafeArray is the proper shape.
    //
    if(SafeArrayGetDim(psa) != 1)
      return ResultFromScode(E_INVALIDARG);

    hresult = SafeArrayGetLBound(psa, 1, &lBound);
    if(FAILED(hresult))
      return hresult;

    if(lBound != 0)
      return ResultFromScode(E_INVALIDARG);

    penum = new FAR CEnumPoint();
    if(penum == NULL)
      return ResultFromScode(E_OUTOFMEMORY);
    penum->AddRef();

    hresult = SafeArrayGetUBound(psa, 1, &lBound);
    if(FAILED(hresult))
      goto LError0;

    penum->m_psa = psa;
    penum->m_celts = lBound + 1;

    *ppenum = penum;

    return NOERROR;

LError0:;
    penum->Release();

    return hresult;
}


//---------------------------------------------------------------------
//                         IUnknown methods
//---------------------------------------------------------------------


STDMETHODIMP
CEnumPoint::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    if(IsEqualIID(riid, IID_IEnumVARIANT) || IsEqualIID(riid, IID_IUnknown))
{
      *ppv = this;
      AddRef();
      return NOERROR;
    }
    *ppv = NULL;
    return ResultFromScode(E_NOINTERFACE);
}


STDMETHODIMP_(unsigned long)
CEnumPoint::AddRef()
```

```
{
    return ++m_refs;
}


STDMETHODIMP_(unsigned long)
CEnumPoint::Release()
{
    if(--m_refs == 0){
      if(m_psa != NULL)
     SafeArrayDestroy(m_psa);
      delete this;
      return 0;
    }
    return m_refs;
}


//---------------------------------------------------------------------
//                        IEnumVARIANT methods
//---------------------------------------------------------------------


/***
*HRESULT CEnumPoint::Next(unsigned long, VARIANT*, unsigned long*)
*Purpose:
*  Attempt to get the next 'celt' items in the enumeration sequence.
*
*Entry:
*  celt = the number of elements to get
*
*Exit:
*  return value = HRESULT
*    S_OK
*    S_FALSE - the end of the sequence was reached
*
*  rgvar = array of the next 'celt' items
*  *pceltFetched = count of the elements actually fetched.
*
***********************************************************************/
STDMETHODIMP
CEnumPoint::Next(
    unsigned long celt,
    VARIANT FAR* rgvar,
    unsigned long FAR* pceltFetched)
{
    long ix;
    unsigned int i;
    HRESULT hresult;


    for(i = 0; i < celt; ++i)
      VariantInit(&rgvar[i]);
    for(i = 0; i < celt; ++i){
      if(m_iCurrent == m_celts){
        hresult = ResultFromScode(S_FALSE);
     goto LDone;
      }

      ix = m_iCurrent++;
      hresult = SafeArrayGetElement(m_psa, &ix, &rgvar[i]);
      if(FAILED(hresult))
     goto LError0;
    }

    hresult = NOERROR;

LDone:;
    if (pceltFetched != NULL)
      *pceltFetched = i;

    return hresult;

LError0:;

    for(i = 0; i < celt; ++i)
      VariantClear(&rgvar[i]);

    return hresult;
}


/***
*HRESULT CEnumPoint::Skip(unsigned long)
*Purpose:
*  Attempt to skip over the next 'celt' elements in the enumeration
*  sequence.
*
*Entry:
*  celt = the count of elements to skip
*
*Exit:
*  return value = HRESULT
*    S_OK
*    S_FALSE -  the end of the sequence was reached
*
***********************************************************************/
STDMETHODIMP
CEnumPoint::Skip(unsigned long celt)
{
    m_iCurrent += celt;

    if(m_iCurrent > m_celts)
     m_iCurrent = m_celts;

    return (m_iCurrent == m_celts)
      ? ResultFromScode(S_FALSE) : NOERROR;
}
```

```
/***
*HRESULT CEnumPoint::Reset(void)
*Purpose:
*  Reset the enumeration sequence back to the beginning.
*
*Entry:
*  None
*
*Exit:
*  return value = SHRESULT CODE
*    S_OK
*
***********************************************************************/
STDMETHODIMP
CEnumPoint::Reset()
{
    m_iCurrent = 0;

    return NOERROR;
}


/***
*HRESULT CEnumPoint::Clone(IEnumVARIANT**)
*Purpose:
*  Retrun a CPoint enumerator with exactly the same state as the
*  current one.
*
*Entry:
*  None
*
*Exit:
*  return value = HRESULT
*    S_OK
*    E_OUTOFMEMORY
*
***********************************************************************/
STDMETHODIMP
CEnumPoint::Clone(IEnumVARIANT FAR* FAR* ppenum)
{
    HRESULT hresult;
    SAFEARRAY FAR* psa;
    CEnumPoint FAR* penum;

    hresult = SafeArrayCopy(m_psa, &psa);
    if(FAILED(hresult))
      return hresult;

    hresult = CEnumPoint::Create(psa, &penum);
    if(FAILED(hresult))
      goto LError0;

    // Assert(penum->m_celts == m_celts);
    penum->m_iCurrent = m_iCurrent;

    *ppenum = penum;

    return NOERROR;

LError0:
    SafeArrayDestroy(psa);

    return hresult;
}
```

CLSID.H   (SPOLY2 Sample)

```
/***
*clsid.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This file defines the CLSIDs
*
*Implementation Notes:
*
****************************************************************************
*/

DEFINE_GUID(CLSID_CPoly2, 0x00020464, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);
DEFINE_GUID(CLSID_CPoint2, 0x00020465, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);


#ifdef INITGUID
# ifdef _MAC
DEFINE_GUID(GUID_NULL, 0L, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);

DEFINE_GUID(IID_IDispatch, 0x00020400L, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);
DEFINE_GUID(IID_IEnumVARIANT, 0x00020404L, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0,
0x46);
DEFINE_GUID(IID_IUnknown, 0x00000000L, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0, 0x46);
DEFINE_GUID(IID_IClassFactory, 0x00000001L, 0, 0, 0xC0, 0, 0, 0, 0, 0, 0,
0x46);
# endif
#endif
```

CLSID.C   (SPOLY2 Sample)

```
/***
*clsid.c
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This file allocates and initializes the CLSIDs.
*
*****************************************************************************
*/

#ifdef _PPCMAC
#pragma data_seg ("_FAR_DATA")
#pragma data_seg ( )
#endif //_PPCMAC

#ifdef _MAC
# include <Types.h>
#ifdef _MSC_VER
# include <Processe.h>
# include <AppleEve.h>
#else //_MSC_VER
# include <Processes.h>
# include <AppleEvents.h>
#endif //_MSC_VER
#else
# include <windows.h>
#endif

#ifndef WIN32
#include <compobj.h>
#endif //!WIN32

// this redefines the DEFINE_GUID() macro to do allocation.
//
#include <initguid.h>

#ifndef INITGUID
# define INITGUID
#endif

// due to the previous header, including this causes the DEFINE_GUID
// definitions in the following header(s) to actually allocate data.
//
#include "clsid.h"
```

CPOINT.H   (SPOLY2 Sample)

```
/***
*cpoint.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  Definition of the CPoint class.
*
*  The CPoint object exposes two properties for programatic access
*  via the IDispatch interface.
*
*  properties:
*    X                 - the 'x' coordinate of the point
*    Y                 - the 'y' coordinate of the point
*
*Implementation Notes:
*
****************************************************************************
*/

#ifndef    CLASS
#ifdef     __TURBOC__
#define CLASS class huge
#else
#define CLASS class FAR
#endif
#endif

class CPoly;

CLASS CPoint : public IDispatch {
    friend class CPoly;

public:
    static CPoint FAR* Create();

    /* IUnknown methods */
    STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppvObj);
    STDMETHOD_(unsigned long, AddRef)(void);
    STDMETHOD_(unsigned long, Release)(void);

    /* IDispatch methods */
    STDMETHOD(GetTypeInfoCount)(unsigned int FAR* pcTypeInfo);

    STDMETHOD(GetTypeInfo)(
      unsigned int iTypeInfo,
      LCID lcid,
      ITypeInfo FAR* FAR* ppTypeInfo);

    STDMETHOD(GetIDsOfNames)(
      REFIID riid,
      OLECHAR FAR* FAR* rgszNames,
      unsigned int cNames,
      LCID lcid,
      DISPID FAR* rgdispid);

    STDMETHOD(Invoke)(
      DISPID dispidMember,
      REFIID riid,
      LCID lcid,
      unsigned short wFlags,
      DISPPARAMS FAR* pdispparams,
      VARIANT FAR* pvarResult,
      EXCEPINFO FAR* pexcepinfo,
      unsigned int FAR* pwArgErr);

    /* Introduced methods */

    virtual short METHODCALLTYPE EXPORT GetX(void);
    virtual void  METHODCALLTYPE EXPORT SetX(short x);
    virtual short METHODCALLTYPE EXPORT GetY(void);
    virtual void  METHODCALLTYPE EXPORT SetY(short y);

private:
    CPoint();

    unsigned long m_refs;

    short m_x;
    short m_y;

    ITypeInfo FAR* m_ptinfo;
};

// member DISPIDs
//
enum IDMEMBER_CPOINT {
    IDMEMBER_CPOINT_GETX = 1,
    IDMEMBER_CPOINT_SETX,
    IDMEMBER_CPOINT_GETY,
    IDMEMBER_CPOINT_SETY,
    IDMEMBER_CPOINT_MAX
};

// member indices - this is an enumeration of all members on CPoint
//
enum IMETH_CPOINT {
    IMETH_CPOINT_QUERYINTERFACE = 0,
    IMETH_CPOINT_ADDREF,
    IMETH_CPOINT_RELEASE,
    IMETH_CPOINT_GETTYPEINFOCOUNT,
    IMETH_CPOINT_GETTYPEINFO,
    IMETH_CPOINT_GETIDSOFNAMES,
    IMETH_CPOINT_INVOKE,

    IMETH_CPOINT_GETX,
    IMETH_CPOINT_SETX,
    IMETH_CPOINT_GETY,
    IMETH_CPOINT_SETY
};

// structure used to link together lists of points
//
struct POINTLINK {
    POINTLINK FAR* next;
    CPoint FAR* ppoint;
};


// The CPoint Class Factory
//
CLASS CPointCF : public IClassFactory
{
public:
    static IClassFactory FAR* Create();

    /* IUnknown methods */
    STDMETHOD(QueryInterface)(REFIID iid, void FAR* FAR* ppv);
    STDMETHOD_(unsigned long, AddRef)(void);
    STDMETHOD_(unsigned long, Release)(void);

    /* IClassFactory methods */
    STDMETHOD(CreateInstance)(
      IUnknown FAR* pUnkOuter, REFIID iid, void FAR* FAR* ppv);
#ifdef _MAC
    STDMETHOD(LockServer)(unsigned long fLock);
#else
    STDMETHOD(LockServer)(BOOL fLock);
#endif

private:
    CPointCF();

    unsigned long m_refs;
};
```

CPOINT.CPP   (SPOLY2 Sample)

```
/***
*cpoint.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*  This module implements the CPoint and CPointCF classes.
*
*  This module is intended as a sample implementation of the IDispatch
*  interface, and its purpose is to demonstrate how an object can
*  expose methods and properties for programatic and cross-process
*  access via the IDispatch interface.
*
*Implementation Notes:
*
****************************************************************************
*/

#include "spoly.h"
#include "cpoint.h"


CPoint::CPoint()
{
    m_x = 0;
    m_y = 0;
    m_refs = 0;
    m_ptinfo = NULL;
}

/***
*CPoint::Create(void)
*Purpose:
*  Create an instance of a CPoint object.
*
*Entry:
*  None
*
*Exit:
*  returns a CPoint*, NULL if creation failed.
*
***********************************************************************/
CPoint FAR*
CPoint::Create()
{
    HRESULT hresult;
    CPoint FAR* ppoint;
    ITypeInfo FAR* ptinfo;
extern INTERFACEDATA NEAR g_idataCPoint;


    if((ppoint = new FAR CPoint()) == NULL)
```

```
      return NULL;
    ppoint->AddRef();

    hresult =
      CreateDispTypeInfo(&g_idataCPoint, LOCALE_SYSTEM_DEFAULT, &ptinfo);
    if(hresult != NOERROR)
      goto LError0;

    ppoint->m_ptinfo = ptinfo;

    return ppoint;

LError0:;
    ppoint->Release();

    return NULL;
}


//---------------------------------------------------------------------
//                      IUnknown Methods
//---------------------------------------------------------------------


STDMETHODIMP
CPoint::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    if(IsEqualIID(riid, IID_IUnknown) || IsEqualIID(riid, IID_IDispatch)){
      *ppv = this;
      AddRef();
      return NOERROR;
    }
    *ppv = NULL;
    return ResultFromScode(E_NOINTERFACE);
}


STDMETHODIMP_(unsigned long)
CPoint::AddRef(void)
{
    return ++m_refs;
}


STDMETHODIMP_(unsigned long)
CPoint::Release(void)
{
    if(--m_refs == 0){
      if(m_ptinfo != NULL){
     m_ptinfo->Release();
      }
      delete this;
      return 0;
    }
    return m_refs;
```

```
}


//---------------------------------------------------------------------
//                     IDispatch methods
//---------------------------------------------------------------------


STDMETHODIMP
CPoint::GetTypeInfoCount(unsigned int FAR* pctinfo)
{
    // this object has a single *introduced* interface
    //
    *pctinfo = 1;

    return NOERROR;
}


STDMETHODIMP
CPoint::GetTypeInfo(unsigned int itinfo, LCID lcid, ITypeInfo FAR* FAR*
pptinfo)
{
    UNUSED(lcid);

    if(itinfo != 0)
      return ResultFromScode(DISP_E_BADINDEX);

    m_ptinfo->AddRef();
    *pptinfo = m_ptinfo;

    return NOERROR;
}


/***
*HRESULT CPoint::GetIDsOfNames(REFIID, char**, unsigned int, LCID, DISPID*)
*Purpose:
*  This method translates the given array of names to a corresponding
*  array of DISPIDs.
*
*  Index 0 of the name array is the member name, and indices 1-N if
*  present represent named parameters on that member.
*
*  The local ID ('lcid') is unused by this naive implementation. A more
*  sophisticated implementation, sensitive to localization and natural
*  language support would use the locale ID to interpret the given names
*  in a correct locale specific context.
*
*Entry:
*  rgszNames = pointer to an array of names
*  cNames = the number of names in the rgszNames array
*  lcid = the callers locale ID
*
```

```
*Exit:
*  return value = HRESULT
*  rgid = array of name IDs corresponding to the rgszNames array
*    this array will contain -1 for each entry that is not known.
*
***********************************************************************/
STDMETHODIMP
CPoint::GetIDsOfNames(
    REFIID riid,
    OLECHAR FAR* FAR* rgszNames,
    unsigned int cNames,
    LCID lcid,
    DISPID FAR* rgdispid)
{
    UNUSED(lcid);

    if(!IsEqualIID(riid,IID_NULL))
      return ResultFromScode(DISP_E_UNKNOWNINTERFACE);

    return DispGetIDsOfNames(m_ptinfo, rgszNames, cNames, rgdispid);
}


/***
*HRESULT CPoint::Invoke(...)
*Purpose:
*  Dispatch a method or property request for objects of type CPoint.
*
*  see the IDispatch document for more information, and a general
*  description of this method.
*
*Entry:
*  dispidMember = the DISPID of the member being requested
*
*  riid = reference to the interface ID of the interface on this object
*    that the requested member belongs to. IID_NULL means to interpret
*    the member as belonging to the implementation defined "default"
*    or "primary" interface.
*
*  lcid = the caller's locale ID
*
*  wFlags = flags indicating the type of access being requested
*
*  pdispparams = pointer to the DISPPARAMS struct containing the
*    requested members arguments (if any) and its named parameter
*    DISPIDs (if any).
*
*Exit:
*  return value = HRESULT
*   see the IDispatch spec for a description of possible success codes.
*
*  pvarResult = pointer to a caller allocated VARIANT containing
*    the members return value (if any).
*
*  pexcepinfo = caller allocated exception info structure, this will
*    be filled in only if an exception was raised that must be passed
*    up through Invoke to an enclosing handler.
*
*  puArgErr = pointer to a caller allocated UINT, that will contain the
*    index of the offending argument if a DISP_E_TYPEMISMATCH error
*    was returned indicating that one of the arguments was of an
*    incorrect type and/or could not be reasonably coerced to a proper
*    type.
*
***********************************************************************/
STDMETHODIMP
CPoint::Invoke(
    DISPID dispidMember,
    REFIID riid,
    LCID lcid,
    unsigned short wFlags,
    DISPPARAMS FAR* pdispparams,
    VARIANT FAR* pvarResult,
    EXCEPINFO FAR* pexcepinfo,
    unsigned int FAR* puArgErr)
{
    UNUSED(lcid);

    if(!IsEqualIID(riid, IID_NULL))
      return ResultFromScode(DISP_E_UNKNOWNINTERFACE);

    return DispInvoke(
      this, m_ptinfo,
      dispidMember, wFlags, pdispparams,
      pvarResult, pexcepinfo, puArgErr);
}


//---------------------------------------------------------------------
//                       Introduced methods
//---------------------------------------------------------------------

short METHODCALLTYPE EXPORT
CPoint::GetX()
{
    return m_x;
}


void METHODCALLTYPE EXPORT
CPoint::SetX(short x)
{
    m_x = x;
}

short METHODCALLTYPE EXPORT
CPoint::GetY()
{
    return m_y;
}
```

```
void METHODCALLTYPE EXPORT
CPoint::SetY(short y)
{
    m_y = y;
}


//---------------------------------------------------------------------
//         Implementation of the CPoint Class Factory
//---------------------------------------------------------------------

CPointCF::CPointCF()
{
    m_refs = 0;
}

IClassFactory FAR*
CPointCF::Create()
{
    CPointCF FAR* pCF;

    if((pCF = new FAR CPointCF()) == NULL)
      return NULL;
    pCF->AddRef();
    return pCF;
}

STDMETHODIMP
CPointCF::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    if(IsEqualIID(riid, IID_IUnknown) || IsEqualIID(riid,
IID_IClassFactory)){
      *ppv = this;
      ++m_refs;
      return NOERROR;
    }
    *ppv = NULL;
    return ResultFromScode(E_NOINTERFACE);
}

STDMETHODIMP_(unsigned long)
CPointCF::AddRef(void)
{
    return ++m_refs;
}

STDMETHODIMP_(unsigned long)
CPointCF::Release(void)
{
    if(--m_refs == 0){
      delete this;
      return 0;
    }
    return m_refs;
}

STDMETHODIMP
CPointCF::CreateInstance(
    IUnknown FAR* punkOuter,
    REFIID riid,
    void FAR* FAR* ppv)
{
    HRESULT hresult;
    CPoint FAR *ppoint;

    UNUSED(punkOuter);

    if((ppoint = CPoint::Create()) == NULL){
      *ppv = NULL;
      return ResultFromScode(E_OUTOFMEMORY);
    }
    hresult = ppoint->QueryInterface(riid, ppv);
    ppoint->Release();
    return hresult;
}

STDMETHODIMP
#ifdef _MAC
CPointCF::LockServer(unsigned long fLock)
#else
CPointCF::LockServer(BOOL fLock)
#endif
{
    UNUSED(fLock);

    return NOERROR;
}
```

CPOLY.H   (SPOLY2 Sample)

```
/***
*cpoly.h
*
*  Copyright (C) 1992-1994 Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  Definition of the CPoly class.
*
*  The CPoly class defines a number of methods and exposes them for
*  external programmability via IDispatch,
*
*  methods:
*    DRAW        - draw the polygon
*    RESET       - delete all points from the polygon
*
*    ADDPOINT(X, Y)    - add a point with coordinates (x,y) to the polygon
*
*    ENUMPOINTS        - return a collection of the polygon's points
*
*    GETXORIGIN        - get and set the X origin of the polygon
*    SETXORIGIN
*
*    GETYORIGIN        - get and set the Y origin of the polygon
*    SETYORIGIN
*
*    GETWIDTH          - get and set the line width of the polygon
*    SETWIDTH
*
*  UNDONE: update description
*
*Implementation Notes:
*
****************************************************************************
*/


#ifndef    CLASS
# ifdef    __TURBOC__
#  define CLASS class huge
# else
#  define CLASS class FAR
# endif
#endif

class CPoint;

CLASS CPoly : public IDispatch
{
public:
    static CPoly FAR* Create();
```

```
/* IUnknown methods */
STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppvObj);
STDMETHOD_(unsigned long, AddRef)(void);
STDMETHOD_(unsigned long, Release)(void);

/* IDispatch methods */
STDMETHOD(GetTypeInfoCount)(unsigned int FAR* pcTypeInfo);

STDMETHOD(GetTypeInfo)(
  unsigned int iTypeInfo,
  LCID lcid,
  ITypeInfo FAR* FAR* ppTypeInfo);

STDMETHOD(GetIDsOfNames)(
  REFIID riid,
  OLECHAR FAR* FAR* rgszNames,
  unsigned int cNames,
  LCID lcid,
  DISPID FAR* rgdispid);

STDMETHOD(Invoke)(
  DISPID dispidMember,
  REFIID riid,
  LCID lcid,
  unsigned short wFlags,
  DISPPARAMS FAR* pdispparams,
  VARIANT FAR* pvarResult,
  EXCEPINFO FAR* pexcepinfo,
  unsigned int FAR* puArgErr);

/* Introduced methods */

virtual void  METHODCALLTYPE EXPORT Draw(void);
virtual void  METHODCALLTYPE EXPORT Reset(void);

// add a point with the given 'x' and 'y' coordinates
virtual HRESULT METHODCALLTYPE EXPORT AddPoint(short x, short y);

// return a collection of the polygon's points
virtual IUnknown FAR* METHODCALLTYPE EXPORT EnumPoints(void);

// get/set the polygon's X origin property
virtual short METHODCALLTYPE EXPORT GetXOrigin(void);
virtual void  METHODCALLTYPE EXPORT SetXOrigin(short x);

// get/set the polygon's Y origin property
virtual short METHODCALLTYPE EXPORT GetYOrigin(void);
virtual void  METHODCALLTYPE EXPORT SetYOrigin(short y);

virtual short METHODCALLTYPE EXPORT GetWidth(void);
virtual void  METHODCALLTYPE EXPORT SetWidth(short width);

virtual short METHODCALLTYPE EXPORT get_red(void);
virtual void  METHODCALLTYPE EXPORT set_red(short red);
```

```
    virtual short METHODCALLTYPE EXPORT get_green(void);
    virtual void  METHODCALLTYPE EXPORT set_green(short green);

    virtual short METHODCALLTYPE EXPORT get_blue(void);
    virtual void  METHODCALLTYPE EXPORT set_blue(short blue);

    // Debug method
    virtual void  METHODCALLTYPE EXPORT Dump(void);

public:

    // Draw all polygons.
    static void PolyDraw(void);

    // Release all polygons.
    static void PolyTerm(void);

    // Dump all polygons to dbwin.
    static void PolyDump(void);


private:
    CPoly();

    short m_xorg;
    short m_yorg;
    short m_width;

    short m_red;
    short m_green;
    short m_blue;

    unsigned long m_refs;
    unsigned int m_cPoints;

    ITypeInfo FAR* m_ptinfo;

    POINTLINK FAR* m_ppointlink;
    POINTLINK FAR* m_ppointlinkLast;
};

// DISPIDs for the members and properties available via IDispatch.
//
enum IDMEMBER_CPOLY {
    IDMEMBER_CPOLY_DRAW = 1,
    IDMEMBER_CPOLY_RESET,
    IDMEMBER_CPOLY_ADDPOINT,
    IDMEMBER_CPOLY_ENUMPOINTS,
    IDMEMBER_CPOLY_GETXORIGIN,
    IDMEMBER_CPOLY_SETXORIGIN,
    IDMEMBER_CPOLY_GETYORIGIN,
    IDMEMBER_CPOLY_SETYORIGIN,
    IDMEMBER_CPOLY_GETWIDTH,
    IDMEMBER_CPOLY_SETWIDTH,
    IDMEMBER_CPOLY_GETRED,
    IDMEMBER_CPOLY_SETRED,
    IDMEMBER_CPOLY_GETGREEN,
    IDMEMBER_CPOLY_SETGREEN,
    IDMEMBER_CPOLY_GETBLUE,
    IDMEMBER_CPOLY_SETBLUE,
    IDMEMBER_CPOLY_DUMP,
    IDMEMBER_CPOLY_MAX
};

// CPoly method indices
//
enum IMETH_CPOLY {
    IMETH_CPOLY_QUERYINTERFACE = 0,
    IMETH_CPOLY_ADDREF,
    IMETH_CPOLY_RELEASE,
    IMETH_CPOLY_GETTYPEINFOCOUNT,
    IMETH_CPOLY_GETTYPEINFO,
    IMETH_CPOLY_GETIDSOFNAMES,
    IMETH_CPOLY_INVOKE,
    IMETH_CPOLY_DRAW,
    IMETH_CPOLY_RESET,
    IMETH_CPOLY_ADDPOINT,
    IMETH_CPOLY_ENUMPOINTS,
    IMETH_CPOLY_GETXORIGIN,
    IMETH_CPOLY_SETXORIGIN,
    IMETH_CPOLY_GETYORIGIN,
    IMETH_CPOLY_SETYORIGIN,
    IMETH_CPOLY_GETWIDTH,
    IMETH_CPOLY_SETWIDTH,
    IMETH_CPOLY_GETRED,
    IMETH_CPOLY_SETRED,
    IMETH_CPOLY_GETGREEN,
    IMETH_CPOLY_SETGREEN,
    IMETH_CPOLY_GETBLUE,
    IMETH_CPOLY_SETBLUE,
    IMETH_CPOLY_DUMP,
    IMETH_CPOLY_MAX
};

// structure used to link together polygons
//
struct POLYLINK {
    POLYLINK FAR* next;
    CPoly FAR* ppoly;
};


// The CPoly class factory
//
CLASS CPolyCF : public IClassFactory
{
public:
    static IClassFactory FAR* Create();
    /* IUnknown methods */
    STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppv);
    STDMETHOD_(unsigned long, AddRef)(void);
    STDMETHOD_(unsigned long, Release)(void);

    /* IClassFactory methods */
    STDMETHOD(CreateInstance)(
      IUnknown FAR* pUnkOuter, REFIID riid, void FAR* FAR* ppv);
#ifdef _MAC
    STDMETHOD(LockServer)(unsigned long fLock);
#else
    STDMETHOD(LockServer)(BOOL fLock);
#endif

private:
    CPolyCF();

    unsigned long m_refs;
};
```

CPOLY.CPP   (SPOLY2 Sample)

```
/***
*cpoly.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*  This module implements the CPoly and CPolyCF classes.
*
*  This module is intended as a sample implementation of the IDispatch
*  interface, and its purpose is to demonstrate how an object can
*  expose methods and properties for programatic and cross-process
*  access via the IDispatch interface.
*
*Implementation Notes:
*
*****************************************************************************
*/

#include "spoly.h"
#include "cpoint.h"
#include "cpoly.h"
#include "cenumpt.h"

#ifndef _MAC
extern CStatBar FAR* g_psb;
#endif

#ifdef _MAC
extern "C" WindowPtr g_pwndClient;
#endif

// our global list of polygons.
//
POLYLINK FAR* g_ppolylink = (POLYLINK FAR*)NULL;

// global count of polygons.
//
int g_cPoly = 0;


CPoly::CPoly()
{
    m_refs = 0;
    m_xorg = 0;
    m_yorg = 0;
    m_width = 0;
    m_cPoints = 0;

    m_red   = 0;
    m_green = 0;
    m_blue  = 0;
    m_ptinfo = NULL;

    m_ppointlink = NULL;
    m_ppointlinkLast = NULL;
}


/***
*CPoly *CPoly::Create(void)
*Purpose:
*  Create an instance of a CPoly object, and add it to the global
*  list of polygons.
*
*Entry:
*  None
*
*Exit:
*  returns a polygon object, NULL the allocation failed.
*
***********************************************************************/
CPoly FAR*
CPoly::Create()
{
    HRESULT hresult;
    CPoly FAR* ppoly;
    ITypeInfo FAR* ptinfo;
    POLYLINK FAR* ppolylink;
extern INTERFACEDATA NEAR g_idataCPoly;


    if((ppolylink = new FAR POLYLINK) == (POLYLINK FAR*)NULL)
      return (CPoly FAR*)NULL;

    if((ppoly = new FAR CPoly()) == (CPoly FAR*)NULL)
      return (CPoly FAR*)NULL;

    ppoly->AddRef();

    // create and attach its TypeInfo
    //
    hresult = CreateDispTypeInfo(&g_idataCPoly, LOCALE_SYSTEM_DEFAULT,
&ptinfo);
    if(hresult != NOERROR)
      goto LError0;

    ppoly->m_ptinfo = ptinfo;

    // push the new polygon onto the front of the polygon list.
    //
    ++g_cPoly;

    ppolylink->ppoly = ppoly;

    ppolylink->next = g_ppolylink;
    g_ppolylink = ppolylink;
```

```
#ifndef _MAC
    SBprintf(g_psb, TSTR("#poly = %d"), g_cPoly);
#endif

    return ppoly;


LError0:;
    ppoly->Release();

    return NULL;
}


//---------------------------------------------------------------------
//                     IUnknown Methods
//---------------------------------------------------------------------


STDMETHODIMP
CPoly::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    if(!IsEqualIID(riid, IID_IUnknown))
      if(!IsEqualIID(riid, IID_IDispatch)) {
     *ppv = NULL;
        return ResultFromScode(E_NOINTERFACE);
    }

    *ppv = this;
    AddRef();
    return NOERROR;
}



STDMETHODIMP_(unsigned long)
CPoly::AddRef()
{
    return ++m_refs;
}


STDMETHODIMP_(unsigned long)
CPoly::Release()
{
    POLYLINK FAR* FAR* pppolylink, FAR* ppolylinkDead;

    if(--m_refs == 0){
      Reset(); // release all CPoints

      // remove ourselves from the global list of polygons
      //
      for( pppolylink = &g_ppolylink;
       *pppolylink != NULL;
```

```
        pppolylink = &(*pppolylink)->next)
      {
     if((*pppolylink)->ppoly == this){
       ppolylinkDead = *pppolylink;
       *pppolylink = (*pppolylink)->next;
       delete ppolylinkDead;
       break;
     }
      }

      if(m_ptinfo != NULL){
     m_ptinfo->Release();
      }

      --g_cPoly;

#ifndef _MAC
      SBprintf(g_psb, TSTR("#poly = %d"), g_cPoly);
#endif

      delete this;
      return 0;
    }
    return m_refs;
}


//---------------------------------------------------------------------
//                      IDispatch Methods
//---------------------------------------------------------------------

STDMETHODIMP
CPoly::GetTypeInfoCount(unsigned int FAR* pctinfo)
{
    // This object has a single *introduced* interface
    //
    *pctinfo = 1;

    return NOERROR;
}


STDMETHODIMP
CPoly::GetTypeInfo(unsigned int itinfo, LCID lcid, ITypeInfo FAR* FAR*
pptinfo)
{
    UNUSED(lcid);

    if(itinfo != 0)
      return ResultFromScode(DISP_E_BADINDEX);

    m_ptinfo->AddRef();
    *pptinfo = m_ptinfo;

    return NOERROR;
```

```
}


/***
*HRESULT CPoly::GetIDsOfNames(char**, unsigned int, LCID, DISPID*)
*Purpose:
*  This method translates the given array of names to a corresponding
*  array of DISPIDs.
*
*  This method deferrs to a common implementation shared by
*  both the CPoly and CPoint objects. See the description of
*  'SPolyGetIDsOfNames()' in dispimpl.cpp for more information.
*
*Entry:
*  rgszNames = pointer to an array of names
*  cNames = the number of names in the rgszNames array
*  lcid = the callers locale ID
*
*Exit:
*  return value = HRESULT
*  rgdispid = array of DISPIDs corresponding to the rgszNames array
*    this array will contain -1 for each entry that is not known.
*
***********************************************************************/
STDMETHODIMP
CPoly::GetIDsOfNames(
    REFIID riid,
    OLECHAR FAR* FAR* rgszNames,
    unsigned int cNames,
    LCID lcid,
    DISPID FAR* rgdispid)
{
    UNUSED(lcid);

    if(!IsEqualIID(riid, IID_NULL))
      return ResultFromScode(DISP_E_UNKNOWNINTERFACE);

    return DispGetIDsOfNames(m_ptinfo, rgszNames, cNames, rgdispid);
}


/***
*HRESULT CPoly::Invoke(...)
*Purpose:
*  Dispatch a method or property request for objects of type CPoly.
*
*  see the IDispatch document for more information, and a general
*  description of this method.
*
*Entry:
*  dispidMember = the DISPID of the member being requested
*
*  riid = reference to the interface ID of the interface on this object
*    that the requested member belongs to. IID_NULL means to interpret
*    the member as belonging to the implementation defined "default"
*    or "primary" interface.
*
*  lcid = the caller's locale ID
*
*  wFlags = flags indicating the type of access being requested
*
*  pdispparams = pointer to the DISPPARAMS struct containing the
*    requested members arguments (if any) and its named parameter
*    DISPIDs (if any).
*
*Exit:
*  return value = HRESULT
*   see the IDispatch spec for a description of possible success codes.
*
*  pvarResult = pointer to a caller allocated VARIANT containing
*    the members return value (if any).
*
*  pexcepinfo = caller allocated exception info structure, this will
*    be filled in only if an exception was raised that must be passed
*    up through Invoke to an enclosing handler.
*
*  puArgErr = pointer to a caller allocated UINT, that will contain the
*    index of the offending argument if a DISP_E_TYPEMISMATCH error
*    was returned indicating that one of the arguments was of an
*    incorrect type and/or could not be reasonably coerced to a proper
*    type.
*
***********************************************************************/
STDMETHODIMP
CPoly::Invoke(
    DISPID dispidMember,
    REFIID riid,
    LCID lcid,
    unsigned short wFlags,
    DISPPARAMS FAR* pdispparams,
    VARIANT FAR* pvarResult,
    EXCEPINFO FAR* pexcepinfo,
    unsigned int FAR* puArgErr)
{
    UNUSED(lcid);

    if(!IsEqualIID(riid, IID_NULL))
      return ResultFromScode(DISP_E_UNKNOWNINTERFACE);

    return DispInvoke(
      this, m_ptinfo,
      dispidMember, wFlags, pdispparams,
      pvarResult, pexcepinfo, puArgErr);
}


//---------------------------------------------------------------------
//                      Introduced Methods
//---------------------------------------------------------------------
```

```
/***
*void CPoly::Draw(void)
*Purpose:
*  Draw the polygon, using the current x/y origin and line width
*  properties.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
void METHODCALLTYPE EXPORT
CPoly::Draw()
{
    short xorg, yorg;
    POINTLINK FAR* ppointlinkFirst, FAR* ppointlink;

    if((ppointlinkFirst = m_ppointlink) == (POINTLINK FAR*)NULL)
      return;

#ifdef _MAC /* { */

    short x,y;
    RGBColor rgb;
    WindowPtr pwndSaved;

    GetPort(&pwndSaved);
    SetPort(g_pwndClient);

    PenNormal();
    PenSize(m_width, m_width);

    rgb.red = m_red;
    rgb.green = m_green;
    rgb.blue = m_blue;
    RGBForeColor(&rgb);

    xorg = m_xorg;
    yorg = m_yorg;

    MoveTo(
      xorg + ppointlinkFirst->ppoint->m_x,
      yorg + ppointlinkFirst->ppoint->m_y);

    for(ppointlink = ppointlinkFirst->next;
     ppointlink != (POINTLINK FAR*)NULL;
     ppointlink = ppointlink->next)
    {
      x = xorg + ppointlink->ppoint->m_x;
      y = yorg + ppointlink->ppoint->m_y;
      LineTo(x, y);
    }

    LineTo(
      xorg + ppointlinkFirst->ppoint->m_x,
      yorg + ppointlinkFirst->ppoint->m_y);

    SetPort(pwndSaved);

#else /* }{ */

    HDC hdc;
    RECT rect;
    HPEN hpen, hpenOld;
extern HWND g_hwndClient;

    GetClientRect(g_hwndClient, &rect);
    xorg = m_xorg + (short) rect.left;
    yorg = m_yorg + (short) rect.top;

    hdc = GetDC(g_hwndClient);
    hpen = CreatePen(PS_SOLID, m_width, RGB(m_red, m_green, m_blue));
    hpenOld = SelectObject(hdc, hpen);

#ifdef WIN32

    MoveToEx(hdc,
      xorg + ppointlinkFirst->ppoint->m_x,
      yorg + ppointlinkFirst->ppoint->m_y, NULL);
#else
    MoveTo(hdc,
      xorg + ppointlinkFirst->ppoint->m_x,
      yorg + ppointlinkFirst->ppoint->m_y);
#endif

    for(ppointlink = ppointlinkFirst->next;
     ppointlink != (POINTLINK FAR*)NULL;
     ppointlink = ppointlink->next)
    {
      LineTo(hdc,
     xorg + ppointlink->ppoint->m_x,
     yorg + ppointlink->ppoint->m_y);
    }

    LineTo(hdc,
      xorg + ppointlinkFirst->ppoint->m_x,
      yorg + ppointlinkFirst->ppoint->m_y);

    SelectObject(hdc, hpenOld);
    DeleteObject(hpen);

    ReleaseDC(g_hwndClient, hdc);

#endif /* } */
}
```

```
/***
*void CPoly::Reset(void)
*Purpose:
*  Release all points referenced by this poly.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
void METHODCALLTYPE EXPORT
CPoly::Reset()
{
    POINTLINK FAR* ppointlink, FAR* ppointlinkNext;

    for(ppointlink = m_ppointlink;
     ppointlink != (POINTLINK FAR*)NULL;
     ppointlink = ppointlinkNext)
    {
      ppointlinkNext = ppointlink->next;
      ppointlink->ppoint->Release();
      delete ppointlink;
    }

    m_cPoints = 0;
    m_ppointlink = NULL;
    m_ppointlinkLast = NULL;
}


/***
*HRESULT CPoly::AddPoint(short, short)
*Purpose:
*  Add a CPoint with the given coordinates to the end of our current
*  list of points.
*
*Entry:
*  x,y = the x and y coordinates of the new point.
*
*Exit:
*  return value = HRESULT
*
***********************************************************************/
HRESULT METHODCALLTYPE
CPoly::AddPoint(short x, short y)
{
    CPoint FAR* ppoint;
    POINTLINK FAR* ppointlink;

    ppoint = CPoint::Create();
    if(ppoint == (CPoint FAR*)NULL)
      return ResultFromScode(E_OUTOFMEMORY);
    ppoint->SetX(x);
    ppoint->SetY(y);

    ppointlink = new FAR POINTLINK;
    if(ppointlink == (POINTLINK FAR*)NULL){
      delete ppoint;
      return ResultFromScode(E_OUTOFMEMORY);
    }

    ppointlink->ppoint = ppoint;
    ppointlink->next = (POINTLINK FAR*)NULL;

    if(m_ppointlinkLast == (POINTLINK FAR*)NULL){
      m_ppointlink = m_ppointlinkLast = ppointlink;
    }else{
      m_ppointlinkLast->next = ppointlink;
      m_ppointlinkLast = ppointlink;
    }

    ++m_cPoints;

    return NOERROR;
}


/***
*IUnknown FAR* CPoly::EnumPoints(void);
*Purpose:
*  Return and enumerator for the points in this polygon.
*
*Entry:
*  None
*
*Exit:
*  return value = HRESULT
*
*  *ppenum = pointer to an IEnumVARIANT for the points in this polygon
*
***********************************************************************/
IUnknown FAR* METHODCALLTYPE EXPORT
CPoly::EnumPoints()
{
    unsigned int i;
    HRESULT hresult;
    VARIANT var;
    SAFEARRAY FAR* psa;
    IUnknown FAR* punk;
    CEnumPoint FAR* penum;
    POINTLINK FAR* ppointlink;
    SAFEARRAYBOUND rgsabound[1];

    rgsabound[0].lLbound = 0;
    rgsabound[0].cElements = m_cPoints;
    psa = SafeArrayCreate(VT_VARIANT, 1, rgsabound);
    if(psa == NULL){
      hresult = ResultFromScode(E_OUTOFMEMORY);
      goto LError0;
    }


    ppointlink = m_ppointlink;
    for(i = 0; i < m_cPoints; ++i){
      long ix[1];

      if(ppointlink == NULL){
        // this indicates an internal consistency error.
     // (this test should probably be an assertion)
        hresult = ResultFromScode(E_FAIL);
        goto LError1;
      }

      V_VT(&var) = VT_DISPATCH;
      hresult = ppointlink->ppoint->QueryInterface(
     IID_IDispatch, (void FAR* FAR*)&V_DISPATCH(&var));
      if(hresult != NOERROR)
        goto LError1;

      ix[0] = i;
      SafeArrayPutElement(psa, ix, &var);

      ppointlink = ppointlink->next;
    }

    hresult = CEnumPoint::Create(psa, &penum);
    if(hresult != NOERROR)
      goto LError1;

    // ownership of the array has passed to the enumerator
    psa = NULL;

    hresult = penum->QueryInterface(IID_IUnknown, (void FAR* FAR*)&punk);
    if(hresult != NOERROR)
      goto LError2;

    penum->Release();

    return punk;

LError2:;
    penum->Release();

LError1:;
    // destroy the array if we were not successful creating the enumerator.
    if(psa != NULL)
      SafeArrayDestroy(psa);

LError0:;
    return NULL;
```

```
}


short METHODCALLTYPE EXPORT
CPoly::GetXOrigin()
{
    return m_xorg;
}

void METHODCALLTYPE EXPORT
CPoly::SetXOrigin(short x)
{
    m_xorg = x;
}

short METHODCALLTYPE EXPORT
CPoly::GetYOrigin()
{
    return m_yorg;
}

void METHODCALLTYPE EXPORT
CPoly::SetYOrigin(short y)
{
    m_yorg = y;
}

short METHODCALLTYPE EXPORT
CPoly::GetWidth()
{
    return m_width;
}


void METHODCALLTYPE EXPORT
CPoly::SetWidth(short width)
{
    m_width = width;
}

short METHODCALLTYPE EXPORT
CPoly::get_red()
{
    return m_red;
}

void METHODCALLTYPE EXPORT
CPoly::set_red(short red)
{
    m_red = red;
}

short METHODCALLTYPE EXPORT
CPoly::get_green()
{
    return m_green;
}

void METHODCALLTYPE EXPORT
CPoly::set_green(short green)
{
    m_green = green;
}

short METHODCALLTYPE EXPORT
CPoly::get_blue()
{
    return m_blue;
}

void METHODCALLTYPE EXPORT
CPoly::set_blue(short blue)
{
    m_blue = blue;
}


/***
*void CPoly::Dump(void)
*Purpose:
*  Output a debug dump of this instance.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
void METHODCALLTYPE EXPORT
CPoly::Dump()
{
#ifdef _MAC

    // REVIEW: implement for the mac

#else


    TCHAR buffer[80];
    POINTLINK FAR* ppointlink;

    wsprintf(buffer, TSTR("CPoly(0x%x) =\n"), (int)this);
    OutputDebugString(buffer);

    wsprintf(buffer,
      TSTR("    xorg = %d, yorg = %d, width = %d, rgb = {%d,%d,%d}\n
points = "),
      m_xorg, m_yorg, m_width,
      get_red(),
      get_green(),
      get_blue());

    OutputDebugString(buffer);

    for(ppointlink = m_ppointlink;
     ppointlink != (POINTLINK FAR*)NULL;
     ppointlink = ppointlink->next)
    {
      wsprintf(buffer, TSTR("{%d,%d}"),
        ppointlink->ppoint->GetX(),
        ppointlink->ppoint->GetY());
      OutputDebugString(buffer);

      wsprintf(buffer, TSTR(" "));
      OutputDebugString(buffer);
    }
    wsprintf(buffer, TSTR("\n"));
    OutputDebugString(buffer);

#endif
}

/***
*void CPoly::PolyDraw(void)
*Purpose:
*  Draw all polygons.
*
*Entry:
*  None
*
*Exit:
*  None
*
************************************************************************/
void
CPoly::PolyDraw()
{
    POLYLINK FAR* polylink;

    for(polylink = g_ppolylink;
     polylink != (POLYLINK FAR*)NULL;
     polylink = polylink->next)
    {
      polylink->ppoly->Draw();
    }
}


/***
*void PolyTerm(void)
*Purpose:
*  Release all polygons.
*
*Entry:
```

```
*  None
*
*Exit:
*  None
*
***********************************************************************/
void
CPoly::PolyTerm()
{
    POLYLINK FAR* ppolylink;
    POLYLINK FAR* ppolylinkNext;

    for(ppolylink = g_ppolylink;
     ppolylink != (POLYLINK FAR*)NULL;
     ppolylink = ppolylinkNext)
    {
      ppolylinkNext = ppolylink->next;
      ppolylink->ppoly->Release();
      delete ppolylink;
    }
    g_ppolylink = NULL;
}


/***
*void PolyDump(void)
*Purpose:
*  Invoke the debug Dump() method on all polygons were currently
*  holding on to.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
void
CPoly::PolyDump()
{
    POLYLINK FAR* ppolylink;

    if(g_ppolylink == (POLYLINK FAR*)NULL){
#ifndef _MAC
      OutputDebugString(TSTR("\t(none)\n"));
#endif
      return;
    }

    for(ppolylink = g_ppolylink;
     ppolylink != (POLYLINK FAR*)NULL;
     ppolylink = ppolylink->next)
    {
      ppolylink->ppoly->Dump();
    }
}


//---------------------------------------------------------------------
//             Implementation of the CPoly Class Factory
//---------------------------------------------------------------------


CPolyCF::CPolyCF()
{
    m_refs = 0;
}

IClassFactory FAR*
CPolyCF::Create()
{
    CPolyCF FAR* pCF;

    if((pCF = new FAR CPolyCF()) == NULL)
      return NULL;
    pCF->AddRef();

    return pCF;
}

STDMETHODIMP
CPolyCF::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    if(IsEqualIID(riid, IID_IUnknown) || IsEqualIID(riid,
IID_IClassFactory)){
      *ppv = this;
      ++m_refs;
      return NOERROR;
    }
    *ppv = NULL;
    return ResultFromScode(E_NOINTERFACE);
}

STDMETHODIMP_(unsigned long)
CPolyCF::AddRef(void)
{
    return ++m_refs;
}

STDMETHODIMP_(unsigned long)
CPolyCF::Release(void)
{
    if(--m_refs == 0){
      delete this;
      return 0;
    }
    return m_refs;
}

STDMETHODIMP
```

```
CPolyCF::CreateInstance(
    IUnknown FAR* punkOuter,
    REFIID iid,
    void FAR* FAR* ppv)
{
    HRESULT hresult;
    CPoly FAR *ppoly;

    UNUSED(punkOuter);

    if((ppoly = CPoly::Create()) == NULL){
      *ppv = NULL;
      return ResultFromScode(E_OUTOFMEMORY);
    }
    hresult = ppoly->QueryInterface(iid, ppv);
    ppoly->Release();
    return hresult;
}

STDMETHODIMP
#ifdef _MAC
CPolyCF::LockServer(unsigned long fLock)
#else
CPolyCF::LockServer(BOOL fLock)
#endif
{
    UNUSED(fLock);

    return NOERROR;
}
```

HOSTENV.H (SPOLY2 Sample)

```
/***
*hostenv.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*  Generic host specific includes.
*
*Implementation Notes:
*
****************************************************************************
*/

#if defined(_MAC)

#if defined(_PPCMAC)
#pragma data_seg("_FAR_DATA")
#pragma data_seg( )
#define MAXLONG 0x7fffffff
#define EventHandlerProcPtr AEEventHandlerUPP
#else //_PPCMAC
#define  GetMenuItemText(mApple,menuItem,daName)
GetItem(mApple,menuItem,daName)
#endif //_PPCMAC

#endif //_MAC

#if defined(_MAC)

#if defined(_MSC_VER)

# include <values.h>
# include <types.h>
# include <string.h>
# include <quickdra.h>
# include <fonts.h>
# include <events.h>
# include <resource.h>
# include <menus.h>
# include <textedit.h>
# include <dialogs.h>
# include <desk.h>
# include <toolutil.h>
# include <memory.h>
# include <files.h>
# include <osutils.h>
# include <osevents.h>
# include <diskinit.h>
# include <packages.h>
# include <traps.h>
# include <AppleEve.h>
```

```
#else //_MSC_VER

# include <values.h>
# include <types.h>
# include <strings.h>
# include <quickdraw.h>
# include <fonts.h>
# include <events.h>
# include <resources.h>
# include <windows.h>
# include <menus.h>
# include <textedit.h>
# include <dialogs.h>
# include <desk.h>
# include <toolutils.h>
# include <memory.h>
# include <files.h>
# include <osutils.h>
# include <osevents.h>
# include <diskinit.h>
# include <packages.h>
# include <traps.h>
# include <AppleEvents.h>

#endif //_MSC_VER

# include <ole2.h>
# include <olenls.h>
# include <dispatch.h>

# define TCHAR               char
# define TSTR(str)           str
# define STRING(str)           (str)
# define WIDESTRING(str)        (str)

#elif defined(WIN32)

# include <windows.h>
# include <ole2.h>
# include <oleauto.h>

# if defined(UNICODE)
    #define TCHAR      WCHAR
    #define TSTR(str)        L##str
    #define STRING(str)        (str)
    #define WIDESTRING(str)  (str)
# else
    #define TCHAR      char
    #define TSTR(str)        str
    #define STRING(str)        AnsiString(str)
    #define WIDESTRING(str)  WideString(str)
    extern "C" char FAR* AnsiString(OLECHAR FAR* strIn);
    extern "C" OLECHAR FAR* WideString(char FAR* strIn);
# endif
```

```
#else /* WIN16 */

# include <windows.h>
# include <ole2.h>
# include <olenls.h>
# include <dispatch.h>

# define TCHAR               char
# define TSTR(str)           str
# define STRING(str)           (str)
# define WIDESTRING(str)        (str)
#endif
```

MISC.CPP (SPOLY2 Sample)

```
/***
*misc.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*
*Implementation Notes:
*
****************************************************************************
*/


#include "spoly.h"
#include "cpoint.h"
#include "cpoly.h"

#include <stdio.h>

#ifdef _MAC
# include <string.h>
# include <ctype.h>
#endif

unsigned long g_dwPolyCF = 0;
unsigned long g_dwPointCF = 0;

IClassFactory FAR* g_ppolyCF = NULL;
IClassFactory FAR* g_ppointCF = NULL;


#ifdef _MAC
struct regentry{
    char *szKey;
    char *szValue;
} g_rgregentry[] = {

      { "CLSID\\{00020464-0000-0000-C000-000000000046}",
     "OLE Automation SPoly2 1.0 Application" }

    , { "CLSID\\{00020464-0000-0000-C000-000000000046}\\LocalServer",
        "SPL2" }

    , { "CLSID\\{00020464-0000-0000-C000-000000000046}\\ProgID",
     "SPoly2.Application" }

    , { "CLSID\\{00020464-0000-0000-C000-000000000046}\\InprocHandler",
     "OLE2:Def$DefFSet" }

    , { "SPL2", "{00020464-0000-0000-C000-000000000046}" }

    , { "SPoly2.Application\\CLSID",
     "{00020464-0000-0000-C000-000000000046}" }

};

HRESULT
EnsureRegistration()
{
    HKEY hkey;

    if(RegOpenKey(HKEY_CLASSES_ROOT, "SPL2", &hkey) == NOERROR){
      RegCloseKey(hkey);
      return NOERROR;
    }

    for(int i = 0; i < DIM(g_rgregentry); ++i){
      if(RegSetValue(HKEY_CLASSES_ROOT, g_rgregentry[i].szKey, REG_SZ,
g_rgregentry[i].szValue, 0) != ERROR_SUCCESS)
        return ResultFromScode(E_FAIL);
    }

    return NOERROR;
}
#endif

/***
*HRESULT InitOle(void)
*Purpose:
*  Initialize Ole, and register our class factories.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
STDAPI
InitOle()
{
    HRESULT hresult;

    if((hresult = OleInitialize(NULL)) != NOERROR)
      goto LError0;

#ifdef _MAC
    if((hresult = EnsureRegistration()) != NOERROR)
      goto LError0;
#endif

    // Register the CPoint Class Factory
    //
    if((g_ppointCF = CPointCF::Create()) == NULL){
      hresult = ResultFromScode(E_OUTOFMEMORY);
      goto LError1;
    }
    hresult = CoRegisterClassObject(
      CLSID_CPoint2,
      g_ppointCF,
      CLSCTX_LOCAL_SERVER,
      REGCLS_MULTIPLEUSE,
      &g_dwPointCF);
    if(hresult != NOERROR)
      goto LError1;

    // Register the CPoly Class Factory.
    //
    if((g_ppolyCF = CPolyCF::Create()) == NULL){
      hresult = ResultFromScode(E_OUTOFMEMORY);
      goto LError1;
    }

    hresult = CoRegisterClassObject(
      CLSID_CPoly2,
      g_ppolyCF,
      CLSCTX_LOCAL_SERVER,
      REGCLS_MULTIPLEUSE,
      &g_dwPolyCF);
    if(hresult != NOERROR)
      goto LError1;

    g_ppolyCF->Release();

    g_ppointCF->Release();

    return NOERROR;


LError1:;
    if(g_ppolyCF != NULL)
      g_ppolyCF->Release();

    if(g_ppointCF != NULL)
      g_ppointCF->Release();

    UninitOle();

LError0:;
    return hresult;
}

STDAPI
UninitOle()
{
    // Tell Ole to release our class factories.
    //
    if(g_dwPointCF != 0L)
      CoRevokeClassObject(g_dwPointCF);
    if(g_dwPolyCF != 0L)
      CoRevokeClassObject(g_dwPolyCF);

    OleUninitialize();

    return NOERROR;
}

// disable unicode expansion for assertions
#undef UNICODE

void
Assert(int fCond, char FAR* file, int line, char FAR* message)
{
    char * fmt;
    char buf[128];

    if(fCond)
      return;

    fmt = (message == NULL)
      ? "Assertion failed: %s(%d)"
      : "Assertion failed: %s(%d) '%s'";
    sprintf(buf, fmt, file, line, message);

#ifdef _MAC
    DebugStr(c2pstr(buf));
#else
#ifdef WIN32
    OutputDebugStringA(buf);
#else //WIN32
    OutputDebugString(buf);
#endif //WIN32
    DebugBreak();
#endif
}

#ifdef _MAC
#if defined(_MSC_VER)
int pascal
#else
pascal int
#endif
stricmp(char *first, char *last)
{
    unsigned short f, l;

    do{
     f = tolower(*first++);
     l = tolower(*last++);
    }while(f && f == l);

    return f - l;
}
#endif
```

RESOURCE.H   (SPOLY2 Sample)

```
/***
*resource.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*
*Implementation Notes:
*
*****************************************************************************
*/

#ifdef _MAC

#define kMinSize 500   /* minimum size (in K) */
#define kPrefSize500   /* preferred size (in K) */

#define    rMenuBar    128   /* menu bar */
#define    rAboutAlert 128   /* about alert */
#define    rUserAlert  129   /* error alert */
#define    rWindow           128   /* application's window */

#define    mApple            128   /* Apple menu */
#define    iAbout            1

#define    mFile       129   /* File menu */
#define    iNew        1
#define    iClose            4
#define    iQuit       12

#define    mEdit       130   /* Edit menu */
#define    iUndo       1
#define    iCut        3
#define    iCopy       4
#define    iPaste            5
#define    iClear            6

#define mSpoly         131
#define iTest          1

#define kMinHeap 21 * 1024
#define kMinSpace8 * 1024

#else /* WIN16 || WIN32 */

# define IDM_CLEAR     1
# define IDM_DUMP2
# define IDM_FIRSTCHILD100

#endif
```

SPOLY.H   (SPOLY2 Sample)

```
/***
*spoly.h - Application-wide definitions
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*
*Implementation Notes:
*
*****************************************************************************
*/

#include "hostenv.h"
#include "resource.h"
#include "clsid.h"

#if defined(_MAC)
# define STRSTR strstr
#elif defined(WIN32)
# include "statbar.h"
# define STRSTR strstr
#else /* WIN16 */
# include "statbar.h"
# define STRSTR _fstrstr
#endif

#ifdef _MAC
# define UNUSED(X) ((void)(void*)&(X))
#else
# define UNUSED(X) (X)
#endif

#define DIM(X) (sizeof(X) / sizeof(X[0]))

extern "C" void Assert(int, char FAR*, int, char FAR*);
#define ASSERT(X) Assert(X, __FILE__, __LINE__, NULL)
#define ASSERTSZ(X, MSG) Assert(X, __FILE__, __LINE__, MSG)

#ifndef EXPORT
# if defined(WIN32)
#  define EXPORT
# elif defined(_MAC)
#  define EXPORT
# else
#  define EXPORT __export
# endif
#endif

#ifndef NEAR
# if defined(WIN32)
#  define NEAR
# elif defined(_MAC)
#  define NEAR
# else
#  define NEAR __near
# endif
#endif

#if defined(WIN32)
# define CC_CALL        CC_STDCALL
# define METHODCALLTYPE __stdcall
#elif defined(_MAC)
# define CC_CALL        CC_CDECL
# define METHODCALLTYPE
#else
# define CC_CALL        CC_PASCAL
# define METHODCALLTYPE __pascal
#endif

STDAPI InitOle();
STDAPI UninitOle();
```

SPOLY2.DEF (SPOLY2 Sample)

```
NAME          SPOLY2

DESCRIPTION  'IDispatch Polygon Test Server #2'

EXETYPE       WINDOWS

STUB          'WINSTUB.EXE'

CODE          PRELOAD MOVEABLE DISCARDABLE
DATA          PRELOAD MULTIPLE

HEAPSIZE      4096
STACKSIZE     8192
```

SPOLY2.RC   (SPOLY2 Sample)

```
#define NOKERNEL
#define NOGDI
#define NOSOUND
#define NOCOMM
#define NODRIVERS
#include "windows.h"
#include "spoly.h"

SPOLY ICON spoly2.ico

SPolyMenu MENU
BEGIN
    MENUITEM     "&Clear"                IDM_CLEAR
    MENUITEM     "&Dump"                       IDM_DUMP
END
```

SPOLY2.REG (SPOLY2 Sample)

```
REGEDIT

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; registration info SPoly2.Application (defaults to SPoly2.Application.1)

HKEY_CLASSES_ROOT\SPoly2.Application = OLE Automation SPoly2 Application
HKEY_CLASSES_ROOT\spoly2.Application\Clsid = {00020464-0000-0000-C000-
000000000046}


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; registration info SPoly2 1.0

; (Application Object)
HKEY_CLASSES_ROOT\SPoly2.Application.1 = OLE Automation SPoly2 1.0
Application
HKEY_CLASSES_ROOT\SPoly2.Application.1\Clsid = {00020464-0000-0000-C000-
000000000046}

HKEY_CLASSES_ROOT\CLSID\{00020464-0000-0000-C000-000000000046} = OLE
Automation Spoly2 1.0 Application
HKEY_CLASSES_ROOT\CLSID\{00020464-0000-0000-C000-000000000046}\ProgID =
SPoly2.Application
HKEY_CLASSES_ROOT\CLSID\{00020464-0000-0000-C000-
000000000046}\VersionIndependentProgID = SPoly2.Application
HKEY_CLASSES_ROOT\CLSID\{00020464-0000-0000-C000-000000000046}\LocalServer32
= spoly2.exe /Automation


HKEY_CLASSES_ROOT\CLSID\{00020465-0000-0000-C000-000000000046} =
SPoly2.SPoint2
HKEY_CLASSES_ROOT\CLSID\{00020465-0000-0000-C000-000000000046}\ProgID =
SPoly2.Application.1
HKEY_CLASSES_ROOT\CLSID\{00020465-0000-0000-C000-
000000000046}\VersionIndependentProgID = SPoly.Application
HKEY_CLASSES_ROOT\CLSID\{00020465-0000-0000-C000-000000000046}\LocalServer32
= spoly2.exe /Automation
```

STATBAR.H (SPOLY2 Sample)

```
/***
*statbar.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*
*Implementation Notes:
*  This file requires windows.h and ole2.h
*
*****************************************************************************
*/

class CStatBar : public IUnknown {
public:
    static CStatBar FAR* Create(HANDLE hinst, HWND hwndFrame);

    // IUnknown methods
    //
    STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppv);
    STDMETHOD_(unsigned long, AddRef)(void);
    STDMETHOD_(unsigned long, Release)(void);

    // Introduced methods
    //
    void Show(void);
    inline void Move(void);
    inline void Update(void);

    inline int GetX(void);
    inline void SetX(int x);

    inline int GetY(void);
    inline void SetY(int y);

    inline int GetHeight(void);
    inline void SetHeight(int height);

    inline int GetWidth(void);
    inline void SetWidth(int width);

    //inline HFONT GetFont(void);
    void SetFont(HFONT hfont);

    //char FAR* GetText(void);
    inline void SetText(OLECHAR FAR* sz);

    void WMPaint(void);
    BOOL Register(HANDLE);

private:
    CStatBar();
    ~CStatBar();

    unsigned longm_refs;

    HWND   m_hwnd;                 // the status bar window handle

    int          m_x;              // x coordinate of upper left corner
    int          m_y;              // y coordinate of upper left corner
    int          m_height;
    int          m_width;

    HFONT  m_hfont;
    int          m_dyFont;         // font height
    int          m_dxFont;         // font width

    BSTR   m_bstrMsg;        // the status bar text

    static TCHAR FAR* m_szWndClass;
};

inline void
CStatBar::Move()
{
    MoveWindow(m_hwnd, m_x, m_y, m_width, m_height, TRUE);
}

inline void
CStatBar::Update()
{
    InvalidateRect(m_hwnd, NULL, TRUE);
    UpdateWindow(m_hwnd);
}

inline int
CStatBar::GetX()
{
    return m_x;
}

inline void
CStatBar::SetX(int x)
{
    m_x = x;
}

inline int
CStatBar::GetY(void)
{
    return m_y;
}

inline void
CStatBar::SetY(int y)
{
    m_y = y;
}

inline int
CStatBar::GetHeight(void)
{
    return m_height;
}

inline void
CStatBar::SetHeight(int height)
{
    m_height = height;
}

inline int
CStatBar::GetWidth(void)
{
    return m_width;
}

inline void
CStatBar::SetWidth(int width)
{
    m_width = width;
}

inline void
CStatBar::SetText(OLECHAR FAR* sz)
{
    SysFreeString(m_bstrMsg);
    m_bstrMsg = SysAllocString(sz);
}

extern "C" void
SBprintf(CStatBar FAR* psb, TCHAR FAR* szFmt, ...);
```

STATBAR.CPP   (SPOLY2 Sample)

```
/***
*statbar.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*
*Implementation Notes:
*
****************************************************************************
*/

#include <stdarg.h>

#include "hostenv.h"
#include "statbar.h"


extern "C" long FAR PASCAL StatBarWndProc(HWND, unsigned int, WPARAM,
LPARAM);


TCHAR FAR* CStatBar::m_szWndClass = TSTR("StatBarWndClass");


CStatBar::CStatBar()
{
    m_refs = 0;

    m_x = 0;
    m_y = 0;
    m_width = 0;
    m_height = 0;

    m_bstrMsg = NULL;

    m_hfont = (HANDLE)0;
}

CStatBar::~CStatBar()
{
    SysFreeString(m_bstrMsg);
}


/***
*PUBLIC CStatBar FAR* CStatBar::Create(HANDLE, HWND)
*
*Purpose:
*
*Entry:
```

```
*
*Exit:
*
***********************************************************************/
CStatBar FAR*
CStatBar::Create(HANDLE hinst, HWND hwndFrame)
{
    CStatBar FAR* psb;

    psb = new FAR CStatBar();
    if(psb == NULL)
      return NULL;
    psb->AddRef();

    if(!psb->Register(hinst))
      goto LFail;

    psb->m_hwnd = CreateWindow(
      CStatBar::m_szWndClass,
      NULL,
      WS_CHILD | WS_CLIPSIBLINGS,
      0, 0, 0, 0,
      hwndFrame,
      0,
      hinst,
      NULL);

    if(!psb->m_hwnd)
      goto LFail;

    // Stash the newly created CStatBar* in the extra bytes of the
    // associated window so we can get at the instance in the message
    // proc.
    //
    // Note: we do not AddRef for this reference. We make sure that the
    // window is destroyed when the refcnt goes to 0.
    //
    SetWindowLong(psb->m_hwnd, 0, (long)psb);

    return psb;

LFail:;
    delete psb;
    return NULL;
}


//---------------------------------------------------------------------
//                     IUnknown Methods
//---------------------------------------------------------------------


STDMETHODIMP
CStatBar::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
    if(IsEqualIID(riid,IID_IUnknown)){
      *ppv = this;
      AddRef();
      return NOERROR;
    }
    *ppv = (void FAR*)NULL;
    return ResultFromScode(E_NOINTERFACE);
}


STDMETHODIMP_(unsigned long)
CStatBar::AddRef(void)
{
    return ++m_refs;
}


STDMETHODIMP_(unsigned long)
CStatBar::Release(void)
{
    if(--m_refs == 0){

      // destroy the status bar window.
      //
      SendMessage(m_hwnd, WM_DESTROY, 0, 0L);

      delete this;
      return 0;
    }

    return m_refs;
}



//---------------------------------------------------------------------
//                     Introduced Methods
//---------------------------------------------------------------------


/***
*PRIVATE BOOL CStatBar::Register(HANDLE)
*
*Purpose:
*  Register the status bar window class.
*
*Entry:
*  None
*
*Exit:
*  return value = BOOL, TRUE if successful, FALSE if not.
*
***********************************************************************/
BOOL
CStatBar::Register(HANDLE hinst)
```

```
{
    WNDCLASS  wc;

    // register the class, unless already registered.
    if(GetClassInfo(hinst, m_szWndClass, &wc) == 0){
      wc.style         = 0;
      wc.lpfnWndProc   = StatBarWndProc;
      wc.cbClsExtra    = 0;
      wc.cbWndExtra    = sizeof(CStatBar FAR*);
      wc.hInstance     = hinst;
      wc.hIcon         = 0;
      wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
      wc.hbrBackground = GetStockObject(LTGRAY_BRUSH);
      wc.lpszMenuName  = 0;
      wc.lpszClassName = CStatBar::m_szWndClass;
      if(!RegisterClass(&wc))
        return FALSE;
    }
    return TRUE;
}


/***
*PUBLIC void CStatBar::Show(void)
*
*Purpose:
*  Show the status bar window associated with this CStatBar instance.
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
void
CStatBar::Show()
{
    ShowWindow(m_hwnd, SW_SHOW);
}

void
CStatBar::SetFont(HFONT hfont)
{
    HDC hdc;
    TEXTMETRIC tm;
    HFONT hfontOld;

    // compute the character sizes given this new font.
    //
    hdc = GetDC(m_hwnd);
    hfontOld = SelectObject(hdc, hfont);
    GetTextMetrics(hdc, &tm);
    m_dxFont = tm.tmAveCharWidth;
    m_dyFont = tm.tmHeight + tm.tmExternalLeading;
    SelectObject(hdc, hfontOld);
    ReleaseDC(m_hwnd, hdc);

    m_hfont = hfont;
}

/***
*PRIVATE CStatBar::WMPaint(void)
*
*Purpose:
*  This method is responsible for drawing the status bar, and is called
*  in response to a WM_PAINT message.
*
*Entry:
*  None
*
*Exit:
*  None
*
************************************************************************/
void
CStatBar::WMPaint()
{
    HDC hdc;
    RECT rcMsg;
    HRGN hrgn;
    HFONT hfontOld;
    PAINTSTRUCT ps;
    HPEN hpenBlack, hpenWhite, hpenGray, hpenOld;

    hdc = BeginPaint(m_hwnd, &ps);

    // compute the message box rect
    //
    rcMsg.top    = 3;
    rcMsg.bottom= m_height - 3;
    rcMsg.left   = m_dxFont;
    rcMsg.right  = m_width - m_dxFont;

    // prepare the pens
    //
    hpenWhite    = GetStockObject(WHITE_PEN);
    hpenBlack    = GetStockObject(BLACK_PEN);
    hpenGray     = CreatePen(PS_SOLID, 1, GetSysColor(COLOR_BTNSHADOW));

    // draw a top gray line
    //
    hpenOld = SelectObject(hdc, hpenGray);
#if   defined(WIN16)
    MoveTo(hdc, ps.rcPaint.left, 0);
#elif defined(WIN32)
    MoveToEx(hdc, ps.rcPaint.left, 0, NULL);
#endif
    LineTo(hdc, ps.rcPaint.right, 0);
    // draw a white line just under
    //
    SelectObject(hdc, hpenWhite);
#if   defined(WIN16)
    MoveTo(hdc, ps.rcPaint.left, 1);
#elif defined(WIN32)
    MoveToEx(hdc, ps.rcPaint.left, 1, NULL);
#endif
    LineTo(hdc, ps.rcPaint.right, 1);


    // do not overwrite the background color
    //
    SetBkMode(hdc, TRANSPARENT);

    // message area
    //
    SelectObject(hdc, hpenBlack);
#if   defined(WIN16)
    MoveTo(hdc, rcMsg.left,  rcMsg.bottom);
#elif defined(WIN32)
    MoveToEx(hdc, rcMsg.left,  rcMsg.bottom, NULL);
#endif
    LineTo(hdc, rcMsg.left,  rcMsg.top);
    LineTo(hdc, rcMsg.right, rcMsg.top);

    SelectObject(hdc, hpenWhite);
    LineTo(hdc, rcMsg.right, rcMsg.bottom);
    LineTo(hdc, rcMsg.left,  rcMsg.bottom);

    // select the black pen for writing
    //
    SelectObject(hdc, hpenBlack);

    // select the status bar font to write in
    //
    hfontOld = SelectObject(hdc, m_hfont);

    // set the clipping region
    //
    hrgn = CreateRectRgn(
      rcMsg.left, rcMsg.top, rcMsg.right, rcMsg.bottom);

    SelectClipRgn(hdc, hrgn);

    // draw the status message
    //
    TextOut(
      hdc,
      rcMsg.left + (m_dxFont / 2),
      rcMsg.top + ((rcMsg.bottom - rcMsg.top - m_dyFont) / 2),
      STRING(m_bstrMsg), (SysStringLen(m_bstrMsg)));

    // cleanup
    //
    SelectObject(hdc, hpenOld);
    SelectObject(hdc, hfontOld);

    DeleteObject(hrgn);
    DeleteObject(hpenGray);

    EndPaint(m_hwnd, &ps);
}

extern "C" long FAR PASCAL
StatBarWndProc(
    HWND hwnd,
    unsigned int message,
    WPARAM wParam,
    LPARAM lParam)
{
    CStatBar FAR* psb;

    switch(message){
    case WM_SIZE:
      return 0;
    case WM_PAINT:
      psb = (CStatBar FAR*)GetWindowLong(hwnd, 0);
      psb->WMPaint();
      return 0;
    }
    return(DefWindowProc(hwnd, message, wParam, lParam));
}


//---------------------------------------------------------------------
//                    Status Bar Utilities
//---------------------------------------------------------------------

extern "C" void
SBprintf(CStatBar FAR* psb, TCHAR FAR* szFmt, ...)
{
    va_list args;
static TCHAR buf[256];

    va_start(args, szFmt);
    wvsprintf(buf, szFmt, args);
    psb->SetText(WIDESTRING(buf));
    psb->Update();
}
```

TDATA.CPP (SPOLY2 Sample)

```
/***
*tdata.cpp
*
*  Copyright (C) 1991-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  CPoly and CPoint type data descriptions.
*
*  These data descriptions are used to construct TypeInfos for these
*  objects at runtime.
*
****************************************************************************
*/

#include "spoly.h"
#include "cpoint.h"
#include "cpoly.h"


//---------------------------------------------------------------------
//                      CPoint type data definitions
//---------------------------------------------------------------------


static PARAMDATA NEAR rgpdataCPointSetX[] =
{
    { OLESTR("X"), VT_I2 }
};

static PARAMDATA NEAR rgpdataCPointSetY[] =
{
    { OLESTR("Y"), VT_I2 }
};

static METHODDATA NEAR rgmdataCPoint[] =
{
    // CPoint::GetX()
    {
     OLESTR("GetX"),
     NULL,
     IDMEMBER_CPOINT_GETX,
     IMETH_CPOINT_GETX,
     CC_CALL,
     0,
     DISPATCH_METHOD,
     VT_I2
    },

    // CPoint::SetX()
    {
     OLESTR("SetX"),
     rgpdataCPointSetX,
     IDMEMBER_CPOINT_SETX,
     IMETH_CPOINT_SETX,
     CC_CALL,
     DIM(rgpdataCPointSetX),
     DISPATCH_METHOD,
     VT_EMPTY
    },

    // CPoint::GetY()
    {
     OLESTR("GetY"),
     NULL,
     IDMEMBER_CPOINT_GETY,
     IMETH_CPOINT_GETY,
     CC_CALL,
     0,
     DISPATCH_METHOD,
     VT_I2
    },

    // CPoint::SetY()
    {
     OLESTR("SetY"),
     rgpdataCPointSetY,
     IDMEMBER_CPOINT_SETY,
     IMETH_CPOINT_SETY,
     CC_CALL,
     DIM(rgpdataCPointSetY),
     DISPATCH_METHOD,
     VT_EMPTY
    }
};

INTERFACEDATA NEAR g_idataCPoint =
{
    rgmdataCPoint, DIM(rgmdataCPoint)
};


//---------------------------------------------------------------------
//                   CPoly type data definitions
//---------------------------------------------------------------------


static PARAMDATA NEAR rgpdataCPolyAddPoint[] =
{
    { OLESTR("x"),     VT_I2 },
    { OLESTR("y"),     VT_I2 }
};

static PARAMDATA NEAR rgpdataCPolySetXOrigin[] =
{
    { OLESTR("x"),     VT_I2 }
};
```

```
static PARAMDATA NEAR rgpdataCPolySetYOrigin[] =
{
    { OLESTR("y"),     VT_I2 }
};

static PARAMDATA NEAR rgpdataCPolySetWidth[] =
{
    { OLESTR("width"), VT_I2 }
};

static PARAMDATA NEAR rgpdataCPolySetRed[] =
{
    { OLESTR("red"),   VT_I2 }
};

static PARAMDATA NEAR rgpdataCPolySetGreen[] =
{
    { OLESTR("green"), VT_I2 }
};

static PARAMDATA NEAR rgpdataCPolySetBlue[] =
{
    { OLESTR("blue"),  VT_I2 }
};

static METHODDATA NEAR rgmdataCPoly[] =
{
    // void CPoly::Draw(void)
    {
     OLESTR("Draw"),
     NULL,
     IDMEMBER_CPOLY_DRAW,
     IMETH_CPOLY_DRAW,
     CC_CALL,
     0,
     DISPATCH_METHOD,
     VT_EMPTY
    },

    // void CPoly::Reset(void)
    {
     OLESTR("Reset"),
     NULL,
     IDMEMBER_CPOLY_RESET,
     IMETH_CPOLY_RESET,
     CC_CALL,
     0,
     DISPATCH_METHOD,
     VT_EMPTY
    },

    // HRESULT CPoly::AddPoint(short x, short y)
    {
```

```
 OLESTR("AddPoint"),
 rgpdataCPolyAddPoint,
 IDMEMBER_CPOLY_ADDPOINT,
 IMETH_CPOLY_ADDPOINT,
 CC_CALL,
 DIM(rgpdataCPolyAddPoint),
 DISPATCH_METHOD,
 VT_ERROR
},

// IUnknown FAR* CPoly::EnumPoints(void)
{
 OLESTR("EnumPoints"),
 NULL,
 IDMEMBER_CPOLY_ENUMPOINTS,
 IMETH_CPOLY_ENUMPOINTS,
 CC_CALL,
 0,
 DISPATCH_METHOD,
 VT_UNKNOWN
},

// short CPoly::GetXOrigin(void)
{
 OLESTR("GetXOrigin"),
 NULL,
 IDMEMBER_CPOLY_GETXORIGIN,
 IMETH_CPOLY_GETXORIGIN,
 CC_CALL,
 0,
 DISPATCH_METHOD,
 VT_I2
},

// void CPoly::SetXOrigin(short x)
{
 OLESTR("SetXOrigin"),
 rgpdataCPolySetXOrigin,
 IDMEMBER_CPOLY_SETXORIGIN,
 IMETH_CPOLY_SETXORIGIN,
 CC_CALL,
 DIM(rgpdataCPolySetXOrigin),
 DISPATCH_METHOD,
 VT_EMPTY
},

// short CPoly::GetYOrigin(void)
{
 OLESTR("GetYOrigin"),
 NULL,
 IDMEMBER_CPOLY_GETYORIGIN,
 IMETH_CPOLY_GETYORIGIN,
 CC_CALL,
 0,
 DISPATCH_METHOD,
 VT_I2
},

// void CPoly::SetYOrigin(short y)
{
 OLESTR("SetYOrigin"),
 rgpdataCPolySetYOrigin,
 IDMEMBER_CPOLY_SETYORIGIN,
 IMETH_CPOLY_SETYORIGIN,
 CC_CALL,
 DIM(rgpdataCPolySetYOrigin),
 DISPATCH_METHOD,
 VT_EMPTY
},

// short CPoly::GetWidth(void)
{
 OLESTR("GetWidth"),
 NULL,
 IDMEMBER_CPOLY_GETWIDTH,
 IMETH_CPOLY_GETWIDTH,
 CC_CALL,
 0,
 DISPATCH_METHOD,
 VT_I2
},

// void CPoly::SetWidth(short width)
{
 OLESTR("SetWidth"),
 rgpdataCPolySetWidth,
 IDMEMBER_CPOLY_SETWIDTH,
 IMETH_CPOLY_SETWIDTH,
 CC_CALL,
 DIM(rgpdataCPolySetWidth),
 DISPATCH_METHOD,
 VT_EMPTY
},

// short CPoly::get_red(void)
{
 OLESTR("get_red"),
 NULL,
 IDMEMBER_CPOLY_GETRED,
 IMETH_CPOLY_GETRED,
 CC_CALL,
 0,
 DISPATCH_METHOD,
 VT_I2
},

// void CPoly::set_red(short red)
{
 OLESTR("set_red"),
 rgpdataCPolySetRed,
 IDMEMBER_CPOLY_SETRED,
 IMETH_CPOLY_SETRED,
 CC_CALL,
 DIM(rgpdataCPolySetRed),
 DISPATCH_METHOD,
 VT_EMPTY
},

// short CPoly::get_green(void)
{
 OLESTR("get_green"),
 NULL,
 IDMEMBER_CPOLY_GETGREEN,
 IMETH_CPOLY_GETGREEN,
 CC_CALL,
 0,
 DISPATCH_METHOD,
 VT_I2
},

// void CPoly::set_green(short green)
{

 OLESTR("set_green"),
 rgpdataCPolySetGreen,
 IDMEMBER_CPOLY_SETGREEN,
 IMETH_CPOLY_SETGREEN,
 CC_CALL,
 DIM(rgpdataCPolySetGreen),
 DISPATCH_METHOD,
 VT_EMPTY
},

// short CPoly::get_blue(void)
{
 OLESTR("get_blue"),
 NULL,
 IDMEMBER_CPOLY_GETBLUE,
 IMETH_CPOLY_GETBLUE,
 CC_CALL,
 0,
 DISPATCH_METHOD,
 VT_I2
},

// void CPoly::set_blue(short blue)
{
 OLESTR("set_blue"),
 rgpdataCPolySetBlue,
 IDMEMBER_CPOLY_SETBLUE,
 IMETH_CPOLY_SETBLUE,
 CC_CALL,
 DIM(rgpdataCPolySetBlue),
 DISPATCH_METHOD,
 VT_EMPTY
```

```
    },

    // void CPoly::Dump(void)
    {
     OLESTR("Dump"),
     NULL,
     IDMEMBER_CPOLY_DUMP,
     IMETH_CPOLY_DUMP,
     CC_CALL,
     0,
     DISPATCH_METHOD,
     VT_EMPTY
    },

};

INTERFACEDATA NEAR g_idataCPoly =
{
    rgmdataCPoly, DIM(rgmdataCPoly)
};
```

WINMAIN.CPP (SPOLY2 Sample)

```
/***
*winmain.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  This module is the main entry point for the sample IDispatch polygon
*  server, spoly2.exe.
*
*  This program is intended to demonstrate an implementation of the
IDispatch
*  interface. Spoly is a very simple app, that implements two simple
objects,
*  CPoly and CPoint and exposes their properties and methods for programatic
*  and cross-process access via IDispatch.
*
*Implementation Notes:
*
*****************************************************************************
*/

#include <stdio.h>
#include <string.h>

#include "spoly.h"
#include "cpoint.h"
#include "cpoly.h"


HANDLE g_hinst = 0;

HWND g_hwndFrame = 0;
HWND g_hwndClient = 0;

TCHAR g_szFrameWndClass[] = TSTR("FrameWClass");

CStatBar FAR* g_psb = NULL;


BOOL InitApplication(HANDLE);
BOOL InitInstance(HANDLE, int);

extern "C" int PASCAL WinMain(HANDLE, HANDLE, LPSTR, int);
extern "C" long FAR PASCAL FrameWndProc(HWND, UINT, WPARAM, LPARAM);


extern "C" int PASCAL
WinMain(
    HANDLE hinst,
    HANDLE hPrevInstance,
    LPSTR lpCmdLine,
    int nCmdShow)
{
    MSG msg;
    int retval;
    HRESULT hresult;

    if(!hPrevInstance)
      if(!InitApplication(hinst))
        return FALSE;

    if((hresult = InitOle()) != NOERROR)
      return FALSE;

    if(!InitInstance(hinst, nCmdShow)){
      retval = FALSE;
      UninitOle();
      goto LExit;
    }

    while(GetMessage(&msg, NULL, NULL, NULL)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }

    CPoly::PolyTerm();

    retval = msg.wParam;

LExit:;

    return retval;
}


BOOL
InitApplication(HANDLE hinst)
{
    WNDCLASS  wc;

    wc.style            = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc      = FrameWndProc;
    wc.cbClsExtra       = 0;
    wc.cbWndExtra       = 0;
    wc.hInstance        = hinst;
    wc.hIcon            = LoadIcon(hinst, TSTR("SPOLY"));
    wc.hCursor          = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground    = (HBRUSH) (COLOR_APPWORKSPACE+1);
    wc.lpszMenuName     = TSTR("SPolyMenu");
    wc.lpszClassName    = g_szFrameWndClass;

    if(!RegisterClass(&wc))
      return FALSE;

    return TRUE;
}
```

```
#ifdef WIN32
#define szAppTitle TSTR("IDispatch Polygon Server #2 (32-bit)")
#else //WIN32
#define szAppTitle TSTR("IDispatch Polygon Server #2")
#endif //WIN32

BOOL
InitInstance(HANDLE hinst, int nCmdShow)
{
    g_hinst = hinst;

    // Create a main frame window
    //
    g_hwndFrame = CreateWindow(
      g_szFrameWndClass,
      szAppTitle,
      WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
      CW_USEDEFAULT,
      CW_USEDEFAULT,
      CW_USEDEFAULT,
      CW_USEDEFAULT,
      NULL,
      NULL,
      hinst,
      NULL);
    if(!g_hwndFrame)
      return FALSE;

    g_hwndClient = GetWindow(g_hwndFrame, GW_CHILD);
    if(!g_hwndClient)
      return FALSE;

    // create the status bar
    //
    g_psb = CStatBar::Create(g_hinst, g_hwndFrame);
    if(!g_psb)
      return FALSE;

    // initialize and show the status bar
    //
    g_psb->SetHeight(GetSystemMetrics(SM_CYCAPTION) - 1);
    g_psb->SetFont(GetStockObject(SYSTEM_FONT));
    g_psb->SetText(OLESTR(""));
    g_psb->Show();

    ShowWindow(g_hwndFrame, nCmdShow);

    UpdateWindow(g_hwndFrame);

    return TRUE;
}
```

```
void
FrameWndOnCreate(HWND hwnd)
{
    CLIENTCREATESTRUCT ccs;

    ccs.hWindowMenu = NULL;
    ccs.idFirstChild = IDM_FIRSTCHILD;

    g_hwndClient = CreateWindow(
      TSTR("MDICLIENT"),
      0,
      WS_CHILD | WS_CLIPCHILDREN | WS_VISIBLE,
      0, 0, 0, 0,
      hwnd,
      (HMENU) 1,
      g_hinst,
      &ccs);
}


void
FrameWndOnSize(HWND hwnd)
{
    RECT rc;
    int height;

    // Get the client rectangle for the frame window
    GetClientRect(hwnd, &rc);

    height = g_psb->GetHeight();

    // adjust the client win to make room for the status bar.
    //
    MoveWindow(
      g_hwndClient,
      rc.left,
      rc.top,
      rc.right - rc.left,
      rc.bottom - rc.top - height,
      TRUE);

    // move the status bar to the bottom of the newly positioned window.
    //
    g_psb->SetX(rc.left);
    g_psb->SetY(rc.bottom - height),
    g_psb->SetWidth(rc.right - rc.left);
    g_psb->Move();
}


extern "C" long FAR PASCAL
FrameWndProc(
    HWND hwnd,
    UINT message,
    WPARAM wParam,
    LPARAM lParam)
{
    switch(message){
    case WM_COMMAND:
      switch(wParam){
      case IDM_DUMP:
        CPoly::PolyDump();
        return 0;

      case IDM_CLEAR:
        InvalidateRect(g_hwndClient, NULL, TRUE);
        return 0;
      }
      break;

    case WM_CREATE:
      FrameWndOnCreate(hwnd);
      break;

    case WM_SIZE:
      FrameWndOnSize(hwnd);
      return 1;

    case WM_PAINT:
      CPoly::PolyDraw();
      break;

    case WM_CLOSE:
      DestroyWindow(hwnd);
      return 0;

    case WM_DESTROY:
      UninitOle();
      PostQuitMessage(0);
      return 0;
    }
    return DefFrameProc(hwnd, g_hwndClient, message, wParam, lParam);
}


#if defined(WIN32)

extern "C" OLECHAR FAR*
ConvertStrAtoW(char FAR* strIn, OLECHAR FAR* buf, UINT size)
{
  MultiByteToWideChar(CP_ACP, MB_PRECOMPOSED,
                       strIn, -1, buf, size) ;
  return buf;
}

extern "C" OLECHAR FAR*
WideString(char FAR* strIn)
{
  static OLECHAR buf[256];
  return (ConvertStrAtoW(strIn, buf, 256));
}

extern "C" char FAR*
ConvertStrWtoA(OLECHAR FAR* strIn, char FAR* buf, UINT size)
{
  int badConversion = FALSE;

  WideCharToMultiByte(CP_ACP, NULL,
                      strIn, -1,
                      buf, size,
                      NULL, &badConversion);
  return buf;
}


extern "C" char FAR*
AnsiString(OLECHAR FAR* strIn)
{
  static char buf[256];

  return (ConvertStrWtoA(strIn, buf, 256));
}

#endif
```

TIBROWSE

-------------------------------------- OLE Automation Sample Program: TiBrowse --------------------------------------

Tibrowse is a sample OLE Automation browser. When started, it asks the user to selected a type library (.tlb) file. The browser then displays the contents of this file.

-------------------------- Shortcomings of this sample -------------------------- 1. This browser is not suitable for use by end users. It is deficient in many ways. A few of them: - Users should never have to know that uuid's exist. Thus,
they should not be displayed in a browser. - Items that are restricted or that start with underscore should not be displayed in a browser. - Duplicates for propery get/set pairs should be removed since users think about properties, not about get/set pairs.

MAKEFILE (TIBROWSE Sample)

```
####
#makefile - makefile for tibrowse.exe
#
#       Copyright (C) 1992, Microsoft Corporation
#
#Purpose:
#  Builds the OLE 2.0 sample IDispatch server, tibrowse.exe.
#
#
#  Usage: NMAKE                  ; build with defaults
#     or: NMAKE option           ; build with the given option(s)
#     or: NMAKE clean            ; erase all compiled files
#
#     option:
#         dev = [win16 | win32] ; dev=win32 is the default
#         DEBUG=[0|1]           ; DEBUG=1 is the default
#
#Notes:
#  This makefile assumes that the PATH, INCLUDE and LIB environment
#  variables are setup properly.
#
###############################################################################
##


##########################################################################
#
# Default Settings
#

!if "$(dev)" == ""
dev = win32
!endif

!if !("$(dev)" == "win16" || "$(dev)" == "win32")
!error Invalid dev option, choose from [win16 | win32]
!endif

!if "$(dev)" == "win16"
TARGET  = WIN16
!endif

!if "$(dev)" == "win32"
TARGET  = WIN32
!endif

!ifdef NODEBUG
DEBUG = 0
!endif

!if "$(DEBUG)" == "0"
NODEBUG = 1
```

```
!endif

!if "$(DEBUG)" == ""
DEBUG = 1
!endif


##########################################################################
#
# WIN16 Settings
#
!if "$(TARGET)" == "WIN16"

CC   = cl
LINK = link

RCFLAGS = -dWIN16
CFLAGS = -c -W3 -AM -GA -GEs -DWIN16
LINKFLAGS = /NOD /NOI /BATCH /ONERROR:NOEXE

LIBS = libw.lib mlibcew.lib

!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS) -Od -Zi -D_DEBUG $(CL)
LINKFLAGS = $(LINKFLAGS) /COD
!else
CFLAGS = $(CFLAGS) -Ox $(CL)
LINKFLAGS = $(LINKFLAGS) /FAR /PACKC
!endif
!endif


##########################################################################
#
# WIN32 Settings
#
!if "$(TARGET)" == "WIN32"

!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

CC = $(cc)
CFLAGS = $(cflags) $(cvarsmt) -DINC_OLE2 -DUNICODE $(cdebug)

!ifndef NODEBUG
CFLAGS = $(CFLAGS) -D_DEBUG
!endif

LINK = $(link)
LINKFLAGS = $(linkdebug) $(guilflags)
RCFLAGS = -DWIN32

!endif


##########################################################################
```

```
#
# Build rules
#

.cpp.obj:
    @echo Compiling $<...
    $(CC) $<

.c.obj:
    @echo Compiling $<...
    $(CC) $<


##########################################################################
#
# Application Settings
#

APPS = tibrowse


!if "$(TARGET)" == "WIN16"
LIBS = ole2.lib compobj.lib ole2disp.lib typelib.lib commdlg.lib $(LIBS)
!endif
!if "$(TARGET)" == "WIN32"
LIBS = $(olelibsmt)
!endif

OBJS = \
         tibrowse.obj


##########################################################################
#
# Default Goal
#

goal : setflags $(APPS).exe

setflags :
        set CL=$(CFLAGS)


##########################################################################
#
# Clean (erase) generated files
#
clean :
    if exist *.obj       del *.obj
    if exist $(APPS).exe del $(APPS).exe
    if exist $(APPS).map del $(APPS).map
    if exist $(APPS).res del $(APPS).res
    if exist *.pdb       del *.pdb
```

```
##########################################################################
#
# Application Build (WIN16 Specific)
#

!if "$(TARGET)" == "WIN16"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
        link $(LINKFLAGS) @<<
$(OBJS),
$@,,
$(LIBS),
$(APPS).def
<<
        rc -k -t $(APPS).res $@
!endif


##########################################################################
#
# Application Build (WIN32 Specific)
#
!if "$(TARGET)" == "WIN32"
$(APPS).exe : $(OBJS) $(APPS).def $(APPS).res $(APPS).ico
      $(LINK) @<<
        $(LINKFLAGS)
        -out:$@
        -map:$*.map
        $(OBJS)
        $(APPS).res
        $(LIBS)
<<
!endif


##########################################################################
#
# Application Build (Common)
#

$(APPS).res : $(APPS).rc
        rc $(RCFLAGS) -r -fo$@ $?


##########################################################################
#
# Dependencies
#


tibrowse.obj : tibrowse.cpp resource.h
    $(CC) tibrowse.cpp
```

RESOURCE.H (TIBROWSE Sample)

```
/***
*resource.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*
*Implementation Notes:
*
*****************************************************************************
*/

#ifdef _MAC
# define IDC_BASE 0

# define kMinSize1000  /* minimum size (in K) */
# define kPrefSize     1000  /* preferred size (in K) */
# define kMinHeap21 * 1024
# define kMinSpace     8 * 1024

# define rMenuBar128   /* menu bar */
# define rAboutAlert   128   /* about alert */
# define rUserAlert    129   /* error alert */
# define rDlg          130

# define mApple        128   /* Apple menu */
# define iAbout        1

# define mFile         129   /* File menu */
# define iOpen         2
# define iClose        4
# define iQuit         12

# define mEdit         130   /* Edit menu */
# define iUndo         1
# define iCut          3
# define iCopy         4
# define iPaste        5
# define iClear        6

#else
# define IDC_BASE 1000
#endif


#define IDC_TYPELIST    (IDC_BASE+1)
#define IDC_MEMBERLIST  (IDC_BASE+2)
#define IDC_PARAMLIST   (IDC_BASE+3)
#define IDC_TYPEKIND    (IDC_BASE+5)
#define IDC_VERSION     (IDC_BASE+7)
#define IDC_GUID        (IDC_BASE+9)
#define IDC_HELPSTRING  (IDC_BASE+11)
```

```
#define IDC_HELPCONTEXT (IDC_BASE+13)

#define IDC_STATIC      -1
```

TIBROWSE.DEF (TIBROWSE Sample)

```
;------------------------------------
; TIBROWSE.DEF module definition file
;------------------------------------

NAME           TIBROWSE

DESCRIPTION    'Type library browser'
EXETYPE        WINDOWS
STUB           'WINSTUB.EXE'
CODE           PRELOAD MOVEABLE DISCARDABLE
DATA           PRELOAD MOVEABLE MULTIPLE
HEAPSIZE       1024
STACKSIZE      10240
```

TIBROWSE.H (TIBROWSE Sample)

```
/***
*tibrowse.h
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*
*Implementation Notes:
*
*****************************************************************************
*/

#if defined(_MAC)

#if defined(_MSC_VER)

# include <values.h>
# include <types.h>
# include <string.h>
# include <quickdra.h>
# include <fonts.h>
# include <events.h>
# include <resource.h>
# include <menus.h>
# include <lists.h>
# include <textedit.h>
# include <dialogs.h>
# include <desk.h>
# include <toolutil.h>
# include <memory.h>
# include <files.h>
# include <osutils.h>
# include <osevents.h>
# include <diskinit.h>
# include <packages.h>
# include <traps.h>
# include <AppleEve.h>
# include <standard.h>

#else //_MSC_VER

# include <values.h>
# include <types.h>
# include <strings.h>
# include <quickdraw.h>
# include <fonts.h>
# include <events.h>
# include <resources.h>
# include <windows.h>
# include <menus.h>
# include <lists.h>
# include <textedit.h>
# include <dialogs.h>
# include <desk.h>
# include <toolutils.h>
# include <memory.h>
# include <files.h>
# include <osutils.h>
# include <osevents.h>
# include <diskinit.h>
# include <packages.h>
# include <traps.h>
# include <AppleEvents.h>
# include <StandardFile.h>

#endif //_MSC_VER

# include <ole2.h>
# include <olenls.h>
# include <dispatch.h>

#elif defined(WIN32)

# include <windows.h>
# include <commdlg.h>
# include <ole2.h>
# include <oleauto.h>

#else /* WIN16 */

# include <windows.h>
# include <commdlg.h>
# include <ole2.h>
# include <olenls.h>
# include <dispatch.h>

#endif

#include "resource.h"

#ifdef WIN32
# define EXPORT
#else
# define EXPORT _export
#endif

#ifdef _MAC
# define UNUSED(X) ((void)(void*)&(X))
#else
# define UNUSED(X) (X)
#endif

#define DIM(X) (sizeof(X) / sizeof(X[0]))

void AssertFail(char FAR*, int);
#define ASSERT(X) if (!(X)) { AssertFail(__FILE__, __LINE__); } else {}
```

```
#define    CHECKRESULT(X) \
  {HRESULT hresult = (X); \
    if(hresult != NOERROR && FAILED(GetScode(hresult)))
MethodError(hresult); }


HRESULT OpenFile(BSTR FAR* pbstrFileName);
void    OpenTypeLib(char FAR*);
void    SetSelectedType(unsigned long);
void    FillMemberList(ITypeInfo FAR *, TYPEATTR FAR *, int cltid);
void    SetSelectedMember(unsigned long);
void    SetSelectedParam(unsigned long dwIndex);
void    UpdateMemberInfo(MEMBERID memid);
void    AssertFail(char FAR*, int);
void    MethodError(HRESULT hresult);
void    Cleanup(void);
void    MemFree(void FAR*);

#ifdef _PPCMAC
#define ROUTINE_DESC(proc)              g_rd##proc
#define DECLARE_ROUTINE_DESC(type,proc) static type ROUTINE_DESC(proc)=0
#define INIT_ROUTINE_DESC(init,proc)    { if (!ROUTINE_DESC(proc))
ROUTINE_DESC(proc)=init(proc); }

#else  // _PPCMAC
#define ROUTINE_DESC(proc)              proc
#define INIT_ROUTINE_DESC(init,proc)
#define DECLARE_ROUTINE_DESC(type,proc)
#endif // _PPCMAC
```

TIBROWSE.RC (TIBROWSE Sample)

```
#include "windows.h"
#include "resource.h"


TIBROWSE                ICON    DISCARDABLE     "TIBROWSE.ICO"

TIBROWSE DIALOG DISCARDABLE  -32768, 0, 272, 192
STYLE WS_MINIMIZEBOX | WS_CAPTION | WS_SYSMENU
CAPTION "Type Library Browser"
CLASS "TiBrowse"
FONT 8, "System"
BEGIN
    LTEXT           "&Type",IDC_STATIC,10,10,62,9
    LISTBOX         IDC_TYPELIST,10,20,80,80,LBS_NOINTEGRALHEIGHT |
                    WS_VSCROLL | WS_TABSTOP
    LTEXT           "&Members",IDC_STATIC,100,10,62,9
    LISTBOX         IDC_MEMBERLIST,100,20,80,80,LBS_NOINTEGRALHEIGHT |
                    WS_VSCROLL | WS_TABSTOP
    LTEXT           "&Parameters",IDC_STATIC,190,10,62,9
    LISTBOX         IDC_PARAMLIST,190,20,75,80,LBS_NOINTEGRALHEIGHT |
                    WS_VSCROLL | WS_TABSTOP
    LTEXT           "",IDC_HELPSTRING,60,170,205,18
    CONTROL         "",IDC_HELPCONTEXT,"Static",SS_LEFTNOWORDWRAP |
WS_GROUP,
                    60,155,140,7
    LTEXT           "Help String:",IDC_STATIC,10,170,45,8
    LTEXT           "Help Context:",IDC_STATIC,10,155,45,8
    LTEXT           "Type Kind:",IDC_STATIC,10,110,45,8
    LTEXT           "",IDC_TYPEKIND,60,110,140,8
    LTEXT           "GUID:",IDC_STATIC,10,140,45,8
    LTEXT           "",IDC_GUID,60,140,160,8
    LTEXT           "Version:",IDC_STATIC,10,125,45,8
    LTEXT           "",IDC_VERSION,60,125,140,8
END
```

TIBROWSE.CPP   (TIBROWSE Sample)

```
/***
*tibrowse.cpp
*
*  Copyright (C) 1992-1994, Microsoft Corporation.  All Rights Reserved.
*  Information Contained Herein Is Proprietary and Confidential.
*
*Purpose:
*  Type Information Browser
*
*
*Implementation Notes:
*
****************************************************************************
*/

#if defined(_MAC)

#if defined(_PPCMAC)
#pragma data_seg("_FAR_DATA")
#pragma data_seg( )
#define MAXLONG 0x7fffffff
#define EventHandlerProcPtr AEEventHandlerUPP
#else //_PPCMAC
#define  GetMenuItemText(mApple,menuItem,daName)
GetItem(mApple,menuItem,daName)
#endif //_PPCMAC

#endif //_MAC

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "tibrowse.h"

#ifdef WIN32
  #define SPRINTFswprintf
#else
  #define SPRINTFsprintf
#endif

#if defined(UNICODE)
  #define TCHAR         WCHAR
  #define TSTR(str)     L##str
#else
  #define TCHAR         char
  #define TSTR(str)     str
  #ifndef LPTSTR
    #define LPTSTR LPSTR
  #endif
#endif
```

```
#ifdef _MAC
DECLARE_ROUTINE_DESC(UserItemUPP,DrawListbox);
#endif

#ifdef _MAC
DialogPtr g_pdlg = NULL;
ListHandle g_rghlist[IDC_PARAMLIST+1] = {NULL};
#else
HWND g_hwnd;
TCHAR g_szAppName[] = TSTR("TiBrowse");
#endif

ITypeLib  FAR *g_ptlib = NULL;
ITypeInfo FAR *g_ptinfoCur = NULL;
TYPEATTR  FAR *g_ptattrCur = NULL;

OLECHAR * g_rgszTKind[] = {
    OLESTR("Enum"),          /* TKIND_ENUM */
    OLESTR("Struct"),        /* TKIND_RECORD */
    OLESTR("Module"),        /* TKIND_MODULE */
    OLESTR("Interface"),     /* TKIND_INTERFACE */
    OLESTR("Dispinterface"), /* TKIND_DISPATCH */
    OLESTR("Coclass"),       /* TKIND_COCLASS */
    OLESTR("Typedef"),       /* TKIND_ALIAS */
    OLESTR("Union"),         /* TKIND_UNION */
};


// Set the text of the control identified by the given control id
void
XSetDlgItemText(int ctlid, OLECHAR FAR* psz)
{
#ifdef _MAC
    Rect rc;
    Handle h;
    short kind;
    char buf[255];

    strcpy(buf, psz);
    GetDItem(g_pdlg, ctlid, &kind, &h, &rc);
    SetIText(h, c2pstr(buf));
#else
#if defined(WIN32) && !defined(UNICODE)
    char bufA[256];
    WideCharToMultiByte(CP_ACP, NULL, psz, -1, bufA, 256, NULL, NULL);
    SendDlgItemMessage(g_hwnd, ctlid, WM_SETTEXT, 0, (LPARAM)&bufA);
#else
    SendDlgItemMessage(g_hwnd, ctlid, WM_SETTEXT, 0, (LPARAM)psz);
#endif
#endif
}

// Clear the listbox identified by the given control id
void
XClrDlgItemList(int ctlid)
```

```
{
#ifdef _MAC
    LDelRow(0, 0, g_rghlist[ctlid]); // delete everything
#else
    SendMessage(GetDlgItem(g_hwnd, ctlid), LB_RESETCONTENT, 0, 0L);
#endif
}

// Add the given string to the listbox identified by the given control id
void
XAddDlgItemList(int ctlid, OLECHAR FAR* psz)
{
#ifdef _MAC
    int row;
    Point pt;
    ListHandle hlist;

    hlist = g_rghlist[ctlid];
    row = LAddRow(1, -1, hlist);
    pt.v = row, pt.h = 0;
    LSetCell(psz, strlen(psz), pt, hlist);
#else
#if defined(WIN32) && !defined(UNICODE)
    char bufA[256];
    WideCharToMultiByte(CP_ACP, NULL, psz, -1, bufA, 256, NULL, NULL);
    SendDlgItemMessage(g_hwnd, ctlid, LB_ADDSTRING, 0, (LPARAM)&bufA);
#else
    SendDlgItemMessage(g_hwnd, ctlid, LB_ADDSTRING, 0, (LPARAM)psz);
#endif
#endif
}


void Cleanup()
{
    if(g_ptinfoCur != NULL){
      if(g_ptattrCur != NULL)
        g_ptinfoCur->ReleaseTypeAttr(g_ptattrCur);
      g_ptinfoCur->Release();
      g_ptinfoCur = NULL;
    }
    if(g_ptlib != NULL){
      g_ptlib->Release();
      g_ptlib = NULL;
    }
}

void Uninit()
{
    OleUninitialize();
#ifdef _MAC
#ifndef _PPCMAC
    UninitOleManager();             // clean up applet
#endif
#endif
}


void
OpenTypeLib(OLECHAR FAR *sztlib)
{
    unsigned int utypeinfoCount, i;
    BSTR bstrName;
    TLIBATTR FAR* ptlibattr;

    /* clear out globals */
    Cleanup();

    /* clear out listboxes */
    XClrDlgItemList(IDC_TYPELIST);
    XClrDlgItemList(IDC_MEMBERLIST);
    XClrDlgItemList(IDC_PARAMLIST);

    /* load the type library */
    CHECKRESULT(LoadTypeLib(sztlib, &g_ptlib));

    /* get library attributes for the fun of it */
    CHECKRESULT(g_ptlib->GetLibAttr(&ptlibattr));

    /* release library attributes */
    g_ptlib->ReleaseTLibAttr(ptlibattr);

    /* Now add each of the names to the type list */
    utypeinfoCount = g_ptlib->GetTypeInfoCount();
    for(i = 0; i < utypeinfoCount; i++){
      CHECKRESULT(g_ptlib->GetDocumentation(i, &bstrName, NULL, NULL,
NULL));
      ASSERT(bstrName);
      XAddDlgItemList(IDC_TYPELIST, bstrName);
      SysFreeString(bstrName);
    }
}

#ifdef WIN32
void
OpenTypeLib(char FAR *sztlib)
{
  OLECHAR buf[MAX_PATH];

  MultiByteToWideChar(CP_ACP, MB_PRECOMPOSED,
                   sztlib, -1, buf, MAX_PATH);
  OpenTypeLib(buf);
}
#endif


/***
*void SetSelectedType
*Purpose:
*  When the user changes the selection of a type, this function updates the
*  dialog by changing the member list and the help for the type. It also
sets
*  g_ptinfoCur to refer to the typeinfo.
*
*Entry:
*  dwIndex =
*
*Exit:
*  return value = None
*
***********************************************************************/
void
SetSelectedType(unsigned long dwIndex)
{
    HRESULT hr;
    BSTR bstrDoc;
    OLECHAR FAR* psz;
    OLECHAR szBuf[40];
    unsigned long dwHelpContext;
    TYPEKIND tkind;

    if(g_ptinfoCur != NULL){
      g_ptinfoCur->ReleaseTypeAttr(g_ptattrCur);
      g_ptinfoCur->Release();
    }

    /* Clear out the member list */
    XClrDlgItemList(IDC_MEMBERLIST);

    if(g_ptlib == NULL)
      return;

    // Note: the index in the list box is conveniently the same as the
    // one to pass to GetTypeInfo/GetTypeInfoType

    // typeinfo type can be obtained without actually loading the typeinfo
    CHECKRESULT(g_ptlib->GetTypeInfoType((unsigned int)dwIndex, &tkind));
    XSetDlgItemText(IDC_TYPEKIND, g_rgszTKind[tkind]);

    CHECKRESULT(g_ptlib->GetTypeInfo((unsigned int)dwIndex, &g_ptinfoCur));
    CHECKRESULT(g_ptinfoCur->GetTypeAttr(&g_ptattrCur));

    // GUID
    hr = StringFromCLSID(g_ptattrCur->guid, &psz);
    ASSERT(hr == NOERROR);
    XSetDlgItemText(IDC_GUID, psz);
    MemFree(psz);

    // Version
    SPRINTF(szBuf, OLESTR("%u.%02u"),
      g_ptattrCur->wMajorVerNum, g_ptattrCur->wMinorVerNum);
    XSetDlgItemText(IDC_VERSION, szBuf);

    CHECKRESULT(
      g_ptlib->GetDocumentation(
        (unsigned int)dwIndex, NULL, &bstrDoc, &dwHelpContext, NULL));

    // Help Context
    SPRINTF(szBuf, OLESTR("%ld"), dwHelpContext);
    XSetDlgItemText(IDC_HELPCONTEXT, szBuf);

    // Documentation string
    psz = (bstrDoc != NULL) ? bstrDoc : OLESTR("<none>");

    XSetDlgItemText(IDC_HELPSTRING, psz);
    SysFreeString(bstrDoc);

    FillMemberList(g_ptinfoCur, g_ptattrCur, IDC_MEMBERLIST);

    XClrDlgItemList(IDC_PARAMLIST);
}

/***
*void FillMemberList
*Purpose:
* Sets the current typeinfo to the typeinfo indexed by dwIndex, and
* then fills in the list box with the members of the type.
*
*Entry:
*  ptinfo =
*  ptypeattr =
*
*Exit:
*  return value = None
*
***********************************************************************/
void
FillMemberList(
    ITypeInfo FAR *ptinfo,
    TYPEATTR FAR *ptypeattr,
    int ctlid)
{
    MEMBERID memid;
    BSTR bstrName;
    unsigned int i;
    FUNCDESC FAR *pfuncdesc;
    VARDESC  FAR *pvardesc;

    /* Now add all of the functions and all of the vars.
     * This is somewhat roundabout.
     * For each one, we need to get the funcdesc, or the vardesc.
     * From that we get the MEMBERID, and finally can get to the name.
     */
    for(i = 0; i < ptypeattr->cFuncs; i++){
      CHECKRESULT(ptinfo->GetFuncDesc(i, &pfuncdesc));
      memid = pfuncdesc->memid;
      CHECKRESULT(ptinfo->GetDocumentation(memid, &bstrName, NULL, NULL,
NULL));
      ptinfo->ReleaseFuncDesc(pfuncdesc);
      pfuncdesc = NULL;
      ASSERT(bstrName);
      XAddDlgItemList(ctlid, bstrName);
      SysFreeString(bstrName);
    }

    for(i = 0; i < ptypeattr->cVars; i++)
    {
      CHECKRESULT(ptinfo->GetVarDesc(i, &pvardesc));
      memid = pvardesc->memid;
      CHECKRESULT(ptinfo->GetDocumentation(memid, &bstrName, NULL, NULL,
NULL));
      ptinfo->ReleaseVarDesc(pvardesc);
      pvardesc = NULL;

      ASSERT(bstrName);
      XAddDlgItemList(ctlid, bstrName);
      SysFreeString(bstrName);
    }
}

/***
*void SetSelectedMember
*Purpose:
*  When a member of a type is selected, update the help to be the help
*  of the member, and if the member is a function update the parameter
*  list to reflect that it is a function.
*
*Entry:
*  dwIndex =
*
*Exit:
*  return value = None
*
***********************************************************************/
void
SetSelectedMember(unsigned long dwIndex)
{
    MEMBERID memid;

    /* In any case, we'll need to clear out the parameter list. */
    XClrDlgItemList(IDC_PARAMLIST);

    if(g_ptattrCur == NULL)
      return;

    /* if this is a function, fill the param list, otherwise just fill
     * in the item info.
     */
    if(dwIndex < g_ptattrCur->cFuncs){
      unsigned short i;
      unsigned int cNames;
      FUNCDESC FAR *pfuncdesc;
      const unsigned int MAX_NAMES = 40;
      BSTR rgNames[MAX_NAMES];
      CHECKRESULT(g_ptinfoCur->GetFuncDesc((unsigned int) dwIndex,
&pfuncdesc));
      memid = pfuncdesc->memid;
      UpdateMemberInfo(memid);

      CHECKRESULT(g_ptinfoCur->GetNames(memid, rgNames, MAX_NAMES,&cNames));
      for(i = 1; i < cNames; i++){
     ASSERT(rgNames[i])
     XAddDlgItemList(IDC_PARAMLIST, rgNames[i]);
     SysFreeString(rgNames[i]);
      }
      ASSERT(rgNames[0]);
      SysFreeString(rgNames[0]);
      g_ptinfoCur->ReleaseFuncDesc(pfuncdesc);
    }
    else
    {
      VARDESC FAR *pvardesc;

      CHECKRESULT(
        g_ptinfoCur->GetVarDesc(
       (unsigned int)(dwIndex - g_ptattrCur->cFuncs), &pvardesc));
      memid = pvardesc->memid;
      UpdateMemberInfo(memid);
      g_ptinfoCur->ReleaseVarDesc(pvardesc);
    }
}

/***
*void UpdateMemberInfo
*Purpose:
*  sets fields on the dialog (such as help string and help context) from
*  the type information.
*
*Entry:
*  memid =
*
*Exit:
*  return value = None
*
***********************************************************************/
void
UpdateMemberInfo(MEMBERID memid)
{
    BSTR bstrDoc;
    OLECHAR buf[40];
    unsigned long dwHelpContext;

    /* get the member information */
    CHECKRESULT(g_ptinfoCur->GetDocumentation(
      memid, NULL, &bstrDoc, &dwHelpContext, NULL));

    /* update the help string displayed in the dialog */
    XSetDlgItemText(IDC_HELPSTRING, (bstrDoc != NULL) ?
                              bstrDoc : OLESTR("<none>"));
    SysFreeString(bstrDoc);

    /* update the help context displayed in the dialog */
    SPRINTF(buf, OLESTR("%ld"), dwHelpContext);
    XSetDlgItemText(IDC_HELPCONTEXT, buf);
}

/***
*void SetSelectedParam
*Purpose:
*  CONSIDER: Enhance to show parameter type information here.
*
*Entry:
*  dwIndex =
*
*Exit:
*  return value = None
*
***********************************************************************/
void
SetSelectedParam(unsigned long dwIndex)
{
}


HRESULT
GetFileName(BSTR FAR* pbstrFileName)
{
    OLECHAR buf[1024];
    BSTR bstrFileName;

#ifdef _MAC

    int i;
    Point pt;
    long dirID;
    char *p, *q;
    SFReply sfr;
    SFTypeList sftl;
    short sRefNumReal;

    SetPt(&pt, 100, 100);
    //sftl[0] = 'OTLB';
    sftl[0] = 0; // currently allow all file types
    SFGetFile(pt, NULL, NULL, -1, sftl, NULL, &sfr);

    if(!sfr.good)
      return ResultFromScode(E_FAIL);

    // translate sfr.vRefNum into a dirID
    {
     WDPBRec wdpbr;
```

```
      wdpbr.ioWDIndex   = 0;
      wdpbr.ioVRefNum   = sfr.vRefNum;
      wdpbr.ioWDVRefNum = 0;
      wdpbr.ioWDIndex   = 0;
      wdpbr.ioWDProcID  = 0;
      wdpbr.ioNamePtr   = (StringPtr)buf;

      if(PBGetWDInfoSync (&wdpbr) != noErr)
        return ResultFromScode(E_FAIL);

      sRefNumReal = wdpbr.ioWDVRefNum;
      dirID = wdpbr.ioWDDirID;
     }

     {
      Str255 stDir;
      CInfoPBRec pb;
      DirInfo *pdirinfo = (DirInfo*)&pb;
      char rgchTemp[256]; // REVIEW: what should the size be?

      pdirinfo->ioNamePtr = stDir;
      pdirinfo->ioVRefNum = sRefNumReal;
      pdirinfo->ioDrParID = dirID;
      pdirinfo->ioFDirIndex = -1;  // get info on a dir

      buf[0] = '\0';

      // loop until we hit the root dir
      do{
        pdirinfo->ioDrDirID = pdirinfo->ioDrParID;
        if(PBGetCatInfoSync (&pb) != noErr)
          return ResultFromScode(E_FAIL);
        strncpy(rgchTemp, (char*)stDir, 64);
        p2cstr((StringPtr)rgchTemp);
        strcat(rgchTemp, ":");
        strcat(rgchTemp, buf);
        strcpy(buf, rgchTemp);
      }while(pdirinfo->ioDrDirID != fsRtDirID);

     }

     q = (char*)sfr.fName;
     i = (int)*q++;
     p = &buf[strlen(buf)];
     while(i-->0)
       *p++ = *q++;
     *p = '\0';

     if((bstrFileName = SysAllocString(buf)) == NULL)
       return ResultFromScode(E_OUTOFMEMORY);

#else

     OPENFILENAME ofn;
```

```
    memset(&ofn, 0, sizeof(OPENFILENAME));
    ofn.lStructSize  = sizeof(OPENFILENAME);
    ofn.hwndOwner    = g_hwnd;
    ofn.lpstrFile    = (LPTSTR) &buf;
    ofn.nMaxFile     = sizeof(buf);
    *buf = OLESTR('\0');
    ofn.lpstrFilter  = TSTR("Type Libraries\0*.tlb\0\0");
    ofn.nFilterIndex = 1;
    ofn.Flags        = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST |
OFN_SHAREAWARE;
    if(GetOpenFileName(&ofn) == 0){
      // REVIEW: we can probably do something smarter with this error
      DWORD dwerr = CommDlgExtendedError();

      return ResultFromScode(E_FAIL);
    }
#if defined(WIN32) && !defined(UNICODE)
    OLECHAR szFileW[_MAX_PATH];
    MultiByteToWideChar(CP_ACP, MB_PRECOMPOSED,
                 ofn.lpstrFile, -1, szFileW, _MAX_PATH);
    if((bstrFileName = SysAllocString(szFileW)) == NULL)
      return ResultFromScode(E_OUTOFMEMORY);
#else
    if((bstrFileName = SysAllocString(ofn.lpstrFile)) == NULL)
      return ResultFromScode(E_OUTOFMEMORY);
#endif

#endif

    *pbstrFileName = bstrFileName;
    return NOERROR;
}

void
AssertFail(char FAR* file, int line)
{
    // Ansi Output for Assertion
    #ifdef UNICODE
      #define OUTPUTDEBUGSTRING    OutputDebugStringA
      #define MESSAGEBOX     MessageBoxA
    #else
      #define OUTPUTDEBUGSTRING    OutputDebugString
      #define MESSAGEBOX     MessageBox
    #endif

    char buf[128];

    sprintf(buf, "Assertion failed: %s(%d)", file, line);

#ifdef _MAC
    DebugStr(c2pstr(buf));
#else
    OUTPUTDEBUGSTRING(buf);
    if(MESSAGEBOX(NULL, buf,
       "TiBrowse Assertion.  OK to continue, CANCEL to quit.",
        MB_OKCANCEL | MB_TASKMODAL) == IDCANCEL)
      exit(1);
    DebugBreak();
#endif
}

#ifdef _MAC /* { */

void Init(void);
void EventLoop(void);

void AdjustMenus(void);
void DoEvent(EventRecord *pevent);
void DoMenuCommand(long menuResult);
void Quit(void);
#ifndef _MSC_VER
#ifndef ConstStr255Param
#define ConstStr255Param StringPtr
#endif
#endif
void Fatal(ConstStr255Param);

Boolean
IsAppWindow(WindowPtr window)
{
    return (window == NULL)
      ? false : ((WindowPeek)window)->windowKind == userKind;
}

Boolean
IsDAWindow(WindowPtr window)
{
    return (window == NULL)
      ? false : (((WindowPeek)window)->windowKind < 0);
}

void
main()
{
    Init();
//    ShowWindow(g_pdlg); // REVIEW: necessarry?
    EventLoop();
}

void
DoDialogEvent(EventRecord *pevent, short sItem)
{
    Point pt;
    GrafPtr pport;
    ListHandle hlist;

    switch(sItem){
    case IDC_TYPELIST:
    case IDC_MEMBERLIST:
    case IDC_PARAMLIST:
      if(pevent->what == mouseDown){
     GetPort(&pport);
     SetPort(g_pdlg);
        GlobalToLocal(&pevent->where);
     hlist = g_rghlist[sItem];
        LClick(pevent->where, pevent->modifiers, hlist);
     SetPt(&pt, 0, 0);
     if(LGetSelect(true, &pt, hlist)){
       switch(sItem){
       case IDC_TYPELIST:
         SetSelectedType(pt.v);
         break;
       case IDC_MEMBERLIST:
         SetSelectedMember(pt.v);
         break;
       case IDC_PARAMLIST:
         SetSelectedParam(pt.v);
         break;
       }
     }
     SetPort(pport);
      }
      break;
    }
}

void
EventLoop()
{
    short sItem;
    DialogPtr pdlg;
    EventRecord  event;
    RgnHandle cursorRgn;

    cursorRgn = NewRgn();
    while(1){
      if(WaitNextEvent(everyEvent, &event, MAXLONG, cursorRgn)){
     if (FrontWindow() != NULL
      && event.what != diskEvt
      && (event.what != keyDown || (event.modifiers & cmdKey) == 0)
      && IsDialogEvent(&event))
     {
       if(DialogSelect(&event, &pdlg, &sItem)){
         ASSERT(pdlg == g_pdlg);
         DoDialogEvent(&event, sItem);
       }
     }else{
       DoEvent(&event);

     }
      }
    }
}

void
```

```
DoEvent(EventRecord *pevent)
{
    char key;
    short part;
    WindowPtr window;

    switch(pevent->what){
    case mouseDown:
      part = FindWindow(pevent->where, &window);
      switch(part){
      case inMenuBar:
     AdjustMenus();
     DoMenuCommand(MenuSelect(pevent->where));
     break;

      case inSysWindow:/* let the system handle the mouseDown */
     SystemClick(pevent, window);
     break;

      case inContent:
     if(window != FrontWindow()){
       SelectWindow(window);
     }
     break;

      case inDrag:
     DragWindow(window, pevent->where, &qd.screenBits.bounds);
     break;
      }
      break;

    case keyDown:
    case autoKey:            /* check for menukey equivalents */
      key = (char)(pevent->message & charCodeMask);
      if(pevent->modifiers & cmdKey){    /* Command key down */
     if(pevent->what == keyDown){
       /* enable/disable/check menu items properly */
       AdjustMenus();
       DoMenuCommand(MenuKey(key));
     }
      }
      break;
    }
}

void
Enable(MenuHandle hmenu, short sItem, Boolean fEnable)
{
    if(fEnable)
      EnableItem(hmenu, sItem);
    else
      DisableItem(hmenu, sItem);
}

void
```

```
AdjustMenus()
{
    Boolean fIsDA;
    MenuHandle hmenu;

    fIsDA = IsDAWindow(FrontWindow());

    /* we can allow desk accessories to be closed from the menu */
    hmenu = GetMHandle(mFile);
    Enable(hmenu, iClose, fIsDA);

    hmenu = GetMHandle(mEdit);
    Enable(hmenu, iUndo,  fIsDA);
    Enable(hmenu, iCut,   fIsDA);
    Enable(hmenu, iCopy,  fIsDA);
    Enable(hmenu, iPaste, fIsDA);
    Enable(hmenu, iClear, fIsDA);
}

void
DoMenuCommand(long menuResult)
{
    short menuID;      /* the resource ID of the selected menu */
    short menuItem;          /* the item number of the selected menu */
    Str255 daName;

    menuID = HiWord(menuResult);
    menuItem = LoWord(menuResult);

    switch(menuID){
    case mApple:
      switch(menuItem){
      case iAbout:           /* bring up alert for About */
     Alert(rAboutAlert, NULL);
     break;
      default:
     GetMenuItemText(GetMHandle(mApple), menuItem, daName);
     OpenDeskAcc(daName);
     break;
      }
      break;

    case mFile:
      switch(menuItem){
      case iOpen:
    {
        BSTR bstrFileName;
        if(GetFileName(&bstrFileName) != NOERROR)
     {
       // REVIEW: shouldnt just bag out here
       Fatal((ConstStr255Param)"\pUnable to open Type Library");
        }
        OpenTypeLib(bstrFileName);
    }
     break;
```

```
      case iQuit:
     Quit();
     break;
      }
      break;

    case mEdit:
      SystemEdit(menuItem-1);
      break;
    }

    HiliteMenu(0);
}

extern "C" {

#if defined(_MSC_VER)
void pascal
#else
pascal void
#endif
DrawListbox(DialogPtr pdlg, short sItem)
{
    Rect rc;
    Handle h;
    short kind;
    GrafPtr port;
    ListHandle hlist;

    ASSERT(sItem == IDC_TYPELIST
     || sItem == IDC_MEMBERLIST
     || sItem == IDC_PARAMLIST);

    GetPort(&port);
    SetPort(pdlg);
    hlist = g_rghlist[sItem];
    LUpdate(pdlg->visRgn, hlist);
    GetDItem(pdlg, sItem, &kind, &h, &rc);
    InsetRect(&rc, -1, -1);
    FrameRect(&rc);
    SetPort(port);
}

}


/***
*ListHandle CreateListBox(DialogPtr, int)
*Purpose:
*  Create a listbox on the given dialog, associated with the given
*  control id.
*
*Entry:
```

```
*  pdlg =
*  ctlid =
*
*Exit:
*  return value = ListHandle
*
***********************************************************************/
ListHandle
CreateListbox(DialogPtr pdlg, int ctlid)
{
    Handle h;
    short kind;
    Point ptCell;
    ListHandle hlist;
    Rect rcView, rcBounds;

    INIT_ROUTINE_DESC(NewUserItemProc,DrawListbox);

    GetDItem(pdlg, ctlid, &kind, &h, &rcView);
    SetDItem(pdlg, ctlid, kind, (Handle) ROUTINE_DESC(DrawListbox),
&rcView);

    // make room for scroll bars
    rcView.right -= 15;

    // list is 1 wide
    SetRect(&rcBounds, 0, 0, 1, 0);

    SetPt(&ptCell, 0, 0);

    hlist = LNew(
      &rcView,
      &rcBounds,
      ptCell,
      0, (WindowPtr)pdlg, true, false, false, true);

    ASSERT(hlist != NULL);

    (*hlist)->selFlags = lUseSense | lNoRect | lNoExtend | lOnlyOne;

    return hlist;
}

/***
*DialogPtr CreateDialog(void)
*Purpose:
*  Create the TiBrowse dialog
*
*Entry:
*  None
*
*Exit:
*  return value = DialogPtr
*
***********************************************************************/
```

```
DialogPtr
CreateDialog()
{
    int i;
    Point pt;
    DialogPtr pdlg;
    ListHandle hlist;
static short rgctl[] = {IDC_TYPELIST, IDC_MEMBERLIST, IDC_PARAMLIST};

    if((pdlg = GetNewDialog(rDlg, NULL, (WindowPtr)-1)) == NULL)
      return NULL;

    for(i = 0; i < DIM(rgctl); ++i){
      g_rghlist[rgctl[i]]  = CreateListbox(pdlg, rgctl[i]);
    }

    SetPt(&pt, 0, 0);
    for(i = 0; i < DIM(rgctl); ++i){
      hlist = g_rghlist[rgctl[i]];
      LSetSelect(true, pt, hlist);
      LDoDraw(true, hlist);
    }

    return pdlg;
}


void
Init()
{
    Handle menuBar;

    MaxApplZone();

    InitGraf((Ptr)&qd.thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(NULL);
    InitCursor();
    FlushEvents(everyEvent, 0);

#ifndef _PPCMAC
    if (InitOleManager(0) != NOERROR)
      Fatal((ConstStr255Param)"\pCould not initialize OLE Applet");
#endif

    if(OleInitialize(NULL) != NOERROR)
      Fatal((ConstStr255Param)"\pUnable to Initialize Ole");

    if((menuBar = GetNewMBar(rMenuBar)) == NULL)
      Fatal((ConstStr255Param)"\pUnable to load menu bar");
    SetMenuBar(menuBar);
    DisposHandle(menuBar);
    AddResMenu(GetMHandle(mApple), 'DRVR');
    DrawMenuBar();

    if((g_pdlg = CreateDialog()) == NULL)
      Fatal((ConstStr255Param)"\pUnable to create dialog");
}

void
Quit()
{
    Cleanup();
    Uninit();
    ExitToShell();
}

/* display fatal error alert, and exit */
void
Fatal(ConstStr255Param msg)
{
    SetCursor(&qd.arrow);
    ParamText(msg, (ConstStr255Param)"\p", (ConstStr255Param)"\p",
(ConstStr255Param)"\p");
    Alert(rUserAlert, nil);
    Quit();
}

#else /* }{ */

long FAR PASCAL EXPORT WndProc (HWND, UINT, WPARAM, LPARAM) ;

int PASCAL
WinMain(
    HANDLE hinst,
    HANDLE hinstPrev,
    LPSTR lpszCmdLine,
    int nCmdShow)
{
    MSG msg;
    WNDCLASS wndclass;

    if(!hinstPrev){
      wndclass.style          = CS_HREDRAW | CS_VREDRAW;
      wndclass.lpfnWndProc    = WndProc ;
      wndclass.cbClsExtra     = 0 ;
      wndclass.cbWndExtra     = DLGWINDOWEXTRA ;
      wndclass.hInstance      = hinst ;
      wndclass.hIcon          = LoadIcon (hinst, g_szAppName) ;
      wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
      wndclass.hbrBackground  = (HBRUSH) (COLOR_WINDOW + 1) ;
      wndclass.lpszMenuName   = NULL ;
      wndclass.lpszClassName  = g_szAppName ;

      RegisterClass (&wndclass) ;
    }
    if(OleInitialize(NULL) != NOERROR){
      MessageBox(NULL, TSTR("unable to initialize Ole"),
              g_szAppName, MB_OK);
      return 0;
    }

    g_hwnd = CreateDialog(hinst, g_szAppName, 0, NULL);

    if(*lpszCmdLine == '\0'){
      BSTR bstrFileName;
      if(GetFileName(&bstrFileName) != NOERROR){
     Cleanup();
     Uninit();
     exit(-1);
      }
      OpenTypeLib(bstrFileName);
      SysFreeString(bstrFileName);
    }else{
      OpenTypeLib(lpszCmdLine);
    }

    ShowWindow(g_hwnd, nCmdShow);

    while(GetMessage (&msg, NULL, 0, 0)){
      TranslateMessage (&msg);
      DispatchMessage (&msg);
    }
    Cleanup();
    Uninit();
    return msg.wParam;
}

long FAR PASCAL EXPORT
WndProc(
    HWND hwnd,
    UINT message,
    WPARAM wParam,
    LPARAM lParam)
{
    unsigned long dwIndex;

    switch(message){
    case WM_COMMAND:
      /* We deal with the following events:
       * The selection changes on the type list and we have
       *  to update the member list & type info.
       * The selection changes on the member list and we have
       *  to update the param list & member info.
       * Selection changes on a parameter and we have to
       *  update the param info.
       */
#ifdef WIN32
# define wParamX LOWORD(wParam)
# define notifyMsg HIWORD(wParam)
#else
# define wParamX wParam
# define notifyMsg HIWORD(lParam)
#endif
      switch(wParamX){
      case IDC_TYPELIST:
        if(notifyMsg == LBN_SELCHANGE){
          dwIndex = SendDlgItemMessage(hwnd, IDC_TYPELIST, LB_GETCURSEL, 0,
0L);
       if(dwIndex != LB_ERR)
         SetSelectedType(dwIndex);
     }
     break;
      case IDC_MEMBERLIST:
        if(notifyMsg == LBN_SELCHANGE){
          dwIndex = SendDlgItemMessage(hwnd, IDC_MEMBERLIST,
LB_GETCURSEL,0,0L);
       if(dwIndex != LB_ERR)
         SetSelectedMember(dwIndex);
     }
        break;
      case IDC_PARAMLIST:
        if(notifyMsg == LBN_SELCHANGE){
          dwIndex = SendDlgItemMessage(hwnd, IDC_PARAMLIST, LB_GETCURSEL,
0,0L);
       if(dwIndex != LB_ERR)
         SetSelectedParam(dwIndex);
     }
     break;
      }
      return 0;

    case WM_DESTROY:
      PostQuitMessage(0);
      return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}

#endif /* } */

void
MethodError(HRESULT hresult)
{
    /* CONSIDER: add code to figure out what specific error this is */
    UNUSED(hresult);

#ifdef _MAC
    Fatal((ConstStr255Param)"\pType Library");
#else
    MessageBox(NULL, TSTR("Error returned from TYPELIB.DLL"),
            g_szAppName, MB_OK);
    exit(1);
#endif
```

```
}

/* free using the task allocator */
void
MemFree(void FAR* pv)
{
    HRESULT hresult;
    IMalloc FAR* pmalloc;

    hresult = CoGetMalloc(MEMCTX_TASK, &pmalloc);

    if(hresult != NOERROR){
#ifdef _MAC
      Fatal((ConstStr255Param)"\pError accessing task allocator");
#else
      MessageBox(NULL, TSTR("Error accessing task allocator"),
              g_szAppName, MB_OK);
#endif
      return;
    }

    pmalloc->Free(pv);
    pmalloc->Release();
}
```

WINHLPRS

These topics contain sample code and header files that create a common dialog, create a window class, and write variables to a stream. For information on compiling and building the samples, see MAKEFILE (WINHLPRS Sample).

MAKEFILE (WINHLPRS Sample)

```
#+--------------------------------------------------------------------------
-
#
#  Microsoft Windows
#  Copyright (C) Microsoft Corporation, 1994.
#
#  File:        makefile
#
#---------------------------------------------------------------------------
-

!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

#
#       builds the WINHLPRS.LIB library
#
#       used by MFract and DFView
#

OLEFLAGS = -I ..\idl

all: winhlprs.lib

clean:
    -del *.lib
    -del *.obj

cwindow.obj: cwindow.cxx cwindow.h
    $(cc) $(cvars) $(cflags) $(UFLAGS) $(cdebug) $(OLEFLAGS) cwindow.cxx

cdialog.obj: cdialog.cxx cdialog.h
    $(cc) $(cvars) $(cflags) $(UFLAGS) $(cdebug) $(OLEFLAGS) cdialog.cxx

strmhelp.obj: strmhelp.cxx strmhelp.h
    $(cc) $(cvars) $(cflags) $(UFLAGS) $(cdebug) $(OLEFLAGS) strmhelp.cxx

winhlprs.lib: cwindow.obj cdialog.obj strmhelp.obj
    lib -out:winhlprs.lib cwindow.obj cdialog.obj strmhelp.obj
```

CDIALOG.H   (WINHLPRS Sample)

```
//
+---------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       cdialog.h
//
//  Contents:   definition for common dialog functionality
//
//  Classes:    CHlprDialog (pure virtual class)
//
//  Functions:  DialogProc
//
//  History:    4-12-94   stevebl   Created
//
//----------------------------------------------------------------------------
--

#ifndef __CDIALOG_H__
#define __CDIALOG_H__

#ifdef __cplusplus
extern "C" {
#endif

BOOL CALLBACK DialogProc(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM
lParam);

#ifdef __cplusplus
}

//
+---------------------------------------------------------------------------
//
//  Class:      CHlprDialog
//
//  Purpose:    virtual base class for wrapping Windows' dialog
functionality
//
//  Interface:  ShowDialog -- analagous to the Windows DialogBox function
//              DialogProc -- pure virtual DialogProc for the dialog box
//              ~CHlprDialog   -- destructor
//
//  History:    4-12-94   stevebl   Created
//
//  Notes:      This class allows a dialog box to be cleanly wrapped in
//              a c++ class.  Specifically, it provides a way for a c++
class
//              to use one of its methods as a DialogProc, giving it a
"this"
//              pointer and allowing it to have direct access to all of its
//              private members.
//
//--------------------------------------------------------------------------
--

class CHlprDialog
{
public:
    virtual int ShowDialog(HINSTANCE hinst, LPCTSTR lpszTemplate, HWND
hwndOwner);
    virtual BOOL DialogProc(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM
lParam) = 0;
    virtual ~CHlprDialog(){};
protected:
    HINSTANCE _hInstance;
};

#endif //__cplusplus

#endif //__CDIALOG_H__
```

CDIALOG.CXX   (WINHLPRS Sample)

```
//
+---------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       dialog.cxx
//
//  Contents:
//
//  Classes:    CHlprDialog
//
//  Functions:  DialogProc
//
//  History:    4-12-94   stevebl   Created
//
//--------------------------------------------------------------------------
--

#include <windows.h>
#include "cdialog.h"

//
+---------------------------------------------------------------------------
//
//  Member:     CHlprDialog::ShowDialog
//
//  Synopsis:   Creates the dialog so that it's DialogProc member function
can
//              be invoked.
//
//  Arguments:  [hinst]        - handle of the application instance
//              [lpszTemplate] - identifies the dialog box template
//              [hwndOwner]    - handle of the owner window
//
//  Returns:    return value from the dialog box
//
//  History:    4-12-94   stevebl   Created
//
//  Notes:      The dialog box object exists until deleted by the caller.
//              It can be shown any number of times.
//
//              This function is analgous to Windows' DialogBox function.
The
//              main difference being that you don't specify a DialogProc;
//              you override the pure virtal function
CHlprDialog::DialogProc.
//
//--------------------------------------------------------------------------
--
```

```
int CHlprDialog::ShowDialog(HINSTANCE hinst, LPCTSTR lpszTemplate, HWND
hwndOwner)
{
    _hInstance = hinst;
    return(DialogBoxParam(hinst, lpszTemplate, hwndOwner,
(DLGPROC)::DialogProc, (long)this));
}

//
+---------------------------------------------------------------------------
//
//  Function:   DialogProc
//
//  Synopsis:   Common DialogProc used by all CHlprDialog class objects.
//
//  Arguments:  [hwndDlg] - handle of dialog box
//              [uMsg]    - message
//              [wParam]  - first message parameter
//              [lParam]  - second message parameter
//
//  Returns:    response from the CHlprDialog::DialogProc method
//
//  History:    4-12-94   stevebl   Created
//
//  Notes:      This procedure is the DialogProc registered for all dialogs
//              created with the CHlprDialog class.  It uses the parameter
//              passed with the WM_INITDIALOG message to identify the dialog
//              classes' "this" pointer which it then stores in the window
//              structure's GWL_USERDATA field.  All subsequent messages
//              can then be forwarded on to the correct dialog class's
//              DialogProc method by using the pointer stored in the
//              GWL_USERDATA field.
//
//---------------------------------------------------------------------------
--

BOOL CALLBACK DialogProc(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{
    CHlprDialog * pdlg;
    switch (uMsg)
    {
    case WM_INITDIALOG:
        // This message is how we identify the dialog object.

        // get a pointer to the window class object_
        pdlg = (CHlprDialog *) lParam;
        // set its USERDATA word to point to the class object
        SetWindowLong(hwndDlg, GWL_USERDATA, (long) pdlg);
        break;
    default:
        // get a pointer to the window class object
        pdlg = (CHlprDialog *) GetWindowLong(hwndDlg, GWL_USERDATA);
        break;
    }
    // and call its message proc method
    if (pdlg != (CHlprDialog *) 0)
    {
        return(pdlg->DialogProc(hwndDlg, uMsg, wParam, lParam));
    }
    else
    {
        return(FALSE);
    }
}
```

CWINDOW.H (WINHLPRS Sample)

```
//
+--------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       cwindow.h
//
//  Contents:   definition of a virtual window class
//
//  Classes:    CHlprWindow
//
//  Functions:  WindowProc
//
//  History:    4-12-94   stevebl   Created
//
//--------------------------------------------------------------------------
--


#ifndef __CWINDOW_H__
#define __CWINDOW_H__

#include <windows.h>

#ifdef __cplusplus
extern "C" {
#endif

LRESULT CALLBACK WindowProc(
    HWND hwnd,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam);

#ifdef __cplusplus
}

//
+--------------------------------------------------------------------------
//
//  Class:      CHlprWindow
//
//  Purpose:    virtual base class for wrapping a window
//
//  Interface:  Create     -- analagous to Windows' CreateWindow function
//              WindowProc -- pure virtual WindowProc for the window
//              ~CHlprWindow   -- destructor
//              CHlprWindow    -- constructor
//
//  History:    4-12-94   stevebl   Created
//
```

```
//  Notes:      This class allows a window to be cleanly wrapped in a
//              c++ class.  Specifically, it provides a way for a c++ class
//              to use one of its methods as a WindowProc, giving it a
"this"
//              pointer and allowing it to have direct access to all of its
//              private members.
//
//-----------------------------------------------------------------------------
--

class CHlprWindow
{
public:
    HWND Create(
        LPCTSTR lpszClassName,
        LPCTSTR lpszWindowName,
        DWORD dwStyle,
        int x,
        int y,
        int nWidth,
        int nHeight,
        HWND hwndParent,
        HMENU hmenu,
        HINSTANCE hinst);
    virtual LRESULT WindowProc(UINT uMsg, WPARAM wParam, LPARAM lParam) = 0;
    virtual ~CHlprWindow(){};
    HWND GetHwnd(void)
    {
        return(_hwnd);
    }
    CHlprWindow()
    {
        _hwnd = NULL;
        _hInstance = NULL;
    };
protected:
friend LRESULT CALLBACK ::WindowProc(
    HWND hwnd,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam);
    HWND _hwnd;
    HINSTANCE _hInstance;
};

#endif

#endif //__CWINDOW_H__
```

CWINDOW.CXX   (WINHLPRS Sample)

```
//
+---------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       cwindow.cxx
//
//  Contents:   implementation for a window class
//
//  Classes:    CHlprWindow
//
//  Functions:  WindowProc
//
//  History:    4-12-94   stevebl   Created
//
//----------------------------------------------------------------------------
--

#include "cwindow.h"

//
+---------------------------------------------------------------------------
//
//  Member:     CHlprWindow::Create
//
//  Synopsis:   Special version of CreateWindow.
//
//  Arguments:  [lpszClassName]  - address of registered class name
//              [lpszWindowName] - address of window name
//              [dwStyle]        - window style
//              [x]              - horizontal position of window
//              [y]              - vertical position of window
//              [nWidth]         - window width
//              [nHeight]        - window height
//              [hwndParent]     - handle of parent or owner window
//              [hmenu]          - handle of menu, or child window
identifier
//              [hinst]          - handle of application instance
//
//  Returns:    HWND of the created window
//
//  Modifies:   _hwnd, _hInstance
//
//  History:    4-12-94   stevebl   Created
//
//  Notes:      The window class must have been previously registered (as
//              is normal Windows procedure) and the callback function
//              must have been registered as ::WindowProc.  ::WindowProc
will
//              then forward all messages on to the CHlprWindow::WindowProc
//              method, allowing the window to directly access class members
//              (i.e. giving the WindowProc access to the "this" pointer).
//
//----------------------------------------------------------------------------
--

HWND CHlprWindow::Create(
    LPCTSTR lpszClassName,
    LPCTSTR lpszWindowName,
    DWORD dwStyle,
    int x,
    int y,
    int nWidth,
    int nHeight,
    HWND hwndParent,
    HMENU hmenu,
    HINSTANCE hinst)
{
    _hInstance = hinst;
    return(_hwnd =
        CreateWindow(
            lpszClassName,
            lpszWindowName,
            dwStyle,
            x,
            y,
            nWidth,
            nHeight,
            hwndParent,
            hmenu,
            hinst,
            this));
}

//
+----------------------------------------------------------------------------
//
//  Function:   WindowProc
//
//  Synopsis:   Standard WindowProc that forwards Windows messages on to the
//              CHlprWindow::WindowProc method.
//
//  Arguments:  [hwnd]   - window handle
//              [uMsg]   - message
//              [wParam] - first message parameter
//              [lParam] - second message parameter
//
//  History:    4-12-94   stevebl   Created
//
//  Notes:      This Window procedure expects that it will receive a "this"
//              pointer as the lpCreateParams member passed as part of the
//              WM_CREATE message.  It saves the "this" pointer in the
//              GWL_USERDATA field of the window structure.
//
//----------------------------------------------------------------------------
--
```

```
LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{
    CHlprWindow * pw;
    switch (uMsg)
    {
    case WM_CREATE:
        // Since this is the first time that we can get ahold of
        // a pointer to the window class object, all messages that might
        // have been sent before this are never seen by the Windows object
        // and only get passed on to te DefWindowProc

        // get a pointer to the window class object
        pw = (CHlprWindow *) ((CREATESTRUCT *)lParam)->lpCreateParams;
        // set its USERDATA DWORD to point to the class object
        SetWindowLong(hwnd, GWL_USERDATA, (long) pw);
        // Set it's protected _hwnd member variable to ensure that
        // member functions have access to the correct window handle.
        pw->_hwnd = hwnd;
        break;
    case WM_DESTROY:
        // This is our signal to destroy the window class object.

        pw = (CHlprWindow *) GetWindowLong(hwnd, GWL_USERDATA);
        SetWindowLong(hwnd, GWL_USERDATA, 0);
        delete pw;
        pw = (CHlprWindow *) 0;
        break;
    default:
        // get a pointer to the window class object
        pw = (CHlprWindow *) GetWindowLong(hwnd, GWL_USERDATA);
        break;
    }
    // and call its message proc method
    if (pw != (CHlprWindow *) 0)
    {
        return(pw->WindowProc(uMsg, wParam, lParam));
    }
    else
    {
        return(DefWindowProc(hwnd, uMsg, wParam, lParam));
    }
}
```

STRMHELP.H (WINHLPRS Sample)

```
//
+-------------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       strmhelp.h
//
//  Contents:   Routines for writting variables to a stream in an
//              architecture independent manner.
//
//  Classes:
//
//  Functions:  WriteDouble
//              ReadDouble
//              SizeDouble
//
//  History:    4-26-94   stevebl   Created
//
//------------------------------------------------------------------------------
--

#ifndef __STRMHELP_H__
#define __STRMHELP_H__

#ifdef __cplusplus
extern "C" {
#endif

DWORD SizeDouble(void);

HRESULT WriteDouble(IStream * pstm, double d, ULONG FAR * pcbWritten);

HRESULT ReadDouble(IStream * pstm, double * pd, ULONG FAR * pcbRead);

#ifdef __cplusplus
}
#endif

#endif //__STRMHELP_H__
```

STRMHELP.CXX   (WINHLPRS Sample)

```
//
+---------------------------------------------------------------------------
//
//  Microsoft Windows
//  Copyright (C) Microsoft Corporation, 1992 - 1994.
//
//  File:       strmhelp.cxx
//
//  Contents:   Routines for writting variables to a stream in an
//              architecture independent manner.
//
//  Classes:
//
//  Functions:  WriteDouble
//              ReadDouble
//              SizeDouble
//
//  History:    4-26-94   stevebl   Created
//
//----------------------------------------------------------------------------
--

#include <windows.h>
#include <ole2.h>
#include "strmhelp.h"
#include <stdio.h>

#define MAX_SIZE 128

//
+---------------------------------------------------------------------------
//
//  Function:   SizeDouble
//
//  Synopsis:   returns the maximum number of bytes that WriteDouble will
//              take to write a double to an IStream
//
//  Arguments:  none
//
//  Returns:    maximum number of bytes needed to write a double to an
IStream
//
//  History:    4-26-94   stevebl   Created
//
//----------------------------------------------------------------------------
--

DWORD SizeDouble(void)
{
    return(MAX_SIZE);
}
```

```
//
+---------------------------------------------------------------------------
//
//  Function:   WriteDouble
//
//  Synopsis:   rrites a double to an OLE stream in a machine independent
manner
//
//  Arguments:  [pstm]       - pointer to an IStream
//              [d]          - the double to be written
//              [pcbWritten] - pointer to the number of bytes actually
written
//                             (may be NULL)
//
//  Returns:    same return values as IStream::Write
//
//  History:    4-26-94   stevebl   Created
//
//--------------------------------------------------------------------------
--

HRESULT WriteDouble(IStream * pstm, double d, ULONG FAR * pcbWritten)
{
    char szTemp[MAX_SIZE];
    sprintf(szTemp,"%.100lg", d);
    WORD cch = strlen(szTemp);
    return(pstm->Write(szTemp, cch + 1, pcbWritten));
}

//
+---------------------------------------------------------------------------
//
//  Function:   ReadDouble
//
//  Synopsis:   reads a double from an OLE stream in a machine independent
//              manner
//
//  Arguments:  [pstm]    - pointer to an IStream
//              [pd]      - pointer to the double to be read
//              [pcbRead] - pointer to the number of bytes actually read
//                          (may be NULL)
//
//  Returns:    same return values as IStream::Read
//
//  History:    4-26-94   stevebl   Created
//
//--------------------------------------------------------------------------
--

HRESULT ReadDouble(IStream * pstm, double * pd, ULONG FAR * pcbRead)
{
    HRESULT hr;
    char szTemp[MAX_SIZE];
    ULONG cbRead;
    DWORD cch = 0;
    do
    {
        hr = pstm->Read(&szTemp[cch], sizeof(char), &cbRead);
        if (pcbRead && SUCCEEDED(hr))
        {
            pcbRead += cbRead;
        }

    } while (SUCCEEDED(hr) && szTemp[cch++]);
    if (SUCCEEDED(hr))
    {
        sscanf(szTemp, "%lf",pd);
    }
    return(hr);
}
```

WRAPUI

This sample demonstrates an implementation of WRAPUI. For information on compiling and building the sample, see MAKEFILE (WRAPUI Sample).

MAKEFILE (WRAPUI Sample)

```
###########################################################################
#
# Makefile for OUTLUI.DLL
#
# Usage:    NMAKE           (builds DEBUG library)
#    or:    NMAKE RELEASE=1 (builds RELEASE library -- no debug symbols)
#
# Environment variables:
#       DEVROOT_DIR=<path>  (root dir for sample code development)
#
###########################################################################

!include <$(MSTOOLS)\samples\ole\include\olesampl.mak>

!ifndef LANG
LANG=USA
!endif

!ifndef LIBRARY
LIBRARY=wrapui
!endif

RESOURCE=res

all: $(LIBRARY).dll

#
-----------------------------------------------------------------------------
#                      O B J E C T   F I L E   L I S T
#
-----------------------------------------------------------------------------

UI_COBJS = busy.obj     \
           common.obj   \
           convert.obj  \
           dballoc.obj  \
           dbgutil.obj  \
           dllfuncs.obj \
           drawicon.obj \
           geticon.obj  \
           hatch.obj    \
           icon.obj     \
           iconbox.obj  \
           insobj.obj   \
           links.obj    \
           msgfiltr.obj \
           enumfetc.obj \
           enumstat.obj \
           objfdbk.obj  \
           ole2ui.obj   \
           olestd.obj   \
           targtdev.obj \
           oleutl.obj   \
           pastespl.obj \
           regdb.obj    \
           resimage.obj \
           stdpal.obj   \
           suminfo.obj  \
           utility.obj

PRECOMPOBJ=precomp.obj

PRECOMP=precomp.pch

#
----------------------------------------------------------------------------
#                       R E S O U R C E   L I S T
#
----------------------------------------------------------------------------
RES =      \
           busy.h                            \
           common.h                          \
           convert.h                         \
           edlinks.h                         \
           geticon.h                         \
           icon.h                            \
           iconbox.h                         \
           insobj.h                          \
           msgfiltr.h                        \
           enumfetc.h                        \
           ole2ui.h                          \
           pastespl.h                        \
           resimage.h                        \
           $(RESOURCE)\STATIC\default.ico    \
           $(RESOURCE)\STATIC\bang.ico       \
           $(RESOURCE)\STATIC\egares.bmp     \
           $(RESOURCE)\STATIC\hivgares.bmp   \
           $(RESOURCE)\STATIC\vgares.bmp     \
           $(RESOURCE)\$(LANG)\strings.rc    \
           $(RESOURCE)\$(LANG)\busy.dlg      \
           $(RESOURCE)\$(LANG)\convert.dlg   \
           $(RESOURCE)\$(LANG)\fileopen.dlg  \
           $(RESOURCE)\$(LANG)\icon.dlg      \
           $(RESOURCE)\$(LANG)\insobj.dlg    \
           $(RESOURCE)\$(LANG)\links.dlg     \
           $(RESOURCE)\$(LANG)\pastespl.dlg  \
           $(RESOURCE)\$(LANG)\prompt.dlg    \
           $(RESOURCE)\ole2ui.rcv   \
           $(RESOURCE)\$(LANG)\verlocal.h

#
----------------------------------------------------------------------------
#                 D E B U G   M A K E   P A R A M E T E R S
#
----------------------------------------------------------------------------

OLE_FLAGS = /DDLL_VER /D_DLL /DOLE2ANSI /DFLAT
```

```
!ifndef NO_DEBUG
OLE_FLAGS = $(OLE_FLAGS) /DDBG /D_DEBUG /D_DEBUGTRACE=0 /DOLE2SHIP
!endif



LANG      = $(LANG)

#select language for version resource if localized version
!if ("$(LANG)"!="USA") && ("$(LANG)"!="usa")
RFLAGS=$(RFLAGS) -D VER_LOC
!endif


.SUFFIXES: .c .asm .obj .res .rc .def .bmp .ico .exe .dll .cod .str

!include "depend"

#
----------------------------------------------------------------------------
#                      I N F E R E N C E   R U L E S
#
----------------------------------------------------------------------------
# compile C file without precompiled headers into object directory\NOPC
# dont compile c files etc for lcoalized builds.

.c.obj:
    @echo Compiling $(@B).c
    $(cc) $(cflags) $(cvars) $(cdebug) $(OLE_FLAGS) -Yuole2ui.h
-Fpprecomp.pch $(@B).c

.rc.res:
    @echo Resource Compiling $(@B).res
    $(rc) -I $(RESOURCE)\$(LANG);$(RESOURCE)\static;$(RESOURCE) -FO $
(@B).res -R $(RFLAGS) $(@B).rc

.c.cod:
    @echo Making $(@B).cod
    $(cc) $(cflags) $(cvars) $(cdebug) $(OLE_FLAGS) -f- -Fc $(@B).c

#
----------------------------------------------------------------------------
#                         G O A L   T A R G E T S
#
----------------------------------------------------------------------------
precomp.pch: precomp.c ole2ui.h
    $(cc) $(cflags) $(cvars) $(cdebug) $(OLE_FLAGS) -Fpprecomp.pch
-Ycole2ui.h precomp.c

suminfo.obj: suminfo.cpp suminfo.h wn_dos.h
    $(cc) $(cflags) $(cvars) $(cdebug) $(OLE_FLAGS) $(@B).cpp

dballoc.obj: dballoc.cpp dballoc.h
    $(cc) $(cflags) $(cvars) $(cdebug) $(OLE_FLAGS) $(@B).cpp
```

```
ole2ui.res: ole2ui.rc $(RES)

ole2ui.rc : $(RESOURCE)\$(LANG)\strings.rc


#
# Build .LIB static library
#

$(LIBRARY).dll: ole2ui.def $(PRECOMPOBJS) $(UI_COBJS) ole2ui.res
    $(implib)  -out:$*.lib -def:ole2ui.def -machine:$(CPU) $(UI_COBJS) $
(PRECOMPOBJ)
    $(link) $(dlllflags) $(linkdebug) $*.exp $(UI_COBJS) $(PRECOMPOBJ)
ole2ui.res -out:$*.dll -map:$*.map $(olelibsdll) advapi32.lib shell32.lib
ole2ansi.lib

clean:
    -del *.obj
    -del *.dll
    -del *.res
    -del *.lib
    -del *.exp
    -del *.pch
    -del *.map
```

DEPEND (WRAPUI Sample)

```
busy.obj : $(PRECOMP) \
        ole2ui.h olestd.h common.h utility.h busy.h

common.obj : $(PRECOMP) \
        ole2ui.h olestd.h common.h utility.h

convert.obj : $(PRECOMP) \
        ole2ui.h olestd.h common.h utility.h geticon.h regdb.h convert.h

dbgutil.obj : $(PRECOMP) \
        ole2ui.h olestd.h

dllfuncs.obj : $(PRECOMP) \
        ole2ui.h olestd.h common.h

drawicon.obj : $(PRECOMP) \
        ole2ui.h olestd.h common.h utility.h geticon.h

enumfetc.obj : $(PRECOMP) \
        ole2ui.h olestd.h enumfetc.h

enumstat.obj : $(PRECOMP) \
        ole2ui.h olestd.h

geticon.obj : $(PRECOMP) \
        ole2ui.h olestd.h common.h utility.h

hatch.obj : $(PRECOMP) \
        ole2ui.h olestd.h

icon.obj : $(PRECOMP) \
        ole2ui.h olestd.h common.h utility.h icon.h geticon.h

iconbox.obj : $(PRECOMP) \
        ole2ui.h olestd.h iconbox.h

insobj.obj : $(PRECOMP) \
        ole2ui.h olestd.h common.h utility.h icon.h insobj.h resimage.h  \
        iconbox.h geticon.h

links.obj : $(PRECOMP) \
        ole2ui.h olestd.h common.h edlinks.h utility.h

msgfiltr.obj : $(PRECOMP) \
        ole2ui.h olestd.h msgfiltr.h

objfdbk.obj : $(PRECOMP) \
        ole2ui.h olestd.h

ole2ui.obj : $(PRECOMP) \
        ole2ui.h olestd.h common.h utility.h resimage.h iconbox.h
```

```
olestd.obj : $(PRECOMP) \
        ole2ui.h olestd.h regdb.h geticon.h common.h

oleutl.obj : $(PRECOMP) \
        ole2ui.h olestd.h

pastespl.obj : $(PRECOMP) \
        ole2ui.h olestd.h pastespl.h common.h utility.h resimage.h iconbox.h
\
        geticon.h icon.h regdb.h

precomp.obj : $(PRECOMP) \
        ole2ui.h olestd.h

regdb.obj : $(PRECOMP) \
        ole2ui.h olestd.h common.h

resimage.obj : $(PRECOMP) \
        ole2ui.h olestd.h resimage.h

stdpal.obj : $(PRECOMP) \
        stdpal.h

targtdev.obj : $(PRECOMP) \
        ole2ui.h olestd.h

template.obj : $(PRECOMP) \
        ole2ui.h olestd.h common.h <FILE>.h

utility.obj : $(PRECOMP) \
        ole2ui.h olestd.h common.h utility.h geticon.h
```

BUSY.H (WRAPUI Sample)

```
/*
 * BUSY.H
 *
 * Internal definitions, structures, and function prototypes for the
 * OLE 2.0 UI Busy dialog.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */

#ifndef _BUSY_H_
#define _BUSY_H_

//Internally used structure
typedef struct tagBUSY
     {
     //Keep this item first as the Standard* functions depend on it here.
     LPOLEUIBUSY     lpOBZ;       //Original structure passed.

     /*
      * What we store extra in this structure besides the original caller's
      * pointer are those fields that we need to modify during the life of
      * the dialog or that we don't want to change in the original structure
      * until the user presses OK.
      */

     DWORD              dwFlags;                // Flags passed in
     HWND               hWndBlocked;            // HWND of app which is
blocking
     } BUSY, *PBUSY, FAR *LPBUSY;

// Internal function prototypes
BOOL    GetTaskInfo(HWND hWnd, HTASK htask, LPSTR FAR* lplpszTaskName, LPSTR
FAR*lplpszWindowName, HWND FAR*lphWnd);
void    BuildBusyDialogString(HWND, DWORD, int, LPSTR, LPSTR);
BOOL CALLBACK EXPORT BusyDialogProc(HWND hDlg, UINT iMsg, WPARAM wParam,
LPARAM lParam);
void    BusyCleanup(HWND hDlg);
BOOL    FBusyInit(HWND hDlg, WPARAM wParam, LPARAM lParam);
BOOL    InitEnumeration(void);
void    UnInitEnumeration(void);
           StartTaskManager(void);
void    MakeWindowActive(HWND hWndSwitchTo);

#endif //_BUSY_H_
```

BUSY.C   (WRAPUI Sample)

```
/*
 * BUSY.C
 *
 * Implements the OleUIBusy function which invokes the "Server Busy"
 * dialog.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */

#define STRICT  1
#include "ole2ui.h"
#include "common.h"
#include "utility.h"
#include "busy.h"
#include <ctype.h> // for tolower() and toupper()

#if !defined( WIN32 )
#include <toolhelp.h>
#endif


/*
 * OleUIBusy
 *
 * Purpose:
 *  Invokes the standard OLE "Server Busy" dialog box which
 *  notifies the user that the server application is not receiving
 *  messages.  The dialog then asks the user to either cancel
 *  the operation, switch to the task which is blocked, or continue
 *  waiting.
 *
 * Parameters:
 *  lpBZ            LPOLEUIBUSY pointing to the in-out structure
 *                  for this dialog.
 *
 * Return Value:
 *              OLEUI_BZERR_HTASKINVALID  : Error
 *              OLEUI_BZ_SWITCHTOSELECTED : Success, user selected "switch
to"
 *              OLEUI_BZ_RETRYSELECTED    : Success, user selected "retry"
 *              OLEUI_CANCEL              : Success, user selected "cancel"
 */

STDAPI_(UINT) OleUIBusy(LPOLEUIBUSY lpOBZ)
     {
     UINT        uRet = 0;
     HGLOBAL     hMemDlg=NULL;

#if !defined( WIN32 )
// BUGBUG32:    this is not yet ported to NT

     uRet=UStandardValidation((LPOLEUISTANDARD)lpOBZ, sizeof(OLEUIBUSY)
                                        , &hMemDlg);

    // Error out if the standard validation failed
    if (OLEUI_SUCCESS!=uRet)
        return uRet;

    // Validate HTASK
    if (!IsTask(lpOBZ->hTask))
        uRet = OLEUI_BZERR_HTASKINVALID;

    // Error out if our secondary validation failed
    if (OLEUI_ERR_STANDARDMIN <= uRet)
        {
        if (NULL!=hMemDlg)
            FreeResource(hMemDlg);

        return uRet;
        }

    // Invoke the dialog.
    uRet=UStandardInvocation(BusyDialogProc, (LPOLEUISTANDARD)lpOBZ,
                                        hMemDlg,
MAKEINTRESOURCE(IDD_BUSY));
#endif

    return uRet;
}


/*
 * BusyDialogProc
 *
 * Purpose:
 *  Implements the OLE Busy dialog as invoked through the OleUIBusy
function.
 *
 * Parameters:
 *  Standard
 *
 * Return Value:
 *  Standard
 *
 */

BOOL CALLBACK EXPORT BusyDialogProc(HWND hDlg, UINT iMsg, WPARAM wParam,
LPARAM lParam)
    {
    LPBUSY          lpBZ;
    UINT            uRet = 0;

    //Declare Win16/Win32 compatible WM_COMMAND parameters.
    COMMANDPARAMS(wID, wCode, hWndMsg);

    //This will fail under WM_INITDIALOG, where we allocate it.
    lpBZ=(LPBUSY)LpvStandardEntry(hDlg, iMsg, wParam, lParam, &uRet);
```

```
      //If the hook processed the message, we're done.
      if (0!=uRet)
            return (BOOL)uRet;

      //Process the temination message
      if (iMsg==uMsgEndDialog)
      {
            BusyCleanup(hDlg);
            StandardCleanup(lpBZ, hDlg);
            EndDialog(hDlg, wParam);
            return TRUE;
      }

      // Process our special "close" message.  If we get this message,
      // this means that the call got unblocked, so we need to
      // return OLEUI_BZ_CALLUNBLOCKED to our calling app.
      if (iMsg == uMsgCloseBusyDlg)
      {
            SendMessage(hDlg, uMsgEndDialog, OLEUI_BZ_CALLUNBLOCKED, 0L);
            return TRUE;
      }

      switch (iMsg)
            {
            case WM_INITDIALOG:
                  FBusyInit(hDlg, wParam, lParam);
                  return TRUE;

            case WM_ACTIVATEAPP:
            {
                  /* try to bring down our Busy/NotResponding dialog as if
                  **    the user entered RETRY.
                  */
                  BOOL fActive = (BOOL)wParam;
                  if (fActive) {
                        // If this is the app BUSY case, then bring down our
                        // dialog when switching BACK to our app
                        if (lpBZ && !(lpBZ->dwFlags &
BZ_NOTRESPONDINGDIALOG))

      SendMessage(hDlg,uMsgEndDialog,OLEUI_BZ_RETRYSELECTED,0L);
                  } else {
                        // If this is the app NOT RESPONDING case, then bring
down
                        // our dialog when switching AWAY to another app
                        if (lpBZ && (lpBZ->dwFlags & BZ_NOTRESPONDINGDIALOG))

      SendMessage(hDlg,uMsgEndDialog,OLEUI_BZ_RETRYSELECTED,0L);
                  }
                  return TRUE;
            }

            case WM_COMMAND:
                  switch (wID)
```

```
                        {
                        case IDBZ_SWITCHTO:
                        {
                              BOOL fNotRespondingDlg =
                                          (BOOL)(lpBZ->dwFlags &
BZ_NOTRESPONDINGDIALOG);

                              // If user selects "Switch To...", switch
activation
                              // directly to the window which is causing the
problem.
                              if (IsWindow(lpBZ->hWndBlocked))
                                    MakeWindowActive(lpBZ->hWndBlocked);
                              else
                                    StartTaskManager(); // Fail safe: Start
Task Manager

                              // If this is the app not responding case, then
we want
                              // to bring down the dialog when "SwitchTo" is
selected.
                              // If the app is busy (RetryRejectedCall
situation) then
                              // we do NOT want to bring down the dialog.
this is
                              // the OLE2.0 user model design.
                              if (fNotRespondingDlg)
                                    SendMessage(hDlg, uMsgEndDialog,
OLEUI_BZ_SWITCHTOSELECTED, 0L);
                              break;
                        }
                        case IDBZ_RETRY:
                              SendMessage(hDlg, uMsgEndDialog,
OLEUI_BZ_RETRYSELECTED, 0L);
                              break;

                        case IDCANCEL:
                              SendMessage(hDlg, uMsgEndDialog, OLEUI_CANCEL,
0L);
                              break;
                        }
                  break;
            }
     return FALSE;
     }


/*
 * FBusyInit
 *
 * Purpose:
 *  WM_INITIDIALOG handler for the Busy dialog box.
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  wParam          WPARAM of the message
 *  lParam          LPARAM of the message
 *
 * Return Value:
 *  BOOL            Value to return for WM_INITDIALOG.
 */

BOOL FBusyInit(HWND hDlg, WPARAM wParam, LPARAM lParam)
     {
     LPBUSY           lpBZ;
     LPOLEUIBUSY      lpOBZ;
     HFONT            hFont;
     LPSTR            lpTaskName;
     LPSTR            lpWindowName;
     HICON            hIcon;

     lpBZ=(LPBUSY)LpvStandardInit(hDlg, sizeof(OLEUIBUSY), TRUE, &hFont);

     // PvStandardInit sent a termination to us already.
     if (NULL==lpBZ)
           return FALSE;

     // Our original structure is in lParam
     lpOBZ = (LPOLEUIBUSY)lParam;

     // Copy it to our instance of the structure (in lpBZ)
     lpBZ->lpOBZ=lpOBZ;

     //Copy other information from lpOBZ that we might modify.
     lpBZ->dwFlags = lpOBZ->dwFlags;

     // Set default information
     lpBZ->hWndBlocked = NULL;

     // Insert HWND of our dialog into the address pointed to by
     // lphWndDialog.  This can be used by the app who called
     // OleUIBusy to bring down the dialog with uMsgCloseBusyDialog
     if (lpOBZ->lphWndDialog &&
           !IsBadWritePtr((VOID FAR *)lpOBZ->lphWndDialog, sizeof(HWND)))
           {
           *lpOBZ->lphWndDialog = hDlg;
           }

     // Update text in text box --
     // GetTaskInfo will return two pointers, one to the task name
     // (file name) and one to the window name.  We need to call
     // OleStdFree on these when we're done with them.  We also
     // get the HWND which is blocked in this call
     //
     // In the case where this call fails, a default message should already
     // be present in the dialog template, so no action is needed

     if (GetTaskInfo(hDlg, lpOBZ->hTask, &lpTaskName, &lpWindowName, &lpBZ-
>hWndBlocked))
           {
           // Build string to present to user, place in IDBZ_MESSAGE1 
control
           BuildBusyDialogString(hDlg, lpBZ->dwFlags, IDBZ_MESSAGE1, 
lpTaskName, lpWindowName);
           OleStdFree(lpTaskName);
           OleStdFree(lpWindowName);
           }

     // Update icon with the system "exclamation" icon
     hIcon = LoadIcon(NULL, IDI_EXCLAMATION);
     SendDlgItemMessage(hDlg, IDBZ_ICON, STM_SETICON, (WPARAM)hIcon, 0L);

     // Disable/Enable controls
     if ((lpBZ->dwFlags & BZ_DISABLECANCELBUTTON) ||
           (lpBZ->dwFlags & BZ_NOTRESPONDINGDIALOG))              // Disable 
cancel for "not responding" dialog
           EnableWindow(GetDlgItem(hDlg, IDCANCEL), FALSE);

     if (lpBZ->dwFlags & BZ_DISABLESWITCHTOBUTTON)
           EnableWindow(GetDlgItem(hDlg, IDBZ_SWITCHTO), FALSE);

     if (lpBZ->dwFlags & BZ_DISABLERETRYBUTTON)
           EnableWindow(GetDlgItem(hDlg, IDBZ_RETRY), FALSE);

     // Call the hook with lCustData in lParam
     UStandardHook((LPVOID)lpBZ, hDlg, WM_INITDIALOG, wParam, lpOBZ-
>lCustData);

     // Update caption if lpszCaption was specified
     if (lpBZ->lpOBZ->lpszCaption && !IsBadReadPtr(lpBZ->lpOBZ->lpszCaption, 
1)
             && lpBZ->lpOBZ->lpszCaption[0] != '\0')
           SetWindowText(hDlg, (LPSTR)lpBZ->lpOBZ->lpszCaption);

     return TRUE;
     }


/*
 * BuildBusyDialogString
 *
 * Purpose:
 *  Builds the string that will be displayed in the dialog from the
 *  task name and window name parameters.
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  dwFlags         DWORD containing flags passed into dialog
 *  iControl        Control ID to place the text string
 *  lpTaskName      LPSTR pointing to name of task (e.g. C:\TEST\TEST.EXE)
 *  lpWindowName    LPSTR for name of window
 *
 * Caveats:
 *  The caller of this function MUST de-allocate the lpTaskName and
 *  lpWindowName pointers itself with OleStdFree
 *
 * Return Value:
 *  void
 */

void BuildBusyDialogString(HWND hDlg, DWORD dwFlags, int iControl, LPSTR
lpTaskName, LPSTR lpWindowName)
{
     LPSTR       pszT, psz1, psz2, psz3;
     UINT        cch;
     LPSTR       pszDot, pszSlash;
     UINT        uiStringNum;

     /*
      * We need scratch memory for loading the stringtable string,
      * the task name, and constructing the final string.  We therefore
      * allocate three buffers as large as the maximum message
      * length (512) plus the object type, guaranteeing that we have enough
      * in all cases.
      */
     cch=512;

     // Use OLE-supplied allocation
     if ((pszT = OleStdMalloc((ULONG)(3*cch))) == NULL)
           return;

     psz1=pszT;
     psz2=psz1+cch;
     psz3=psz2+cch;

     // Parse base name out of path name, use psz2 for the task
     // name to display

     _fstrcpy(psz2, lpTaskName);
     pszDot = _fstrrchr(psz2, '.');
     if (pszDot != NULL)
       *pszDot = '\0'; // Null terminate at the DOT

     pszSlash = _fstrrchr(psz2, '\\'); // Find last backslash in path
     if (pszSlash != NULL)
       psz2 = pszSlash + 1; // Nuke everything up to this point

#ifdef LOWERCASE_NAME
     // Compile this with /DLOWERCASE_NAME if you want the lower-case
     // module name to be displayed in the dialog rather than the
     // all-caps name.
     {
     int i,l;

     // Now, lowercase all letters except first one
     l = _fstrlen(psz2);
     for(i=0;i<l;i++)
       psz2[i] = tolower(psz2[i]);

     psz2[0] = toupper(psz2[0]);
     }
#endif

     // Check size of lpWindowName.  We can reasonably fit about 80
     // characters into the text control, so truncate more than 80 chars
     if (_fstrlen(lpWindowName) > 80)
       lpWindowName[80] = '\0';

     // Load the format string out of stringtable, choose a different
     // string depending on what flags are passed in to the dialog
     if (dwFlags & BZ_NOTRESPONDINGDIALOG)
           uiStringNum = IDS_BZRESULTTEXTNOTRESPONDING;
     else
           uiStringNum = IDS_BZRESULTTEXTBUSY;

     if (LoadString(ghInst, uiStringNum, psz1, cch) == 0)
       return;

     // Build the string. The format string looks like this:
     // "This action cannot be completed because the '%s' application
     // (%s) is [busy | not responding]. Choose \"Switch To\" to activate
'%s' and
     // correct the problem."

     wsprintf(psz3, psz1, (LPSTR)psz2, (LPSTR)lpWindowName, (LPSTR)psz2);
     SetDlgItemText(hDlg, iControl, (LPSTR)psz3);
     OleStdFree(pszT);

     return;
}



/*
 * BusyCleanup
 *
 * Purpose:
 *  Performs busy-specific cleanup before termination.
 *
 * Parameters:
 *  hDlg            HWND of the dialog box so we can access controls.
 *
 * Return Value:
 *  None
 */
void BusyCleanup(HWND hDlg)
{
   return;
}



/*
 * GetTaskInfo()
 *
 * Purpose:  Gets information about the specified task and places the
 * module name, window name and top-level HWND for the task in the specified
 * pointers
 *
 * NOTE: The two string pointers allocated in this routine are
 * the responsibility of the CALLER to de-allocate.
 *
 * Parameters:
 *    hWnd             HWND who called this function
 *    htask            HTASK which we want to find out more info about
 *    lplpszTaskName   Location that the module name is returned
 *    lplpszWindowName Location where the window name is returned
 *
 */

BOOL GetTaskInfo(HWND hWnd, HTASK htask, LPSTR FAR* lplpszTaskName, LPSTR
FAR*lplpszWindowName, HWND FAR*lphWnd)
{
     BOOL        fRet = FALSE;
#if !defined( WIN32 )
     TASKENTRY   te;
     HINSTANCE   hinstToolHelp = NULL;
     typedef BOOL (WINAPI* LPFNTASKFINDHANDLE) (TASKENTRY FAR*, HTASK);
     LPFNTASKFINDHANDLE lpfnTaskFindHandle;
     BOOL        fFoundTaskHandle = FALSE;
#endif
     HWND        hwndNext;
     LPSTR       lpszTN = NULL;
     LPSTR       lpszWN = NULL;
     HWND        hwndFind = NULL;

     // Clear out return values in case of error
     *lplpszTaskName = NULL;
     *lplpszWindowName = NULL;

#if !defined( WIN32 )
     te.dwSize = sizeof(TASKENTRY);

#if !defined( LINK_TOOLHELP )
     // Load the TOOLHELP.DLL library module
     hinstToolHelp = LoadLibrary("TOOLHELP.DLL");
     if (hinstToolHelp > HINSTANCE_ERROR) {  /* loaded successfully */
           // Retrieve the address of the TaskFindHandle function
           lpfnTaskFindHandle = (LPFNTASKFINDHANDLE)
                       GetProcAddress(hinstToolHelp, "TaskFindHandle");
           if (lpfnTaskFindHandle != NULL) {
                 // Call the TaskFindHandle function
                 fFoundTaskHandle=
                             (*lpfnTaskFindHandle) ((TASKENTRY FAR*)&te,
(HTASK)htask);
           }
           FreeLibrary(hinstToolHelp);
     }
#else
     fFoundTaskHandle = TaskFindHandle((TASKENTRY FAR*)&te, htask);
```

```
#endif

     if (fFoundTaskHandle)
#endif
           {
           // Now, enumerate top-level windows in system
           hwndNext = GetWindow(hWnd, GW_HWNDFIRST);
           while (hwndNext)
                 {
                 // See if we can find a non-owned top level window whose
                 // hInstance matches the one we just got passed.  If we
find one,
                 // we can be fairly certain that this is the top-level
window for
                 // the task which is blocked.
                 //
                 // REVIEW:  Will this filter hold true for InProcServer
DLL-created
                 // windows?
                 //
                 if ((hwndNext != hWnd) &&
#if !defined( WIN32 )
                       (GetWindowWord(hwndNext, GWW_HINSTANCE) ==
(UINT)te.hInst) &&
#else
                       (GetWindowThreadProcessId(hwndNext,NULL) ==
(DWORD)htask) &&
#endif
                       (IsWindowVisible(hwndNext)) &&
                       !GetWindow(hwndNext, GW_OWNER))
                       {
                       // We found our window!  Alloc space for new strings
                       if ((lpszTN = OleStdMalloc(OLEUI_CCHPATHMAX)) ==
NULL)
                             return TRUE;  // continue task window
enumeration

                       if ((lpszWN = OleStdMalloc(OLEUI_CCHPATHMAX)) ==
NULL)
                             return TRUE;  // continue task window
enumeration

                       // We found the window we were looking for, copy info
to
                       // local vars
                       GetWindowText(hwndNext, lpszWN, OLEUI_CCHPATHMAX);
#if !defined( WIN32 )
                        LSTRCPYN(lpszTN, te.szModule, OLEUI_CCHPATHMAX);
#else
                       /* WIN32 NOTE: we are not able to get a module name
                       **    given a thread process id on WIN32. the best we
                       **    can do is use the window title as the
module/app
                       **    name.
                       */
```

```
                        LSTRCPYN(lpszTN, lpszWN, OLEUI_CCHPATHMAX);
#endif
                       hwndFind = hwndNext;

                       fRet = TRUE;
                       goto OKDone;
                       }

                 hwndNext = GetWindow(hwndNext, GW_HWNDNEXT);
                 }
           }

OKDone:

     // OK, everything was successful. Set string pointers to point to
     // our data.

     *lplpszTaskName = lpszTN;
     *lplpszWindowName = lpszWN;
     *lphWnd = hwndFind;

     return fRet;
}


/*
 * StartTaskManager()
 *
 * Purpose: Starts Task Manager.  Used to bring up task manager to
 * assist in switching to a given blocked task.
 *
 */

StartTaskManager()
{
     WinExec("taskman.exe", SW_SHOW);
     return TRUE;
}



/*
 * MakeWindowActive()
 *
 * Purpose: Makes specified window the active window.
 *
 */

void MakeWindowActive(HWND hWndSwitchTo)
{
     // Move the new window to the top of the Z-order
     SetWindowPos(hWndSwitchTo, HWND_TOP, 0, 0, 0, 0,
                   SWP_NOSIZE | SWP_NOMOVE);

     // If it's iconic, we need to restore it.
     if (IsIconic(hWndSwitchTo))
           ShowWindow(hWndSwitchTo, SW_RESTORE);
}
```

COMMON.H   (WRAPUI Sample)

```
/*
 * COMMON.H
 *
 * Structures and definitions applicable to all OLE 2.0 UI dialogs.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */


#ifndef _COMMON_H_
#define _COMMON_H_


//Macros to handle control message packing between Win16 and Win32
#ifdef WIN32

#ifndef COMMANDPARAMS
#define COMMANDPARAMS(wID, wCode, hWndMsg)                          \
     UINT        wID     = LOWORD(wParam);                           \
     UINT        wCode   = HIWORD(wParam);                           \
     HWND        hWndMsg = (HWND)(UINT)lParam;
#endif  //COMMANDPARAMS

#ifndef SendCommand
#define SendCommand(hWnd, wID, wCode, hControl)                     \
                 SendMessage(hWnd, WM_COMMAND, MAKELONG(wID, wCode)      \
                                   , (LPARAM)hControl)
#endif  //SendCommand

#ifndef SendSetSel
#define SendSetSel(hWnd,nStart,nEnd)                               \
            SendMessage(hWnd, EM_SETSEL, (WPARAM) nStart, (LPARAM) nEnd)
#endif  //SendSetSel

#else   //Start !WIN32

#ifndef COMMANDPARAMS
#define COMMANDPARAMS(wID, wCode, hWndMsg)                          \
     UINT        wID     = LOWORD(wParam);                           \
     UINT        wCode   = HIWORD(lParam);                           \
     HWND        hWndMsg = (HWND)(UINT)lParam;
#endif  //COMMANDPARAMS

#ifndef SendCommand
#define SendCommand(hWnd, wID, wCode, hControl)                     \
                 SendMessage(hWnd, WM_COMMAND, wID                       \
                                   , MAKELONG(hControl, wCode))
#endif  //SendCommand

#ifndef SendSetSel
#define SendSetSel(hWnd,nStart,nEnd)                               \
            SendMessage(hWnd, EM_SETSEL, 0, MAKELPARAM(nStart, nEnd))
```

```
#endif  //SendSetSel

#endif  //!WIN32



//Property labels used to store dialog structures and fonts
#define STRUCTUREPROP       "Structure"
#define FONTPROP            "Font"


/*
 * Standard structure for all dialogs.  This commonality lets us make
 * a single piece of code that will validate this entire structure and
 * perform any necessary initialization.
 */

typedef struct tagOLEUISTANDARD
     {
     //These IN fields are standard across all OLEUI dialog functions.
     DWORD           cbStruct;       //Structure Size
     DWORD           dwFlags;        //IN-OUT:  Flags
     HWND            hWndOwner;      //Owning window
     LPCSTR          lpszCaption;    //Dialog caption bar contents
     LPFNOLEUIHOOK   lpfnHook;       //Hook callback
     LPARAM          lCustData;      //Custom data to pass to hook
     HINSTANCE       hInstance;      //Instance for customized template name
     LPCSTR          lpszTemplate;   //Customized template name
     HRSRC           hResource;      //Customized template handle
     } OLEUISTANDARD, *POLEUISTANDARD, FAR *LPOLEUISTANDARD;



//Function prototypes
//COMMON.C
UINT  WINAPI  UStandardValidation(const LPOLEUISTANDARD, const UINT, const
HGLOBAL FAR *);
UINT  WINAPI  UStandardInvocation(DLGPROC, LPOLEUISTANDARD, HGLOBAL,
LPCSTR);
LPVOID WINAPI LpvStandardInit(HWND, UINT, BOOL, HFONT FAR *);
LPVOID WINAPI LpvStandardEntry(HWND, UINT, WPARAM, LPARAM, UINT FAR *);
UINT WINAPI   UStandardHook(LPVOID, HWND, UINT, WPARAM, LPARAM);
void WINAPI   StandardCleanup(LPVOID, HWND);
void WINAPI   StandardShowDlgItem(HWND hDlg, int idControl, int nCmdShow);


//DRAWICON.C

//Structure for label and source extraction from a metafile
typedef struct tagLABELEXTRACT
     {
     LPSTR       lpsz;
     UINT        Index;      // index in lpsz (so we can retrieve 2+ lines)
     DWORD       PrevIndex;  // index of last line (so we can mimic word
wrap)
```

```
     union
          {
          UINT    cch;        //Length of label for label extraction
          UINT    iIcon;      //Index of icon in source extraction.
          } u;

     //For internal use in enum procs
     BOOL        fFoundIconOnly;
     BOOL        fFoundSource;
     BOOL        fFoundIndex;
     } LABELEXTRACT, FAR * LPLABELEXTRACT;


//Structure for extracting icons from a metafile (CreateIcon parameters)
typedef struct tagICONEXTRACT
     {
     HICON       hIcon;          //Icon created in the enumeration proc.

     /*
      * Since we want to handle multitasking well we have the caller
      * of the enumeration proc instantiate these variables instead of
      * using statics in the enum proc (which would be bad).
      */
     BOOL        fAND;
     HGLOBAL     hMemAND;        //Enumeration proc allocates and copies
     } ICONEXTRACT, FAR * LPICONEXTRACT;


//Structure to use to pass info to EnumMetafileDraw
typedef struct tagDRAWINFO
     {
     RECT     Rect;
     BOOL     fIconOnly;
     } DRAWINFO, FAR * LPDRAWINFO;


int CALLBACK EXPORT EnumMetafileIconDraw(HDC, HANDLETABLE FAR *, METARECORD
FAR *, int, LPARAM);
int CALLBACK EXPORT EnumMetafileExtractLabel(HDC, HANDLETABLE FAR *,
METARECORD FAR *, int, LPLABELEXTRACT);
int CALLBACK EXPORT EnumMetafileExtractIcon(HDC, HANDLETABLE FAR *,
METARECORD FAR *, int, LPICONEXTRACT);
int CALLBACK EXPORT EnumMetafileExtractIconSource(HDC, HANDLETABLE FAR *,
METARECORD FAR *, int, LPLABELEXTRACT);


//Shared globals:  our instance, registered messages used from all dialogs
and clipboard
// formats used by the PasteSpecial dialog
extern HINSTANCE  ghInst;

extern UINT       uMsgHelp;
extern UINT       uMsgEndDialog;
extern UINT       uMsgBrowse;
extern UINT       uMsgChangeIcon;
extern UINT       uMsgFileOKString;
extern UINT       uMsgCloseBusyDlg;

extern UINT       cfObjectDescriptor;
extern UINT       cfLinkSrcDescriptor;
extern UINT       cfEmbedSource;
extern UINT       cfEmbeddedObject;
extern UINT       cfLinkSource;
extern UINT       cfOwnerLink;
extern UINT       cfFileName;

//Standard control identifiers
#define ID_NULL                         98

#endif //_COMMON_H_
```

COMMON.C   (WRAPUI Sample)

```
/*
 * COMMON.C
 *
 * Standardized (and centralized) pieces of each OLE2UI dialog function:
 *  UStandardValidation     Validates standard fields in each dialog
structure
 *  UStandardInvocation     Invokes a dialog through DialogBoxIndirectParam
 *  LpvStandardInit         Common WM_INITDIALOG processing
 *  LpvStandardEntry        Common code to execute on dialog proc entry.
 *  FStandardHook           Centralized hook calling function.
 *  StandardCleanup         Common exit/cleanup code.
 *  OleUIShowDlgItem        Show-Enable/Hide-Disable dialog item
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */

#define STRICT  1
#include "ole2ui.h"
#include "common.h"
#include "utility.h"
#include <malloc.h>


/*
 * UStandardValidation
 *
 * Purpose:
 *  Performs validation on the standard pieces of any dialog structure,
 *  that is, the fields defined in the OLEUISTANDARD structure.
 *
 * Parameters:
 *  lpUI            const LPOLEUISTANDARD pointing to the shared data of
 *                  all structs.
 *  cbExpect        const UINT structure size desired by the caller.
 *  phDlgMem        const HGLOBAL FAR * in which to store a loaded
customized
 *                  template, if one exists.
 *
 * Return Value:
 *  UINT            OLEUI_SUCCESS if all validation succeeded.  Otherwise
 *                  it will be one of the standard error codes.
 */

UINT WINAPI UStandardValidation(const LPOLEUISTANDARD lpUI, const UINT
cbExpect
                                    , const HGLOBAL FAR *phMemDlg)
     {
     HRSRC       hRes=NULL;
     HGLOBAL     hMem=NULL;


     /*
```

```
      * 1.  Validate non-NULL pointer parameter.  Note:  We don't validate
      *     phDlg since it's not passed from an external source.
      */
     if (NULL==lpUI)
           return OLEUI_ERR_STRUCTURENULL;

     //2.  Validate that the structure is readable and writable.
     if (IsBadReadPtr(lpUI, cbExpect) || IsBadWritePtr(lpUI, cbExpect))
           return OLEUI_ERR_STRUCTUREINVALID;

     //3.  Validate the structure size
     if (cbExpect!=lpUI->cbStruct)
           return OLEUI_ERR_CBSTRUCTINCORRECT;

     //4.  Validate owner-window handle.  NULL is considered valid.
     if (NULL!=lpUI->hWndOwner && !IsWindow(lpUI->hWndOwner))
           return OLEUI_ERR_HWNDOWNERINVALID;

     //5.  Validate the dialog caption.  NULL is considered valid.
     if (NULL!=lpUI->lpszCaption && IsBadReadPtr(lpUI->lpszCaption, 1))
           return OLEUI_ERR_LPSZCAPTIONINVALID;

     //6.  Validate the hook pointer.  NULL is considered valid.
     if ((LPFNOLEUIHOOK)NULL!=lpUI->lpfnHook
           && IsBadCodePtr((FARPROC)lpUI->lpfnHook))
           return OLEUI_ERR_LPFNHOOKINVALID;

     /*
      * 7.  If hInstance is non-NULL, we have to also check lpszTemplate.
      *     Otherwise, lpszTemplate is not used and requires no validation.
      *     lpszTemplate cannot be NULL if used.
      */
     if (NULL!=lpUI->hInstance)
           {
           //Best we can try is one character
           if (NULL==lpUI->lpszTemplate || IsBadReadPtr(lpUI->lpszTemplate,
1))
                 return OLEUI_ERR_LPSZTEMPLATEINVALID;

           hRes=FindResource(lpUI->hInstance, lpUI->lpszTemplate,
RT_DIALOG);

           //This is the only thing that catches invalid non-NULL hInstance
           if (NULL==hRes)
                 return OLEUI_ERR_FINDTEMPLATEFAILURE;

           hMem=LoadResource(lpUI->hInstance, hRes);

           if (NULL==hMem)
                 return OLEUI_ERR_LOADTEMPLATEFAILURE;
           }


     //8.  If hResource is non-NULL, be sure we can lock it.
     if (NULL!=lpUI->hResource)
```

```
           {
           if ((LPSTR)NULL==GlobalLock(lpUI->hResource))
                 return OLEUI_ERR_HRESOURCEINVALID;

           GlobalUnlock(lpUI->hResource);
           }

     /*
      * Here we have hMem==NULL if we should use the standard template
      * or the one in lpUI->hResource.  If hMem is non-NULL, then we
      * loaded one from the calling application's resources which the
      * caller of this function has to free if it sees any other error.
      */
     *(HGLOBAL FAR *)phMemDlg=hMem;
     return OLEUI_SUCCESS;
     }





/*
 * UStandardInvocation
 *
 * Purpose:
 *  Provides standard template loading and calling on DialogBoxIndirectParam
 *  for all the OLE UI dialogs.
 *
 * Parameters:
 *  lpDlgProc       DLGPROC of the dialog function.
 *  lpUI            LPOLEUISTANDARD containing the dialog structure.
 *  hMemDlg         HGLOBAL containing the dialog template.  If this
 *                  is NULL and lpUI->hResource is NULL, then we load
 *                  the standard template given the name in lpszStdTemplate
 *  lpszStdTemplate LPCSTR standard template to load if hMemDlg is NULL
 *                  and lpUI->hResource is NULL.
 *
 * Return Value:
 *  UINT            OLEUI_SUCCESS if all is well, otherwise and error
 *                  code.
 */

UINT WINAPI UStandardInvocation(DLGPROC lpDlgProc, LPOLEUISTANDARD lpUI
                                    , HGLOBAL hMemDlg, LPCSTR
lpszStdTemplate)
     {
     HGLOBAL     hTemplate=hMemDlg;
     HRSRC       hRes;
     int         iRet;

     //Make sure we have a template, then lock it down
     if (NULL==hTemplate)
           hTemplate=lpUI->hResource;

     if (NULL==hTemplate)
```

```
            {
            hRes=FindResource(ghInst, lpszStdTemplate, RT_DIALOG);

            if (NULL==hRes)
                  {
                  return OLEUI_ERR_FINDTEMPLATEFAILURE;
                  }

            hTemplate=LoadResource(ghInst, hRes);

            if (NULL==hTemplate)
                  {
                  return OLEUI_ERR_LOADTEMPLATEFAILURE;
                  }
            }

      /*
       * hTemplate has the template to use, so now we can invoke the dialog.
       * Since we have exported all of our dialog procedures using the
       * _export keyword, we do not need to call MakeProcInstance,
       * we can ue the dialog procedure address directly.
       */

      iRet=DialogBoxIndirectParam(ghInst, hTemplate, lpUI->hWndOwner
                                          , lpDlgProc, (LPARAM)lpUI);

      /*
       * Cleanup the template if we explicitly loaded it.  Caller is
       * responsible for already loaded template resources.
       */
      if (hTemplate!=lpUI->hResource)
            FreeResource(hTemplate);

      if (-1==iRet)
            return OLEUI_ERR_DIALOGFAILURE;

      //Return the code from EndDialog, generally OLEUI_OK or OLEUI_CANCEL
      return (UINT)iRet;
      }
```

```
/*
 * LpvStandardInit
 *
 * Purpose:
 *  Default actions for WM_INITDIALOG handling in the dialog, allocating
 *  a dialog-specific structure, setting that memory as a dialog property,
 *  and creating a small font if necessary setting that font as a property.
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  cbStruct        UINT size of dialog-specific structure to allocate.
 *  fCreateFont     BOOL indicating if we need to create a small Helv
 *                  font for this dialog.
 *  phFont          HFONT FAR * in which to place a created font.  Can be
 *                  NULL if fCreateFont is FALSE.
 *
 * Return Value:
 *  LPVOID          Pointer to global memory allocated for the dialog.
 *                  The memory will have been set as a dialog property
 *                  using the STRUCTUREPROP label.
 */

LPVOID WINAPI LpvStandardInit(HWND hDlg, UINT cbStruct, BOOL fCreateFont,
HFONT FAR * phFont)
     {
     LPVOID      lpv;
     HFONT       hFont;
     LOGFONT     lf;
     HGLOBAL     gh;

     //Must have at least sizeof(OLEUISTANDARD) bytes in cbStruct
     if (sizeof(OLEUISTANDARD) > cbStruct || (fCreateFont && NULL==phFont))
           {
           PostMessage(hDlg, uMsgEndDialog, OLEUI_ERR_GLOBALMEMALLOC, 0L);
           return NULL;
           }

     gh=GlobalAlloc(GMEM_MOVEABLE|GMEM_ZEROINIT, cbStruct);

     if (NULL==gh)
           {
           PostMessage(hDlg, uMsgEndDialog, OLEUI_ERR_GLOBALMEMALLOC, 0L);
           return NULL;
           }
     lpv = GlobalLock(gh);
     SetProp(hDlg, STRUCTUREPROP, gh);

     if (fCreateFont) {
           //Create the non-bold font for result and file texts.  We call
           hFont=(HFONT)SendMessage(hDlg, WM_GETFONT, 0, 0L);
           GetObject(hFont, sizeof(LOGFONT), &lf);
           lf.lfWeight=FW_NORMAL;

           //Attempt to create the font.  If this fails, then we return no
font.
           *phFont=CreateFontIndirect(&lf);

           //If we couldn't create the font, we'll do with the default.
           if (NULL!=*phFont)
                 SetProp(hDlg, FONTPROP, (HANDLE)*phFont);
     }

     return lpv;
     }
```

```
/*
 * LpvStandardEntry
 *
 * Purpose:
 *  Retrieves the dialog's structure property and calls the hook
 *  as necessary.  This should be called on entry into all dialog
 *  procedures.
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  iMsg            UINT message to the dialog
 *  wParam, lParam  WPARAM, LPARAM message parameters
 *  puHookResult    UINT FAR * in which this function stores the return
value
 *                  from the hook if it is called.  If no hook is available,
 *                  this will be FALSE.
 *
 * Return Value:
 *  LPVOID          Pointer to the dialog's extra structure held in the
 *                  STRUCTUREPROP property.
 */

LPVOID WINAPI LpvStandardEntry(HWND hDlg, UINT iMsg, WPARAM wParam, LPARAM
lParam
                               , UINT FAR * puHookResult)
     {
     LPVOID              lpv = NULL;
     HGLOBAL             gh;

     // This will fail under WM_INITDIALOG, where we allocate using
StandardInit
     gh = GetProp(hDlg, STRUCTUREPROP);

     if (NULL!=puHookResult && NULL!=gh)
           {
           *puHookResult=0;

           // gh was locked previously, lock and unlock to get lpv
           lpv = GlobalLock(gh);
           GlobalUnlock(gh);

           //Call the hook for all messages except WM_INITDIALOG
           if (NULL!=lpv && WM_INITDIALOG!=iMsg)
                 *puHookResult=UStandardHook(lpv, hDlg, iMsg, wParam,
lParam);
           }

     return lpv;
     }
```

```
/*
 * UStandardHook
 *
 * Purpose:
 *  Provides a generic hook calling function assuming that all private
 *  dialog structures have a far pointer to their assocated public
 *  structure as the first field, and that the first part of the public
 *  structure matches an OLEUISTANDARD.
 *
 * Parameters:
 *  pv              PVOID to the dialog structure.
 *  hDlg            HWND to send with the call to the hook.
 *  iMsg            UINT message to send to the hook.
 *  wParam, lParam  WPARAM, LPARAM message parameters
 *
 * Return Value:
 *  UINT            Return value from the hook, zero to indicate that
 *                  default action should occur,  nonzero to specify
 *                  that the hook did process the message.  In some
 *                  circumstances it will be important for the hook to
 *                  return a non-trivial non-zero value here, such as
 *                  a brush from WM_CTLCOLOR, in which case the caller
 *                  should return that value from the dialog procedure.
 */

UINT WINAPI UStandardHook(LPVOID lpv, HWND hDlg, UINT iMsg, WPARAM wParam,
LPARAM lParam)
     {
     LPOLEUISTANDARD     lpUI;
     UINT                uRet=0;

     lpUI=*((LPOLEUISTANDARD FAR *)lpv);

     if (NULL!=lpUI && NULL!=lpUI->lpfnHook)
           {
           /*
            * In order for the hook to have the proper DS, they should be
            * compiling with -GA -GEs so and usin __export to get everything
            * set up properly.
            */
           uRet=(*lpUI->lpfnHook)(hDlg, iMsg, wParam, lParam);
           }

     return uRet;
     }





/*
 * StandardCleanup
 *
 * Purpose:
 *  Removes properties and reverses any other standard initiazation
 *  done through StandardSetup.
 *
 * Parameters:
 *  lpv             LPVOID containing the private dialog structure.
 *  hDlg            HWND of the dialog closing.
 *
 * Return Value:
 *  None
 */

void WINAPI StandardCleanup(LPVOID lpv, HWND hDlg)
     {
     HFONT       hFont;
     HGLOBAL     gh;

     hFont=(HFONT)GetProp(hDlg, FONTPROP);

     if (NULL!=hFont)
           DeleteObject(hFont);

     RemoveProp(hDlg, FONTPROP);

     gh = RemoveProp(hDlg, STRUCTUREPROP);
     if (gh)
           {
           GlobalUnlock(gh);
           GlobalFree(gh);
           }
     return;
     }


/* StandardShowDlgItem
** -------------------
**    Show & Enable or Hide & Disable a dialog item as appropriate.
**    it is NOT sufficient to simply hide the item; it must be disabled
**    too or the keyboard accelerator still functions.
*/
void WINAPI StandardShowDlgItem(HWND hDlg, int idControl, int nCmdShow)
{
     if (SW_HIDE == nCmdShow) {
           ShowWindow(GetDlgItem(hDlg, idControl), SW_HIDE);
           EnableWindow(GetDlgItem(hDlg, idControl), FALSE);
     } else {
           ShowWindow(GetDlgItem(hDlg, idControl), SW_SHOWNORMAL);
           EnableWindow(GetDlgItem(hDlg, idControl), TRUE);
     }
}
```

CONVERT.H (WRAPUI Sample)

```
/*
 * CONVERT.H
 *
 * Internal definitions, structures, and function prototypes for the
 * OLE 2.0 UI Convert dialog.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */


#ifndef _CONVERT_H_
#define _CONVERT_H_


//Internally used structure
typedef struct tagCONVERT
     {
     //Keep this item first as the Standard* functions depend on it here.
     LPOLEUICONVERT     lpOCV;       //Original structure passed.

     /*
      * What we store extra in this structure besides the original caller's
      * pointer are those fields that we need to modify during the life of
      * the dialog but that we don't want to change in the original
structure
      * until the user presses OK.
      */

     DWORD               dwFlags;  // Flags passed in
     HWND                hListVisible;  // listbox that is currently visible
     HWND                hListInvisible;  // listbox that is currently
hidden
     CLSID               clsid;    // Class ID sent in to dialog: IN only
     DWORD               dvAspect;
     BOOL                fCustomIcon;
     UINT                IconIndex;         // index (in exe) of current
icon
     LPSTR               lpszIconSource;    // path to current icon source
     LPSTR               lpszCurrentObject;
     LPSTR               lpszConvertDefault;
     LPSTR               lpszActivateDefault;
     } CONVERT, *PCONVERT, FAR *LPCONVERT;



//Internal function prototypes in CONVERT.C
BOOL CALLBACK EXPORT ConvertDialogProc(HWND, UINT, WPARAM, LPARAM);
BOOL            FConvertInit(HWND hDlg, WPARAM, LPARAM);
UINT            FPopulateListbox(HWND hListbox, CLSID cID);
BOOL            IsValidClassID(CLSID cID);
void            SetConvertResults(HWND, LPCONVERT);
UINT FillClassList(
           CLSID clsid,
           HWND hList,
           HWND hListInvisible,
           LPSTR FAR *lplpszCurrentClass,
           BOOL fIsLinkedObject,
           WORD wFormat,
           UINT cClsidExclude,
           LPCLSID lpClsidExclude);
BOOL            FormatIncluded(LPSTR szStringToSearch, UINT wFormat);
void            UpdateCVClassIcon(HWND hDlg, LPCONVERT lpCV, HWND hList);
void            SwapWindows(HWND, HWND, HWND);
void            ConvertCleanup(HWND hDlg, LPCONVERT lpCV);

#endif // _CONVERT_H_
```

CONVERT.C   (WRAPUI Sample)

```
/*
 * CONVERT.C
 *
 * Implements the OleUIConvert function which invokes the complete
 * Convert dialog.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */

#define STRICT  1
#include "ole2ui.h"
#include <stdlib.h>
#include "common.h"
#include "utility.h"
#include "geticon.h"
#include "regdb.h"
#include "convert.h"

#define CF_CLIPBOARDMIN   0xc000
#define CF_CLIPBOARDMAX   0xffff

#define AUXUSERTYPE_SHORTNAME  USERCLASSTYPE_SHORT  // short name

static char szOLE2DLL[] = "ole2.dll";   // name of OLE 2.0 library
static char szVanillaDocIcon[] = "DefIcon";

/*
 * OleUIConvert
 *
 * Purpose:
 *  Invokes the standard OLE Change Type dialog box allowing the user
 *  to change the type of the single specified object, or change the
 *  type of all OLE objects of a specified type.
 *
 * Parameters:
 *  lpCV            LPOLEUICONVERT pointing to the in-out structure
 *                  for this dialog.
 *
 * Return Value:
 *  UINT            One of the following codes, indicating success or error:
 *                      OLEUI_SUCCESS           Success
 *                      OLEUI_ERR_STRUCTSIZE    The dwStructSize value is
wrong
 */

STDAPI_(UINT) OleUIConvert(LPOLEUICONVERT lpCV)
     {
     UINT        uRet;
     HGLOBAL     hMemDlg=NULL;

     uRet=UStandardValidation((LPOLEUISTANDARD)lpCV, sizeof(OLEUICONVERT)
                                          , &hMemDlg);
     if (OLEUI_SUCCESS!=uRet)
           return uRet;

     // Validate structure members passed in.
#if defined( OBSOLETE )
     if (!IsValidClassID(lpCV->clsid))
        uRet = OLEUI_CTERR_CLASSIDINVALID;
#endif

     if ( (lpCV->dwFlags & CF_SETCONVERTDEFAULT)
            && (!IsValidClassID(lpCV->clsidConvertDefault)) )
        uRet = OLEUI_CTERR_CLASSIDINVALID;

     if ( (lpCV->dwFlags & CF_SETACTIVATEDEFAULT)
            && (!IsValidClassID(lpCV->clsidActivateDefault)) )
        uRet = OLEUI_CTERR_CLASSIDINVALID;

     if ( (lpCV->dvAspect != DVASPECT_ICON)
            && (lpCV->dvAspect != DVASPECT_CONTENT) )
        uRet = OLEUI_CTERR_DVASPECTINVALID;

     if ( (lpCV->wFormat >= CF_CLIPBOARDMIN)
            && (lpCV->wFormat <= CF_CLIPBOARDMAX) )
     {
            char szTemp[8];

            if (0 == GetClipboardFormatName(lpCV->wFormat, (LPSTR)szTemp,
8))
              uRet = OLEUI_CTERR_CBFORMATINVALID;
     }


     if ( (NULL != lpCV->lpszUserType)
           && (IsBadReadPtr(lpCV->lpszUserType, 1)) )
        uRet = OLEUI_CTERR_STRINGINVALID;

     if ( (NULL != lpCV->lpszDefLabel)
           && (IsBadReadPtr(lpCV->lpszDefLabel, 1)) )
        uRet = OLEUI_CTERR_STRINGINVALID;

     if (0!=lpCV->cClsidExclude)
           {
           if (NULL!=lpCV->lpClsidExclude && IsBadReadPtr(lpCV-
>lpClsidExclude
                 , lpCV->cClsidExclude*sizeof(CLSID)))
           uRet=OLEUI_IOERR_LPCLSIDEXCLUDEINVALID;
           }


     if (OLEUI_ERR_STANDARDMIN <= uRet)
           {
           if (NULL!=hMemDlg)
                 FreeResource(hMemDlg);
            return uRet;
            }

      //Now that we've validated everything, we can invoke the dialog.
      uRet=UStandardInvocation(ConvertDialogProc, (LPOLEUISTANDARD)lpCV,
                                          hMemDlg,
MAKEINTRESOURCE(IDD_CONVERT));

      return uRet;
      }





/*
 * ConvertDialogProc
 *
 * Purpose:
 *  Implements the OLE Convert dialog as invoked through the
 *  OleUIConvert function.
 *
 * Parameters:
 *  Standard
 *
 * Return Value:
 *  Standard
 *
 */

BOOL CALLBACK EXPORT ConvertDialogProc(HWND hDlg, UINT iMsg, WPARAM wParam,
LPARAM lParam)
      {
      LPCONVERT           lpCV;
      UINT                uRet = 0;
      OLEUICHANGEICON     ci;

      //Declare Win16/Win32 compatible WM_COMMAND parameters.
      COMMANDPARAMS(wID, wCode, hWndMsg);

      //This will fail under WM_INITDIALOG, where we allocate it.
      lpCV=(LPCONVERT)LpvStandardEntry(hDlg, iMsg, wParam, lParam, (UINT FAR
*)&uRet);

      //If the hook processed the message, we're done.
      if (0!=uRet)
            return (BOOL)uRet;

      //Process the temination message
      if (iMsg==uMsgEndDialog)
      {
            ConvertCleanup(hDlg, lpCV);
            StandardCleanup(lpCV, hDlg);
            EndDialog(hDlg, wParam);
            return TRUE;
      }

      // Process help message from Change Icon
      if (iMsg == uMsgHelp)
      {

        PostMessage(lpCV->lpOCV->hWndOwner, uMsgHelp, wParam, lParam);
        return FALSE;

      }

      switch (iMsg)
            {
            case WM_INITDIALOG:
                  FConvertInit(hDlg, wParam, lParam);
                  return TRUE;

            case WM_COMMAND:
                  switch (wID)
                  {
                        case IDCV_ACTIVATELIST:
                        case IDCV_CONVERTLIST:
                              switch (wCode)
                              {
                                    case LBN_SELCHANGE:

                                          // Change "Results" window to
reflect current selection
                                          SetConvertResults(hDlg, lpCV);

                                          // Update the icon we display, if
we are indeed
                                          // displaying an icon.
                                          if ( (lpCV->dwFlags &
CF_SELECTCONVERTTO)

                                                && (lpCV->dvAspect ==
DVASPECT_ICON)

                                                && (!lpCV->fCustomIcon) )
                                            UpdateCVClassIcon(hDlg, lpCV,
hWndMsg);


                                          break;

                                    case LBN_DBLCLK:
                                          //Same as pressing OK.
                                          SendCommand(hDlg, IDOK, BN_CLICKED,
hWndMsg);
                                          break;
                              }
                              break;

                        case IDCV_CONVERTTO:
                        case IDCV_ACTIVATEAS:
                        {
                              HWND    hList, hListInvisible;
```

```
                             LRESULT lRetVal;
                             BOOL    fState;

                             hList = lpCV->hListVisible;
                             hListInvisible = lpCV->hListInvisible;


                             if (IDCV_CONVERTTO == wParam)
                             {

                                // User just click on the button again - it
was
                                // already selected.
                                if (lpCV->dwFlags & CF_SELECTCONVERTTO)
                                     break;


                                // Turn painting updates off.
                                SendMessage(hDlg, WM_SETREDRAW, FALSE, 0L);


                                // If we're working with a linked object,
don't
                                // add the activate list - just the object's
                                // class should appear in the listbox.

                                SwapWindows(hDlg,
                                                  hList,
                                                  hListInvisible);

                                lpCV->hListVisible = hListInvisible;
                                lpCV->hListInvisible = hList;

                                EnableWindow(lpCV->hListInvisible, FALSE);
                                EnableWindow(lpCV->hListVisible, TRUE);

                                // Update our flags.
                                lpCV->dwFlags &= ~CF_SELECTACTIVATEAS;
                                lpCV->dwFlags |= CF_SELECTCONVERTTO;

                             }
                             else
                             {
                                if (lpCV->dwFlags & CF_SELECTACTIVATEAS)
                                     break;

                                // Turn painting updates off.
                                SendMessage(hDlg, WM_SETREDRAW, FALSE, 0L);

                                SwapWindows(hDlg,
                                                  hList,
                                                  hListInvisible);

                                lpCV->hListVisible = hListInvisible;
                                lpCV->hListInvisible = hList;
```

```
                              EnableWindow(lpCV->hListInvisible, FALSE);
                              EnableWindow(lpCV->hListVisible, TRUE);


                              // Update our flags.
                              lpCV->dwFlags |= CF_SELECTACTIVATEAS;
                              lpCV->dwFlags &= ~CF_SELECTCONVERTTO;
                           }


                           if (lpCV->dwFlags & CF_SELECTCONVERTTO)
                              lRetVal = SendMessage(lpCV->hListVisible,
LB_SELECTSTRING, (WPARAM)-1, (LPARAM)lpCV->lpszConvertDefault);

                           else
                              lRetVal = SendMessage(lpCV->hListVisible,
LB_SELECTSTRING, (WPARAM)-1, (LPARAM)lpCV->lpszActivateDefault);

                           if (LB_ERR == lRetVal)
                           {
                              char szCurrentObject[40];

                              GetDlgItemText(hDlg, IDCV_OBJECTTYPE,
(LPSTR)szCurrentObject, 40);
                              SendMessage(lpCV->hListVisible,
LB_SELECTSTRING, (WPARAM)-1, (LPARAM)(LPSTR)szCurrentObject);
                           }

                           // Turn updates back on.
                           SendMessage(hDlg, WM_SETREDRAW, TRUE, 0L);

                           InvalidateRect(lpCV->hListVisible, NULL, TRUE);
                           UpdateWindow(lpCV->hListVisible);

                           if ((lpCV->dvAspect & DVASPECT_ICON) && (lpCV-
>dwFlags & CF_SELECTCONVERTTO))
                             UpdateCVClassIcon(hDlg, lpCV, lpCV-
>hListVisible);

                           // Hide the icon stuff when Activate is
selected...show
                           // it again when Convert is selected.

                           fState = ((lpCV->dwFlags & CF_SELECTACTIVATEAS)
||
                                         (lpCV->dwFlags &
CF_DISABLEDISPLAYASICON)) ?
                                         SW_HIDE : SW_SHOW;

                           StandardShowDlgItem(hDlg, IDCV_DISPLAYASICON,
fState);

                           // Only display the icon if convert is selected
AND
```

```
                              // display as icon is checked.
                              if ((SW_SHOW==fState) && (DVASPECT_ICON!=lpCV-
>dvAspect))
                                 fState = SW_HIDE;

                              StandardShowDlgItem(hDlg, IDCV_CHANGEICON,
fState);
                              StandardShowDlgItem(hDlg, IDCV_ICON, fState);
                              StandardShowDlgItem(hDlg, IDCV_ICONLABEL1,
fState);
                              StandardShowDlgItem(hDlg, IDCV_ICONLABEL2,
fState);

                          SetConvertResults(hDlg, lpCV);

                        }
                        break;


                        case IDOK:
                        {

                              UINT  iCurSel;
                              LPSTR lpszCLSID;
                              char  szBuffer[256];

                              // Set OUT parameters

                              //
                              // Set output flags to current ones
                              //
                              lpCV->lpOCV->dwFlags = lpCV->dwFlags;


                              // Update the dvAspect and fObjectsIconChanged
members
                              // as appropriate.
                              //
                              if (lpCV->dwFlags & CF_SELECTACTIVATEAS)
                              {
                                // DON'T update aspect if activate as was
selected.
                                lpCV->lpOCV->fObjectsIconChanged = FALSE;
                              }
                              else
                                lpCV->lpOCV->dvAspect = lpCV->dvAspect;


                              //
                              // Get the new clsid
                              //
                              iCurSel = (UINT)SendMessage(lpCV->hListVisible,
LB_GETCURSEL, 0, 0);
                              SendMessage(lpCV->hListVisible, LB_GETTEXT,
iCurSel, (LPARAM)(LPSTR)szBuffer);
```

```
                              lpszCLSID = PointerToNthField((LPSTR)szBuffer,
2, '\t');

                              CLSIDFromString(lpszCLSID, (LPCLSID)(&(lpCV-
>lpOCV->clsidNew)));



                              // Free the hMetaPict we got in.
                              OleUIMetafilePictIconFree(lpCV->lpOCV-
>hMetaPict);

                              //
                              // Get the hMetaPict (if display as icon is
checked)
                              //
                              if (DVASPECT_ICON == lpCV->dvAspect)
                              {
                                 HICON hIcon;
                                 char  szLabel[OLEUI_CCHLABELMAX];
                                 int   Index;


                                 // Create the hMetaPict here from icon,
label,
                                 // index, and path

                                 hIcon = (HICON)SendDlgItemMessage(hDlg,
IDCV_ICON, STM_GETICON, 0, 0L);

                                 // the combined length of the 2 label lines
won't ever be more than
                                 // OLEUI_CCHLABELMAX.
                                 Index = (int)SendDlgItemMessage(hDlg,
IDCV_ICONLABEL1, WM_GETTEXT, OLEUI_CCHLABELMAX, (LPARAM)(LPSTR)szLabel);

                                 if (Index < OLEUI_CCHLABELMAX)
                                 {
                                      LPSTR lpszSecondLine = szLabel + Index;


                                      SendDlgItemMessage(hDlg,
IDCV_ICONLABEL2, WM_GETTEXT,

OLEUI_CCHLABELMAX-Index,
                                                                             (LPARAM)
(LPSTR)lpszSecondLine);
                                 }
                                 lpCV->lpOCV->hMetaPict =

OleUIMetafilePictFromIconAndLabel(hIcon,

                     (LPSTR)szLabel,
                     lpCV->lpszIconSource,

                     lpCV->IconIndex);

                              }
                              else
                                 lpCV->lpOCV->hMetaPict = (HGLOBAL)NULL;


                              //
                              // End the dialog
                              //
                              SendMessage(hDlg, uMsgEndDialog, OLEUI_OK, 0L);
                        }
                        break;

                        case IDCANCEL:
                              SendMessage(hDlg, uMsgEndDialog, OLEUI_CANCEL,
0L);
                              break;


                        case ID_OLEUIHELP:
                              PostMessage(lpCV->lpOCV->hWndOwner,
                                                uMsgHelp, (WPARAM)hDlg,
MAKELPARAM(IDD_CONVERT, 0));
                              break;

                        case IDCV_DISPLAYASICON:
                        {

                              int i;
                              BOOL fCheck;

                              fCheck=IsDlgButtonChecked(hDlg, wID);

                              if (fCheck)
                                    lpCV->dvAspect = DVASPECT_ICON;
                              else
                                    lpCV->dvAspect = DVASPECT_CONTENT;

                              if (fCheck && (!lpCV->fCustomIcon))
                                 UpdateCVClassIcon(hDlg, lpCV, lpCV-
>hListVisible);

                              //Show or hide the icon depending on the check
state.

                              i=(fCheck) ? SW_SHOWNORMAL : SW_HIDE;

                              StandardShowDlgItem(hDlg, IDCV_CHANGEICON, i);
                              StandardShowDlgItem(hDlg, IDCV_ICON, i);
                              StandardShowDlgItem(hDlg, IDCV_ICONLABEL1, i);
                              StandardShowDlgItem(hDlg, IDCV_ICONLABEL2, i);
```

```
                              SetConvertResults(hDlg, lpCV);

                        }
                        break;

                        case IDCV_CHANGEICON:
                        {
                              LPMALLOC pIMalloc;
                              LPSTR    pszString, pszCLSID;
                              int      iSel;
                              HICON    hIcon;
                              char     szLabel[OLEUI_CCHLABELMAX];
                              int      Index;


                              //Initialize the structure for the hook.
                              _fmemset((LPOLEUICHANGEICON)&ci, 0,
sizeof(ci));


                              // Create the hMetaPict here from icon, label,
                              // index, and path

                              hIcon = (HICON)SendDlgItemMessage(hDlg,
IDCV_ICON, STM_GETICON, 0, 0L);

                              // the combined length of the 2 label lines
won't ever be more than
                              // OLEUI_CCHLABELMAX.

                              Index = (int)SendDlgItemMessage(hDlg,
IDCV_ICONLABEL1, WM_GETTEXT, OLEUI_CCHLABELMAX, (LPARAM)(LPSTR)szLabel);

                              if (Index < OLEUI_CCHLABELMAX)
                              {
                                 LPSTR    lpszSecondLine;

                                 lpszSecondLine = szLabel + Index;

                                 SendDlgItemMessage(hDlg, IDCV_ICONLABEL2,
WM_GETTEXT,

OLEUI_CCHLABELMAX-Index,
                                                              (LPARAM)
(LPSTR)lpszSecondLine);
                              }

                              ci.hMetaPict =

OleUIMetafilePictFromIconAndLabel(hIcon,

                  (LPSTR)szLabel,

                  lpCV->lpszIconSource,
                 lpCV->IconIndex);

                              ci.cbStruct =sizeof(ci);
                              ci.hWndOwner=hDlg;
                              ci.dwFlags  = CIF_SELECTCURRENT;

                              // Only show help if we're showing it for this
dialog.
                              if (lpCV->dwFlags & CF_SHOWHELPBUTTON)
                                ci.dwFlags  |= CIF_SHOWHELP;

                              iSel = (int)SendMessage(lpCV->hListVisible,
LB_GETCURSEL, 0, 0L);

                              CoGetMalloc(MEMCTX_TASK, &pIMalloc);

                              pszString = (LPSTR)pIMalloc->lpVtbl-
>Alloc(pIMalloc, OLEUI_CCHLABELMAX + OLEUI_CCHCLSIDSTRING);

                              // Get whole string
                              SendMessage(lpCV->hListVisible, LB_GETTEXT,
iSel, (LONG)pszString);

                              // Set pointer to CLSID (string)
                              pszCLSID = PointerToNthField(pszString, 2,
'\t');

                              // Get the clsid to pass to change icon.
                              CLSIDFromString((LPSTR)pszCLSID,
(LPCLSID)&(ci.clsid));

                              pIMalloc->lpVtbl->Free(pIMalloc,
(LPVOID)pszString);
                              pIMalloc->lpVtbl->Release(pIMalloc);

                              //Let the hook in to customize Change Icon if
desired.
                              uRet=UStandardHook(lpCV, hDlg, uMsgChangeIcon
                                                           , 0, (LONG)
(LPSTR)&ci);

                              if (0==uRet)
                                    uRet=(UINT)
(OLEUI_OK==OleUIChangeIcon((LPOLEUICHANGEICON)&ci));

                              //Update the display if necessary.
                              if (0!=uRet)
                              {
                                    HICON hIcon;
                                    char  szLabel[OLEUI_CCHLABELMAX];
                                    DWORD dwWrapIndex;
```

```
                                   hIcon =
OleUIMetafilePictExtractIcon(ci.hMetaPict);

                                   SendDlgItemMessage(hDlg, IDCV_ICON,
STM_SETICON, (WPARAM)hIcon, 0L);


     OleUIMetafilePictExtractIconSource(ci.hMetaPict, lpCV->lpszIconSource,
&(lpCV->IconIndex));


     OleUIMetafilePictExtractLabel(ci.hMetaPict, szLabel, OLEUI_CCHLABELMAX,
&dwWrapIndex);

                                   if (0 == dwWrapIndex)  // no second line
                                   {
                                      SendDlgItemMessage(hDlg,
IDCV_ICONLABEL1, WM_SETTEXT, 0, (LPARAM)(LPSTR)szLabel);
                                      SendDlgItemMessage(hDlg,
IDCV_ICONLABEL2, WM_SETTEXT, 0, (LPARAM)(LPSTR)"");
                                   }
                                   else
                                   {

                                      LPSTR lpszSecondLine;

                                      lpszSecondLine = szLabel +
dwWrapIndex;
                                      SendDlgItemMessage(hDlg,
IDCV_ICONLABEL2,

WM_SETTEXT, 0, (LPARAM)lpszSecondLine);

                                      *lpszSecondLine = '\0';
                                      SendDlgItemMessage(hDlg,
IDCV_ICONLABEL1,

WM_SETTEXT, 0, (LPARAM)(LPSTR)szLabel);
                                   }


                                   // Update our custom/default flag

                                   if (ci.dwFlags & CIF_SELECTDEFAULT)
                                      lpCV->fCustomIcon = FALSE;   // we're
in default mode (icon changes on each LB selchange)
                                   else if (ci.dwFlags & CIF_SELECTFROMFILE)
                                      lpCV->fCustomIcon = TRUE;    // we're
in custom mode (icon doesn't change)
                                   // no change in fCustomIcon if user
selected current


                                   lpCV->lpOCV->fObjectsIconChanged = TRUE;
                             }
```

```
                        }
                        break;

                  }
                  break;
            }
     return FALSE;
     }


/*
 * FConvertInit
 *
 * Purpose:
 *  WM_INITIDIALOG handler for the Convert dialog box.
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  wParam          WPARAM of the message
 *  lParam          LPARAM of the message
 *
 * Return Value:
 *  BOOL            Value to return for WM_INITDIALOG.
 */

BOOL FConvertInit(HWND hDlg, WPARAM wParam, LPARAM lParam)
     {
     LPCONVERT               lpCV;
     LPOLEUICONVERT          lpOCV;
     LPMALLOC                pIMalloc;
     HFONT                   hFont;  // non-bold version of dialog's font
     RECT                    rc;
     DWORD                   dw;
     int                     cItemsActivate;
     HKEY                    hKey;
     LONG                    lRet;
     UINT                    nRet;


     //Copy the structure at lParam into our instance memory.
     lpCV=(LPCONVERT)LpvStandardInit(hDlg, sizeof(CONVERT), TRUE, (HFONT FAR
*)&hFont);

     //PvStandardInit send a termination to us already.
     if (NULL==lpCV)
           return FALSE;

     lpOCV=(LPOLEUICONVERT)lParam;

     lpCV->lpOCV=lpOCV;

     lpCV->fCustomIcon = FALSE;

     //Copy other information from lpOCV that we might modify.
     lpCV->dwFlags = lpOCV->dwFlags;
     lpCV->clsid = lpOCV->clsid;
     lpCV->dvAspect = lpOCV->dvAspect;
     lpCV->hListVisible = GetDlgItem(hDlg, IDCV_ACTIVATELIST);
     lpCV->hListInvisible = GetDlgItem(hDlg, IDCV_CONVERTLIST);
     lpCV->lpszCurrentObject = lpOCV->lpszUserType;

     lpOCV->clsidNew = CLSID_NULL;

     lpOCV->fObjectsIconChanged = FALSE;

     //Allocate space for our strings
     if (NOERROR != CoGetMalloc(MEMCTX_TASK, &pIMalloc))
        return FALSE;

     lpCV->lpszConvertDefault = (LPSTR)pIMalloc->lpVtbl->Alloc(pIMalloc,
OLEUI_CCHLABELMAX);
     lpCV->lpszActivateDefault = (LPSTR)pIMalloc->lpVtbl->Alloc(pIMalloc,
OLEUI_CCHLABELMAX);
     lpCV->lpszIconSource = (LPSTR)pIMalloc->lpVtbl->Alloc(pIMalloc,
OLEUI_CCHPATHMAX);
     pIMalloc->lpVtbl->Release(pIMalloc);

     //If we got a font, send it to the necessary controls.
     if (NULL!=hFont)
           {
           SendDlgItemMessage(hDlg, IDCV_OBJECTTYPE, WM_SETFONT,
(WPARAM)hFont, 0L);
           SendDlgItemMessage(hDlg, IDCV_RESULTTEXT, WM_SETFONT,
(WPARAM)hFont, 0L);
           SendDlgItemMessage(hDlg, IDCV_ICONLABEL1, WM_SETFONT,
(WPARAM)hFont, 0L);
           SendDlgItemMessage(hDlg, IDCV_ICONLABEL2, WM_SETFONT,
(WPARAM)hFont, 0L);
           }

     //Hide the help button if necessary
     if (!(lpCV->dwFlags & CF_SHOWHELPBUTTON))
           StandardShowDlgItem(hDlg, ID_OLEUIHELP, SW_HIDE);

     //Fill the Object Type listbox with entries from the reg DB.
     nRet = FillClassList(lpOCV->clsid,
                       lpCV->hListVisible,
                       lpCV->hListInvisible,
                       &(lpCV->lpszCurrentObject),
                       lpOCV->fIsLinkedObject,
                       lpOCV->wFormat,
                       lpOCV->cClsidExclude,
                       lpOCV->lpClsidExclude);

     if (nRet == -1) {
           // bring down dialog if error when filling list box
           PostMessage(hDlg, uMsgEndDialog, OLEUI_ERR_LOADSTRING, 0L);
     }

     // Set the name of the current object.
     SetDlgItemText(hDlg, IDCV_OBJECTTYPE, (LPSTR)lpCV->lpszCurrentObject);

     // Disable the "Activate As" button if the Activate list doesn't
     // have any objects in it.

     cItemsActivate = (int)SendMessage(lpCV->hListVisible, LB_GETCOUNT, 0,
0L);

     if (1 >= cItemsActivate || (lpCV->dwFlags & CF_DISABLEACTIVATEAS))
       EnableWindow(GetDlgItem(hDlg, IDCV_ACTIVATEAS), FALSE);

     //Set the tab width in the list to push all the tabs off the side.
     GetClientRect(lpCV->hListVisible, (LPRECT)&rc);
     dw=GetDialogBaseUnits();
     rc.right =(8*rc.right)/LOWORD(dw);  //Convert pixels to 2x dlg units.
     SendMessage(lpCV->hListVisible, LB_SETTABSTOPS, 1, (LPARAM)(LPINT)
(&rc.right));
     SendMessage(lpCV->hListInvisible, LB_SETTABSTOPS, 1, (LPARAM)(LPINT)
(&rc.right));


     // Make sure that either "Convert To" or "Activate As" is selected
     // and initialize listbox contents and selection accordingly
     if (lpCV->dwFlags & CF_SELECTACTIVATEAS)
     {
       // Don't need to adjust listbox here because FillClassList
       // initializes to the "Activate As" state.
        CheckRadioButton(hDlg, IDCV_CONVERTTO, IDCV_ACTIVATEAS,
IDCV_ACTIVATEAS);

        // Hide the icon stuff when Activate is selected...it gets shown
        // again when Convert is selected.

        StandardShowDlgItem(hDlg, IDCV_DISPLAYASICON, SW_HIDE);
        StandardShowDlgItem(hDlg, IDCV_CHANGEICON, SW_HIDE);
        StandardShowDlgItem(hDlg, IDCV_ICON, SW_HIDE);
        StandardShowDlgItem(hDlg, IDCV_ICONLABEL1, SW_HIDE);
        StandardShowDlgItem(hDlg, IDCV_ICONLABEL2, SW_HIDE);
     }
     else
     {
       // Default case.  If user hasn't selected either flag, we will
       // come here anyway.
       // swap listboxes.

       HWND hWndTemp = lpCV->hListVisible;

       if ( lpCV->dwFlags & CF_DISABLEDISPLAYASICON ) {
             StandardShowDlgItem(hDlg, IDCV_DISPLAYASICON, SW_HIDE);
             StandardShowDlgItem(hDlg, IDCV_CHANGEICON, SW_HIDE);
             StandardShowDlgItem(hDlg, IDCV_ICON, SW_HIDE);
             StandardShowDlgItem(hDlg, IDCV_ICONLABEL1, SW_HIDE);
             StandardShowDlgItem(hDlg, IDCV_ICONLABEL2, SW_HIDE);
       }
       lpCV->dwFlags |= CF_SELECTCONVERTTO; // Make sure flag is set
       CheckRadioButton(hDlg, IDCV_CONVERTTO, IDCV_ACTIVATEAS,
IDCV_CONVERTTO);

       SwapWindows(hDlg, lpCV->hListVisible, lpCV->hListInvisible);

       lpCV->hListVisible = lpCV->hListInvisible;
       lpCV->hListInvisible = hWndTemp;

       EnableWindow(lpCV->hListInvisible, FALSE);
       EnableWindow(lpCV->hListVisible, TRUE);
     }



     // Initialize Default strings.

     // Default convert string is easy...just user the user type name from
     // the clsid we got, or the current object
     if ( (lpCV->dwFlags & CF_SETCONVERTDEFAULT)
            && (IsValidClassID(lpCV->lpOCV->clsidConvertDefault)) )
     {
           dw = OleStdGetUserTypeOfClass((LPCLSID)(&lpCV->lpOCV-
>clsidConvertDefault),
                                                        lpCV-
>lpszConvertDefault,
                                                        OLEUI_CCHLABELMAX,
                                                        NULL);

           if (0 == dw)
              lstrcpy((LPSTR)lpCV->lpszConvertDefault, (LPSTR)lpCV-
>lpszCurrentObject);
     }
     else
           lstrcpy((LPSTR)lpCV->lpszConvertDefault, (LPSTR)lpCV-
>lpszCurrentObject);



   // Default activate is a bit trickier.  We want to use the user type
   // name if from the clsid we got (assuming we got one), or the current
   // object if it fails or we didn't get a clsid.  But...if there's a
   // Treat As entry in the reg db, then we use that instead.  So... the
   // logic boils down to this:
   //
   // if ("Treat As" in reg db)
   //    use it;
   // else
   //    if (CF_SETACTIVATEDEFAULT)
   //      use it;
   //    else
   //      use current object;
```

```
     lRet = RegOpenKey(HKEY_CLASSES_ROOT, "CLSID", (HKEY FAR *)&hKey);

     if (lRet != ERROR_SUCCESS)
       goto CheckInputFlag;

     else
     {
        LPSTR lpszCLSID;
        char  szKey[OLEUI_CCHKEYMAX];
        CLSID clsid;
        char  szValue[OLEUI_CCHKEYMAX];

        StringFromCLSID(&(lpCV->lpOCV->clsid), &lpszCLSID);
        lstrcpy(szKey, lpszCLSID);
        lstrcat(szKey, (LPSTR)"\\TreatAs");

        dw = OLEUI_CCHKEYMAX;
        lRet = RegQueryValue(hKey, (LPSTR)szKey, (LPSTR)szValue,
(LPDWORD)&dw);

        if (lRet != ERROR_SUCCESS)
        {

             RegCloseKey(hKey);
             OleStdFreeString(lpszCLSID, NULL);
             goto CheckInputFlag;
        }
        else
        {
             CLSIDFromString((LPSTR)szValue, &clsid);
             if (0 == OleStdGetUserTypeOfClass(&clsid,
                                               lpCV->lpszActivateDefault,
                                               OLEUI_CCHLABELMAX,
                                               NULL))
             {
                  RegCloseKey(hKey);
                  OleStdFreeString(lpszCLSID, NULL);
                  goto CheckInputFlag;
             }
        }
        RegCloseKey(hKey);
        OleStdFreeString(lpszCLSID, NULL);
        goto SelectStringInListbox;
     }


CheckInputFlag:
     if ( (lpCV->dwFlags & CF_SETACTIVATEDEFAULT)
            && (IsValidClassID(lpCV->lpOCV->clsidActivateDefault)) )
     {
           dw = OleStdGetUserTypeOfClass((LPCLSID)(&lpCV->lpOCV-
>clsidActivateDefault),
                                                    lpCV-
>lpszActivateDefault,
                                                    OLEUI_CCHLABELMAX,
                                                          NULL);

           if (0 == dw)
              lstrcpy((LPSTR)lpCV->lpszActivateDefault, (LPSTR)lpCV-
>lpszCurrentObject);
     }
     else
           lstrcpy((LPSTR)(lpCV->lpszActivateDefault), (LPSTR)lpCV-
>lpszCurrentObject);


SelectStringInListbox:

     if (lpCV->dwFlags & CF_SELECTCONVERTTO)
        lRet = SendMessage(lpCV->hListVisible, LB_SELECTSTRING, (WPARAM)-1,
(LPARAM)(LPSTR)(lpCV->lpszConvertDefault));

     else
        lRet = SendMessage(lpCV->hListVisible, LB_SELECTSTRING, (WPARAM)-1,
(LPARAM)(LPSTR)(lpCV->lpszActivateDefault));

     if (LB_ERR == lRet)
        SendMessage(lpCV->hListVisible, LB_SETCURSEL, (WPARAM)0, 0L);


     // Initialize icon stuff
     if (DVASPECT_ICON == lpCV->dvAspect )
     {
       SendDlgItemMessage(hDlg, IDCV_DISPLAYASICON, BM_SETCHECK, TRUE, 0L);

       if ((HGLOBAL)NULL != lpOCV->hMetaPict)
       {
            char  szLabel[OLEUI_CCHLABELMAX];
            HICON hIcon;
            DWORD dwWrapIndex;


            // Set the icon to the icon from the hMetaPict,
            // set the label to the label from the hMetaPict.

            if (0 != OleUIMetafilePictExtractLabel(lpOCV->hMetaPict,
(LPSTR)szLabel, OLEUI_CCHLABELMAX, &dwWrapIndex))
            {
                  if (0 == dwWrapIndex)  // no second line
                  {
                       SendDlgItemMessage(hDlg, IDCV_ICONLABEL1, WM_SETTEXT,
0, (LPARAM)(LPSTR)szLabel);
                       SendDlgItemMessage(hDlg, IDCV_ICONLABEL2, WM_SETTEXT,
0, (LPARAM)(LPSTR)"");
                  }
                  else
                  {

                       LPSTR lpszSecondLine;
                       lpszSecondLine = szLabel + dwWrapIndex;
                       SendDlgItemMessage(hDlg, IDCV_ICONLABEL2,
                                                  WM_SETTEXT, 0,
(LPARAM)lpszSecondLine);

                       *lpszSecondLine = '\0';
                       SendDlgItemMessage(hDlg, IDCV_ICONLABEL1,
                                                  WM_SETTEXT, 0, (LPARAM)
(LPSTR)szLabel);
                  }


            }

            hIcon = OleUIMetafilePictExtractIcon(lpOCV->hMetaPict);

            if (NULL != hIcon)
            {
              SendDlgItemMessage(hDlg, IDCV_ICON, STM_SETICON,
(WPARAM)hIcon, 0L);
              lpCV->fCustomIcon = TRUE;
            }

            OleUIMetafilePictExtractIconSource(lpOCV->hMetaPict,
                                                                 (LPSTR)
(lpCV->lpszIconSource),
                                                                 &(lpCV-
>IconIndex));

       }
       else
            UpdateCVClassIcon(hDlg, lpCV, lpCV->hListVisible);
     }
     else
     {
       // Hide & disable icon stuff
       StandardShowDlgItem(hDlg, IDCV_ICON, SW_HIDE);
       StandardShowDlgItem(hDlg, IDCV_ICONLABEL1, SW_HIDE);
       StandardShowDlgItem(hDlg, IDCV_ICONLABEL2, SW_HIDE);
       StandardShowDlgItem(hDlg, IDCV_CHANGEICON, SW_HIDE);
     }

     // Call the hook with lCustData in lParam
     UStandardHook((LPVOID)lpCV, hDlg, WM_INITDIALOG, wParam, lpOCV-
>lCustData);
     // Update results window
     SetConvertResults(hDlg, lpCV);

     // Update caption if lpszCaption was specified
     if (lpCV->lpOCV->lpszCaption && !IsBadReadPtr(lpCV->lpOCV->lpszCaption,
1)
             && lpCV->lpOCV->lpszCaption[0] != '\0')
           SetWindowText(hDlg, (LPSTR)lpCV->lpOCV->lpszCaption);

     return TRUE;
     }


/*
 * FillClassList
 *
 * Purpose:
 *  Enumerates available OLE object classes from the registration
 *  database that we can convert or activate the specified clsid from.
 *
 *  Note that this function removes any prior contents of the listbox.
 *
 * Parameters:
 *  clsid           Class ID for class to find convert classes for
 *  hList           HWND to the listbox to fill.
 *  hListInvisible  HWND to invisible listbox that stores "activate as"
list.
 *  lpszCurrentClass LPSTR to put the (hr) class name of the clsid; we
 *                   do it here since we've already got the reg db open.
 *  fIsLinkedObject BOOL is the original object a linked object
 *  wFormat         WORD specifying the format of the original class.
 *  cClsidExclude   UINT number of entries in exclude list
 *  lpClsidExclude  LPCLSID array classes to exclude for list
 *
 * Return Value:
 *  UINT            Number of strings added to the listbox, -1 on failure.
 */
UINT FillClassList(
           CLSID clsid,
           HWND hList,
           HWND hListInvisible,
           LPSTR FAR *lplpszCurrentClass,
           BOOL fIsLinkedObject,
           WORD wFormat,
           UINT cClsidExclude,
           LPCLSID lpClsidExclude)
{

     DWORD       dw;
     UINT        cStrings=0;
     HKEY        hKey;
     LONG        lRet;
     char        szFormatKey[OLEUI_CCHKEYMAX];
     char        szClass[OLEUI_CCHKEYMAX];
     char        szFormat[OLEUI_CCHKEYMAX];
     char        szHRClassName[OLEUI_CCHKEYMAX];
     CLSID       clsidForList;

     LPSTR       lpszCLSID;


     //Clean out the existing strings.
     SendMessage(hList, LB_RESETCONTENT, 0, 0L);
     SendMessage(hListInvisible, LB_RESETCONTENT, 0, 0L);
     //Open up the root key.
     lRet=RegOpenKey(HKEY_CLASSES_ROOT, (LPSTR)"CLSID", (HKEY FAR *)&hKey);

     if ((LONG)ERROR_SUCCESS!=lRet)
           return (UINT)-1;

     if (NULL == *lplpszCurrentClass)
     {
        // alloc buffer here...

           LPMALLOC pIMalloc = NULL;
           HRESULT  hrErr;


           hrErr = CoGetMalloc(MEMCTX_TASK, &pIMalloc);

           if (hrErr != NOERROR)
           {
                 RegCloseKey(hKey);
                 return FALSE;
           }

           // Allocate space for lpszCurrentClass
           *lplpszCurrentClass = (LPSTR)pIMalloc->lpVtbl->Alloc(pIMalloc,
OLEUI_CCHKEYMAX);
            pIMalloc->lpVtbl->Release(pIMalloc);

           lRet = OleStdGetUserTypeOfClass((REFCLSID)&clsid,

     *lplpszCurrentClass,

     OLEUI_CCHLABELMAX,
                                                           NULL);

           if (0 ==lRet)
           {
                 int n = LoadString(ghInst, (UINT)IDS_PSUNKNOWNTYPE,
*lplpszCurrentClass,
                             OLEUI_CCHKEYMAX);
                 if (!n)
                 {
                       RegCloseKey(hKey);
                       return (UINT)-1;
                 }
           }
     }

     // Get the class name of the original class.
     StringFromCLSID((REFCLSID)&clsid, (LPSTR FAR *)&lpszCLSID);


     // Here, we step through the entire registration db looking for
     // class that can read or write the original class' format.  We
     // maintain two lists - an activate list and a convert list.  The
```

```
    // activate list is a subset of the convert list - activate ==
read/write
    // and convert == read. We swap the listboxes as needed with
    // SwapWindows, and keep track of which is which in the lpCV structure.

    // Every item has the following format:
    //
    //     Class Name\tclsid\0


    cStrings=0;
    lRet=RegEnumKey(hKey, cStrings++, (LPSTR)szClass, OLEUI_CCHKEYMAX);

    while ((LONG)ERROR_SUCCESS==lRet)
    {
        int j;
        BOOL fExclude=FALSE;


        //Check if this CLSID is in the exclusion list.
        CLSIDFromString((LPSTR)szClass, (LPCLSID)&clsidForList);
        for (j=0; j < (int)cClsidExclude; j++)
        {
            if (IsEqualCLSID(&clsidForList, (LPCLSID)
(lpClsidExclude+j)))
            {
                fExclude=TRUE;
                break;
            }
        }
        if (fExclude)
            goto Next;   // don't add this class to list

        // Check for a \Conversion\Readwritable\Main - if its
        // readwriteable, then the class can be added to the ActivateAs
        // list.
        // NOTE: the presence of this key should NOT automatically be
        //       used to add the class to the CONVERT list.

        lstrcpy((LPSTR)szFormatKey, (LPSTR)szClass);
        lstrcat((LPSTR)szFormatKey, (LPSTR)"\\Conversion\\Readwritable\
\Main");

        dw=OLEUI_CCHKEYMAX;

        lRet=RegQueryValue(hKey, (LPSTR)szFormatKey, (LPSTR)szFormat,
(LONG FAR *)&dw);

        if ( ((LONG)ERROR_SUCCESS==lRet)
             && (FormatIncluded((LPSTR)szFormat, wFormat)) )
        {
            // Here, we've got a list of formats that this class can
read
            // and write. We need to see if the original class' format
is
```

```
                 // in this list.  We do that by looking for wFormat in
                 // szFormat - if it in there, then we add this class to the
                 // ACTIVATEAS list only. we do NOT automatically add it to
the
                 // CONVERT list. Readable and Readwritable format lists
should
                 // be handled separately.

                 dw=OLEUI_CCHKEYMAX;
                 lRet=RegQueryValue(hKey, (LPSTR)szClass,
(LPSTR)szHRClassName, (LPDWORD)&dw);

                 if ((LONG)ERROR_SUCCESS==lRet)
                 {
                       lstrcat((LPSTR)szHRClassName, (LPSTR)"\t");

                       // only add if not already in list
                       if (LB_ERR==SendMessage(hList,LB_FINDSTRING, 0,
                                   (LPARAM)(LPSTR)szHRClassName)) {
                             lstrcat((LPSTR)szHRClassName, (LPSTR)szClass);
                             SendMessage(hList, LB_ADDSTRING, 0,
                                         (DWORD)(LPSTR)szHRClassName);
                       }
                 }

           }


           // Here we'll check to see if the original class' format is in
the
           // readable list. if so, we will add the class to the CONVERTLIST


           // We've got a special case for a linked object here.
           // If an object is linked, then the only class that
           // should appear in the convert list is the object's
           // class.  So, here we check to see if the object is
           // linked.  If it is, then we compare the classes.  If
           // they aren't the same, then we just go to the next key.

           if ( (!fIsLinkedObject)
                       || (lstrcmp((LPSTR)lpszCLSID, (LPSTR)szClass) == 0) )
           {

                 //Check for a \Conversion\Readable\Main entry
                 lstrcpy((LPSTR)szFormatKey, (LPSTR)szClass);
                 lstrcat((LPSTR)szFormatKey, (LPSTR)"\\Conversion\\Readable\
\Main");

                 dw=OLEUI_CCHKEYMAX;

                 // Check to see if this class can read the original class
                 // format.  If it can, add the string to the listbox as
                 // CONVERT_LIST.
                 lRet=RegQueryValue(hKey, (LPSTR)szFormatKey,
(LPSTR)szFormat, (LPDWORD)&dw);

                 if ( ((LONG)ERROR_SUCCESS==lRet)
                        && (FormatIncluded((LPSTR)szFormat, wFormat)) )
                 {


                       dw=OLEUI_CCHKEYMAX;
                       lRet=RegQueryValue(hKey, (LPSTR)szClass,
(LPSTR)szHRClassName, (LPDWORD)&dw);

                       if ((LONG)ERROR_SUCCESS==lRet)
                       {

                             lstrcat((LPSTR)szHRClassName, (LPSTR)"\t");

                             // only add if not already in list
                             if
(LB_ERR==SendMessage(hListInvisible,LB_FINDSTRING, 0,
                                         (LPARAM)(LPSTR)szHRClassName)) {
                                   lstrcat((LPSTR)szHRClassName,
(LPSTR)szClass);
                                   SendMessage(hListInvisible, LB_ADDSTRING,
0,
                                               (DWORD)(LPSTR)szHRClassName);
                             }
                       }  // end if

                 } // end if
           } // end else
Next:
           //Continue with the next key.
           lRet=RegEnumKey(hKey, cStrings++, (LPSTR)szClass,
OLEUI_CCHKEYMAX);

     }  // end while

     // If the original class isn't already in the list, add it.

     lstrcpy((LPSTR)szHRClassName, *lplpszCurrentClass);
     lstrcat((LPSTR)szHRClassName, (LPSTR)"\t");

     lRet = SendMessage(hList, LB_FINDSTRING, (WPARAM)-1, (LPARAM)
(LPSTR)szHRClassName);

     // only add it if it's not there already.
     if (LB_ERR == lRet) {
           lstrcat((LPSTR)szHRClassName, lpszCLSID);
           SendMessage(hList, LB_ADDSTRING, 0, (LPARAM)
(LPSTR)szHRClassName);
     }

     lRet = SendMessage(hListInvisible, LB_FINDSTRING, (WPARAM)-1, (LPARAM)
(LPSTR)szHRClassName);
```

```
     // only add it if it's not there already.
     if (LB_ERR == lRet)
           SendMessage(hListInvisible, LB_ADDSTRING, 0, (LPARAM)
(LPSTR)szHRClassName);

     // Free the string we got from StringFromCLSID.
     // OLE2NOTE:  StringFromCLSID uses your IMalloc to alloc a
     // string, so you need to be sure to free the string you
     // get back, otherwise you'll have leaky memory.

     OleStdFreeString(lpszCLSID, NULL);

     RegCloseKey(hKey);

     return cStrings;
}


/*
 * OleUICanConvertOrActivateAs
 *
 * Purpose:
 *  Determine if there is any OLE object class from the registration
 *  database that we can convert or activate the specified clsid from.
 *
 * Parameters:
 *  rClsid          REFCLSID Class ID for class to find convert classes for
 *  fIsLinkedObject BOOL is the original object a linked object
 *  wFormat         WORD specifying the format of the original class.
 *
 * Return Value:
 *  BOOL            TRUE if Convert command should be enabled, else FALSE
 */

STDAPI_(BOOL) OleUICanConvertOrActivateAs(
           REFCLSID    rClsid,
           BOOL        fIsLinkedObject,
           WORD        wFormat
)
{

     DWORD       dw;
     UINT        cStrings=0;
     HKEY        hKey;
     LONG        lRet;
     char        szFormatKey[OLEUI_CCHKEYMAX];
     char        szClass[OLEUI_CCHKEYMAX];
     char        szFormat[OLEUI_CCHKEYMAX];
     char        szHRClassName[OLEUI_CCHKEYMAX];
     BOOL        fEnableConvert = FALSE;

     LPSTR       lpszCLSID;

     //Open up the root key.
     lRet=RegOpenKey(HKEY_CLASSES_ROOT, (LPSTR)"CLSID", (HKEY FAR *)&hKey);

     if ((LONG)ERROR_SUCCESS!=lRet)
           return FALSE;

     // Get the class name of the original class.
     StringFromCLSID(rClsid, (LPSTR FAR *)&lpszCLSID);

     // Here, we step through the entire registration db looking for
     // class that can read or write the original class' format.
     // This loop stops if a single class is found.

     cStrings=0;
     lRet=RegEnumKey(hKey, cStrings++, (LPSTR)szClass, OLEUI_CCHKEYMAX);

     while ((LONG)ERROR_SUCCESS==lRet)
     {
           if (lstrcmp(lpszCLSID, szClass) == 0)
                 goto next;   // we don't want to consider the source class

           // Check for a \Conversion\ReadWriteable\Main entry first - if
its
           // readwriteable, then we don't need to bother checking to see if
           // its readable.

           lstrcpy((LPSTR)szFormatKey, (LPSTR)szClass);
           lstrcat((LPSTR)szFormatKey, (LPSTR)"\\Conversion\\Readwritable\
\Main");

           dw=OLEUI_CCHKEYMAX;

           lRet=RegQueryValue(hKey, (LPSTR)szFormatKey, (LPSTR)szFormat,
(LONG FAR *)&dw);

           if ( (LONG)ERROR_SUCCESS != lRet)
           {
             // Try \\DataFormats\DefaultFile too

             lstrcpy((LPSTR)szFormatKey, (LPSTR)szClass);
             lstrcat((LPSTR)szFormatKey, (LPSTR)"\\DataFormats\
\DefaultFile");

             dw=OLEUI_CCHKEYMAX;

             lRet=RegQueryValue(hKey, (LPSTR)szFormatKey, (LPSTR)szFormat,
(LONG FAR *)&dw);
           }


           if ( ((LONG)ERROR_SUCCESS==lRet)
                  && (FormatIncluded((LPSTR)szFormat, wFormat)) )
           {

                 // Here, we've got a list of formats that this class can
read
```

```
                 // and write. We need to see if the original class' format
is
                 // in this list.  We do that by looking for wFormat in
                 // szFormat - if it in there, then we add this class to the
                 // both lists and continue.  If not, then we look at the
                 // class' readable formats.


                 dw=OLEUI_CCHKEYMAX;
                 lRet=RegQueryValue(hKey, (LPSTR)szClass,
(LPSTR)szHRClassName, (LPDWORD)&dw);

                 if ((LONG)ERROR_SUCCESS==lRet)
                 {
                       fEnableConvert = TRUE;
                       break;  // STOP -- found one!
                 }

           }


           // We either didn't find the readwritable key, or the
           // list of readwritable formats didn't include the
           // original class format.  So, here we'll check to
           // see if its in the readable list.


           // We've got a special case for a linked object here.
           // If an object is linked, then the only class that
           // should appear in the convert list is the object's
           // class.  So, here we check to see if the object is
           // linked.  If it is, then we compare the classes.  If
           // they aren't the same, then we just go to the next key.

           else if ( (!fIsLinkedObject)
                       || (lstrcmp((LPSTR)lpszCLSID, (LPSTR)szClass) == 0) )
           {

                 //Check for a \Conversion\Readable\Main entry
                 lstrcpy((LPSTR)szFormatKey, (LPSTR)szClass);
                 lstrcat((LPSTR)szFormatKey, (LPSTR)"\\Conversion\\Readable\
\Main");

                 dw=OLEUI_CCHKEYMAX;

                 // Check to see if this class can read the original class
                 // format.  If it can, add the string to the listbox as
                 // CONVERT_LIST.

                 lRet=RegQueryValue(hKey, (LPSTR)szFormatKey,
(LPSTR)szFormat, (LPDWORD)&dw);

                 if ( ((LONG)ERROR_SUCCESS==lRet)
                        && (FormatIncluded((LPSTR)szFormat, wFormat)) )
                 {
                       dw=OLEUI_CCHKEYMAX;
                       lRet=RegQueryValue(hKey, (LPSTR)szClass,
(LPSTR)szHRClassName, (LPDWORD)&dw);

                       if ((LONG)ERROR_SUCCESS==lRet)
                       {

                             fEnableConvert = TRUE;
                             break;  // STOP -- found one!
                       }  // end if

                 } // end if
           } // end else
next:
           //Continue with the next key.
           lRet=RegEnumKey(hKey, cStrings++, (LPSTR)szClass,
OLEUI_CCHKEYMAX);

     }  // end while

     // Free the string we got from StringFromCLSID.
     // OLE2NOTE:  StringFromCLSID uses your IMalloc to alloc a
     // string, so you need to be sure to free the string you
     // get back, otherwise you'll have leaky memory.

     OleStdFreeString(lpszCLSID, NULL);

     RegCloseKey(hKey);

     return fEnableConvert;
}


/*
 * FormatIncluded
 *
 * Purpose:
 *  Parses the string for format from the word.
 *
 * Parameters:
 *  szStringToSearch  String to parse
 *  wFormat           format to find
 *
 * Return Value:
 *  BOOL        TRUE if format is found in string,
 *              FALSE otherwise.
 */
BOOL FormatIncluded(LPSTR szStringToSearch, UINT wFormat)
{

   LPSTR        lpToken;
   char         seps[] = ",";
   static char  szFormat[255];  // max size of atom (what GetClipboardName
returns)


   if (wFormat < 0xC000)              // RegisterClipboardFormat returns
values
        _itoa(wFormat, szFormat, 10);  // between 0xC000 and 0xFFFF.

   else
        GetClipboardFormatName(wFormat, szFormat, 255);


   lpToken = (LPSTR)_fstrtok(szStringToSearch, seps);

   while (lpToken != NULL)
   {

      if (0 == lstrcmpi(lpToken, szFormat))
           return TRUE;

      else
           lpToken = (LPSTR)_fstrtok(NULL, seps);
   }

   return FALSE;
}


/*
 * UpdateCVClassIcon
 *
 * Purpose:
 *  Handles LBN_SELCHANGE for the Object Type listbox.  On a selection
 *  change, we extract an icon from the server handling the currently
 *  selected object type using the utility function HIconFromClass.
 *  Note that we depend on the behavior of FillClassList to stuff the
 *  object class after a tab in the listbox string that we hide from
 *  view (see WM_INITDIALOG).
 *
 * Parameters
 *  hDlg            HWND of the dialog box.
 *  lpCV            LPCONVERT pointing to the dialog structure
 *  hList           HWND of the Object Type listbox.
 *
 * Return Value:
 *  None
 */

void UpdateCVClassIcon(HWND hDlg, LPCONVERT lpCV, HWND hList)
     {
     UINT        iSel;
     DWORD       cb;
     HGLOBAL     hMem;
     LPSTR       pszName, pszCLSID;
     CLSID       clsid;
     HICON       hIcon, hOldIcon;
     UINT        cch, uWrapIndex;
     RECT        LabelRect;
     char        szLabel[OLEUI_CCHLABELMAX];
     LPSTR       lpszLabel = szLabel;
     HFONT       hFont;
     HWND        hLabel1;

     /*
      * When we change object type selection, get the new icon for that
      * type into our structure and update it in the display.
      */

     iSel=(UINT)SendMessage(hList, LB_GETCURSEL, 0, 0L);

     if (LB_ERR==(int)iSel)
           return;

     //Allocate a string to hold the entire listbox string
     cb=SendMessage(hList, LB_GETTEXTLEN, iSel, 0L);

     hMem=GlobalAlloc(GHND, cb+1);

     if (NULL==hMem)
           return;

     pszName=GlobalLock(hMem);

     // Get whole string
     SendMessage(hList, LB_GETTEXT, iSel, (LONG)pszName);

     // Set pointer to CLSID (string)
     pszCLSID = PointerToNthField(pszName, 2, '\t');

     //Create the class ID with this string.
     CLSIDFromString((LPSTR)pszCLSID, (LPCLSID)&clsid);

     hIcon = HIconAndSourceFromClass(&clsid, (LPSTR)(lpCV->lpszIconSource),
&(lpCV->IconIndex));

     if (NULL == hIcon)  // Use Vanilla Document
     {
           lstrcpy((LPSTR)(lpCV->lpszIconSource), (LPSTR)szOLE2DLL);
           lpCV->IconIndex = 0;    // 1st icon in OLE2.DLL
           hIcon = ExtractIcon(ghInst,
                                     (LPSTR)(lpCV->lpszIconSource),
                                     lpCV->IconIndex);
     }

     //Replace the current display with this new one.
     hOldIcon = (HICON)SendDlgItemMessage(hDlg, IDCV_ICON, STM_SETICON,
(WPARAM)hIcon, 0L);

     hLabel1 = GetDlgItem(hDlg, IDCV_ICONLABEL1);
     GetWindowRect(hLabel1, &LabelRect);

     // Get the label
     if (lpCV->lpOCV->lpszDefLabel) {
           // width is used as 1.5 times width of icon window
           lpszLabel = ChopText(hLabel1, ((LabelRect.right-
LabelRect.left)*3)/2, (LPSTR)lpCV->lpOCV->lpszDefLabel);
           LSTRCPYN(szLabel, lpCV->lpOCV->lpszDefLabel, sizeof(szLabel));
     } else {
           if ((cch = OleStdGetAuxUserType(&clsid, AUXUSERTYPE_SHORTNAME,
                       (LPSTR)szLabel, OLEUI_CCHLABELMAX, NULL)) == 0) {
                 // If we can't get the AuxUserType2, then try the long name
                 if ((cch = OleStdGetUserTypeOfClass(&clsid, (LPSTR)szLabel,
                             sizeof(szLabel), NULL)) == 0) {
                       // last resort; use "Document" as label
                       LoadString(ghInst,(UINT)IDS_DEFICONLABEL,
(LPSTR)szLabel,sizeof(szLabel));
                       cch = lstrlen((LPSTR)szLabel);
                 }
           }
     }

     hFont = (HFONT)SendMessage(hLabel1, WM_GETFONT, 0, 0L);

     // Figure out where to split the label
     uWrapIndex = OleStdIconLabelTextOut(NULL, hFont, 0, 0, 0, &LabelRect,
(LPSTR)lpszLabel, cch, NULL);

     if (0 == uWrapIndex)
     {
        SendMessage(hLabel1, WM_SETTEXT, 0, (LPARAM)(LPSTR)lpszLabel);
        SendDlgItemMessage(hDlg, IDCV_ICONLABEL2, WM_SETTEXT, 0, (LPARAM)
(LPSTR)"");
     }
     else
     {
        char  chKeep;
        LPSTR lpszSecondLine;

        chKeep = szLabel[uWrapIndex];
        lpszLabel[uWrapIndex] = '\0';

        SendMessage(hLabel1, WM_SETTEXT, 0, (LPARAM)(LPSTR)lpszLabel);

        lpszLabel[uWrapIndex] = chKeep;
        lpszSecondLine = lpszLabel + uWrapIndex;

        SendDlgItemMessage(hDlg, IDCV_ICONLABEL2, WM_SETTEXT, 0,
(LPARAM)lpszSecondLine);
     }

     // get rid of the old icon
     if ((HICON)NULL != hOldIcon)
       DestroyIcon(hOldIcon);
     GlobalUnlock(hMem);
     GlobalFree(hMem);
     return;
     }





BOOL IsValidClassID(CLSID cID)
{
     if (0 == _fmemcmp(&cID, &CLSID_NULL, sizeof(CLSID)))  // if (CLSID_NULL
== cID)
       return FALSE;
     else
       return TRUE;
}



/*
 * SetConvertResults
 *
 * Purpose:
 *  Centralizes setting of the Result display in the Convert
 *  dialog.  Handles loading the appropriate string from the module's
 *  resources and setting the text, displaying the proper result picture,
 *  and showing the proper icon.
 *
 * Parameters:
 *  hDlg            HWND of the dialog box so we can access controls.
 *  lpCV            LPCONVERT in which we assume that the dwFlags is
 *                  set to the appropriate radio button selection, and
 *                  the list box has the appropriate class selected.
 *
 * Return Value:
 *  None
 */

void SetConvertResults(HWND hDlg, LPCONVERT lpCV)
   {
     LPSTR       pszT,        // temp
                       lpszOutput,  // text sent in SetDlgItemText
                       lpszDefObj,  // string containing default object
class
                       lpszSelObj,  // string containing selected object
class
                       lpszString;  // stirng we get from loadstring

     UINT        i, cch;
     HGLOBAL     hMem;

     HWND        hList;  // handle to listbox (so we can just use SendMsg i
                                   // instead of SendDlgItemMsg).
     hList = lpCV->hListVisible;
     /*
      * We need scratch memory for loading the stringtable string, loading
      * the object type from the listbox, loading the source object
      * type, and constructing the final string.  We therefore allocate
      * four buffers as large as the maximum message length (512) plus
      * the object type, guaranteeing that we have enough
      * in all cases.
      */
     i=(UINT)SendMessage(hList, LB_GETCURSEL, 0, 0L);

     cch=512+(UINT)SendMessage(hList, LB_GETTEXTLEN, i, 0L);

     hMem=GlobalAlloc(GHND, (DWORD)(4*cch));

     if (NULL==hMem)
           return;

     lpszOutput = (LPSTR)GlobalLock(hMem);
     lpszSelObj = lpszOutput + cch;
     lpszDefObj = lpszSelObj + cch;
     lpszString = lpszDefObj + cch;

     // Get selected object and null terminate human-readable name (1st
field).
     SendMessage(hList, LB_GETTEXT, i, (LONG)lpszSelObj);

     pszT = PointerToNthField(lpszSelObj, 2, '\t');

     pszT = AnsiPrev(lpszSelObj, pszT);

     *pszT = '\0';


     // Get default object

     GetDlgItemText(hDlg, IDCV_OBJECTTYPE, lpszDefObj, 512);


     //Default is an empty string.
     *lpszOutput=0;


     if (lpCV->dwFlags & CF_SELECTCONVERTTO)
     {

           if (lpCV->lpOCV->fIsLinkedObject)  // working with linked object
             LoadString(ghInst, (UINT)IDS_CVRESULTCONVERTLINK, lpszOutput,
cch);


           else
           {
                 // converting to a new class
             if (0 != lstrcmp(lpszDefObj, lpszSelObj))
             {
                 if (0 != LoadString(ghInst, (UINT)IDS_CVRESULTCONVERTTO,
lpszString, cch))
                    wsprintf(lpszOutput, lpszString, lpszDefObj,
lpszSelObj);

             }
             else  // converting to the same class (no conversion)
             {

                  if (0 != LoadString(ghInst, (UINT)IDS_CVRESULTNOCHANGE,
lpszString, cch))
                    wsprintf(lpszOutput, lpszString, lpszDefObj);
             }

           }

           if (lpCV->dvAspect == DVASPECT_ICON)  // Display as icon is
checked
           {
              if (0 != LoadString(ghInst, (UINT)IDS_CVRESULTDISPLAYASICON,
lpszString, cch))
                       lstrcat(lpszOutput, lpszString);
           }
     }

     if (lpCV->dwFlags & CF_SELECTACTIVATEAS)
     {

        if (0!=LoadString(ghInst, (UINT)IDS_CVRESULTACTIVATEAS, lpszString,
cch))
             wsprintf(lpszOutput, lpszString, lpszDefObj, lpszSelObj);

        // activating as a new class
        if (0 != lstrcmp(lpszDefObj, lpszSelObj))
        {
             if (0!=LoadString(ghInst, (UINT)IDS_CVRESULTACTIVATEDIFF,
lpszString, cch))
                  lstrcat(lpszOutput, lpszString);
        }
        else // activating as itself.
        {
            lstrcat(lpszOutput, ".");
        }
     }


     //If LoadString failed, we simply clear out the results (*lpszOutput=0
above)
     SetDlgItemText(hDlg, IDCV_RESULTTEXT, lpszOutput);

     GlobalUnlock(hMem);
     GlobalFree(hMem);
     return;
     }






/*
 * ConvertCleanup
 *
 * Purpose:
 *  Performs convert-specific cleanup before Convert termination.
 *
 * Parameters:
 *  hDlg            HWND of the dialog box so we can access controls.
 *
 * Return Value:
 *  None
 */
void ConvertCleanup(HWND hDlg, LPCONVERT lpCV)
{

   LPMALLOC pIMalloc;


   // Free our strings. Zero out the user type name string
   // the the calling app doesn't free to it.

   if (NOERROR == CoGetMalloc(MEMCTX_TASK, &pIMalloc))
   {
        pIMalloc->lpVtbl->Free(pIMalloc, (LPVOID)lpCV->lpszConvertDefault);
        pIMalloc->lpVtbl->Free(pIMalloc, (LPVOID)lpCV->lpszActivateDefault);
        pIMalloc->lpVtbl->Free(pIMalloc, (LPVOID)lpCV->lpszIconSource);
        if (lpCV->lpOCV->lpszUserType) {
              pIMalloc->lpVtbl->Free(pIMalloc,(LPVOID)lpCV->lpOCV-
>lpszUserType);
              lpCV->lpOCV->lpszUserType = NULL;
        }
        if (lpCV->lpOCV->lpszDefLabel) {
              pIMalloc->lpVtbl->Free(pIMalloc,(LPVOID)lpCV->lpOCV-
>lpszDefLabel);
              lpCV->lpOCV->lpszDefLabel = NULL;
        }

        pIMalloc->lpVtbl->Release(pIMalloc);
   }

   return;
}
```

```
/*
 * SwapWindows
 *
 * Purpose:
 *  Moves hWnd1 to hWnd2's position and hWnd2 to hWnd1's position.
 *  Does NOT change sizes.
 *
 *
 * Parameters:
 *  hDlg            HWND of the dialog box so we can turn redraw off
 *
 * Return Value:
 *  None
 */
void SwapWindows(HWND hDlg, HWND hWnd1, HWND hWnd2)
{

   RECT Rect1, Rect2;


   GetWindowRect(hWnd1, &Rect1);
   GetWindowRect(hWnd2, &Rect2);

   ScreenToClient(hDlg, (LPPOINT)&Rect1.left);
   ScreenToClient(hDlg, (LPPOINT)&Rect1.right);

   ScreenToClient(hDlg, (LPPOINT)&Rect2.left);
   ScreenToClient(hDlg, (LPPOINT)&Rect2.right);

   SetWindowPos(hWnd1,
                    NULL,
                    Rect2.left,
                    Rect2.top,
                    0,
                    0,
                    SWP_NOZORDER | SWP_NOSIZE);

   SetWindowPos(hWnd2,
                    NULL,
                    Rect1.left,
                    Rect1.top,
                    0,
                    0,
                    SWP_NOZORDER | SWP_NOSIZE);

   return;

}
```

DBALLOC.H   (WRAPUI Sample)

```
/***
*dballoc.h
*
*  Copyright (C) 1992-93, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*  This file contains the definition of CDbAlloc - A debug implementation
*  of the IMalloc interface.
*
*Implementation Notes:
*
*****************************************************************************
*/

#ifndef DBALLOC_H_INCLUDED /* { */
#define DBALLOC_H_INCLUDED


interface IDbOutput : public IUnknown
{
     STDMETHOD(QueryInterface)(THIS_ REFIID riid, void FAR* FAR* ppv) PURE;
     STDMETHOD_(ULONG, AddRef)(THIS) PURE;
     STDMETHOD_(ULONG, Release)(THIS) PURE;

     STDMETHOD_(void, Printf)(THIS_
       char FAR* szFmt, ...) PURE;

     STDMETHOD_(void, Assertion)(THIS_
       BOOL cond,
       char FAR* szExpr,
       char FAR* szFile,
       UINT uLine,
       char FAR* szMsg) PURE;
};


#endif /* } DBALLOC_H_INCLUDED */
```

DBALLOC.CPP   (WRAPUI Sample)

```
/***
*dballoc.cpp
*
*  Copyright (C) 1992-93, Microsoft Corporation.  All Rights Reserved.
*
*Purpose:
*  This file contains a debug implementation of the IMalloc interface.
*
*  This implementation is basically a simple wrapping of the C runtime,
*  with additional work to detect memory leakage, and memory overwrite.
*
*  Leakage is detected by tracking each allocation in an address
*  instance table, and then checking to see if the table is empty
*  when the last reference to the allocator is released.
*
*  Memory overwrite is detected by placing a signature at the end
*  of every allocated block, and checking to make sure the signature
*  is unchanged when the block is freed.
*
*  This implementation also has additional param validation code, as
*  well as additional check make sure that instances that are passed
*  to Free() were actually allocated by the corresponding instance
*  of the allocator.
*
*
*  Creating an instance of this debug allocator that uses the default
*  output interface would look like the following,
*
*
*  BOOL init_application_instance()
*  {
*    HRESULT hresult;
*    IMalloc FAR* pmalloc;
*
*    pmalloc = NULL;
*
*    if((hresult = OleStdCreateDbAlloc(0,&pmalloc))!=NOERROR)
*      goto LReturn;
*
*    hresult = OleInitialize(pmalloc);
*
*    // release pmalloc to let OLE hold the only ref to the it. later
*    // when OleUnitialize is called, memory leaks will be reported.
*    if(pmalloc != NULL)
*      pmalloc->Release();
*
*  LReturn:
*
*    return (hresult == NOERROR) ? TRUE : FALSE;
*  }
*
*
```

```
*  CONSIDER: could add an option to force error generation, something
*   like DBALLOC_ERRORGEN
*
*  CONSIDER: add support for heap-checking. say for example,
*   DBALLOC_HEAPCHECK would do a heapcheck every free? every 'n'
*   calls to free? ...
*
*
*Implementation Notes:
*
*  The method IMalloc::DidAlloc() is allowed to always return
*  "Dont Know" (-1).  This method is called by Ole, and they take
*  some appropriate action when they get this answer.
*
****************************************************************************
*/


// Note: this file is designed to be stand-alone; it includes a
// carefully chosen, minimal set of headers.
//
// For conditional compilation we use the ole2 conventions,
//    _MAC      = mac
//    WIN32     = Win32 (NT really)
//    <nothing> = defaults to Win16


// REVIEW: the following needs to modified to handle _MAC
#define STRICT
#include <windows.h>

#include <ole2.h>

#if defined( __TURBOC__)
#define __STDC__ (1)
#endif

#define WINDLL  1           // make far pointer version of stdargs.h
#include <stdarg.h>

#if defined( __TURBOC__)
#undef __STDC__
#endif

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <limits.h>

#include "dballoc.h"

extern "C" DWORD g_dwObjectCount; // since we don't include ole2ui.h

#define DIM(X) (sizeof(X)/sizeof((X)[0]))
```

```
#define UNREACHED 0

#if defined(WIN32)
# define MEMCMP(PV1, PV2, CB)   memcmp((PV1), (PV2), (CB))
# define MEMCPY(PV1, PV2, CB)   memcpy((PV1), (PV2), (CB))
# define MEMSET(PV,  VAL, CB)   memset((PV),  (VAL), (CB))
# define MALLOC(CB)     malloc(CB)
# define REALLOC(PV, CB)    realloc((PV), (CB))
# define FREE(PV)       free(PV)
# define HEAPMIN()
#elif defined(_MAC)
# define MEMCMP(PV1, PV2)   ERROR -- NYI
# define MEMCPY(PV1, PV2, CB)   ERROR -- NYI
# define MEMSET(PV,  VAL, CB)   ERROR -- NYI
# define MALLOC(CB)     ERROR -- NYI
# define REALLOC(PV, CB)    ERROR -- NYI
# define FREE(PV)       ERROR -- NYI
# define HEAPMIN()      ERROR -- NYI
#else
# define MEMCMP(PV1, PV2, CB)   _fmemcmp((PV1), (PV2), (CB))
# define MEMCPY(PV1, PV2, CB)   _fmemcpy((PV1), (PV2), (CB))
# define MEMSET(PV,  VAL, CB)   _fmemset((PV),  (VAL), (CB))
# define MALLOC(CB)     _fmalloc(CB)
# define REALLOC(PV, CB)    _frealloc(PV, CB)
# define FREE(PV)       _ffree(PV)
# define HEAPMIN()      _fheapmin()
#endif

/*************************************************************************
** DEBUG ASSERTION ROUTINES
*************************************************************************/

#ifdef DBG
#include <assert.h>
#define FnAssert(lpstrExpr, lpstrMsg, lpstrFileName, iLine)      \
        (_assert(lpstrMsg ? lpstrMsg : lpstrExpr,                \
                 lpstrFileName,                                  \
                 iLine), NOERROR)
#endif //DBG

#if defined( __TURBOC__ )
#define classmodel _huge
#else
#define classmodel FAR
#endif

class classmodel CStdDbOutput : public IDbOutput {
public:
     static IDbOutput FAR* Create();

     // IUnknown methods

     STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppv);
     STDMETHOD_(ULONG, AddRef)(void);
     STDMETHOD_(ULONG, Release)(void);
     // IDbOutput methods

     STDMETHOD_(void, Printf)(char FAR* szFmt, ...);

     STDMETHOD_(void, Assertion)(
       BOOL cond,
       char FAR* szExpr,
       char FAR* szFile,
       UINT uLine,
       char FAR* szMsg);


     void FAR* operator new(size_t cb){
       return MALLOC(cb);
     }
     void operator delete(void FAR* pv){
       FREE(pv);
     }

     CStdDbOutput(){
       g_dwObjectCount++ ;
       m_refs = 0;
     }

     ~CStdDbOutput() { g_dwObjectCount-- ; }


private:
     ULONG m_refs;

     char m_rgch[128]; // buffer for output formatting
};


//---------------------------------------------------------------------
//                implementation of the debug allocator
//---------------------------------------------------------------------

class FAR CAddrNode
{
public:
     void FAR*      m_pv;    // instance
     ULONG      m_cb;    // size of allocation in BYTES
     ULONG          m_nAlloc;    // the allocation pass count
     CAddrNode FAR* m_next;

     void FAR* operator new(size_t cb){
       return MALLOC(cb);
     }
     void operator delete(void FAR* pv){
       FREE(pv);
     }
};
```

```
class classmodel CDbAlloc : public IMalloc
{
public:
     static HRESULT Create(
       ULONG options, IDbOutput FAR* pdbout, IMalloc FAR* FAR* ppmalloc);

     // IUnknown methods

     STDMETHOD(QueryInterface)(REFIID riid, void FAR* FAR* ppv);
     STDMETHOD_(ULONG, AddRef)(void);
     STDMETHOD_(ULONG, Release)(void);

     // IMalloc methods

     STDMETHOD_(void FAR*, Alloc)(ULONG cb);
     STDMETHOD_(void FAR*, Realloc)(void FAR* pv, ULONG cb);
     STDMETHOD_(void, Free)(void FAR* pv);
     STDMETHOD_(ULONG, GetSize)(void FAR* pv);
     STDMETHOD_(int, DidAlloc)(void FAR* pv);
     STDMETHOD_(void, HeapMinimize)(void);


     void FAR* operator new(size_t cb){
       return MALLOC(cb);
     }
     void operator delete(void FAR* pv){
       FREE(pv);
     }

     CDbAlloc(){
       m_refs = 1;
       m_pdbout = NULL;
       m_cAllocCalls = 0;
       m_nBreakAtNthAlloc = 0;
       m_nBreakAtAllocSize = 0;
       MEMSET(m_rganode, 0, sizeof(m_rganode));

       g_dwObjectCount++ ;

     }

     ~CDbAlloc() {   g_dwObjectCount-- ; }


private:

     ULONG m_refs;
     ULONG m_cAllocCalls;        // total count of allocation calls
     ULONG m_nBreakAtNthAlloc;   // allocation number to break to debugger
                                            //  this value should be set
typically in the
                                            //  debugger.
     ULONG m_nBreakAtAllocSize;  // allocation size to break to debugger
```

```
                                                //  this value should be set
typically in the
                                                //  debugger.
     IDbOutput FAR* m_pdbout;        // output interface
     CAddrNode FAR* m_rganode[64];   // address instance table


     // instance table methods

     BOOL IsEmpty(void);

     void AddInst(void FAR* pv, ULONG nAlloc, ULONG cb);
     void DelInst(void FAR* pv);
     CAddrNode FAR* GetInst(void FAR* pv);

     void DumpInst(CAddrNode FAR* pn);
     void DumpInstTable(void);

     inline UINT HashInst(void FAR* pv) const {
       return ((UINT)((ULONG)pv >> 4)) % DIM(m_rganode);
     }


     // output method(s)

     inline void Assertion(
       BOOL cond,
       char FAR* szExpr,
       char FAR* szFile,
       UINT uLine,
       char FAR* szMsg)
     {
       m_pdbout->Assertion(cond, szExpr, szFile, uLine, szMsg);
     }

     #define ASSERT(X) Assertion(X, #X, __FILE__, __LINE__, NULL)

     #define ASSERTSZ(X, SZ) Assertion(X, #X, __FILE__, __LINE__, SZ)

     static const unsigned char m_rgchSig[4];
};


const unsigned char CDbAlloc::m_rgchSig[] = { 0xDE, 0xAD, 0xBE, 0xEF };


/***
*HRESULT OleStdCreateDbAlloc(ULONG reserved, IMalloc** ppmalloc)
* Purpose:
*  Create an instance of CDbAlloc -- a debug implementation
*  of IMalloc.
*
* Parameters:
*   ULONG reserved               - reserved for future use. must be 0.
*   IMalloc FAR* FAR* ppmalloc  - (OUT) pointer to an IMalloc interface
*                                    of new debug allocator object
* Returns:
*   HRESULT
*       NOERROR          - if no error.
*       E_OUTOFMEMORY    - allocation failed.
*
**********************************************************************/
STDAPI OleStdCreateDbAlloc(ULONG reserved,IMalloc FAR* FAR* ppmalloc)
{
     return CDbAlloc::Create(reserved, NULL, ppmalloc);
}


HRESULT
CDbAlloc::Create(
     ULONG options,
     IDbOutput FAR* pdbout,
     IMalloc FAR* FAR* ppmalloc)
{
     HRESULT hresult;
     CDbAlloc FAR* pmalloc;


     // default the instance of IDbOutput if the user didn't supply one
     if(pdbout == NULL && ((pdbout = CStdDbOutput::Create()) == NULL)){
       hresult = ResultFromScode(E_OUTOFMEMORY);
       goto LError0;
     }

     pdbout->AddRef();

     if((pmalloc = new FAR CDbAlloc()) == NULL){
       hresult = ResultFromScode(E_OUTOFMEMORY);
       goto LError1;
     }

     pmalloc->m_pdbout = pdbout;

     *ppmalloc = pmalloc;

     return NOERROR;

LError1:;
     pdbout->Release();

LError0:;
     return hresult;
}

STDMETHODIMP
CDbAlloc::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
     if(riid == IID_IUnknown || riid == IID_IMalloc){
       *ppv = this;
       AddRef();
       return NOERROR;
     }
     return ResultFromScode(E_NOINTERFACE);
}

STDMETHODIMP_(ULONG)
CDbAlloc::AddRef()
{
     return ++m_refs;
}

STDMETHODIMP_(ULONG)
CDbAlloc::Release()
{
     if(--m_refs == 0){

       // check for memory leakage
       if(IsEmpty()){
             m_pdbout->Printf("No Memory Leaks.\n");
       }else{
             m_pdbout->Printf("Memory Leak Detected,\n");
             DumpInstTable();
       }

       m_pdbout->Release();
       delete this;
       return 0;
     }
     return m_refs;
}

STDMETHODIMP_(void FAR*)
CDbAlloc::Alloc(ULONG cb)
{
     size_t size;
     void FAR* pv;

     ++m_cAllocCalls;

     if (m_nBreakAtNthAlloc && m_cAllocCalls == m_nBreakAtNthAlloc) {
           ASSERTSZ(FALSE, "DBALLOC: NthAlloc Break target reached\r\n");
     } else if (m_nBreakAtAllocSize && cb == m_nBreakAtAllocSize) {
           ASSERTSZ(FALSE, "DBALLOC: AllocSize Break target reached\r\n");
     }

     // REVIEW: need to add support for huge allocations (on win16)
     if((cb + sizeof(m_rgchSig)) > UINT_MAX)
       return NULL;

     size = (size_t)cb;

     if((pv = MALLOC(size + sizeof(m_rgchSig))) == NULL)
       return NULL;

     // set allocated block to some non-zero value
     MEMSET(pv, -1, size);

     // put signature at end of allocated block
     MEMCPY((char FAR*)pv + size, m_rgchSig, sizeof(m_rgchSig));

     AddInst(pv, m_cAllocCalls, size);

     return pv;
}

STDMETHODIMP_(void FAR*)
CDbAlloc::Realloc(void FAR* pv, ULONG cb)
{
     size_t size;

     // REVIEW: need to add support for huge realloc
     if((cb + sizeof(m_rgchSig)) > UINT_MAX)
       return NULL;

     if(pv == NULL){
       return Alloc(cb);
     }

     ++m_cAllocCalls;

     ASSERT(GetInst(pv) != NULL);

     DelInst(pv);

     if(cb == 0){
       Free(pv);
       return NULL;
     }

     size = (size_t)cb;

     if((pv = REALLOC(pv, size + sizeof(m_rgchSig))) == NULL)
       return NULL;

     // put signature at end of allocated block
     MEMCPY((char FAR*)pv + size, m_rgchSig, sizeof(m_rgchSig));

     AddInst(pv, m_cAllocCalls, size);

     return pv;
}

STDMETHODIMP_(void)
CDbAlloc::Free(void FAR* pv)
{
     CAddrNode FAR* pn;
static char szSigMsg[] = "Signature Check Failed";

     if (pv == NULL) return;
     pn = GetInst(pv);

     // check for attempt to free an instance we didnt allocate
     if(pn == NULL){
       ASSERTSZ(FALSE, "pointer freed by wrong allocator");
       return;
     }

     // verify the signature
     if(MEMCMP((char FAR*)pv + pn->m_cb, m_rgchSig, sizeof(m_rgchSig)) != 0)
{
       m_pdbout->Printf(szSigMsg); m_pdbout->Printf("\n");
       DumpInst(GetInst(pv));
       ASSERTSZ(FALSE, szSigMsg);
     }

     // stomp on the contents of the block
     MEMSET(pv, 0xCC, (size_t)pn->m_cb + sizeof(m_rgchSig));

     DelInst(pv);

     FREE(pv);
}


STDMETHODIMP_(ULONG)
CDbAlloc::GetSize(void FAR* pv)
{
     CAddrNode FAR* pn;

     pn = GetInst(pv);

     if (pn == NULL) {
           ASSERT(pn != NULL);
           return 0;
     }

     return pn->m_cb;
}


/***
*PUBLIC HRESULT CDbAlloc::DidAlloc
*Purpose:
*  Answer if the given address belongs to a block allocated by
*  this allocator.
*
*Entry:
*  pv = the instance to lookup
*
*Exit:
*  return value = int
*    1 - did alloc
*    0 - did *not* alloc
*   -1 - dont know (according to the ole2 spec it is always legal
*        for the allocator to answer "dont know")
*
**********************************************************************/
STDMETHODIMP_(int)
CDbAlloc::DidAlloc(void FAR* pv)
{
     return -1; // answer "I dont know"
}


STDMETHODIMP_(void)
CDbAlloc::HeapMinimize()
{
     HEAPMIN();
}


//---------------------------------------------------------------------
//                      instance table methods
//---------------------------------------------------------------------

/***
*PRIVATE CDbAlloc::AddInst
*Purpose:
*  Add the given instance to the address instance table.
*
*Entry:
*  pv = the instance to add
*  nAlloc = the allocation passcount of this instance
*
*Exit:
*  None
*
**********************************************************************/
void
CDbAlloc::AddInst(void FAR* pv, ULONG nAlloc, ULONG cb)
{
     UINT hash;
     CAddrNode FAR* pn;


     ASSERT(pv != NULL);

     pn = (CAddrNode FAR*)new FAR CAddrNode();

     if (pn == NULL) {
           ASSERT(pn != NULL);
           return;
     }

     pn->m_pv = pv;
     pn->m_cb = cb;
     pn->m_nAlloc = nAlloc;

     hash = HashInst(pv);
     pn->m_next = m_rganode[hash];
     m_rganode[hash] = pn;
}


/***
*UNDONE
*Purpose:
*  Remove the given instance from the address instance table.
*
*Entry:
*  pv = the instance to remove
*
*Exit:
*  None
*
***********************************************************************/
void
CDbAlloc::DelInst(void FAR* pv)
{
     CAddrNode FAR* FAR* ppn, FAR* pnDead;

     for(ppn = &m_rganode[HashInst(pv)]; *ppn != NULL; ppn = &(*ppn)-
>m_next){
       if((*ppn)->m_pv == pv){
     pnDead = *ppn;
     *ppn = (*ppn)->m_next;
     delete pnDead;
     // make sure it doesnt somehow appear twice
     ASSERT(GetInst(pv) == NULL);
     return;
       }
     }

     // didnt find the instance
     ASSERT(UNREACHED);
}


CAddrNode FAR*
CDbAlloc::GetInst(void FAR* pv)
{
     CAddrNode FAR* pn;

     for(pn = m_rganode[HashInst(pv)]; pn != NULL; pn = pn->m_next){
       if(pn->m_pv == pv)
           return pn;
     }
     return NULL;
}


void
CDbAlloc::DumpInst(CAddrNode FAR* pn)
{
     if (pn == NULL)
           return;

     m_pdbout->Printf("[0x%lx]  nAlloc=%ld  size=%ld\n",
       pn->m_pv, pn->m_nAlloc, GetSize(pn->m_pv));
}


/***
*PRIVATE BOOL IsEmpty
*Purpose:
*  Answer if the address instance table is empty.
*
*Entry:
*  None
*
*Exit:
*  return value = BOOL, TRUE if empty, FALSE otherwise
*
***********************************************************************/
BOOL
CDbAlloc::IsEmpty()
{
     UINT u;

     for(u = 0; u < DIM(m_rganode); ++u){
       if(m_rganode[u] != NULL)
     return FALSE;
     }

     return TRUE;
}


/***
*PRIVATE CDbAlloc::Dump
*Purpose:
*  Print the current contents of the address instance table,
*
*Entry:
*  None
*
*Exit:
*  None
*
***********************************************************************/
void
CDbAlloc::DumpInstTable()
{
     UINT u;
     CAddrNode FAR* pn;

     for(u = 0; u < DIM(m_rganode); ++u){
       for(pn = m_rganode[u]; pn != NULL; pn = pn->m_next){
             DumpInst(pn);
       }
     }
}


//----------------------------------------------------------------------
//                implementation of CStdDbOutput
//----------------------------------------------------------------------

IDbOutput FAR*
CStdDbOutput::Create()
{
     return (IDbOutput FAR*)new FAR CStdDbOutput();
}

STDMETHODIMP
CStdDbOutput::QueryInterface(REFIID riid, void FAR* FAR* ppv)
{
     if(riid == IID_IUnknown){
       *ppv = this;
       AddRef();
       return NOERROR;
     }
     return ResultFromScode(E_NOINTERFACE);
}

STDMETHODIMP_(ULONG)
CStdDbOutput::AddRef()
{
     return ++m_refs;
}

STDMETHODIMP_(ULONG)
CStdDbOutput::Release()
{
     if(--m_refs == 0){
       delete this;
       return 0;
     }
     return m_refs;
}

STDMETHODIMP_(void)
CStdDbOutput::Printf(char FAR* lpszFmt, ...)
{
     va_list args;
     char szBuf[256];
#if defined( OBSOLETE )
     char *pn, FAR* pf;
static char rgchFmtBuf[128];
static char rgchOutputBuf[128];

     // copy the 'far' format string to a near buffer so we can use
     // a medium model vsprintf, which only supports near data pointers.
     //
     pn = rgchFmtBuf, pf=szFmt;
     while(*pf != '\0')
       *pn++ = *pf++;
     *pn = '\0';
#endif

     va_start(args, lpszFmt);

//    wvsprintf(rgchOutputBuf, rgchFmtBuf, args);
     wvsprintf(szBuf, lpszFmt, args);

     OutputDebugString(szBuf);
}

STDMETHODIMP_(void)
CStdDbOutput::Assertion(
     BOOL cond,
     char FAR* szExpr,
     char FAR* szFile,
     UINT uLine,
     char FAR* szMsg)
{
     if(cond)
       return;

#ifdef _DEBUG
     // following is from compobj.dll (ole2)
     FnAssert(szExpr, szMsg, szFile, uLine);
#else
     // REVIEW: should be able to do something better that this...
     DebugBreak();
#endif
}
```

DBGUTIL.C   (WRAPUI Sample)

```
/*************************************************************************
**
**    OLE 2.0 Common Utilities
**
**    dbgutil.h
**
**    This file contains file contains functions to support debug output.
**
**    (c) Copyright Microsoft Corp. 1990 - 1992 All Rights Reserved
**
*************************************************************************/

#define STRICT  1
#include "ole2ui.h"

static int s_nDbgIndent = 0;        // indent level for debug message
#if defined( _DEBUG )
static int s_nDbgLevel = 0;   // default dbg level printed
#else
static int s_nDbgLevel = 0;   // default dbg level printed
#endif

STDAPI_(void) OleDbgPrint(
                int     nDbgLvl,
                LPSTR   lpszPrefix,
                LPSTR   lpszMsg,
                int     nIndent
)
{
        if (nDbgLvl <= s_nDbgLevel)
                OleDbgPrintAlways(lpszPrefix, lpszMsg, nIndent);
}


STDAPI_(void) OleDbgPrintAlways(LPSTR lpszPrefix, LPSTR lpszMsg, int
nIndent)
{
        int i;

        if (nIndent < 0)
                OleDbgIndent(nIndent);

        if (lpszPrefix && *lpszPrefix != '\0') {
                OutputDebugString("| ");
                for (i = 0; i < s_nDbgIndent; i++)
                        OutputDebugString("----");

                OutputDebugString(lpszPrefix);
                OutputDebugString(": ");
        }

        OutputDebugString(lpszMsg);
        if (nIndent > 0)
                OleDbgIndent(nIndent);
}

STDAPI_(void) OleDbgSetDbgLevel(int nDbgLvl)
{
        s_nDbgLevel = nDbgLvl;
}

STDAPI_(int) OleDbgGetDbgLevel( void )
{
        return s_nDbgLevel;
}

STDAPI_(void) OleDbgIndent(int n)
{
        switch (n) {
                case -1:
                        s_nDbgIndent--;
                        break;
                case 1:
                        s_nDbgIndent++;
                        break;
                case -2:
                        s_nDbgIndent = 0;
                        break;
        }
}


STDAPI_(void) OleDbgPrintRefCnt(
                int         nDbgLvl,
                LPSTR       lpszPrefix,
                LPSTR       lpszMsg,
                LPVOID      lpObj,
                ULONG       refcnt
)
{
        if (nDbgLvl <= s_nDbgLevel)
                OleDbgPrintRefCntAlways(lpszPrefix, lpszMsg, lpObj, refcnt);
}


STDAPI_(void) OleDbgPrintRefCntAlways(
                LPSTR       lpszPrefix,
                LPSTR       lpszMsg,
                LPVOID      lpObj,
                ULONG       refcnt
)
{
#if defined( _DEBUG )
        char szBuf[256];

        wsprintf(szBuf, "[obj=(0x%lx) cnt=%ld] %s", lpObj, refcnt, lpszMsg);
        OleDbgPrintAlways(lpszPrefix, szBuf, 0);
```

```
#endif
}


STDAPI_(void) OleDbgPrintRect(
                int         nDbgLvl,
                LPSTR       lpszPrefix,
                LPSTR       lpszMsg,
                LPRECT      lpRect
)
{
        if (nDbgLvl <= s_nDbgLevel)
                OleDbgPrintRectAlways(lpszPrefix, lpszMsg, lpRect);
}


STDAPI_(void) OleDbgPrintRectAlways(
                LPSTR       lpszPrefix,
                LPSTR       lpszMsg,
                LPRECT      lpRect
)
{
#if defined( _DEBUG )
        char szBuf[256];

        wsprintf(
                        szBuf,
                        "%s: (%d,%d)-(%d,%d) %dx%d\r\n",
                        lpszMsg,
                        lpRect->left,
                        lpRect->top,
                        lpRect->right,
                        lpRect->bottom,
                        (lpRect->right-lpRect->left),
                        (lpRect->bottom-lpRect->top)
        );
        OleDbgPrintAlways(lpszPrefix, szBuf, 0);
#endif
}


#define CASE_SCODE(sc)  \
                case sc: \
                        lstrcpy((LPSTR)szErrName, (LPSTR)#sc); \
                        break;

STDAPI_(void) OleDbgPrintScodeAlways(LPSTR lpszPrefix, LPSTR lpszMsg, SCODE
sc)
{
#if defined( _DEBUG )
        char szBuf[256];
        char szErrName[40];

        switch (sc) {
```

```
/* SCODE's defined in SCODE.H */

CASE_SCODE(S_OK)
CASE_SCODE(S_FALSE)
CASE_SCODE(E_UNEXPECTED)
CASE_SCODE(E_OUTOFMEMORY)
CASE_SCODE(E_INVALIDARG)
CASE_SCODE(E_NOINTERFACE)
CASE_SCODE(E_POINTER)
CASE_SCODE(E_HANDLE)
CASE_SCODE(E_ABORT)
CASE_SCODE(E_FAIL)
CASE_SCODE(E_ACCESSDENIED)

/* SCODE's defined in OLE2.H */

CASE_SCODE(OLE_E_OLEVERB)
CASE_SCODE(OLE_E_ADVF)
CASE_SCODE(OLE_E_ENUM_NOMORE)
CASE_SCODE(OLE_E_ADVISENOTSUPPORTED)
CASE_SCODE(OLE_E_NOCONNECTION)
CASE_SCODE(OLE_E_NOTRUNNING)
CASE_SCODE(OLE_E_NOCACHE)
CASE_SCODE(OLE_E_BLANK)
CASE_SCODE(OLE_E_CLASSDIFF)
CASE_SCODE(OLE_E_CANT_GETMONIKER)
CASE_SCODE(OLE_E_CANT_BINDTOSOURCE)
CASE_SCODE(OLE_E_STATIC)
CASE_SCODE(OLE_E_PROMPTSAVECANCELLED)
CASE_SCODE(OLE_E_INVALIDRECT)
CASE_SCODE(OLE_E_WRONGCOMPOBJ)
CASE_SCODE(OLE_E_INVALIDHWND)
CASE_SCODE(OLE_E_NOT_INPLACEACTIVE)
CASE_SCODE(OLE_E_CANTCONVERT)
CASE_SCODE(OLE_E_NOSTORAGE)

CASE_SCODE(DV_E_FORMATETC)
CASE_SCODE(DV_E_DVTARGETDEVICE)
CASE_SCODE(DV_E_STGMEDIUM)
CASE_SCODE(DV_E_STATDATA)
CASE_SCODE(DV_E_LINDEX)
CASE_SCODE(DV_E_TYMED)
CASE_SCODE(DV_E_CLIPFORMAT)
CASE_SCODE(DV_E_DVASPECT)
CASE_SCODE(DV_E_DVTARGETDEVICE_SIZE)
CASE_SCODE(DV_E_NOIVIEWOBJECT)

CASE_SCODE(OLE_S_USEREG)
CASE_SCODE(OLE_S_STATIC)
CASE_SCODE(OLE_S_MAC_CLIPFORMAT)

CASE_SCODE(CONVERT10_E_OLESTREAM_GET)
CASE_SCODE(CONVERT10_E_OLESTREAM_PUT)
CASE_SCODE(CONVERT10_E_OLESTREAM_FMT)
CASE_SCODE(CONVERT10_E_OLESTREAM_BITMAP_TO_DIB)
CASE_SCODE(CONVERT10_E_STG_FMT)
CASE_SCODE(CONVERT10_E_STG_NO_STD_STREAM)
CASE_SCODE(CONVERT10_E_STG_DIB_TO_BITMAP)
CASE_SCODE(CONVERT10_S_NO_PRESENTATION)

CASE_SCODE(CLIPBRD_E_CANT_OPEN)
CASE_SCODE(CLIPBRD_E_CANT_EMPTY)
CASE_SCODE(CLIPBRD_E_CANT_SET)
CASE_SCODE(CLIPBRD_E_BAD_DATA)
CASE_SCODE(CLIPBRD_E_CANT_CLOSE)

CASE_SCODE(DRAGDROP_E_NOTREGISTERED)
CASE_SCODE(DRAGDROP_E_ALREADYREGISTERED)
CASE_SCODE(DRAGDROP_E_INVALIDHWND)
CASE_SCODE(DRAGDROP_S_DROP)
CASE_SCODE(DRAGDROP_S_CANCEL)
CASE_SCODE(DRAGDROP_S_USEDEFAULTCURSORS)

CASE_SCODE(OLEOBJ_E_NOVERBS)
CASE_SCODE(OLEOBJ_E_INVALIDVERB)
CASE_SCODE(OLEOBJ_S_INVALIDVERB)
CASE_SCODE(OLEOBJ_S_CANNOT_DOVERB_NOW)
CASE_SCODE(OLEOBJ_S_INVALIDHWND)
CASE_SCODE(INPLACE_E_NOTUNDOABLE)
CASE_SCODE(INPLACE_E_NOTOOLSPACE)
CASE_SCODE(INPLACE_S_TRUNCATED)

/* SCODE's defined in COMPOBJ.H */

CASE_SCODE(CO_E_NOTINITIALIZED)
CASE_SCODE(CO_E_ALREADYINITIALIZED)
CASE_SCODE(CO_E_CANTDETERMINECLASS)
CASE_SCODE(CO_E_CLASSSTRING)
CASE_SCODE(CO_E_IIDSTRING)
CASE_SCODE(CO_E_APPNOTFOUND)
CASE_SCODE(CO_E_APPSINGLEUSE)
CASE_SCODE(CO_E_ERRORINAPP)
CASE_SCODE(CO_E_DLLNOTFOUND)
CASE_SCODE(CO_E_ERRORINDLL)
CASE_SCODE(CO_E_WRONGOSFORAPP)
CASE_SCODE(CO_E_OBJNOTREG)
CASE_SCODE(CO_E_OBJISREG)
CASE_SCODE(CO_E_OBJNOTCONNECTED)
CASE_SCODE(CO_E_APPDIDNTREG)
CASE_SCODE(CLASS_E_NOAGGREGATION)
CASE_SCODE(CLASS_E_CLASSNOTAVAILABLE)
CASE_SCODE(REGDB_E_READREGDB)
CASE_SCODE(REGDB_E_WRITEREGDB)
CASE_SCODE(REGDB_E_KEYMISSING)
CASE_SCODE(REGDB_E_INVALIDVALUE)
CASE_SCODE(REGDB_E_CLASSNOTREG)
CASE_SCODE(REGDB_E_IIDNOTREG)
CASE_SCODE(RPC_E_CALL_REJECTED)
CASE_SCODE(RPC_E_CALL_CANCELED)
CASE_SCODE(RPC_E_CANTPOST_INSENDCALL)
```

```
                CASE_SCODE(RPC_E_CANTCALLOUT_INASYNCCALL)
                CASE_SCODE(RPC_E_CANTCALLOUT_INEXTERNALCALL)
                CASE_SCODE(RPC_E_CONNECTION_TERMINATED)
                CASE_SCODE(RPC_E_SERVER_DIED)
                CASE_SCODE(RPC_E_CLIENT_DIED)
                CASE_SCODE(RPC_E_INVALID_DATAPACKET)
                CASE_SCODE(RPC_E_CANTTRANSMIT_CALL)
                CASE_SCODE(RPC_E_CLIENT_CANTMARSHAL_DATA)
                CASE_SCODE(RPC_E_CLIENT_CANTUNMARSHAL_DATA)
                CASE_SCODE(RPC_E_SERVER_CANTMARSHAL_DATA)
                CASE_SCODE(RPC_E_SERVER_CANTUNMARSHAL_DATA)
                CASE_SCODE(RPC_E_INVALID_DATA)
                CASE_SCODE(RPC_E_INVALID_PARAMETER)
//                CASE_SCODE(RPC_E_CANTCALLOUT_AGAIN)
                CASE_SCODE(RPC_E_UNEXPECTED)

                /* SCODE's defined in DVOBJ.H */

                CASE_SCODE(DATA_S_SAMEFORMATETC)
                CASE_SCODE(VIEW_E_DRAW)
                CASE_SCODE(VIEW_S_ALREADY_FROZEN)
                CASE_SCODE(CACHE_E_NOCACHE_UPDATED)
                CASE_SCODE(CACHE_S_FORMATETC_NOTSUPPORTED)
                CASE_SCODE(CACHE_S_SAMECACHE)
                CASE_SCODE(CACHE_S_SOMECACHES_NOTUPDATED)

                /* SCODE's defined in STORAGE.H */

                CASE_SCODE(STG_E_INVALIDFUNCTION)
                CASE_SCODE(STG_E_FILENOTFOUND)
                CASE_SCODE(STG_E_PATHNOTFOUND)
                CASE_SCODE(STG_E_TOOMANYOPENFILES)
                CASE_SCODE(STG_E_ACCESSDENIED)
                CASE_SCODE(STG_E_INVALIDHANDLE)
                CASE_SCODE(STG_E_INSUFFICIENTMEMORY)
                CASE_SCODE(STG_E_INVALIDPOINTER)
                CASE_SCODE(STG_E_NOMOREFILES)
                CASE_SCODE(STG_E_DISKISWRITEPROTECTED)
                CASE_SCODE(STG_E_SEEKERROR)
                CASE_SCODE(STG_E_WRITEFAULT)
                CASE_SCODE(STG_E_READFAULT)
                CASE_SCODE(STG_E_SHAREVIOLATION)
                CASE_SCODE(STG_E_LOCKVIOLATION)
                CASE_SCODE(STG_E_FILEALREADYEXISTS)
                CASE_SCODE(STG_E_INVALIDPARAMETER)
                CASE_SCODE(STG_E_MEDIUMFULL)
                CASE_SCODE(STG_E_ABNORMALAPIEXIT)
                CASE_SCODE(STG_E_INVALIDHEADER)
                CASE_SCODE(STG_E_INVALIDNAME)
                CASE_SCODE(STG_E_UNKNOWN)
                CASE_SCODE(STG_E_UNIMPLEMENTEDFUNCTION)
                CASE_SCODE(STG_E_INVALIDFLAG)
                CASE_SCODE(STG_E_INUSE)
                CASE_SCODE(STG_E_NOTCURRENT)
                CASE_SCODE(STG_E_REVERTED)
                CASE_SCODE(STG_E_CANTSAVE)
                CASE_SCODE(STG_E_OLDFORMAT)
                CASE_SCODE(STG_E_OLDDLL)
                CASE_SCODE(STG_E_SHAREREQUIRED)
                CASE_SCODE(STG_E_NOTFILEBASEDSTORAGE)
                CASE_SCODE(STG_E_EXTANTMARSHALLINGS)
                CASE_SCODE(STG_S_CONVERTED)

                /* SCODE's defined in MONIKER.H */

                CASE_SCODE(MK_E_CONNECTMANUALLY)
                CASE_SCODE(MK_E_EXCEEDEDDEADLINE)
                CASE_SCODE(MK_E_NEEDGENERIC)
                CASE_SCODE(MK_E_UNAVAILABLE)
                CASE_SCODE(MK_E_SYNTAX)
                CASE_SCODE(MK_E_NOOBJECT)
                CASE_SCODE(MK_E_INVALIDEXTENSION)
                CASE_SCODE(MK_E_INTERMEDIATEINTERFACENOTSUPPORTED)
                CASE_SCODE(MK_E_NOTBINDABLE)
                CASE_SCODE(MK_E_NOTBOUND)
                CASE_SCODE(MK_E_CANTOPENFILE)
                CASE_SCODE(MK_E_MUSTBOTHERUSER)
                CASE_SCODE(MK_E_NOINVERSE)
                CASE_SCODE(MK_E_NOSTORAGE)
                CASE_SCODE(MK_E_NOPREFIX)
                CASE_SCODE(MK_S_REDUCED_TO_SELF)
                CASE_SCODE(MK_S_ME)
                CASE_SCODE(MK_S_HIM)
                CASE_SCODE(MK_S_US)
                CASE_SCODE(MK_S_MONIKERALREADYREGISTERED)

                default:
                        lstrcpy(szErrName, "UNKNOWN SCODE");
        }

        wsprintf(szBuf, "%s %s (0x%lx)\n", lpszMsg, (LPSTR)szErrName, sc);
        OleDbgPrintAlways(lpszPrefix, szBuf, 0);
#endif  // _DEBUG
}

//+------------------------------------------------------------
// Function:    PopUpError
//
// Synopsis:    Displays a dialog box using provided text,
//              and presents the user with the option to
//              continue or cancel.
//
// Arguments:
//      szMsg --        The string to display in main body of dialog
//      iLine --        Line number of file in error
//      szFile --       Filename of file in error
//
// Returns:
//      IDCANCEL --     User selected the CANCEL button
//      IDOK     --     User selected the OK button
```

```
//-------------------------------------------------------------

STDAPI_(int)
PopUpError(
    char const *szMsg,
    int iLine,
    char const *szFile)
{

    int id;
    static char szAssertCaption[100];

#ifdef FLAT
    DWORD tid = GetCurrentThreadId();
    wsprintf(szAssertCaption,"File: %s line %u, thread id %d",
        szFile, iLine, tid);
#else  // FLAT
    wsprintf(szAssertCaption,"File: %s line %u",szFile,iLine);
#endif // FLAT

    id = MessageBoxA(NULL,
                     (char *) szMsg,
                     (LPSTR) szAssertCaption,
                     MB_SETFOREGROUND |
                     MB_TASKMODAL | MB_ICONEXCLAMATION | MB_OKCANCEL);

    return id;
}

STDAPI_(void)
Win4AssertEx(
    char const * szFile,
    int iLine,
    char const * szMessage)
{
    int id = PopUpError(szMessage,iLine,szFile);

        if (id == IDCANCEL)
        {
#ifdef FLAT
            DebugBreak();
#else
            _asm int 3;
#endif
        }
}
```

DEBUG.H   (WRAPUI Sample)

```
/*
 * DEBUG.H
 *
 * Definitions, structures, types, and function prototypes for debugging
 * purposes.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved,
 * as applied to redistribution of this source code in source form
 * License is granted to use of compiled code in shipped binaries.
 */

#ifndef _DEBUG_H_
#define _DEBUG_H_

#ifdef DEBUG

//Basic debug macros
#define D(x)        {x;}
#define ODS(x)      D(OutputDebugString(x);OutputDebugString("\r\n"))

#define ODSsz(f, s)  {\
                              char        szDebug[128];\
                              wsprintf(szDebug, f, (LPSTR)s);\
                              ODS(szDebug);\
                              }


#define ODSu(f, u)  {\
                             char        szDebug[128];\
                             wsprintf(szDebug, f, (UINT)u);\
                             ODS(szDebug);\
                             }


#define ODSlu(f, lu) {\
                              char        szDebug[128];\
                              wsprintf(szDebug, f, (DWORD)lu);\
                              ODS(szDebug);\
                              }

#define ODSszu(f, s, u) {\
                                   char        szDebug[128];\
                                   wsprintf(szDebug, f, (LPSTR)s, (UINT)u);\
                                   ODS(szDebug);\
                                   }


#define ODSszlu(f, s, lu) {\
                                     char        szDebug[128];\
                                     wsprintf(szDebug, f, (LPSTR)s,
(DWORD)lu);\
                                     ODS(szDebug);\
                                        }


#else   //NO DEBUG

#define D(x)
#define ODS(x)

#define ODSsz(f, s)
#define ODSu(f, u)
#define ODSlu(f, lu)
#define ODSszu(f, s, u)
#define ODSszlu(f, s, lu)


#endif //DEBUG

#endif //_DEBUG_H_
```

DLLFUNCS.C   (WRAPUI Sample)

```
/*
 * DLLFUNCS.C
 *
 * Contains entry and exit points for the DLL implementation
 * of the OLE 2.0 User Interface Support Library.
 *
 * This file is not needed if we are linking the static library
 * version of this library.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */

#define STRICT  1
#include "ole2ui.h"
#include "common.h"

OLEDBGDATA


/*
 * LibMain
 *
 * Purpose:
 *  DLL-specific entry point called from LibEntry.  Initializes
 *  the DLL's heap and registers the GizmoBar GizmoBar.
 *
 * Parameters:
 *  hInst           HINSTANCE instance of the DLL.
 *  wDataSeg        WORD segment selector of the DLL's data segment.
 *  wHeapSize       WORD byte count of the heap.
 *  lpCmdLine       LPSTR to command line used to start the module.
 *
 * Return Value:
 *  HANDLE          Instance handle of the DLL.
 *
 */

#ifdef WIN32

int CALLBACK WEP(int);

BOOL WINAPI DllMain
(
     HINSTANCE hInst,
     ULONG Reason,
     PCONTEXT Context
)
{
     UNREFERENCED_PARAMETER(Context);
     if (Reason == DLL_PROCESS_DETACH)
         WEP(0);
     // Initialize OLE UI libraries.  If you're linking with the static LIB
     // version of this library, you need to make a call to OleUIInitialize
     // explicitly in your application (because this LibMain won't be
executed)
    if (Reason == DLL_PROCESS_ATTACH)
         OleUIInitialize(hInst, (HINSTANCE)0);

     return TRUE;

}

#else

int FAR PASCAL LibMain(HINSTANCE hInst, WORD wDataSeg
                   , WORD cbHeapSize, LPSTR lpCmdLine)
     {
     OleDbgOut2("LibMain: OLE2UI.DLL loaded\r\n");

     // Initialize OLE UI libraries.  If you're linking with the static LIB
version
     // of this library, you need to make the below call in your application
(because
     // this LibMain won't be executed).

     // The symbol LIBNAME is defined on the compiler command line

     OleUIInitialize(hInst, (HINSTANCE)0);

     //All done...
     if (0!=cbHeapSize)
     UnlockData(0);

     return (int)hInst;
     }

#endif

/*
 * WEP
 *
 * Purpose:
 *  Required DLL Exit function.
 *
 * Parameters:
 *  bSystemExit     BOOL indicating if the system is being shut
 *                  down or the DLL has just been unloaded.
 *
 * Return Value:
 *  void
 *
 */
int CALLBACK EXPORT WEP(int bSystemExit)
{
     OleUIUnInitialize();
     return 0;
```

```
}


/*
 * DllCanUnloadNow
 *
 * Purpose:
 *   Determines when it is safe to go away.
 *
 * Return Value:
 *   HRESULT    NOERROR it is safe to go away, S_FALSE this code must stay
 *              loaded.
 *
 */
STDAPI DllCanUnloadNow(void)
{
     return OleUICanUnloadNow();
}



/*
 * OleUILockLibrary
 *
 *
 * Purpose:
 *   Increments & decrements a lock count to force this library to stay
 *   alive, all OLE 2.x DLL objects servers (ie. INPROC server DLL) which
 *   use the OLE2UI library in a DLL form must call this function passing
TRUE
 *   in their InitInstance() routine and balance with a call, passing FALSE
 *   in their ExitInstance(). This is done so as to keep DllCanUnloadNow()
 *   from returning NOERROR until the OLE 2.x DLL server goes away.
 *   While the INPROC server DLL is loaded, it wants to force the OLE2UI
 *   DLL to remain loaded (ie. force it to return S_FALSE from the OLE2UI's
 *   DllCanUnloadNow function). It is possible that the INROC server DLL
 *   gets unloaded from a call to CoFreeUnusedLibraries, but it is still
 *   necessary for the OLE2UI library to remain in use (eg. because an
 *   enumerator returned from OleStdEnumFmtEtc_Create still exists). only
 *   after all explicit "OleUILockLibrary" locks and instances of objects
 *   created have been release will the DllCanUnloadNow function for the
 *   OLE2UI library DLL return NOERROR.
 *
 *   If the INPROC server DLL links to the OLE2UI library as a static
 *   library, then this OleUILockLibrary function should NOT be used.
 *   instead the INPROC server DLL should call OleUICanUnloadNow API from
 *   within its own DllCanUnloadNow function. The idea here is, that if
there
 *   are any existing instance of objects created by the OLE2UI library
 *   functions (eg. EnumFORMATETC objects created by
OleStdEnumFmtEtc_Create)
 *   then, the INPROC server DLL must NOT let itself be unloaded.
 *
 *   An EXE based object using the OLE2UI libray need NOT use either the
 *   OleUILockLibrary or OleUICanUnloadNow functions. All objects created
 *   by the OLE2UI library will have LRPC proxies and stubs created to
 *   manage remoting method calls. the worst that can happen when the EXE
 *   exits is that any outstanding proxies for unreleased objects will get
 *   RPC_E_SERVERDIED errors; they will not GPFault.
 *
 * Return Value:
 *   HRESULT    NOERROR it is safe to go away, S_FALSE this code must stay
 *              loaded.
 */

STDAPI OleUILockLibrary(BOOL fLock)
{
  HRESULT hRes = NOERROR;

  if (fLock == TRUE) {
     if ((DWORD)CoLoadLibrary("ole2ui" ".dll", TRUE) != NOERROR) {
       g_dwObjectCount ++;

     }else{
       hRes = ResultFromScode(CO_E_DLLNOTFOUND);
     }

  }else{
     g_dwObjectCount --;
  }

  return hRes;
}
```

DRAWICON.C   (WRAPUI Sample)

```
/*
 * DRAWICON.C
 *
 * Functions to handle creation of metafiles with icons and labels
 * as well as functions to draw such metafiles with or without the label.
 *
 * The metafile is created with a comment that marks the records containing
 * the label code.  Drawing the metafile enumerates the records, draws
 * all records up to that point, then decides to either skip the label
 * or draw it.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */

#define STRICT  1
#include "ole2ui.h"
#include "common.h"
#include "utility.h"
#include "geticon.h"

/*
 * Strings for metafile comments.  KEEP THESE IN SYNC WITH THE
 * STRINGS IN GETICON.C.
 */

static char szIconOnly[]="IconOnly";        //Where to stop to exclude
label.




/*
 * OleUIMetafilePictIconFree
 *
 * Purpose:
 *  Deletes the metafile contained in a METAFILEPICT structure and
 *  frees the memory for the structure itself.
 *
 * Parameters:
 *  hMetaPict       HGLOBAL metafilepict structure created in
 *                  OleUIMetafilePictFromIconAndLabel
 *
 * Return Value:
 *  None
 */

STDAPI_(void) OleUIMetafilePictIconFree(HGLOBAL hMetaPict)
     {
     LPMETAFILEPICT      pMF;

     if (NULL==hMetaPict)
          return;
     pMF=(LPMETAFILEPICT)GlobalLock(hMetaPict);

     if (NULL!=pMF)
           {
           if (NULL!=pMF->hMF)
                 DeleteMetaFile(pMF->hMF);
           }

     GlobalUnlock(hMetaPict);
     GlobalFree(hMetaPict);
     return;
     }
```

```
/*
 * OleUIMetafilePictIconDraw
 *
 * Purpose:
 *  Draws the metafile from OleUIMetafilePictFromIconAndLabel, either with
 *  the label or without.
 *
 * Parameters:
 *  hDC             HDC on which to draw.
 *  pRect           LPRECT in which to draw the metafile.
 *  hMetaPict       HGLOBAL to the METAFILEPICT from
 *                  OleUIMetafilePictFromIconAndLabel
 *  fIconOnly       BOOL specifying to draw the label or not.
 *
 * Return Value:
 *  BOOL            TRUE if the function is successful, FALSE if the
 *                  given metafilepict is invalid.
 */

STDAPI_(BOOL) OleUIMetafilePictIconDraw(HDC hDC, LPRECT pRect, HGLOBAL
hMetaPict
                                                       , BOOL fIconOnly)
     {
     LPMETAFILEPICT  pMF;
     DRAWINFO        di;
     int             cx, cy;
     SIZE            size;
     POINT           point;

     if (NULL==hMetaPict)
           return FALSE;

     pMF=GlobalLock(hMetaPict);
     if (NULL==pMF)
           return FALSE;

     di.Rect = *pRect;
     di.fIconOnly = fIconOnly;

     //Transform to back to pixels
     cx=XformWidthInHimetricToPixels(hDC, pMF->xExt);
     cy=XformHeightInHimetricToPixels(hDC, pMF->yExt);

     SaveDC(hDC);

     SetMapMode(hDC, pMF->mm);
     SetViewportOrgEx(hDC, (pRect->right - cx) / 2, 0, &point);

     SetViewportExtEx(hDC, min ((pRect->right - cx) / 2 + cx, cx), cy,
&size);

     if (fIconOnly)
           {
           // Since we've used the __export keyword on the
           // EnumMetafileIconDraw proc, we do not need to use
           // MakeProcInstance
           EnumMetaFile(hDC, pMF->hMF, (MFENUMPROC)EnumMetafileIconDraw
                 , (LPARAM)(LPDRAWINFO)&di);
           }
     else
        PlayMetaFile(hDC, pMF->hMF);

     RestoreDC(hDC, -1);

     GlobalUnlock(hMetaPict);
     return TRUE;
     }




/*
 * EnumMetafileIconDraw
 *
 * Purpose:
 *  EnumMetaFile callback function that draws either the icon only or
 *  the icon and label depending on given flags.
 *
 * Parameters:
 *  hDC              HDC into which the metafile should be played.
 *  phTable          HANDLETABLE FAR * providing handles selected into the
DC.
 *  pMFR             METARECORD FAR * giving the enumerated record.
 *  lParam           LPARAM flags passed in EnumMetaFile.
 *
 * Return Value:
 *  int              0 to stop enumeration, 1 to continue.
 */
```

```
int CALLBACK EXPORT EnumMetafileIconDraw(HDC hDC, HANDLETABLE FAR *phTable
     , METARECORD FAR *pMFR, int cObj, LPARAM lParam)
     {
     LPDRAWINFO lpdi = (LPDRAWINFO)lParam;

     /*
      * We play everything blindly except for DIBBITBLT (or DIBSTRETCHBLT)
      * and ESCAPE with MFCOMMENT.  For the BitBlts we change the x,y to
      * draw at (0,0) instead of wherever it was written to draw.  The
      * comment tells us there to stop if we don't want to draw the label.
      */

     //If we're playing icon only, stop enumeration at the comment.
     if (lpdi->fIconOnly)
           {
           if (META_ESCAPE==pMFR->rdFunction && MFCOMMENT==pMFR->rdParm[0])
                 {
                 if (0==lstrcmpi(szIconOnly, (LPSTR)&pMFR->rdParm[2]))
                       return 0;
                 }

           /*
            * Check for the records in which we want to munge the
coordinates.
            * destX is offset 6 for BitBlt, offset 9 for StretchBlt, either
of
            * which may appear in the metafile.
            */
           if (META_DIBBITBLT==pMFR->rdFunction)
                 pMFR->rdParm[6]=0;

           if (META_DIBSTRETCHBLT==pMFR->rdFunction)
                   pMFR->rdParm[9] = 0;

           }


     PlayMetaFileRecord(hDC, phTable, pMFR, cObj);
     return 1;
     }





/*
 * OleUIMetafilePictExtractLabel
 *
 * Purpose:
 *  Retrieves the label string from metafile representation of an icon.
 *
 * Parameters:
 *  hMetaPict       HGLOBAL to the METAFILEPICT containing the metafile.
 *  lpszLabel       LPSTR in which to store the label.
 *  cchLabel        UINT length of lpszLabel.
 *  lpWrapIndex     DWORD index of first character in last line. Can be NULL
 *                  if calling function doesn't care about word wrap.
 *
 * Return Value:
 *  UINT            Number of characters copied.
 */
STDAPI_(UINT) OleUIMetafilePictExtractLabel(HGLOBAL hMetaPict, LPSTR
lpszLabel
                                                            , UINT
cchLabel, LPDWORD lpWrapIndex)
     {
     LPMETAFILEPICT  pMF;
     LABELEXTRACT    le;
     HDC             hDC;

     /*
      * We extract the label by getting a screen DC and walking the metafile
      * records until we see the ExtTextOut record we put there.  That
      * record will have the string embedded in it which we then copy out.
      */

     if (NULL==hMetaPict || NULL==lpszLabel || 0==cchLabel)
           return FALSE;

     pMF=GlobalLock(hMetaPict);

     if (NULL==pMF)
           return FALSE;

     le.lpsz=lpszLabel;
     le.u.cch=cchLabel;
     le.Index=0;
     le.fFoundIconOnly=FALSE;
     le.fFoundSource=FALSE;  //Unused for this function.
     le.fFoundIndex=FALSE;   //Unused for this function.
     le.PrevIndex = 0;

     //Use a screen DC so we have something valid to pass in.
     hDC=GetDC(NULL);

     // Since we've used the EXPORT keyword on the
     // EnumMetafileExtractLabel proc, we do not need to use
     // MakeProcInstance

     EnumMetaFile(hDC, pMF->hMF, (MFENUMPROC)EnumMetafileExtractLabel,
(LONG)(LPLABELEXTRACT)&le);

     ReleaseDC(NULL, hDC);

     GlobalUnlock(hMetaPict);

     //Tell where we wrapped (if calling function cares)
     if (NULL != lpWrapIndex)
        *lpWrapIndex = le.PrevIndex;
     //Return amount of text copied
     return le.u.cch;
     }





/*
 * EnumMetafileExtractLabel
 *
 * Purpose:
 *  EnumMetaFile callback function that walks a metafile looking for
 *  ExtTextOut, then concatenates the text from each one into a buffer
 *  in lParam.
 *
 * Parameters:
 *  hDC             HDC into which the metafile should be played.
 *  phTable         HANDLETABLE FAR * providing handles selected into the
DC.
 *  pMFR            METARECORD FAR * giving the enumerated record.
 *  pLE             LPLABELEXTRACT providing the destination buffer and
length.
 *
 * Return Value:
 *  int             0 to stop enumeration, 1 to continue.
 */

int CALLBACK EXPORT EnumMetafileExtractLabel(HDC hDC, HANDLETABLE FAR
*phTable
     , METARECORD FAR *pMFR, int cObj, LPLABELEXTRACT pLE)
     {

     /*
      * We don't allow anything to happen until we see "IconOnly"
      * in an MFCOMMENT that is used to enable everything else.
      */
     if (!pLE->fFoundIconOnly)
           {
           if (META_ESCAPE==pMFR->rdFunction && MFCOMMENT==pMFR->rdParm[0])
                 {
                 if (0==lstrcmpi(szIconOnly, (LPSTR)&pMFR->rdParm[2]))
                       pLE->fFoundIconOnly=TRUE;
                 }

           return 1;
           }

     //Enumerate all records looking for META_EXTTEXTOUT - there can be more
     //than one.
     if (META_EXTTEXTOUT==pMFR->rdFunction)
           {
           UINT        cchMax;
           LPSTR       lpszTemp;
           /*
            * If ExtTextOut has NULL fuOptions, then the rectangle is
omitted
            * from the record, and the string starts at rdParm[4].  If
            * fuOptions is non-NULL, then the string starts at rdParm[8]
            * (since the rectange takes up four WORDs in the array).  In
            * both cases, the string continues for (rdParm[2]+1) >> 1
            * words.  We just cast a pointer to rdParm[8] to an LPSTR and
            * lstrcpyn into the buffer we were given.
            *
            * Note that we use element 8 in rdParm instead of 4 because we
            * passed ETO_CLIPPED in for the options on ExtTextOut--docs say
            * [4] which is rect doesn't exist if we passed zero there.
            *
            */

           cchMax=min(pLE->u.cch - pLE->Index, (UINT)pMFR->rdParm[2]);
           lpszTemp = pLE->lpsz + pLE->Index;

           lstrcpyn(lpszTemp, (LPSTR)&(pMFR->rdParm[8]), cchMax + 1);
//         lstrcpyn(lpszTemp, (LPSTR)&(pMFR->rdParm[4]), cchMax + 1);

           pLE->PrevIndex = pLE->Index;

           pLE->Index += cchMax;
           }

     return 1;
     }





/*
 * OleUIMetafilePictExtractIcon
 *
 * Purpose:
 *  Retrieves the icon from metafile into which DrawIcon was done before.
 *
 * Parameters:
 *  hMetaPict       HGLOBAL to the METAFILEPICT containing the metafile.
 *
 * Return Value:
 *  HICON           Icon recreated from the data in the metafile.
 */
STDAPI_(HICON) OleUIMetafilePictExtractIcon(HGLOBAL hMetaPict)
     {
     LPMETAFILEPICT  pMF;
     HDC             hDC;
     ICONEXTRACT     ie;

     /*
      * We extract the label by getting a screen DC and walking the metafile
      * records until we see the ExtTextOut record we put there.  That
      * record will have the string embedded in it which we then copy out.
      */

     if (NULL==hMetaPict)
          return NULL;

     pMF=GlobalLock(hMetaPict);

     if (NULL==pMF)
          return FALSE;

     //Use a screen DC so we have something valid to pass in.
     hDC=GetDC(NULL);
     ie.fAND=TRUE;

     // We get information back in the ICONEXTRACT structure.
     // (Since we've used the EXPORT keyword on the
     // EnumMetafileExtractLabel proc, we do not need to use
     // MakeProcInstance)
     EnumMetaFile(hDC, pMF->hMF, (MFENUMPROC)EnumMetafileExtractIcon, (LONG)
(LPICONEXTRACT)&ie);

     ReleaseDC(NULL, hDC);
     GlobalUnlock(hMetaPict);

     return ie.hIcon;
     }




/*
 * EnumMetafileExtractIcon
 *
 * Purpose:
 *  EnumMetaFile callback function that walks a metafile looking for
 *  StretchBlt (3.1) and BitBlt (3.0) records.  We expect to see two
 *  of them, the first being the AND mask and the second being the XOR
 *  data.  We
 *  ExtTextOut, then copies the text into a buffer in lParam.
 *
 * Parameters:
 *  hDC             HDC into which the metafile should be played.
 *  phTable         HANDLETABLE FAR * providing handles selected into the
DC.
 *  pMFR            METARECORD FAR * giving the enumerated record.
 *  pIE             LPICONEXTRACT providing the destination buffer and
length.
 *
 * Return Value:
 *  int             0 to stop enumeration, 1 to continue.
 */
```

```
int CALLBACK EXPORT EnumMetafileExtractIcon(HDC hDC, HANDLETABLE FAR
*phTable
     , METARECORD FAR *pMFR, int cObj, LPICONEXTRACT pIE)
     {
     LPBITMAPINFO        lpBI;
     LPBITMAPINFOHEADER  lpBH;
     LPBYTE              lpbSrc;
     LPBYTE              lpbDst;
     UINT                uWidth, uHeight;
     DWORD               cb;
     HGLOBAL             hMem;
     BITMAP              bm;
     HBITMAP             hBmp;
     int                 cxIcon, cyIcon;


     //Continue enumeration if we don't see the records we want.
     if (META_DIBBITBLT!=pMFR->rdFunction && META_DIBSTRETCHBLT!=pMFR-
>rdFunction)
           return 1;

     /*
      * Windows 3.0 DrawIcon uses META_DIBBITBLT in whereas 3.1 uses
      * META_DIBSTRETCHBLT so we have to handle each case separately.
      */

     if (META_DIBBITBLT==pMFR->rdFunction)       //Win3.0
           {
           //Get dimensions and the BITMAPINFO struct.
           uHeight=pMFR->rdParm[1];
           uWidth =pMFR->rdParm[2];
           lpBI=(LPBITMAPINFO)&(pMFR->rdParm[8]);
           }

     if (META_DIBSTRETCHBLT==pMFR->rdFunction)   //Win3.1
           {
           //Get dimensions and the BITMAPINFO struct.
           uHeight=pMFR->rdParm[2];
           uWidth =pMFR->rdParm[3];
           lpBI=(LPBITMAPINFO)&(pMFR->rdParm[10]);
           }

     lpBH=(LPBITMAPINFOHEADER)&(lpBI->bmiHeader);

     //Pointer to the bits which follows the BITMAPINFO structure.
     lpbSrc=(LPBYTE)lpBI+lpBH->biSize;

     //Add the length of the color table.
     if (0!=lpBH->biClrUsed)
           lpbSrc+=(DWORD)(lpBH->biClrUsed*sizeof(RGBQUAD));
     else
           {
        if (lpBH->biBitCount <= 8)
            {
               //1 << bc gives 2, 16, 256 for 1, 4, or 256 colors
               lpbSrc+=(DWORD)((1 << (lpBH->biBitCount))*sizeof(RGBQUAD));
            }
           }


     /*
      * All the bits we have in lpbSrc are device-independent, so we
      * need to change them over to be device-dependent using SetDIBits.
      * Once we have a bitmap with the device-dependent bits, we can
      * GetBitmapBits to have buffers with the real data.
      *
      * For each pass we have to allocate memory for the bits.  We save
      * the memory for the mask between passes.
      */

     //Use CreateBitmap for ANY monochrome bitmaps
     if (pIE->fAND || 1==lpBH->biBitCount || lpBH->biBitCount > 8)
           hBmp=CreateBitmap((UINT)lpBH->biWidth, (UINT)lpBH->biHeight, 1,
1, NULL);
     else if (lpBH->biBitCount <= 8)
           hBmp=CreateCompatibleBitmap(hDC, (UINT)lpBH->biWidth, (UINT)lpBH-
>biHeight);

     if (!hBmp || !SetDIBits(hDC, hBmp, 0, (UINT)lpBH->biHeight,
(LPVOID)lpbSrc, lpBI, DIB_RGB_COLORS))
           {
           if (!pIE->fAND)
                 GlobalFree(pIE->hMemAND);

           DeleteObject(hBmp);
           return 0;
           }

     //Allocate memory and get the DDBits into it.
     GetObject(hBmp, sizeof(bm), &bm);

     cb=bm.bmHeight*bm.bmWidthBytes * bm.bmPlanes;

//    if (cb % 4 != 0)        // dword align
//      cb += 4 - (cb % 4);

     hMem=GlobalAlloc(GHND, cb);

     if (NULL==hMem)
           {
           if (NULL!=pIE->hMemAND)
                 GlobalFree(pIE->hMemAND);

           DeleteObject(hBmp);
           return 0;
           }

     lpbDst=(LPBYTE)GlobalLock(hMem);
     GetBitmapBits(hBmp, cb, (LPVOID)lpbDst);
     DeleteObject(hBmp);
     GlobalUnlock(hMem);


     /*
      * If this is the first pass (pIE->fAND==TRUE) then save the memory
      * of the AND bits for the next pass.
      */
     if (pIE->fAND)
           {
           pIE->fAND=FALSE;
           pIE->hMemAND=hMem;

           //Continue enumeration looking for the next blt record.
           return 1;
           }
     else
           {
           //Get the AND pointer again.
           lpbSrc=(LPBYTE)GlobalLock(pIE->hMemAND);

           /*
            * Create the icon now that we have all the data.  lpbDst already
            * points to the XOR bits.
            */
           cxIcon = GetSystemMetrics(SM_CXICON);
           cyIcon = GetSystemMetrics(SM_CYICON);

           pIE->hIcon=CreateIcon(ghInst,
                                           uWidth,
                                           uHeight,
                                           (BYTE)bm.bmPlanes,
                                           (BYTE)bm.bmBitsPixel,
                                           (LPVOID)lpbSrc,
                                           (LPVOID)lpbDst);

           GlobalUnlock(pIE->hMemAND);
           GlobalFree(pIE->hMemAND);
           GlobalFree(hMem);

           //We're done so we can stop.
           return 0;
           }
     }




/*
 * OleUIMetafilePictExtractIconSource
 *
 * Purpose:
 *  Retrieves the filename and index of the icon source from a metafile
 *  created with OleUIMetafilePictFromIconAndLabel.
 *
 * Parameters:
 *  hMetaPict       HGLOBAL to the METAFILEPICT containing the metafile.
 *  lpszSource      LPSTR in which to store the source filename.  This
 *                  buffer should be OLEUI_CCHPATHMAX characters.
 *  piIcon          UINT FAR * in which to store the icon's index
 *                  within lpszSource
 *
 * Return Value:
 *  BOOL            TRUE if the records were found, FALSE otherwise.
 */
STDAPI_(BOOL) OleUIMetafilePictExtractIconSource(HGLOBAL hMetaPict
     , LPSTR lpszSource, UINT FAR *piIcon)
     {
     LPMETAFILEPICT  pMF;
     LABELEXTRACT    le;
     HDC             hDC;

     /*
      * We will walk the metafile looking for the two comment records
      * following the IconOnly comment.  The flags fFoundIconOnly and
      * fFoundSource indicate if we have found IconOnly and if we have
      * found the source comment already.
      */

     if (NULL==hMetaPict || NULL==lpszSource || NULL==piIcon)
           return FALSE;

     pMF=GlobalLock(hMetaPict);

     if (NULL==pMF)
           return FALSE;

     le.lpsz=lpszSource;
     le.fFoundIconOnly=FALSE;
     le.fFoundSource=FALSE;
     le.fFoundIndex=FALSE;

     //Use a screen DC so we have something valid to pass in.
     hDC=GetDC(NULL);

     EnumMetaFile(hDC, pMF->hMF, (MFENUMPROC)EnumMetafileExtractIconSource,
(LONG)(LPLABELEXTRACT)&le);

     ReleaseDC(NULL, hDC);
     GlobalUnlock(hMetaPict);

     //Copy the icon index to the caller's variable.
     *piIcon=le.u.iIcon;

     //Check that we found everything.
     return (le.fFoundIconOnly && le.fFoundSource && le.fFoundIndex);
     }
```

```
/*
 * EnumMetafileExtractIconSource
 *
 * Purpose:
 *  EnumMetaFile callback function that walks a metafile skipping the first
 *  comment record, extracting the source filename from the second, and
 *  the index of the icon in the third.
 *
 * Parameters:
 *  hDC             HDC into which the metafile should be played.
 *  phTable         HANDLETABLE FAR * providing handles selected into the
DC.
 *  pMFR            METARECORD FAR * giving the enumerated record.
 *  pLE             LPLABELEXTRACT providing the destination buffer and
 *                  area to store the icon index.
 *
 * Return Value:
 *  int             0 to stop enumeration, 1 to continue.
 */

int CALLBACK EXPORT EnumMetafileExtractIconSource(HDC hDC, HANDLETABLE FAR
*phTable
     , METARECORD FAR *pMFR, int cObj, LPLABELEXTRACT pLE)
     {
     LPSTR       psz;

     /*
      * We don't allow anything to happen until we see "IconOnly"
      * in an MFCOMMENT that is used to enable everything else.
      */
     if (!pLE->fFoundIconOnly)
           {
           if (META_ESCAPE==pMFR->rdFunction && MFCOMMENT==pMFR->rdParm[0])
                 {
                 if (0==lstrcmpi(szIconOnly, (LPSTR)&pMFR->rdParm[2]))
                       pLE->fFoundIconOnly=TRUE;
                 }

           return 1;
           }

     //Now see if we find the source string.
     if (!pLE->fFoundSource)
           {
           if (META_ESCAPE==pMFR->rdFunction && MFCOMMENT==pMFR->rdParm[0])
                 {
                 LSTRCPYN(pLE->lpsz, (LPSTR)&pMFR->rdParm[2],
OLEUI_CCHPATHMAX);
                 pLE->lpsz[OLEUI_CCHPATHMAX-1] = '\0';
                 pLE->fFoundSource=TRUE;
                 }
```

```
        return 1;
        }

//Next comment will be the icon index.
if (META_ESCAPE==pMFR->rdFunction && MFCOMMENT==pMFR->rdParm[0])
        {
        /*
         * This string contains the icon index in string form,
         * so we need to convert back to a UINT.  After we see this
         * we can stop the enumeration.  The comment will have
         * a null terminator because we made sure to save it.
         */
        psz=(LPSTR)&pMFR->rdParm[2];
        pLE->u.iIcon=0;

        //Do Ye Olde atoi
        while (*psz)
                pLE->u.iIcon=(10*pLE->u.iIcon)+((*psz++)-'0');

        pLE->fFoundIndex=TRUE;
        return 0;
        }

return 1;
}
```

EDLINKS.H (WRAPUI Sample)

```
/*
 * EDLINKS.H
 *
 * Internal definitions, structures, and function prototypes for the
 * OLE 2.0 UI Edit Links dialog.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */


#ifndef _LINKS_H_
#define _LINKS_H_

//INTERNAL INFORMATION STARTS HERE
#define OLEUI_SZMAX 255
#define LINKTYPELEN 9
#define szNULL    "\0"

typedef UINT (CALLBACK* COMMDLGHOOKPROC)(HWND, UINT, WPARAM, LPARAM);

//Internally used structure

typedef struct tagLINKINFO
  {
  DWORD     dwLink;             // app specific identifier of a link
  LPSTR     lpszDisplayName;    // file based part of name
  LPSTR     lpszItemName;       // object part of name
  LPSTR     lpszShortFileName;  // filename without path
  LPSTR     lpszShortLinkType;  // Short link type - progID
  LPSTR     lpszFullLinkType;   // Full link type - user friendly name
  LPSTR     lpszAMX;          // Is the link auto (A) man (M) or dead (X)
  ULONG     clenFileName;       // count of file part of mon.
  BOOL      fSourceAvailable;   // bound or not - on boot assume yes??
  BOOL      fIsAuto;                      // 1 =automatic, 0=manual update
  BOOL      fIsMarked;                  // 1 = marked, 0 = not
  BOOL      fDontFree;          // Don't free this data since it's being
reused
  BOOL      fIsSelected;        // item selected or to be selected
  } LINKINFO, FAR* LPLINKINFO;

  /*
  * What we store extra in this structure besides the original caller's
  * pointer are those fields that we need to modify during the life of
  * the dialog but that we don't want to change in the original structure
  * until the user presses OK.
  */

typedef struct tagEDITLINKS
     {
  //Keep this item first as the Standard* functions depend on it here.

  LPOLEUIEDITLINKS    lpOEL;       //Original structure passed.
  BOOL        fClose;             //  Does the button read cancel (0) or
                                                 //  close (1)?
  int         *rgIndex;           //  Array to hold indexes of selected
items
  int         cSelItems;          //  Number of selected items
  BOOL        fItemsExist;        //  TRUE, items in lbox, FALSE, none
  UINT        nChgSrcHelpID;      //  ID for Help callback from ChangeSrc
dlg
  char        szClose[50];        //  Text for Close button
                                                 //    (when Cancel button
gets renamed)
} EDITLINKS, *PEDITLINKS, FAR *LPEDITLINKS;

// Data to and from the ChangeSource dialog hook
typedef struct tagOLEUICHANGESOURCEHOOKDATA
{
     //These IN fields are standard across all OLEUI dialog functions.
     DWORD           cbStruct;         //Structure Size
     DWORD           dwFlags;          //IN-OUT:  Flags
     HWND            hWndOwner;        //Owning window
     LPCSTR          lpszCaption;      //Dialog caption bar contents
     LPFNOLEUIHOOK   lpfnHook;         //Hook callback
     LPARAM          lCustData;        //Custom data to pass to hook
     HINSTANCE       hInstance;        //Instance for customized template
name
     LPCSTR          lpszTemplate;     //Customized template name
     HRSRC           hResource;        //Customized template handle

     //Specifics for OLEUIINSERTOBJECT.  All are IN-OUT unless otherwise
spec.
     LPLINKINFO  lpLI;                 // IN: ptr to LinkInfo entry
     LPEDITLINKS lpEL;                 // IN: ptr to EditLinks dialog struct
     BOOL        fValidLink;           // OUT: was link source validated
     LPSTR       lpszFrom;             // OUT: string containing prefix of
                                                       //      source
changed from
     LPSTR       lpszTo;               // OUT: string containing prefix of
                                                       //      source
changed to
} OLEUICHANGESOURCEHOOKDATA, *POLEUICHANGESOURCEHOOKDATA,
  FAR *LPOLEUICHANGESOURCEHOOKDATA;


// Data to and from the ChangeSource dialog hook
typedef struct tagCHANGESOURCEHOOKDATA
{
     LPOLEUICHANGESOURCEHOOKDATA lpOCshData;  //Original structure passed.
     LPOPENFILENAME lpOfn;
     BOOL        fValidLink;
     int         nFileLength;
     int         nEditLength;
     char        szFileName[OLEUI_CCHPATHMAX];
     char        szItemName[OLEUI_CCHPATHMAX];
     BOOL        bFileNameStored;
     BOOL        bItemNameStored;
     char        szEdit[OLEUI_CCHPATHMAX];
     LPSTR       lpszFrom;           // string containing prefix of source
                                                     // changed from
     LPSTR       lpszTo;             // string containing prefix of source
                                                     // source changed to
} CHANGESOURCEHOOKDATA, *PCHANGESOURCEHOOKDATA, FAR *LPCHANGESOURCEHOOKDATA;


//Internal function prototypes
//LINKS.C
BOOL CALLBACK EXPORT EditLinksDialogProc(HWND, UINT, WPARAM, LPARAM);
BOOL            FEditLinksInit(HWND, WPARAM, LPARAM);
BOOL      Container_ChangeSource(HWND, LPEDITLINKS);
HRESULT   Container_AutomaticManual(HWND, BOOL, LPEDITLINKS);
HRESULT   CancelLink(HWND, LPEDITLINKS);
HRESULT   Container_UpdateNow(HWND, LPEDITLINKS);
HRESULT   Container_OpenSource(HWND, LPEDITLINKS);
int AddLinkLBItem(HWND hListBox, LPOLEUILINKCONTAINER lpOleUILinkCntr,
LPLINKINFO lpLI, BOOL fGetSelected, int nIndex);
VOID    BreakString(LPLINKINFO);
int     GetSelectedItems(HWND, int FAR* FAR*);
BOOL WINAPI ChangeSource(HWND hWndOwner,
                          LPSTR lpszFile,
                          UINT cchFile,
                          UINT iFilterString,
                          COMMDLGHOOKPROC lpfnBrowseHook,
                          LPOLEUICHANGESOURCEHOOKDATA lpLbhData);
UINT CALLBACK EXPORT ChangeSourceHook(HWND hDlg, UINT uMsg, WPARAM wParam,
LPARAM lParam);
VOID InitControls(HWND hDlg, LPEDITLINKS lpEL);
VOID UpdateLinkLBItem(HWND hListBox, int nIndex, LPEDITLINKS lpEL, BOOL
bSelect);
VOID DiffPrefix(LPCSTR lpsz1, LPCSTR lpsz2, char FAR* FAR* lplpszPrefix1,
char FAR* FAR* lplpszPrefix2);
int PopupMessage(HWND hwndParent, UINT idTitle, UINT idMessage, UINT
fuStyle);
BOOL FChangeAllLinks(HWND hLIstBox, LPOLEUILINKCONTAINER lpOleUILinkCntr,
LPSTR lpszFrom, LPSTR lpszTo);
BOOL CFindLinks(HWND hListBox, LPSTR lpsz);
int LoadLinkLB(HWND hListBox, LPOLEUILINKCONTAINER lpOleUILinkCntr);
VOID RefreshLinkLB(HWND hListBox, LPOLEUILINKCONTAINER lpOleUILinkCntr);
#endif // __LINKS_H__
```

ENUMFETC.H (WRAPUI Sample)

```
// This file is now OBSOLETE (include olestd.h instead)

/************************************************************************
**
**    OLE 2 Utility Code
**
**    enumfetc.c
**
**    (c) Copyright Microsoft Corp. 1990 - 1992 All Rights Reserved
**
************************************************************************/

// Function prototypes moved to olestd.h
```

ENUMFETC.C (WRAPUI Sample)

```
/*************************************************************************
**
**    OLE 2 Utility Code
**
**    enumfetc.c
**
**    This file contains a standard implementation of IEnumFormatEtc
**    interface.
**    This file is part of the OLE 2.0 User Interface support library.
**
**    (c) Copyright Microsoft Corp. 1990 - 1992 All Rights Reserved
**
*************************************************************************/

#define STRICT  1
#include "ole2ui.h"
#include "enumfetc.h"


typedef struct tagOleStdEnumFmtEtc {
  IEnumFORMATETCVtbl FAR* lpVtbl;
  ULONG m_dwRefs;       /* referance count */
  ULONG m_nIndex;       /* current index in list */
  ULONG m_nCount;       /* how many items in list */
  LPFORMATETC m_lpEtc;  /* list of formatetc */
} OLESTDENUMFMTETC, FAR* LPOLESTDENUMFMTETC;

VOID  OleStdEnumFmtEtc_Destroy(LPOLESTDENUMFMTETC pEtc);

STDMETHODIMP OleStdEnumFmtEtc_QueryInterface(
           LPENUMFORMATETC lpThis, REFIID riid, LPVOID FAR* ppobj);
STDMETHODIMP_(ULONG)  OleStdEnumFmtEtc_AddRef(LPENUMFORMATETC lpThis);
STDMETHODIMP_(ULONG)  OleStdEnumFmtEtc_Release(LPENUMFORMATETC lpThis);
STDMETHODIMP  OleStdEnumFmtEtc_Next(LPENUMFORMATETC lpThis, ULONG celt,
                                                 LPFORMATETC rgelt, ULONG
FAR* pceltFetched);
STDMETHODIMP  OleStdEnumFmtEtc_Skip(LPENUMFORMATETC lpThis, ULONG celt);
STDMETHODIMP  OleStdEnumFmtEtc_Reset(LPENUMFORMATETC lpThis);
STDMETHODIMP  OleStdEnumFmtEtc_Clone(LPENUMFORMATETC lpThis,
                                                      LPENUMFORMATETC FAR*
ppenum);

static IEnumFORMATETCVtbl g_EnumFORMATETCVtbl = {
           OleStdEnumFmtEtc_QueryInterface,
           OleStdEnumFmtEtc_AddRef,
           OleStdEnumFmtEtc_Release,
           OleStdEnumFmtEtc_Next,
           OleStdEnumFmtEtc_Skip,
           OleStdEnumFmtEtc_Reset,
           OleStdEnumFmtEtc_Clone,
};
```

```
////////////////////////////////////////////////////////////////////////////
/


STDAPI_(LPENUMFORMATETC)
  OleStdEnumFmtEtc_Create(ULONG nCount, LPFORMATETC lpEtc)
//--------------------------------------------------------------------------
--
//
//--------------------------------------------------------------------------
--
{
  LPMALLOC lpMalloc=NULL;
  LPOLESTDENUMFMTETC lpEF=NULL;
  DWORD dwSize;
  UINT i;
  HRESULT hRes;

  hRes = CoGetMalloc(MEMCTX_TASK, &lpMalloc);
  if (hRes != NOERROR) {
     return NULL;
  }

  lpEF = (LPOLESTDENUMFMTETC)lpMalloc->lpVtbl->Alloc(lpMalloc,

sizeof(OLESTDENUMFMTETC));
  if (lpEF == NULL) {
     goto errReturn;
  }

  lpEF->lpVtbl = &g_EnumFORMATETCVtbl;
  lpEF->m_dwRefs = 1;
  lpEF->m_nCount = nCount;
  lpEF->m_nIndex = 0;

  dwSize = sizeof(FORMATETC) * lpEF->m_nCount;

  lpEF->m_lpEtc = (LPFORMATETC)lpMalloc->lpVtbl->Alloc(lpMalloc, dwSize);
  if (lpEF->m_lpEtc == NULL)
     goto errReturn;

  lpMalloc->lpVtbl->Release(lpMalloc);

  for (i=0; i<nCount; i++) {
     OleStdCopyFormatEtc(
                  (LPFORMATETC)&(lpEF->m_lpEtc[i]),
(LPFORMATETC)&(lpEtc[i]));
  }

  g_dwObjectCount++ ;

  return (LPENUMFORMATETC)lpEF;

errReturn:
  if (lpEF != NULL)
```

```
     lpMalloc->lpVtbl->Free(lpMalloc, lpEF);

  if (lpMalloc != NULL)
     lpMalloc->lpVtbl->Release(lpMalloc);

  return NULL;

} /* OleStdEnumFmtEtc_Create()
   */


VOID
  OleStdEnumFmtEtc_Destroy(LPOLESTDENUMFMTETC lpEF)
//----------------------------------------------------------------------------
--
//
//----------------------------------------------------------------------------
--
{
     LPMALLOC lpMalloc=NULL;
     UINT i;

     if (lpEF != NULL) {

           if (CoGetMalloc(MEMCTX_TASK, &lpMalloc) == NOERROR) {

                 /* OLE2NOTE: we MUST free any memory that was allocated for
                 **    TARGETDEVICES contained within the FORMATETC
elements.
                 */
                 for (i=0; i<lpEF->m_nCount; i++) {
                       OleStdFree(lpEF->m_lpEtc[i].ptd);
                 }

                 if (lpEF->m_lpEtc != NULL) {
                       lpMalloc->lpVtbl->Free(lpMalloc, lpEF->m_lpEtc);
                 }

                 lpMalloc->lpVtbl->Free(lpMalloc, lpEF);
                 lpMalloc->lpVtbl->Release(lpMalloc);
           }
     }

     g_dwObjectCount-- ;

} /* OleStdEnumFmtEtc_Destroy()
   */


STDMETHODIMP
  OleStdEnumFmtEtc_QueryInterface(
                       LPENUMFORMATETC lpThis, REFIID riid, LPVOID FAR*
ppobj)
//----------------------------------------------------------------------------
--
```

```
//
//----------------------------------------------------------------------------
--
{
  LPOLESTDENUMFMTETC lpEF = (LPOLESTDENUMFMTETC)lpThis;
  *ppobj = NULL;

  if (IsEqualIID(riid,&IID_IUnknown) ||
IsEqualIID(riid,&IID_IEnumFORMATETC)){
     *ppobj = (LPVOID)lpEF;
  }

  if (*ppobj == NULL) return ResultFromScode(E_NOINTERFACE);
  else{
     OleStdEnumFmtEtc_AddRef(lpThis);
     return NOERROR;
  }

} /* OleStdEnumFmtEtc_QueryInterface()
   */


STDMETHODIMP_(ULONG)
  OleStdEnumFmtEtc_AddRef(LPENUMFORMATETC lpThis)
//----------------------------------------------------------------------------
--
//
//----------------------------------------------------------------------------
--
{
  LPOLESTDENUMFMTETC lpEF = (LPOLESTDENUMFMTETC)lpThis;
  return lpEF->m_dwRefs++;

} /* OleStdEnumFmtEtc_AddRef()
   */


STDMETHODIMP_(ULONG)
  OleStdEnumFmtEtc_Release(LPENUMFORMATETC lpThis)
//----------------------------------------------------------------------------
--
//
//----------------------------------------------------------------------------
--
{
  LPOLESTDENUMFMTETC lpEF = (LPOLESTDENUMFMTETC)lpThis;
  DWORD dwRefs = --lpEF->m_dwRefs;

  if (dwRefs == 0)
     OleStdEnumFmtEtc_Destroy(lpEF);

  return dwRefs;

} /* OleStdEnumFmtEtc_Release()
   */
```

```
STDMETHODIMP
  OleStdEnumFmtEtc_Next(LPENUMFORMATETC lpThis, ULONG celt, LPFORMATETC
rgelt,
                                ULONG FAR* pceltFetched)
//----------------------------------------------------------------------------
--
//
//----------------------------------------------------------------------------
--
{
  LPOLESTDENUMFMTETC lpEF = (LPOLESTDENUMFMTETC)lpThis;
  ULONG i=0;
  ULONG nOffset;

  if (rgelt == NULL) {
     return ResultFromScode(E_INVALIDARG);
  }

  while (i < celt) {
     nOffset = lpEF->m_nIndex + i;

     if (nOffset < lpEF->m_nCount) {
       OleStdCopyFormatEtc(
                 (LPFORMATETC)&(rgelt[i]), (LPFORMATETC)&(lpEF-
>m_lpEtc[nOffset]));
       i++;
     }else{
       break;
     }
  }

  lpEF->m_nIndex += (WORD)i;

  if (pceltFetched != NULL) {
     *pceltFetched = i;
  }

  if (i != celt) {
     return ResultFromScode(S_FALSE);
  }

  return NOERROR;
} /* OleStdEnumFmtEtc_Next()
   */


STDMETHODIMP
  OleStdEnumFmtEtc_Skip(LPENUMFORMATETC lpThis, ULONG celt)
//----------------------------------------------------------------------------
--
//
//----------------------------------------------------------------------------
--
```

```
{
  LPOLESTDENUMFMTETC lpEF = (LPOLESTDENUMFMTETC)lpThis;
  ULONG i=0;
  ULONG nOffset;

  while (i < celt) {
     nOffset = lpEF->m_nIndex + i;

     if (nOffset < lpEF->m_nCount) {
       i++;
     }else{
       break;
     }
  }

  lpEF->m_nIndex += (WORD)i;

  if (i != celt) {
     return ResultFromScode(S_FALSE);
  }

  return NOERROR;
} /* OleStdEnumFmtEtc_Skip()
   */


STDMETHODIMP
  OleStdEnumFmtEtc_Reset(LPENUMFORMATETC lpThis)
//----------------------------------------------------------------------------
--
//
//----------------------------------------------------------------------------
--
{
  LPOLESTDENUMFMTETC lpEF = (LPOLESTDENUMFMTETC)lpThis;
  lpEF->m_nIndex = 0;

  return NOERROR;
} /* OleStdEnumFmtEtc_Reset()
   */


STDMETHODIMP
  OleStdEnumFmtEtc_Clone(LPENUMFORMATETC lpThis, LPENUMFORMATETC FAR*
ppenum)
//----------------------------------------------------------------------------
--
//
//----------------------------------------------------------------------------
--
{
  LPOLESTDENUMFMTETC lpEF = (LPOLESTDENUMFMTETC)lpThis;

  if (ppenum == NULL) {
     return ResultFromScode(E_INVALIDARG);
```

```
    }

    *ppenum = OleStdEnumFmtEtc_Create(lpEF->m_nCount, lpEF->m_lpEtc);

    // make sure cloned enumerator has same index state as the original
    if (*ppenum) {
         LPOLESTDENUMFMTETC lpEFClone = (LPOLESTDENUMFMTETC)*ppenum;
         lpEFClone->m_nIndex = lpEF->m_nIndex;
         return NOERROR;
    } else
         return ResultFromScode(E_OUTOFMEMORY);

} /* OleStdEnumFmtEtc_Clone()
   */
```

ENUMSTAT.C   (WRAPUI Sample)

```
/*************************************************************************
**
**    OLE 2 Utility Code
**
**    enumstat.c
**
**    This file contains a standard implementation of IEnumStatData
**    interface.
**    This file is part of the OLE 2.0 User Interface support library.
**
**    (c) Copyright Microsoft Corp. 1990 - 1992 All Rights Reserved
**
*************************************************************************/

#define STRICT  1
#include "ole2ui.h"


typedef struct tagOleStdEnumStatData {
  IEnumSTATDATAVtbl FAR* lpVtbl;
  ULONG m_dwRefs;       /* referance count */
  ULONG m_nIndex;       /* current index in list */
  ULONG m_nCount;       /* how many items in list */
  LPSTATDATA m_lpStat;  /* list of STATDATA */
} OLESTDENUMSTATDATA, FAR* LPOLESTDENUMSTATDATA;

VOID  OleStdEnumStatData_Destroy(LPOLESTDENUMSTATDATA pStat);

STDMETHODIMP OleStdEnumStatData_QueryInterface(
           LPENUMSTATDATA lpThis, REFIID riid, LPVOID FAR* ppobj);
STDMETHODIMP_(ULONG)  OleStdEnumStatData_AddRef(LPENUMSTATDATA lpThis);
STDMETHODIMP_(ULONG)  OleStdEnumStatData_Release(LPENUMSTATDATA lpThis);
STDMETHODIMP  OleStdEnumStatData_Next(LPENUMSTATDATA lpThis, ULONG celt,
                                                 LPSTATDATA rgelt, ULONG
FAR* pceltFetched);
STDMETHODIMP  OleStdEnumStatData_Skip(LPENUMSTATDATA lpThis, ULONG celt);
STDMETHODIMP  OleStdEnumStatData_Reset(LPENUMSTATDATA lpThis);
STDMETHODIMP  OleStdEnumStatData_Clone(LPENUMSTATDATA lpThis,
                                                      LPENUMSTATDATA FAR*
ppenum);

static IEnumSTATDATAVtbl g_EnumSTATDATAVtbl = {
           OleStdEnumStatData_QueryInterface,
           OleStdEnumStatData_AddRef,
           OleStdEnumStatData_Release,
           OleStdEnumStatData_Next,
           OleStdEnumStatData_Skip,
           OleStdEnumStatData_Reset,
           OleStdEnumStatData_Clone,
};
```

```
////////////////////////////////////////////////////////////////////////////
/

STDAPI_(BOOL)
  OleStdCopyStatData(LPSTATDATA pDest, LPSTATDATA pSrc)
//--------------------------------------------------------------------------
--
//
//--------------------------------------------------------------------------
--
{
  if ((pDest == NULL) || (pSrc == NULL)) {
     return FALSE;
  }

  if (OleStdCopyFormatEtc(&pDest->formatetc, &pSrc->formatetc) == FALSE) {
     return FALSE;
  }

  pDest->advf = pSrc->advf;
  pDest->pAdvSink = pSrc->pAdvSink;
  pDest->dwConnection = pSrc->dwConnection;

  if (pDest->pAdvSink != NULL) {
     pDest->pAdvSink->lpVtbl->AddRef(pDest->pAdvSink);
  }

  return TRUE;

} /* OleStdCopyStatData()
   */

STDAPI_(LPENUMSTATDATA)
  OleStdEnumStatData_Create(ULONG nCount, LPSTATDATA lpStatOrg)
//--------------------------------------------------------------------------
--
//
//--------------------------------------------------------------------------
--
{
  LPMALLOC lpMalloc=NULL;
  LPOLESTDENUMSTATDATA lpSD=NULL;
  DWORD dwSize;
  WORD i;
  HRESULT hRes;

  hRes = CoGetMalloc(MEMCTX_TASK, &lpMalloc);
  if (hRes != NOERROR) {
     return NULL;
  }

  lpSD = (LPOLESTDENUMSTATDATA)lpMalloc->lpVtbl->Alloc(lpMalloc,

sizeof(OLESTDENUMSTATDATA));
  if (lpSD == NULL) {
     goto errReturn;
  }

  lpSD->lpVtbl = &g_EnumSTATDATAVtbl;
  lpSD->m_dwRefs = 1;
  lpSD->m_nCount = nCount;
  lpSD->m_nIndex = 0;

  dwSize = sizeof(STATDATA) * lpSD->m_nCount;

  lpSD->m_lpStat = (LPSTATDATA)lpMalloc->lpVtbl->Alloc(lpMalloc, dwSize);
  if (lpSD->m_lpStat == NULL)
     goto errReturn;

  lpMalloc->lpVtbl->Release(lpMalloc);

  for (i=0; i<nCount; i++) {
     OleStdCopyStatData(
                 (LPSTATDATA)&(lpSD->m_lpStat[i]),
(LPSTATDATA)&(lpStatOrg[i]));
  }

  g_dwObjectCount++ ;

  return (LPENUMSTATDATA)lpSD;

errReturn:
  if (lpSD != NULL)
     lpMalloc->lpVtbl->Free(lpMalloc, lpSD);

  if (lpMalloc != NULL)
     lpMalloc->lpVtbl->Release(lpMalloc);

  return NULL;

} /* OleStdEnumStatData_Create()
   */


VOID
  OleStdEnumStatData_Destroy(LPOLESTDENUMSTATDATA lpSD)
//---------------------------------------------------------------------------
--
//
//---------------------------------------------------------------------------
--
{
     LPMALLOC lpMalloc=NULL;
     WORD i;

     if (lpSD != NULL) {

           if (CoGetMalloc(MEMCTX_TASK, &lpMalloc) == NOERROR) {

                 /* OLE2NOTE: we MUST free any memory that was allocated for
```

```
                 **    TARGETDEVICES contained within the STATDATA elements.
                 */
                 for (i=0; i<lpSD->m_nCount; i++) {
                       if( lpSD->m_lpStat[i].pAdvSink )
                             lpSD->m_lpStat[i].pAdvSink->lpVtbl-
>Release(lpSD->m_lpStat[i].pAdvSink);

                       OleStdFree(lpSD->m_lpStat[i].formatetc.ptd);
                 }

                 if (lpSD->m_lpStat != NULL) {
                       lpMalloc->lpVtbl->Free(lpMalloc, lpSD->m_lpStat);
                 }

                 lpMalloc->lpVtbl->Free(lpMalloc, lpSD);
                 lpMalloc->lpVtbl->Release(lpMalloc);
           }
     }

     g_dwObjectCount-- ;

} /* OleStdEnumStatData_Destroy()
   */


STDMETHODIMP
  OleStdEnumStatData_QueryInterface(
                       LPENUMSTATDATA lpThis, REFIID riid, LPVOID FAR*
ppobj)
//----------------------------------------------------------------------------
--
//
//----------------------------------------------------------------------------
--
{
  LPOLESTDENUMSTATDATA lpSD = (LPOLESTDENUMSTATDATA)lpThis;
  *ppobj = NULL;

  if (IsEqualIID(riid,&IID_IUnknown) || IsEqualIID(riid,&IID_IEnumSTATDATA))
{
     *ppobj = (LPVOID)lpSD;
  }

  if (*ppobj == NULL) return ResultFromScode(E_NOINTERFACE);
  else{
     OleStdEnumStatData_AddRef(lpThis);
     return NOERROR;
  }

} /* OleStdEnumStatData_QueryInterface()
   */


STDMETHODIMP_(ULONG)
  OleStdEnumStatData_AddRef(LPENUMSTATDATA lpThis)
```

```
//------------------------------------------------------------------------------
--
//
//------------------------------------------------------------------------------
--
{
  LPOLESTDENUMSTATDATA lpSD = (LPOLESTDENUMSTATDATA)lpThis;
  return lpSD->m_dwRefs++;

} /* OleStdEnumStatData_AddRef()
   */


STDMETHODIMP_(ULONG)
  OleStdEnumStatData_Release(LPENUMSTATDATA lpThis)
//------------------------------------------------------------------------------
--
//
//------------------------------------------------------------------------------
--
{
  LPOLESTDENUMSTATDATA lpSD = (LPOLESTDENUMSTATDATA)lpThis;
  DWORD dwRefs = --lpSD->m_dwRefs;

  if (dwRefs == 0)
     OleStdEnumStatData_Destroy(lpSD);

  return dwRefs;

} /* OleStdEnumStatData_Release()
   */


STDMETHODIMP
  OleStdEnumStatData_Next(LPENUMSTATDATA lpThis, ULONG celt, LPSTATDATA
rgelt,
                                ULONG FAR* pceltFetched)
//------------------------------------------------------------------------------
--
//
//------------------------------------------------------------------------------
--
{
  LPOLESTDENUMSTATDATA lpSD = (LPOLESTDENUMSTATDATA)lpThis;
  ULONG i=0;
  ULONG nOffset;

  if (rgelt == NULL) {
     return ResultFromScode(E_INVALIDARG);
  }

  while (i < celt) {
     nOffset = lpSD->m_nIndex + i;

     if (nOffset < lpSD->m_nCount) {
       OleStdCopyStatData(
                  (LPSTATDATA)&(rgelt[i]), (LPSTATDATA)&(lpSD-
>m_lpStat[nOffset]));
       i++;
     }else{
       break;
     }
  }

  lpSD->m_nIndex += (WORD)i;

  if (pceltFetched != NULL) {
     *pceltFetched = i;
  }

  if (i != celt) {
     return ResultFromScode(S_FALSE);
  }

  return NOERROR;
} /* OleStdEnumStatData_Next()
   */


STDMETHODIMP
  OleStdEnumStatData_Skip(LPENUMSTATDATA lpThis, ULONG celt)
//---------------------------------------------------------------------------
--
//
//---------------------------------------------------------------------------
--
{
  LPOLESTDENUMSTATDATA lpSD = (LPOLESTDENUMSTATDATA)lpThis;
  ULONG i=0;
  ULONG nOffset;

  while (i < celt) {
     nOffset = lpSD->m_nIndex + i;

     if (nOffset < lpSD->m_nCount) {
       i++;
     }else{
       break;
     }
  }

  lpSD->m_nIndex += (WORD)i;

  if (i != celt) {
     return ResultFromScode(S_FALSE);
  }

  return NOERROR;
} /* OleStdEnumStatData_Skip()
   */
```

```
STDMETHODIMP
  OleStdEnumStatData_Reset(LPENUMSTATDATA lpThis)
//-----------------------------------------------------------------------------
--
//
//-----------------------------------------------------------------------------
--
{
  LPOLESTDENUMSTATDATA lpSD = (LPOLESTDENUMSTATDATA)lpThis;
  lpSD->m_nIndex = 0;

  return NOERROR;
} /* OleStdEnumStatData_Reset()
   */


STDMETHODIMP
  OleStdEnumStatData_Clone(LPENUMSTATDATA lpThis, LPENUMSTATDATA FAR*
ppenum)
//-----------------------------------------------------------------------------
--
//
//-----------------------------------------------------------------------------
--
{
  LPOLESTDENUMSTATDATA lpSD = (LPOLESTDENUMSTATDATA)lpThis;

  if (ppenum == NULL) {
     return ResultFromScode(E_INVALIDARG);
  }

  *ppenum = OleStdEnumStatData_Create(lpSD->m_nCount, lpSD->m_lpStat);

  // make sure cloned enumerator has same index state as the original
  if (*ppenum) {
       LPOLESTDENUMSTATDATA lpSDClone = (LPOLESTDENUMSTATDATA)*ppenum;
       lpSDClone->m_nIndex = lpSD->m_nIndex;
       return NOERROR;
  } else
       return ResultFromScode(E_OUTOFMEMORY);

} /* OleStdEnumStatData_Clone()
   */
```

GETICON.H (WRAPUI Sample)

```
// This file is now OBSOLETE (include olestd.h instead)
/*
 * GETICON.H
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */

/**************************************************************************
**
** The following API's are now OBSOLETE because equivalent API's have been
** added to the OLE2.DLL library
**      GetIconOfFile       superceeded by OleGetIconOfFile
**      GetIconOfClass      superceeded by OleGetIconOfClass
**      OleUIMetafilePictFromIconAndLabel
**                          superceeded by OleMetafilePictFromIconAndLabel
**************************************************************************/

// Other function prototypes moved to olestd.h
```

GETICON.C (WRAPUI Sample)

```
/*************************************************************************
**
** The following API's are now OBSOLETE because equivalent API's have been
** added to the OLE2.DLL library
**      GetIconOfFile       superceeded by OleGetIconOfFile
**      GetIconOfClass      superceeded by OleGetIconOfClass
**      OleUIMetafilePictFromIconAndLabel
**                          superceeded by OleMetafilePictFromIconAndLabel
*************************************************************************/

/*
 *  GETICON.C
 *
 *  Functions to create DVASPECT_ICON metafile from filename or classname.
 *
 *  GetIconOfFile
 *  GetIconOfClass
 *  OleUIMetafilePictFromIconAndLabel
 *  HIconAndSourceFromClass Extracts the first icon in a class's server path
 *                          and returns the path and icon index to caller.
 *  FIconFileFromClass      Retrieves the path to the exe/dll containing the
 *                           default icon, and the index of the icon.
 *  OleStdIconLabelTextOut  Draw icon label text (line break if necessary)
 *
 *    (c) Copyright Microsoft Corp. 1992-1993 All Rights Reserved
 */


/*******
 *
 * ICON (DVASPECT_ICON) METAFILE FORMAT:
 *
 * The metafile generated with OleUIMetafilePictFromIconAndLabel contains
 * the following records which are used by the functions in DRAWICON.C
 * to draw the icon with and without the label and to extract the icon,
 * label, and icon source/index.
 *
 *  SetWindowOrg
 *  SetWindowExt
 *  DrawIcon:
 *      Inserts records of DIBBITBLT or DIBSTRETCHBLT, once for the
 *      AND mask, one for the image bits.
 *  Escape with the comment "IconOnly"
 *      This indicates where to stop record enumeration to draw only
 *      the icon.
 *  SetTextColor
 *  SetTextAlign
 *  SetBkColor
 *  CreateFont
 *  SelectObject on the font.
 *  ExtTextOut
 *      One or more ExtTextOuts occur if the label is wrapped.  The
 *      text in these records is used to extract the label.
 *  SelectObject on the old font.
 *  DeleteObject on the font.
 *  Escape with a comment that contains the path to the icon source.
 *  Escape with a comment that is the ASCII of the icon index.
 *
 *******/

#define STRICT  1
#include "ole2ui.h"

#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <commdlg.h>
#include <memory.h>
#include <cderr.h>
#include "common.h"
#include "utility.h"

static char szSeparators[] = " \t\\/!:";

#define IS_SEPARATOR(c)         ( (c) == ' ' || (c) == '\\' || (c) == '/' ||
(c) == '\t' || (c) == '!' || (c) == ':' )
#define IS_FILENAME_DELIM(c)    ( (c) == '\\' || (c) == '/' || (c) == ':' )


#if defined( OBSOLETE )
static HINSTANCE  s_hInst;

static char szMaxWidth[] ="WWWWWWWWWW";

//Strings for metafile comments.
static char szIconOnly[]="IconOnly";        //Where to stop to exclude
label.


static char szOLE2DLL[] = "ole2.dll";   // name of OLE 2.0 library

#define ICONINDEX              0

#define AUXUSERTYPE_SHORTNAME  USERCLASSTYPE_SHORT  // short name
#define HIMETRIC_PER_INCH   2540      // number HIMETRIC units per inch
#define PTS_PER_INCH          72      // number points (font size) per inch

#define MAP_PIX_TO_LOGHIM(x,ppli)   MulDiv(HIMETRIC_PER_INCH, (x), (ppli))
#define MAP_LOGHIM_TO_PIX(x,ppli)   MulDiv((ppli), (x), HIMETRIC_PER_INCH)

static char szVanillaDocIcon[] = "DefIcon";

static char szDocument[40] = "";


/*
 * GetIconOfFile(HINSTANCE hInst, LPSTR lpszPath, BOOL fUseFileAsLabel)
 *
 * Purpose:
 *  Returns a hMetaPict containing an icon and label (filename) for the
 *  specified filename.
 *
 * Parameters:
 *  hinst
 *  lpszPath        LPSTR path including filename to use
 *  fUseFileAsLabel BOOL TRUE if the icon's label is the filename, FALSE if
 *                  there should be no label.
 *
 * Return Value:
 *  HGLOBAL         hMetaPict containing the icon and label - if there's no
 *                  class in reg db for the file in lpszPath, then we use
 *                  Document.  If lpszPath is NULL, then we return NULL.
 */

STDAPI_(HGLOBAL) GetIconOfFile(HINSTANCE hInst, LPSTR lpszPath, BOOL
fUseFileAsLabel)
{
  char     szIconFile[OLEUI_CCHPATHMAX];
  char     szLabel[OLEUI_CCHLABELMAX];
  LPSTR    lpszClsid = NULL;
  CLSID    clsid;
  HICON    hDefIcon = NULL;
  UINT     IconIndex = 0;
  HGLOBAL  hMetaPict;
  HRESULT  hResult;

  if (NULL == lpszPath)  // even if fUseFileAsLabel is FALSE, we still
     return NULL;             // need a valid filename to get the class.

  s_hInst = hInst;

  hResult = GetClassFile(lpszPath, &clsid);

  if (NOERROR == hResult)  // use the clsid we got to get to the icon
  {
      hDefIcon = HIconAndSourceFromClass(&clsid,

(LPSTR)szIconFile,
                                                                  &IconIndex);
  }

  if ( (NOERROR != hResult) || (NULL == hDefIcon) )
  {
      // Here, either GetClassFile failed or HIconAndSourceFromClass failed.

      LPSTR lpszTemp;

      lpszTemp = lpszPath;

      while ((*lpszTemp != '.') && (*lpszTemp != '\0'))
           lpszTemp++;
      if ('.' != *lpszTemp)
        goto UseVanillaDocument;


      if (FALSE == GetAssociatedExecutable(lpszTemp, (LPSTR)szIconFile))
        goto UseVanillaDocument;

      hDefIcon = ExtractIcon(s_hInst, szIconFile, IconIndex);
  }

  if (hDefIcon <= (HICON)1) // ExtractIcon returns 1 if szExecutable is not
exe,
  {                          // 0 if there are no icons.
UseVanillaDocument:

     lstrcpy((LPSTR)szIconFile, (LPSTR)szOLE2DLL);
     IconIndex = ICONINDEX;
     hDefIcon = ExtractIcon(s_hInst, szIconFile, IconIndex);

  }

  // Now let's get the label we want to use.

  if (fUseFileAsLabel)   // strip off path, so we just have the filename.
  {
      int istrlen;
      LPSTR lpszBeginFile;

      istrlen = lstrlen(lpszPath);

      // set pointer to END of path, so we can walk backwards through it.
      lpszBeginFile = lpszPath + istrlen -1;

      while ( (lpszBeginFile >= lpszPath)
                  && (!IS_FILENAME_DELIM(*lpszBeginFile)) )
       lpszBeginFile--;


      lpszBeginFile++;  // step back over the delimiter


      LSTRCPYN(szLabel, lpszBeginFile, sizeof(szLabel));
  }

  else   // use the short user type (AuxUserType2) for the label
  {

       if (0 == OleStdGetAuxUserType(&clsid, AUXUSERTYPE_SHORTNAME,
                                                 (LPSTR)szLabel,
OLEUI_CCHLABELMAX, NULL)) {

            if ('\0'==szDocument[0]) {
                  LoadString(
s_hInst,IDS_DEFICONLABEL,szDocument,sizeof(szDocument));
            }
            lstrcpy(szLabel, szDocument);
       }
  }


  hMetaPict = OleUIMetafilePictFromIconAndLabel(hDefIcon,

     szLabel,

     (LPSTR)szIconFile,

     IconIndex);

  DestroyIcon(hDefIcon);

  return hMetaPict;

}



/*
 * GetIconOfClass(HINSTANCE hInst, REFCLSID rclsid, LPSTR lpszLabel, BOOL
fUseTypeAsLabel)
 *
 * Purpose:
 *  Returns a hMetaPict containing an icon and label (human-readable form
 *  of class) for the specified clsid.
 *
 * Parameters:
 *  hinst
 *  rclsid          REFCLSID pointing to clsid to use.
 *  lpszLabel       label to use for icon.
 *  fUseTypeAsLabel Use the clsid's user type name as the icon's label.
 *
 * Return Value:
 *  HGLOBAL         hMetaPict containing the icon and label - if we
 *                  don't find the clsid in the reg db then we
 *                  return NULL.
 */

STDAPI_(HGLOBAL)    GetIconOfClass(HINSTANCE hInst, REFCLSID rclsid, LPSTR
lpszLabel, BOOL fUseTypeAsLabel)
{

  char    szLabel[OLEUI_CCHLABELMAX];
  char    szIconFile[OLEUI_CCHPATHMAX];
  HICON   hDefIcon;
  UINT    IconIndex;
  HGLOBAL hMetaPict;
  s_hInst = hInst;

  if (!fUseTypeAsLabel)  // Use string passed in as label
  {
     if (NULL != lpszLabel)
        LSTRCPYN(szLabel, lpszLabel, sizeof(szLabel));
     else
        *szLabel = '\0';
  }
  else   // Use AuxUserType2 (short name) as label
  {

       if (0 == OleStdGetAuxUserType(rclsid,
                                                     AUXUSERTYPE_SHORTNAME,
                                                     (LPSTR)szLabel,
                                                     OLEUI_CCHLABELMAX,
                                                     NULL))

        // If we can't get the AuxUserType2, then try the long name
        if (0 == OleStdGetUserTypeOfClass(rclsid, szLabel, OLEUI_CCHKEYMAX,
NULL)) {
            if ('\0'==szDocument[0]) {
                  LoadString(

s_hInst,IDS_DEFICONLABEL,szDocument,sizeof(szDocument));
            }
            lstrcpy(szLabel, szDocument);  // last resort
        }
  }

  // Get the icon, icon index, and path to icon file
  hDefIcon = HIconAndSourceFromClass(rclsid,
                        (LPSTR)szIconFile,
                        &IconIndex);

  if (NULL == hDefIcon)  // Use Vanilla Document
  {
     lstrcpy((LPSTR)szIconFile, (LPSTR)szOLE2DLL);
     IconIndex = ICONINDEX;
     hDefIcon = ExtractIcon(s_hInst, szIconFile, IconIndex);
  }

  // Create the metafile
  hMetaPict = OleUIMetafilePictFromIconAndLabel(hDefIcon, szLabel,

     (LPSTR)szIconFile, IconIndex);

  DestroyIcon(hDefIcon);

  return hMetaPict;

}


/*
```

```
 * OleUIMetafilePictFromIconAndLabel
 *
 * Purpose:
 *  Creates a METAFILEPICT structure that container a metafile in which
 *  the icon and label are drawn.  A comment record is inserted between
 *  the icon and the label code so our special draw function can stop
 *  playing before the label.
 *
 * Parameters:
 *  hIcon           HICON to draw into the metafile
 *  pszLabel        LPSTR to the label string.
 *  pszSourceFile   LPSTR containing the local pathname of the icon
 *                  as we either get from the user or from the reg DB.
 *  iIcon           UINT providing the index into pszSourceFile where
 *                  the icon came from.
 *
 * Return Value:
 *  HGLOBAL         Global memory handle containing a METAFILEPICT where
 *                  the metafile uses the MM_ANISOTROPIC mapping mode.  The
 *                  extents reflect both icon and label.
 */

STDAPI_(HGLOBAL) OleUIMetafilePictFromIconAndLabel(HICON hIcon, LPSTR
pszLabel
     , LPSTR pszSourceFile, UINT iIcon)
     {
     HDC             hDC, hDCScreen;
     HMETAFILE       hMF;
     HGLOBAL         hMem;
     LPMETAFILEPICT  pMF;
     UINT            cxIcon, cyIcon;
     UINT            cxText, cyText;
     UINT            cx, cy;
     UINT            cchLabel = 0;
     HFONT           hFont, hFontT;
     int             cyFont;
     char            szIndex[10];
     RECT            TextRect;
     SIZE            size;
     POINT           point;
     UINT            fuAlign;

     if (NULL==hIcon)  // null label is valid but NOT a null icon
           return NULL;

     //Create a memory metafile
     hDC=(HDC)CreateMetaFile(NULL);

     if (NULL==hDC)
           return NULL;

     //Allocate the metafilepict
     hMem=GlobalAlloc(GMEM_MOVEABLE | GMEM_DDESHARE, sizeof(METAFILEPICT));

     if (NULL==hMem)
```

```
      {
      hMF=CloseMetaFile(hDC);
      DeleteMetaFile(hMF);
      return NULL;
      }


if (NULL!=pszLabel)
      {
      cchLabel=lstrlen(pszLabel);

      if (cchLabel >= OLEUI_CCHLABELMAX)
         pszLabel[cchLabel] = '\0';   // truncate string
      }

//Need to use the screen DC for these operations
hDCScreen=GetDC(NULL);
cyFont=-(8*GetDeviceCaps(hDCScreen, LOGPIXELSY))/72;

//cyFont was calculated to give us 8 point.
hFont=CreateFont(cyFont, 5, 0, 0, FW_NORMAL, 0, 0, 0, ANSI_CHARSET
      , OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, PROOF_QUALITY
      , FF_SWISS, "MS Sans Serif");

hFontT=SelectObject(hDCScreen, hFont);

GetTextExtentPoint(hDCScreen,szMaxWidth,lstrlen(szMaxWidth),&size);
SelectObject(hDCScreen, hFontT);

cxText = size.cx;
cyText = size.cy * 2;

cxIcon = GetSystemMetrics(SM_CXICON);
cyIcon = GetSystemMetrics(SM_CYICON);


// If we have no label, then we want the metafile to be the width of
// the icon (plus margin), not the width of the fattest string.
if ( (NULL == pszLabel) || ('\0' == *pszLabel) )
      cx = cxIcon + cxIcon / 4;
else
      cx = max(cxText, cxIcon);

cy=cyIcon+cyText+4;

//Set the metafile size to fit the icon and label
SetWindowOrgEx(hDC, 0, 0, &point);
SetWindowExtEx(hDC, cx, cy, &size);

//Set up rectangle to pass to OleStdIconLabelTextOut
SetRectEmpty(&TextRect);

TextRect.right = cx;
TextRect.bottom = cy;
```

```
     //Draw the icon and the text, centered with respect to each other.
     DrawIcon(hDC, (cx-cxIcon)/2, 0, hIcon);

     //String that indicates where to stop if we're only doing icons
     Escape(hDC, MFCOMMENT, lstrlen(szIconOnly)+1, szIconOnly, NULL);

     SetTextColor(hDC, GetSysColor(COLOR_WINDOWTEXT));
     SetBkMode(hDC, TRANSPARENT);
     fuAlign = SetTextAlign(hDC, TA_LEFT | TA_TOP | TA_NOUPDATECP);

     OleStdIconLabelTextOut(hDC,
                                      hFont,
                                      0,
                                      cy - cyText,
                                      ETO_CLIPPED,
                                      &TextRect,
                                      pszLabel,
                                      cchLabel,
                                      NULL);

     //Write comments containing the icon source file and index.
     if (NULL!=pszSourceFile)
           {
           //+1 on string lengths insures the null terminator is embedded.
           Escape(hDC, MFCOMMENT, lstrlen(pszSourceFile)+1, pszSourceFile,
NULL);

           cchLabel=wsprintf(szIndex, "%u", iIcon);
           Escape(hDC, MFCOMMENT, cchLabel+1, szIndex, NULL);
           }

     SetTextAlign(hDC, fuAlign);

     //All done with the metafile, now stuff it all into a METAFILEPICT.
     hMF=CloseMetaFile(hDC);

     if (NULL==hMF)
           {
           GlobalFree(hMem);
           ReleaseDC(NULL, hDCScreen);
           return NULL;
           }

     //Fill out the structure
     pMF=(LPMETAFILEPICT)GlobalLock(hMem);

     //Transform to HIMETRICS
     cx=XformWidthInPixelsToHimetric(hDCScreen, cx);
     cy=XformHeightInPixelsToHimetric(hDCScreen, cy);
     ReleaseDC(NULL, hDCScreen);

     pMF->mm=MM_ANISOTROPIC;
     pMF->xExt=cx;
     pMF->yExt=cy;
     pMF->hMF=hMF;
     GlobalUnlock(hMem);

     DeleteObject(hFont);

     return hMem;
     }

#endif  // OBSOLETE


/*
 * GetAssociatedExecutable
 *
 * Purpose:  Finds the executable associated with the provided extension
 *
 * Parameters:
 *   lpszExtension   LPSTR points to the extension we're trying to find
 *                   an exe for. Does **NO** validation.
 *
 *   lpszExecutable  LPSTR points to where the exe name will be returned.
 *                   No validation here either - pass in 128 char buffer.
 *
 * Return:
 *   BOOL            TRUE if we found an exe, FALSE if we didn't.
 *
 */

BOOL FAR PASCAL GetAssociatedExecutable(LPSTR lpszExtension, LPSTR
lpszExecutable)

{
   HKEY    hKey;
   LONG    dw;
   LRESULT lRet;
   char    szValue[OLEUI_CCHKEYMAX];
   char    szKey[OLEUI_CCHKEYMAX];
   LPSTR   lpszTemp, lpszExe;


   lRet = RegOpenKey(HKEY_CLASSES_ROOT, NULL, &hKey);

   if (ERROR_SUCCESS != lRet)
       return FALSE;

   dw = OLEUI_CCHPATHMAX;
   lRet = RegQueryValue(hKey, lpszExtension, (LPSTR)szValue, &dw);  //ProgId

   if (ERROR_SUCCESS != lRet)
   {
       RegCloseKey(hKey);
       return FALSE;
   }
```

```
    // szValue now has ProgID
    lstrcpy(szKey, szValue);
    lstrcat(szKey, "\\Shell\\Open\\Command");


    dw = OLEUI_CCHPATHMAX;
    lRet = RegQueryValue(hKey, (LPSTR)szKey, (LPSTR)szValue, &dw);

    if (ERROR_SUCCESS != lRet)
    {
        RegCloseKey(hKey);
        return FALSE;
    }

    // szValue now has an executable name in it.  Let's null-terminate
    // at the first post-executable space (so we don't have cmd line
    // args.

    lpszTemp = (LPSTR)szValue;

    while (('\0' != *lpszTemp) && (isspace(*lpszTemp)))
        lpszTemp++;     // Strip off leading spaces

    lpszExe = lpszTemp;

    while (('\0' != *lpszTemp) && (!isspace(*lpszTemp)))
        lpszTemp++;     // Step through exe name

    *lpszTemp = '\0';  // null terminate at first space (or at end).


    lstrcpy(lpszExecutable, lpszExe);

    return TRUE;

}


/*
 * HIconAndSourceFromClass
 *
 * Purpose:
 *  Given an object class name, finds an associated executable in the
 *  registration database and extracts the first icon from that
 *  executable.  If none is available or the class has no associated
 *  executable, this function returns NULL.
 *
 * Parameters:
 *  rclsid          pointer to clsid to look up.
 *  pszSource       LPSTR in which to place the source of the icon.
 *                  This is assumed to be OLEUI_CCHPATHMAX
 *  puIcon          UINT FAR * in which to store the index of the
 *                  icon in pszSource.
 *
 * Return Value:
```

```
 *  HICON           Handle to the extracted icon if there is a module
 *                  associated to pszClass.  NULL on failure to either
 *                  find the executable or extract and icon.
 */

HICON WINAPI HIconAndSourceFromClass(REFCLSID rclsid, LPSTR pszSource, UINT
FAR *puIcon)
     {
     HICON           hIcon;
     UINT            IconIndex;

     if (NULL==rclsid || NULL==pszSource ||
IsEqualCLSID(rclsid,&CLSID_NULL))
           return NULL;

     if (!FIconFileFromClass(rclsid, pszSource, OLEUI_CCHPATHMAX,
&IconIndex))
           return NULL;

     hIcon=ExtractIcon(ghInst, pszSource, IconIndex);

     if ((HICON)32 > hIcon)
           hIcon=NULL;
     else
           *puIcon= IconIndex;

     return hIcon;
     }


/*
 * FIconFileFromClass
 *
 * Purpose:
 *  Looks up the path to executable that contains the class default icon.
 *
 * Parameters:
 *  rclsid          pointer to CLSID to look up.
 *  pszEXE          LPSTR at which to store the server name
 *  cch             UINT size of pszEXE
 *  lpIndex         LPUINT to index of icon within executable
 *
 * Return Value:
 *  BOOL            TRUE if one or more characters were loaded into pszEXE.
 *                  FALSE otherwise.
 */

BOOL WINAPI FIconFileFromClass(REFCLSID rclsid, LPSTR pszEXE, UINT cch, UINT
FAR *lpIndex)
{

     LONG          dw;
     LONG          lRet;
     HKEY          hKey;
     LPMALLOC      lpIMalloc;
     HRESULT       hrErr;
     LPSTR         lpBuffer;
     LPSTR         lpIndexString;
     UINT          cBufferSize = 136;// room for 128 char path and icon's
index
     char          szKey[64];
     LPSTR         pszClass;


     if (NULL==rclsid || NULL==pszEXE || 0==cch ||
IsEqualCLSID(rclsid,&CLSID_NULL))
           return FALSE;

     //Here, we use CoGetMalloc and alloc a buffer (maxpathlen + 8) to
     //pass to RegQueryValue.  Then, we copy the exe to pszEXE and the
     //index to *lpIndex.

     hrErr = CoGetMalloc(MEMCTX_TASK, &lpIMalloc);

     if (NOERROR != hrErr)
       return FALSE;

     lpBuffer = (LPSTR)lpIMalloc->lpVtbl->Alloc(lpIMalloc, cBufferSize);

     if (NULL == lpBuffer)
     {
       lpIMalloc->lpVtbl->Release(lpIMalloc);
       return FALSE;
     }


     if (CoIsOle1Class(rclsid))
     {

       LPSTR lpszProgID;

       // we've got an ole 1.0 class on our hands, so we look at
       // progID\protocol\stdfileedting\server to get the
       // name of the executable.

       ProgIDFromCLSID(rclsid, &lpszProgID);

       //Open up the class key
       lRet=RegOpenKey(HKEY_CLASSES_ROOT, lpszProgID, &hKey);

       if (ERROR_SUCCESS != lRet)
       {
            lpIMalloc->lpVtbl->Free(lpIMalloc, lpszProgID);
            lpIMalloc->lpVtbl->Free(lpIMalloc, lpBuffer);
            lpIMalloc->lpVtbl->Release(lpIMalloc);
            return FALSE;
       }

       dw=(LONG)cBufferSize;
```

```
        lRet = RegQueryValue(hKey, "Protocol\\StdFileEditing\\Server",
lpBuffer, &dw);

        if (ERROR_SUCCESS != lRet)
        {

             RegCloseKey(hKey);
             lpIMalloc->lpVtbl->Free(lpIMalloc, lpszProgID);
             lpIMalloc->lpVtbl->Free(lpIMalloc, lpBuffer);
             lpIMalloc->lpVtbl->Release(lpIMalloc);
             return FALSE;
        }


        // Use server and 0 as the icon index
        LSTRCPYN(pszEXE, lpBuffer, cch);

        *lpIndex = 0;

        RegCloseKey(hKey);
        lpIMalloc->lpVtbl->Free(lpIMalloc, lpszProgID);
        lpIMalloc->lpVtbl->Free(lpIMalloc, lpBuffer);
        lpIMalloc->lpVtbl->Release(lpIMalloc);
        return TRUE;

      }




      /*
       * We have to go walking in the registration database under the
       * classname, so we first open the classname key and then check
       * under "\\DefaultIcon" to get the file that contains the icon.
       */

      StringFromCLSID(rclsid, &pszClass);

      lstrcpy(szKey, "CLSID\\");
      lstrcat(szKey, pszClass);

      //Open up the class key
      lRet=RegOpenKey(HKEY_CLASSES_ROOT, szKey, &hKey);

      if (ERROR_SUCCESS != lRet)
      {
            lpIMalloc->lpVtbl->Free(lpIMalloc, lpBuffer);
            lpIMalloc->lpVtbl->Free(lpIMalloc, pszClass);
            lpIMalloc->lpVtbl->Release(lpIMalloc);
            return FALSE;
      }

      //Get the executable path and icon index.

      dw=(LONG)cBufferSize;
      lRet=RegQueryValue(hKey, "DefaultIcon", lpBuffer, &dw);
```

```
     if (ERROR_SUCCESS != lRet)
     {
       // no DefaultIcon  key...try LocalServer

       dw=(LONG)cBufferSize;
#ifdef WIN32
      lRet=RegQueryValue(hKey, "LocalServer32", lpBuffer, &dw);
#else
      lRet=RegQueryValue(hKey, "LocalServer", lpBuffer, &dw);
#endif

       if (ERROR_SUCCESS != lRet)
       {
            // no LocalServer entry either...they're outta luck.

            RegCloseKey(hKey);
            lpIMalloc->lpVtbl->Free(lpIMalloc, lpBuffer);
            lpIMalloc->lpVtbl->Free(lpIMalloc, pszClass);
            lpIMalloc->lpVtbl->Release(lpIMalloc);
            return FALSE;
       }


       // Use server from LocalServer or Server and 0 as the icon index
       LSTRCPYN(pszEXE, lpBuffer, cch);

       *lpIndex = 0;

       RegCloseKey(hKey);
       lpIMalloc->lpVtbl->Free(lpIMalloc, lpBuffer);
       lpIMalloc->lpVtbl->Free(lpIMalloc, pszClass);
       lpIMalloc->lpVtbl->Release(lpIMalloc);
       return TRUE;
     }

     RegCloseKey(hKey);

     // lpBuffer contains a string that looks like
"<pathtoexe>,<iconindex>",
     // so we need to separate the path and the icon index.

     lpIndexString = PointerToNthField(lpBuffer, 2, ',');

     if ('\0' == *lpIndexString)  // no icon index specified - use 0 as
default.
     {
        *lpIndex = 0;

     }
     else
     {
        LPSTR lpTemp;
        static char  szTemp[16];
        lstrcpy((LPSTR)szTemp, lpIndexString);

        // Put the icon index part into *pIconIndex
        *lpIndex = atoi((const char *)szTemp);

        // Null-terminate the exe part.
        lpTemp = AnsiPrev(lpBuffer, lpIndexString);
        *lpTemp = '\0';
     }

     if (!LSTRCPYN(pszEXE, lpBuffer, cch))
     {
        lpIMalloc->lpVtbl->Free(lpIMalloc, lpBuffer);
        lpIMalloc->lpVtbl->Free(lpIMalloc, pszClass);
        lpIMalloc->lpVtbl->Release(lpIMalloc);
        return FALSE;
     }

     // Free the memory we alloc'd and leave.
     lpIMalloc->lpVtbl->Free(lpIMalloc, lpBuffer);
     lpIMalloc->lpVtbl->Free(lpIMalloc, pszClass);
     lpIMalloc->lpVtbl->Release(lpIMalloc);
     return TRUE;
}



/*
 * OleStdIconLabelTextOut
 *
 * Purpose:
 *  Replacement for DrawText to be used in the "Display as Icon" metafile.
 *  Uses ExtTextOut to output a string center on (at most) two lines.
 *  Uses a very simple word wrap algorithm to split the lines.
 *
 * Parameters:  (same as for ExtTextOut, except for hFont)
 *  hDC           device context to draw into; if this is NULL, then we
don't
 *                ETO the text, we just return the index of the beginning
 *                of the second line
 *  hFont         font to use
 *  nXStart       x-coordinate of starting position
 *  nYStart       y-coordinate of starting position
 *  fuOptions     rectangle type
 *  lpRect        rect far * containing rectangle to draw text in.
 *  lpszString    string to draw
 *  cchString     length of string (truncated if over OLEUI_CCHLABELMAX)
 *  lpDX          spacing between character cells
 *
 * Return Value:
 *  UINT          Index of beginning of last line (0 if there's only one
 *                line of text).
 *
 */
```

```
STDAPI_(UINT) OleStdIconLabelTextOut(HDC        hDC,
                                                      HFONT      hFont,
                                                      int        nXStart,
                                                      int        nYStart,
                                                      UINT       fuOptions,
                                                      RECT FAR * lpRect,
                                                      LPSTR      lpszString,
                                                      UINT       cchString,
                                                      int FAR *  lpDX)
{

  HDC          hDCScreen;
  static char  szTempBuff[OLEUI_CCHLABELMAX];
  int          cxString, cyString, cxMaxString;
  int          cxFirstLine, cyFirstLine, cxSecondLine;
  int          index;
  int          cch = cchString;
  char         chKeep;
  LPSTR        lpszSecondLine;
  HFONT        hFontT;
  BOOL         fPrintText = TRUE;
  UINT         iLastLineStart = 0;
  SIZE         size;

  // Initialization stuff...

  if (NULL == hDC)  // If we got NULL as the hDC, then we don't actually
call ETO
     fPrintText = FALSE;


  // Make a copy of the string (NULL or non-NULL) that we're using
  if (NULL == lpszString)
     *szTempBuff = '\0';

  else
     LSTRCPYN(szTempBuff, lpszString, sizeof(szTempBuff));

  // set maximum width
  cxMaxString = lpRect->right - lpRect->left;

  // get screen DC to do text size calculations
  hDCScreen = GetDC(NULL);

  hFontT=SelectObject(hDCScreen, hFont);

  // get the extent of our label
  GetTextExtentPoint(hDCScreen, szTempBuff, cch, &size);

  cxString = size.cx;
  cyString = size.cy;

  // Select in the font we want to use
  if (fPrintText)
      SelectObject(hDC, hFont);
```

```
   // String is smaller than max string - just center, ETO, and return.
   if (cxString <= cxMaxString)
   {

      if (fPrintText)
         ExtTextOut(hDC,
                           nXStart + (lpRect->right - cxString) / 2,
                           nYStart,
                           fuOptions,
                           lpRect,
                           szTempBuff,
                           cch,
                           NULL);

      iLastLineStart = 0;  // only 1 line of text
      goto CleanupAndLeave;
   }

   // String is too long...we've got to word-wrap it.


   // Are there any spaces, slashes, tabs, or bangs in string?

   if (lstrlen(szTempBuff) != (int)strcspn(szTempBuff, szSeparators))
   {
       // Yep, we've got spaces, so we'll try to find the largest
       // space-terminated string that will fit on the first line.

       index = cch;


       while (index >= 0)
       {

         char cchKeep;

         // scan the string backwards for spaces, slashes, tabs, or bangs

         while (!IS_SEPARATOR(szTempBuff[index]) )
             index--;


         if (index <= 0)
             break;

         cchKeep = szTempBuff[index];  // remember what char was there

         szTempBuff[index] = '\0';  // just for now

         GetTextExtentPoint(
                     hDCScreen,
(LPSTR)szTempBuff,lstrlen((LPSTR)szTempBuff),&size);

         cxFirstLine = size.cx;
         cyFirstLine = size.cy;

         szTempBuff[index] = cchKeep;   // put the right char back

         if (cxFirstLine <= cxMaxString)
         {

               iLastLineStart = index + 1;

               if (!fPrintText)
                   goto CleanupAndLeave;

               ExtTextOut(hDC,
                                nXStart +  (lpRect->right - cxFirstLine) / 2,
                                nYStart,
                                fuOptions,
                                lpRect,
                                (LPSTR)szTempBuff,
                                index + 1,
                                lpDX);

               lpszSecondLine = (LPSTR)szTempBuff;

               lpszSecondLine += index + 1;

               GetTextExtentPoint(hDCScreen,
                                                      lpszSecondLine,

      lstrlen(lpszSecondLine),
                                                      &size);

               // If the second line is wider than the rectangle, we
               // just want to clip the text.
               cxSecondLine = min(size.cx, cxMaxString);

               ExtTextOut(hDC,
                                nXStart + (lpRect->right - cxSecondLine) / 2,
                                nYStart + cyFirstLine,
                                fuOptions,
                                lpRect,
                                lpszSecondLine,
                                lstrlen(lpszSecondLine),
                                lpDX);

               goto CleanupAndLeave;

         }  // end if

         index--;

       }  // end while

   }  // end if

   // Here, there are either no spaces in the string (strchr(szTempBuff, ' ')
   // returned NULL), or there spaces in the string, but they are
   // positioned so that the first space terminated string is still
   // longer than one line. So, we walk backwards from the end of the
   // string until we find the largest string that will fit on the first
   // line , and then we just clip the second line.

   cch = lstrlen((LPSTR)szTempBuff);

   chKeep = szTempBuff[cch];
   szTempBuff[cch] = '\0';

   GetTextExtentPoint(hDCScreen, szTempBuff, lstrlen(szTempBuff),&size);

   cxFirstLine = size.cx;
   cyFirstLine = size.cy;

   while (cxFirstLine > cxMaxString)
   {
       // We allow 40 characters in the label, but the metafile is
       // only as wide as 10 W's (for aesthetics - 20 W's wide looked
       // dumb.  This means that if we split a long string in half (in
       // terms of characters), then we could still be wider than the
       // metafile.  So, if this is the case, we just step backwards
       // from the halfway point until we get something that will fit.
       // Since we just let ETO clip the second line

       szTempBuff[cch--] = chKeep;
       if (0 == cch)
         goto CleanupAndLeave;

       chKeep = szTempBuff[cch];
       szTempBuff[cch] = '\0';

       GetTextExtentPoint(
                   hDCScreen, szTempBuff, lstrlen(szTempBuff), &size);
       cxFirstLine = size.cx;
   }

   iLastLineStart = cch;

   if (!fPrintText)
      goto CleanupAndLeave;

   ExtTextOut(hDC,
                   nXStart + (lpRect->right - cxFirstLine) / 2,
                   nYStart,
                   fuOptions,
                   lpRect,
                   (LPSTR)szTempBuff,
                   lstrlen((LPSTR)szTempBuff),
                   lpDX);

   szTempBuff[cch] = chKeep;
   lpszSecondLine = szTempBuff;
   lpszSecondLine += cch;
```

```
  GetTextExtentPoint(
             hDCScreen, (LPSTR)lpszSecondLine, lstrlen(lpszSecondLine),
&size);

  // If the second line is wider than the rectangle, we
  // just want to clip the text.
  cxSecondLine = min(size.cx, cxMaxString);

  ExtTextOut(hDC,
                  nXStart + (lpRect->right - cxSecondLine) / 2,
                  nYStart + cyFirstLine,
                  fuOptions,
                  lpRect,
                  lpszSecondLine,
                  lstrlen(lpszSecondLine),
                  lpDX);

CleanupAndLeave:
  SelectObject(hDCScreen, hFontT);
  ReleaseDC(NULL, hDCScreen);
  return iLastLineStart;

}



/*
 * PointerToNthField
 *
 * Purpose:
 *  Returns a pointer to the beginning of the nth field.
 *  Assumes null-terminated string.
 *
 * Parameters:
 *  lpszString        string to parse
 *  nField            field to return starting index of.
 *  chDelimiter       char that delimits fields
 *
 * Return Value:
 *  LPSTR             pointer to beginning of nField field.
 *                    NOTE: If the null terminator is found
 *                          Before we find the Nth field, then
 *                          we return a pointer to the null terminator -
 *                          calling app should be sure to check for
 *                          this case.
 *
 */
LPSTR WINAPI PointerToNthField(LPSTR lpszString, int nField, char
chDelimiter)
{
   LPSTR lpField = lpszString;
   int   cFieldFound = 1;

   if (1 ==nField)
```

```
       return lpszString;

   while (*lpField != '\0')
   {

       if (*lpField++ == chDelimiter)
       {

            cFieldFound++;

            if (nField == cFieldFound)
                 return lpField;
       }
   }

   return lpField;

}
```

HATCH.C (WRAPUI Sample)

```
/*
 * HATCH.C
 *
 * Miscellaneous API's to generate hatch window for in-place active
 * objects. This is part of the OLE 2.0 User Interface Support Library.
 *
 * Copyright (c)1993 Microsoft Corporation, All Right Reserved
 */

#define STRICT  1
#include "ole2ui.h"

// offsets in the extra bytes stored with the hatch window
#define EB_HATCHWIDTH       0
#define EB_HATCHRECT_LEFT   2
#define EB_HATCHRECT_TOP    4
#define EB_HATCHRECT_RIGHT  6
#define EB_HATCHRECT_BOTTOM 8

// class name of hatch window
#define CLASS_HATCH     "Hatch Window"

// local function prototypes
LRESULT WINAPI EXPORT HatchWndProc(HWND hWnd, UINT Message, WPARAM wParam,
LPARAM lParam);


/*
 * HatchRegisterClass
 *
 * Purpose:
 *  Register the hatch window
 *
 * Parameters:
 *  hInst           Process instance
 *
 * Return Value:
 *  TRUE            if successful
 *  FALSE           if failed
 *
 */
STDAPI_(BOOL) RegisterHatchWindowClass(HINSTANCE hInst)
{
     WNDCLASS wc;

     // Register Hatch Window Class
     wc.style = CS_BYTEALIGNWINDOW;
     wc.lpfnWndProc = HatchWndProc;
     wc.cbClsExtra = 0;
     wc.cbWndExtra = 5 * sizeof(int);    // extra bytes stores
                                                           //
uHatchWidth
```

```
                                                                      //
rcHatchRect
     wc.hInstance = hInst;
     wc.hIcon = NULL;
     wc.hCursor = LoadCursor(NULL, IDC_ARROW);
     wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
     wc.lpszMenuName = NULL;
     wc.lpszClassName = CLASS_HATCH;

     if (!RegisterClass(&wc))
           return FALSE;
     else
           return TRUE;
}


/*
 * CreateHatchWindow
 *
 * Purpose:
 *  Create the hatch window
 *
 * Parameters:
 *  hWndParent          parent of hatch window
 *  hInst               instance handle
 *
 * Return Value:
 *  pointer to hatch window         if successful
 *  NULL                            if failed
 *
 */
STDAPI_(HWND) CreateHatchWindow(HWND hWndParent, HINSTANCE hInst)
{
     HWND         hWnd;

     if (!hWndParent || !hInst)
           return NULL;

     hWnd = CreateWindow(
           CLASS_HATCH,
           "Hatch Window",
           WS_CHILDWINDOW | WS_CLIPCHILDREN | WS_CLIPSIBLINGS,
           0, 0, 0, 0,
           hWndParent,
           (HMENU)NULL,
           hInst,
           0L
     );

     if (!hWnd)
           return NULL;

     return hWnd;
}
```

```
/*
 *  GetHatchWidth
 *
 *  Purpose:
 *      Get width of hatch border
 *
 *  Parameters:
 *      hWndHatch       hatch window handle
 *
 *  Return Value:
 *      width of the hatch border
 */
STDAPI_(UINT) GetHatchWidth(HWND hWndHatch)
{
     if (!IsWindow(hWndHatch))
           return 0;

     return (UINT)GetWindowWord(hWndHatch, EB_HATCHWIDTH);
}

/*
 *  GetHatchRect
 *
 *  Purpose:
 *      Get hatch rect. this is the size of the hatch window if it were
 *      not clipped by the ClipRect.
 *
 *  Parameters:
 *      hWndHatch       hatch window handle
 *      lprcHatchRect   hatch rect
 *
 *  Return Value:
 *      none
 */
STDAPI_(void) GetHatchRect(HWND hWndHatch, LPRECT lprcHatchRect)
{
     if (!IsWindow(hWndHatch)) {
           SetRect(lprcHatchRect, 0, 0, 0, 0);
           return;
     }

     lprcHatchRect->left = GetWindowWord(hWndHatch, EB_HATCHRECT_LEFT);
     lprcHatchRect->top = GetWindowWord(hWndHatch, EB_HATCHRECT_TOP);
     lprcHatchRect->right = GetWindowWord(hWndHatch, EB_HATCHRECT_RIGHT);
     lprcHatchRect->bottom = GetWindowWord(hWndHatch, EB_HATCHRECT_BOTTOM);
}


/* SetHatchRect
 *
 *
 *  Purpose:
 *      Store hatch rect with HatchRect window.
 *      this rect is the size of the hatch window if it were
 *      not clipped by the ClipRect.
 *
 *  Parameters:
 *      hWndHatch       hatch window handle
 *      lprcHatchRect   hatch rect
 *
 *  Return Value:
 *      none
 */
STDAPI_(void) SetHatchRect(HWND hWndHatch, LPRECT lprcHatchRect)
{
     if (!IsWindow(hWndHatch))
           return;

     SetWindowWord(hWndHatch, EB_HATCHRECT_LEFT,  (WORD)lprcHatchRect-
>left);
     SetWindowWord(hWndHatch, EB_HATCHRECT_TOP,   (WORD)lprcHatchRect->top);
     SetWindowWord(hWndHatch, EB_HATCHRECT_RIGHT, (WORD)lprcHatchRect-
>right);
     SetWindowWord(hWndHatch, EB_HATCHRECT_BOTTOM,(WORD)lprcHatchRect-
>bottom);
}


/* SetHatchWindowSize
 *
 *
 *  Purpose:
 *      Move/size the HatchWindow correctly given the rect required by the
 *      in-place server object window and the lprcClipRect imposed by the
 *      in-place container. both rect's are expressed in the client coord.
 *      of the in-place container's window (which is the parent of the
 *      HatchWindow).
 *
 *      OLE2NOTE: the in-place server must honor the lprcClipRect specified
 *      by its in-place container. it must NOT draw outside of the ClipRect.
 *      in order to achieve this, the hatch window is sized to be
 *      exactly the size that should be visible (rcVisRect). the
 *      rcVisRect is defined as the intersection of the full size of
 *      the HatchRect window and the lprcClipRect.
 *      the ClipRect could infact clip the HatchRect on the
 *      right/bottom and/or on the top/left. if it is clipped on the
 *      right/bottom then it is sufficient to simply resize the hatch
 *      window. but if the HatchRect is clipped on the top/left then
 *      in-place server document window (child of HatchWindow) must be moved
 *      by the delta that was clipped. the window origin of the
 *      in-place server window will then have negative coordinates relative
 *      to its parent HatchWindow.
 *
 *  Parameters:
 *      hWndHatch       hatch window handle
 *      lprcIPObjRect   full size of in-place server object window
 *      lprcClipRect    clipping rect imposed by in-place container
 *      lpptOffset      offset required to position in-place server object
 *                      window properly. caller should call:
```

```
 *                          OffsetRect(&rcObjRect,lpptOffset->x,lpptOffset-
>y)
 *
 *  Return Value:
 *      none
 */
STDAPI_(void) SetHatchWindowSize(
           HWND        hWndHatch,
           LPRECT      lprcIPObjRect,
           LPRECT      lprcClipRect,
           LPPOINT       lpptOffset
)
{
     RECT        rcHatchRect;
     RECT        rcVisRect;
     UINT        uHatchWidth;
     POINT       ptOffset;

     if (!IsWindow(hWndHatch))
           return;

     rcHatchRect = *lprcIPObjRect;
     uHatchWidth = GetHatchWidth(hWndHatch);
     InflateRect((LPRECT)&rcHatchRect, uHatchWidth + 1, uHatchWidth + 1);

     IntersectRect((LPRECT)&rcVisRect, (LPRECT)&rcHatchRect, lprcClipRect);
     MoveWindow(
                 hWndHatch,
                 rcVisRect.left,
                 rcVisRect.top,
                 rcVisRect.right-rcVisRect.left,
                 rcVisRect.bottom-rcVisRect.top,
                 TRUE    /* fRepaint */
     );
     InvalidateRect(hWndHatch, NULL, TRUE);

     ptOffset.x = -rcHatchRect.left + (rcHatchRect.left - rcVisRect.left);
     ptOffset.y = -rcHatchRect.top + (rcHatchRect.top - rcVisRect.top);

     /* convert the rcHatchRect into the client coordinate system of the
     **    HatchWindow itself
     */
     OffsetRect((LPRECT)&rcHatchRect, ptOffset.x, ptOffset.y);

     SetHatchRect(hWndHatch, (LPRECT)&rcHatchRect);

     // calculate offset required to position in-place server doc window
     lpptOffset->x = ptOffset.x;
     lpptOffset->y = ptOffset.y;
}


/*
 *  HatchWndProc
 *
 *  Purpose:
 *      WndProc for hatch window
 *
 *  Parameters:
 *      hWnd
 *      Message
 *      wParam
 *      lParam
 *
 *  Return Value:
 *      message dependent
 */
LRESULT WINAPI EXPORT HatchWndProc(HWND hWnd, UINT Message, WPARAM wParam,
LPARAM lParam)
{
     int nBorderWidth;

     switch (Message) {

           case WM_CREATE:
                 nBorderWidth = GetProfileInt(
                       "windows",
                       "oleinplaceborderwidth",
                       DEFAULT_HATCHBORDER_WIDTH
                 );
                 SetWindowWord(hWnd, EB_HATCHWIDTH, (WORD)nBorderWidth);
                 break;

           case WM_PAINT:
           {
                 HDC hDC;
                 PAINTSTRUCT ps;
                 RECT rcHatchRect;

                 nBorderWidth = GetHatchWidth(hWnd);
                 hDC = BeginPaint(hWnd, &ps);
                 GetHatchRect(hWnd, (LPRECT)&rcHatchRect);
                 OleUIDrawShading(&rcHatchRect, hDC, OLEUI_SHADE_BORDERIN,
                             nBorderWidth);
                 InflateRect((LPRECT)&rcHatchRect, -nBorderWidth,
-nBorderWidth);
                 OleUIDrawHandles(&rcHatchRect, hDC, OLEUI_HANDLES_OUTSIDE,
                             nBorderWidth+1, TRUE);
                 EndPaint(hWnd, &ps);
                 break;
           }

           /* OLE2NOTE: Any window that is used during in-place activation
           **    must handle the WM_SETCURSOR message or else the cursor
           **    of the in-place parent will be used. if WM_SETCURSOR is
           **    not handled, then DefWindowProc sends the message to the
           **    window's parent.
           */
           case WM_SETCURSOR:
                 SetCursor(LoadCursor( NULL, MAKEINTRESOURCE(IDC_ARROW) ) );
                 return (LRESULT)TRUE;

           default:
                 return DefWindowProc(hWnd, Message, wParam, lParam);
     }

     return 0L;
}
```

ICON.H   (WRAPUI Sample)

```
/*
 * ICON.H
 *
 * Internal definitions, structures, and function prototypes for the
 * OLE 2.0 UI Change Icon dialog.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */


#ifndef _ICON_H_
#define _ICON_H_

#ifndef RC_INVOKED
#pragma message ("INCLUDING ICON.H from " __FILE__)
#endif  /* RC_INVOKED */

#define CXICONPAD                   12
#define CYICONPAD                   4

// Property used by ChangeIcon dialog to give its parent window access to
// its hDlg. The PasteSpecial dialog may need to force the ChgIcon dialog
// down if the clipboard contents change underneath it. if so it will send
// a IDCANCEL command to the ChangeIcon dialog.
#define PROP_HWND_CHGICONDLG    "HWND_CIDLG"

//Internally used structure
typedef struct tagCHANGEICON
     {
     LPOLEUICHANGEICON   lpOCI;      //Original structure passed.

     /*
      * What we store extra in this structure besides the original caller's
      * pointer are those fields that we need to modify during the life of
      * the dialog but that we don't want to change in the original
structure
      * until the user presses OK.
      */
     DWORD               dwFlags;
     HICON               hCurIcon;
     char                szLabel[OLEUI_CCHLABELMAX+1];
     char                szFile[OLEUI_CCHPATHMAX];
     UINT                iIcon;
     HICON               hDefIcon;
     char                szDefIconFile[OLEUI_CCHPATHMAX];
     UINT                iDefIcon;
     UINT                nBrowseHelpID;      // Help ID callback for Browse
dlg
     } CHANGEICON, *PCHANGEICON, FAR *LPCHANGEICON;


//Internal function prototypes
```

```
//ICON.C
BOOL CALLBACK EXPORT ChangeIconDialogProc(HWND, UINT, WPARAM, LPARAM);
BOOL            FChangeIconInit(HWND, WPARAM, LPARAM);
UINT            UFillIconList(HWND, UINT, LPSTR);
BOOL            FDrawListIcon(LPDRAWITEMSTRUCT);
void            UpdateResultIcon(LPCHANGEICON, HWND, UINT);


#endif //_ICON_H_
```

ICON.C   (WRAPUI Sample)

```
/*
 * ICON.C
 *
 * Implements the OleUIChangeIcon function which invokes the complete
 * Change Icon dialog.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */

#define STRICT  1
#include "ole2ui.h"
#include "common.h"
#include "utility.h"
#include "icon.h"
#include "geticon.h"

/*
 * OleUIChangeIcon
 *
 * Purpose:
 *  Invokes the standard OLE Change Icon dialog box allowing the user
 *  to select an icon from an icon file, executable, or DLL.
 *
 * Parameters:
 *  lpCI            LPOLEUIChangeIcon pointing to the in-out structure
 *                  for this dialog.
 *
 * Return Value:
 *  UINT            OLEUI_SUCCESS or OLEUI_OK if all is well, otherwise
 *                  an error value.
 */

STDAPI_(UINT) OleUIChangeIcon(LPOLEUICHANGEICON lpCI)
     {
     UINT        uRet;
     HGLOBAL     hMemDlg=NULL;

     uRet=UStandardValidation((LPOLEUISTANDARD)lpCI, sizeof(OLEUICHANGEICON)
                                         , &hMemDlg);

     if (OLEUI_SUCCESS!=uRet)
           return uRet;

#if defined( OBSOLETE )
     if (NULL==lpCI->hMetaPict)
           uRet=OLEUI_CIERR_MUSTHAVECURRENTMETAFILE;
#endif

     if (lpCI->dwFlags & CIF_USEICONEXE)
     {
       if (   (NULL == lpCI->szIconExe)
            || (IsBadReadPtr(lpCI->szIconExe, lpCI->cchIconExe))
```

```
             || (IsBadWritePtr(lpCI->szIconExe, lpCI->cchIconExe)) )
            uRet = OLEUI_CIERR_SZICONEXEINVALID;

     }

 // REVIEW: how do we validate the CLSID?
/*
     if ('\0'==*((LPSTR)&lpCI->clsid))
           uRet=OLEUI_CIERR_MUSTHAVECLSID;
*/
     if (OLEUI_ERR_STANDARDMIN <= uRet)
           {
           if (NULL!=hMemDlg)
                 FreeResource(hMemDlg);

           return uRet;
           }

     //Now that we've validated everything, we can invoke the dialog.
     return UStandardInvocation(ChangeIconDialogProc, (LPOLEUISTANDARD)lpCI
                                            , hMemDlg,
MAKEINTRESOURCE(IDD_CHANGEICON));
     }





/*
 * ChangeIconDialogProc
 *
 * Purpose:
 *  Implements the OLE Change Icon dialog as invoked through the
 *  OleUIChangeIcon function.
 *
 * Parameters:
 *  Standard
 *
 * Return Value:
 *  Standard
 */

BOOL CALLBACK EXPORT ChangeIconDialogProc(HWND hDlg, UINT iMsg
                                                      , WPARAM wParam,
LPARAM lParam)
     {
     LPCHANGEICON            lpCI;
     HICON                   hIcon;
     HGLOBAL                 hMetaPict;
     BOOL                    fOK=FALSE;
     UINT                    uRet=0;
     LPSTR                   psz;
     char                    szTemp[OLEUI_CCHPATHMAX];

     //Declare Win16/Win32 compatible WM_COMMAND parameters.
```

```
    COMMANDPARAMS(wID, wCode, hWndMsg);

    lpCI=(LPCHANGEICON)LpvStandardEntry(hDlg, iMsg, wParam, lParam, &uRet);

    //If the hook processed the message, we're done.
    if (0!=uRet)
        return uRet;

    //Process the temination message
    if (iMsg==uMsgEndDialog)
        {
        //Insure that icons are properly destroyed.
        SendDlgItemMessage(hDlg, ID_ICONLIST, LB_RESETCONTENT, 0, 0L);

        StandardCleanup(lpCI, hDlg);
        EndDialog(hDlg, wParam);
        return TRUE;
        }

    switch (iMsg)
        {
        case WM_INITDIALOG:
            FChangeIconInit(hDlg, wParam, lParam);
            return TRUE;


        case WM_MEASUREITEM:
            {
            LPMEASUREITEMSTRUCT     lpMI=(LPMEASUREITEMSTRUCT)lParam;

            //All icons are system metric+padding in width and height
            lpMI->itemWidth =GetSystemMetrics(SM_CXICON)+CXICONPAD;
            lpMI->itemHeight=GetSystemMetrics(SM_CYICON)+CYICONPAD;
            }
            break;


        case WM_DRAWITEM:
            return FDrawListIcon((LPDRAWITEMSTRUCT)lParam);


        case WM_DELETEITEM:
            //Free the GDI object for the item
            DestroyIcon((HICON)LOWORD(((LPDELETEITEMSTRUCT)lParam)-
>itemData));
            break;


        case WM_COMMAND:
            switch (wID)
                {
                case ID_CURRENT:
                case ID_DEFAULT:
                case ID_FROMFILE:
                    UpdateResultIcon(lpCI, hDlg, wID);
```

```
                             break;

                       case ID_LABELEDIT:
                             //When the edit loses focus, update the result
display
                             if (EN_KILLFOCUS==wCode)
                                   {
                                   GetDlgItemText(hDlg, ID_LABELEDIT,
szTemp, sizeof(szTemp));
                                   SetDlgItemText(hDlg, ID_RESULTLABEL,
szTemp);
                                   }
                             break;

                       case ID_FROMFILEEDIT:
                             //If the text changed, remove any selection in
the list.
                             GetDlgItemText(hDlg, ID_FROMFILEEDIT, szTemp,
sizeof(szTemp));

                             if (lpCI && lstrcmpi(szTemp, lpCI->szFile))
                                   {
                                   SendDlgItemMessage(hDlg, ID_ICONLIST,
LB_SETCURSEL
                                                              , (WPARAM)-1,
0);

                                   //Also force selection of ID_FROMFILE
                                   CheckRadioButton(hDlg, ID_CURRENT,
ID_FROMFILE, ID_FROMFILE);
                                   }
                             break;


                       case ID_ICONLIST:
                             switch (wCode)
                                   {
                                   case LBN_SETFOCUS:
                                         //If we got the focus, see about
updating.
                                         GetDlgItemText(hDlg,
ID_FROMFILEEDIT, szTemp
                                                              ,
sizeof(szTemp));

                                         //Check if file changed and update
the list if so
                                         if (lpCI && 0!=lstrcmpi(szTemp,
lpCI->szFile))
                                               {
                                               lstrcpy(lpCI->szFile,
szTemp);
                                               UFillIconList(hDlg,
ID_ICONLIST, lpCI->szFile);
```

```
                                                UpdateResultIcon(lpCI, hDlg,
ID_FROMFILE);
                                                }
                                          break;

                                    case LBN_SELCHANGE:
                                          UpdateResultIcon(lpCI, hDlg,
ID_FROMFILE);
                                          break;

                                    case LBN_DBLCLK:
                                          //Same as pressing OK.
                                          SendCommand(hDlg, IDOK, BN_CLICKED,
hWndMsg);
                                          break;
                                    }
                              break;


                        case ID_BROWSE:
                        {
                              DWORD dwOfnFlags;

                              /*
                               * To allow the hook to customize the browse
dialog, we
                               * send OLEUI_MSG_BROWSE.  If the hook returns
FALSE
                               * we use the default, otherwise we trust that
it retrieved
                               * a filename for us.  This mechanism prevents
hooks from
                               * trapping ID_BROWSE to customize the dialog
and from
                               * trying to figure out what we do after we
have the name.
                               */

                              //Copy for reference
                              LSTRCPYN(szTemp, lpCI->szFile, sizeof(szTemp));

                              uRet=UStandardHook(lpCI, hDlg, uMsgBrowse,
OLEUI_CCHPATHMAX
                                                            , (LONG)(LPSTR)lpCI-
>szFile);

                              dwOfnFlags = OFN_FILEMUSTEXIST;
                              if (lpCI->lpOCI->dwFlags & CIF_SHOWHELP)
                                 dwOfnFlags |= OFN_SHOWHELP;

                              if (0==uRet)
                                    uRet=(BOOL)Browse(hDlg, lpCI->szFile,
NULL, OLEUI_CCHPATHMAX, (UINT)IDS_ICONFILTERS, (UINT)IDS_BROWSE,
dwOfnFlags);
```

```
                             /*
                              * Only reinitialize if the file changed, so if
we got
                              * TRUE from the hook but the user hit Cancel,
we don't
                              * spend time unecessarily refilling the list.
                              */
                             if (0!=uRet && 0!=lstrcmpi(szTemp, lpCI-
>szFile))
                             {
                                   CheckRadioButton(hDlg, ID_CURRENT,
ID_FROMFILE, ID_FROMFILE);
                                   SetDlgItemText(hDlg, ID_FROMFILEEDIT,
lpCI->szFile);
                                   UFillIconList(hDlg, ID_ICONLIST, lpCI-
>szFile);
                                   UpdateResultIcon(lpCI, hDlg,
ID_FROMFILE);
                             }
                       }
                       break;


                       case IDOK:
                             /*
                              * If the user pressed enter, compare the
current file
                              * and the one we have stored.  If they match,
then
                              * refill the listbox instead of closing down.
This is
                              * so the user can press Enter in the edit
control as
                              * they would expect to be able to do.
                              */
                             GetDlgItemText(hDlg, ID_FROMFILEEDIT, szTemp,
sizeof(szTemp));

                             //Check if the file changed at all.
                             if (0!=lstrcmpi(szTemp, lpCI->szFile))
                                   {
                                   lstrcpy(lpCI->szFile, szTemp);
                                   UFillIconList(hDlg, ID_ICONLIST, lpCI-
>szFile);
                                   UpdateResultIcon(lpCI, hDlg,
ID_FROMFILE);

                                   //Eat this message to prevent focus
change.
                                   return TRUE;
                                   }

                             // Check if the file name is valid
                             //  (if FromFile is enabled)
                             if (ID_FROMFILE & lpCI->dwFlags)
```

```
                                        {
                                        OFSTRUCT    of;
                                        HWND hWnd;
                                        if (HFILE_ERROR==DoesFileExist(lpCI-
>szFile, &of))
                                              {
                                              OpenFileError(hDlg, of.nErrCode,
lpCI->szFile);
                                              hWnd = GetDlgItem(hDlg,
ID_FROMFILEEDIT);
                                              SetFocus(hWnd);
                                              SendMessage(hWnd, EM_SETSEL, 0,
MAKELPARAM(0, (WORD)-1));
                                              return TRUE;  // eat this message
                                              }
                                        }

                                  if ((HWND)LOWORD(lParam) != GetFocus())
                                     SetFocus((HWND)LOWORD(lParam));

                                  /*
                                   * On closing, create a new metafilepict with
the
                                   * current icon and label, destroying the old
structure.
                                   *
                                   * Since we make a copy of the icon by placing
it into
                                   * the metafile, we have to make sure we delete
the
                                   * icon in the current field.  When the listbox
is
                                   * destroyed WM_DELETEITEMs will clean it up
appropriately.
                                   */

                                  hIcon=(HICON)SendDlgItemMessage(hDlg,
ID_RESULTICON

     , STM_GETICON, 0, 0L);

                                  /*
                                   * If default is selected then we get the
source
                                   * information from registrion database for the
                                   * current class to put in the metafile.  If
current
                                   * is selected the we just retrieve the
original file
                                   * again and recreate the metafile.  If from
file is
                                   * selected we use the current filename from
the
                                   * control and the current listbox selection.
                                   */
```

```
                              psz=lpCI->szFile;

                              if (lpCI->dwFlags & CIF_SELECTDEFAULT)
                                    {
                                    psz=lpCI->szDefIconFile;
                                    lpCI->iIcon=lpCI->iDefIcon;
                                    hIcon=lpCI->hDefIcon;
                                    }

                              if (lpCI->dwFlags & CIF_SELECTCURRENT)
                                    {
                                    //Go get the current icon source back.
                                    OleUIMetafilePictExtractIconSource(lpCI-
>lpOCI->hMetaPict
                                          , psz, &lpCI->iIcon);
                                    }

                              if (lpCI->dwFlags & CIF_SELECTFROMFILE)
                                    {
                                    GetDlgItemText(hDlg, ID_FROMFILEEDIT,
psz, OLEUI_CCHPATHMAX);

                                    lpCI->iIcon=(UINT)SendDlgItemMessage(hDlg
                                          , ID_ICONLIST, LB_GETCURSEL, 0,
0L);
                                    }


                              //Get the label and go create the metafile
                              GetDlgItemText(hDlg, ID_LABELEDIT, szTemp,
sizeof(szTemp));

                              //If psz is NULL (default) we get no source
comments.

     hMetaPict=OleUIMetafilePictFromIconAndLabel(hIcon
                                    , szTemp, psz, lpCI->iIcon);

                              //Clean up the current icon that we extracted.
                              hIcon=(HICON)SendDlgItemMessage(hDlg,
ID_CURRENTICON

     , STM_GETICON, 0, 0L);
                              DestroyIcon(hIcon);

                              //Clean up the default icon
                              DestroyIcon(lpCI->hDefIcon);

                              // Remove the prop set on our parent
                              RemoveProp(lpCI->lpOCI->hWndOwner,
PROP_HWND_CHGICONDLG);

                              if (NULL==hMetaPict)
```

```
                                   SendMessage(hDlg, uMsgEndDialog,
OLEUI_FALSE, 0L);

                             OleUIMetafilePictIconFree(lpCI->lpOCI-
>hMetaPict);
                             lpCI->lpOCI->hMetaPict=hMetaPict;

                             lpCI->lpOCI->dwFlags = lpCI->dwFlags;

                             SendMessage(hDlg, uMsgEndDialog, OLEUI_OK, 0L);
                             break;


                       case IDCANCEL:
                             //Clean up the current icon that we extracted.
                             hIcon=(HICON)SendDlgItemMessage(hDlg,
ID_CURRENTICON

     , STM_GETICON, 0, 0L);
                             DestroyIcon(hIcon);

                             //Clean up the default icon
                             DestroyIcon(lpCI->hDefIcon);

                             // Remove the prop set on our parent
                             RemoveProp(lpCI->lpOCI->hWndOwner,
PROP_HWND_CHGICONDLG);

                             //We leave hMetaPict intact on Cancel; caller's
responsibility
                             SendMessage(hDlg, uMsgEndDialog, OLEUI_CANCEL,
0L);
                             break;


                       case ID_OLEUIHELP:
                             PostMessage(lpCI->lpOCI->hWndOwner, uMsgHelp,
                                               (WPARAM)hDlg,
MAKELPARAM(IDD_CHANGEICON, 0));
                             break;
                       }
                 break;

           default:
           {
                 if (lpCI && iMsg == lpCI->nBrowseHelpID) {
                       PostMessage(lpCI->lpOCI->hWndOwner, uMsgHelp,
                                   (WPARAM)hDlg,
MAKELPARAM(IDD_CHANGEICONBROWSE, 0));
                 }
           }
           break;
           }
     return FALSE;
     }
```

```
/*
 * FChangeIconInit
 *
 * Purpose:
 *  WM_INITIDIALOG handler for the Change Icon dialog box.
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  wParam          WPARAM of the message
 *  lParam          LPARAM of the message
 *
 * Return Value:
 *  BOOL            Value to return for WM_INITDIALOG.
 */

BOOL FChangeIconInit(HWND hDlg, WPARAM wParam, LPARAM lParam)
     {
     LPCHANGEICON            lpCI;
     LPOLEUICHANGEICON       lpOCI;
     HFONT                   hFont;
     HWND                    hList;
     UINT                    cyList;
     RECT                    rc, rcG;
     UINT                    uID;

     //1.  Copy the structure at lParam into our instance memory.
     lpCI=(LPCHANGEICON)LpvStandardInit(hDlg, sizeof(CHANGEICON), TRUE,
&hFont);

     //PvStandardInit send a termination to us already.
     if (NULL==lpCI)
           return FALSE;

     //Save the original pointer and copy necessary information.
     lpOCI=(LPOLEUICHANGEICON)lParam;

     lpCI->lpOCI  =lpOCI;
     lpCI->dwFlags=lpOCI->dwFlags;

     //Go extract the icon source from the metafile.
     OleUIMetafilePictExtractIconSource(lpOCI->hMetaPict, lpCI->szFile,
&lpCI->iIcon);

     //Go extract the icon and the label from the metafile
     OleUIMetafilePictExtractLabel(lpOCI->hMetaPict, lpCI->szLabel,
OLEUI_CCHLABELMAX, NULL);
     lpCI->hCurIcon=OleUIMetafilePictExtractIcon(lpOCI->hMetaPict);

     //2.  If we got a font, send it to the necessary controls.
     if (NULL!=hFont)
           {
```

```
           SendDlgItemMessage(hDlg, ID_RESULTLABEL, WM_SETFONT
                                      , (WPARAM)hFont, 0L);
           }


     //3.  Show or hide the help button
     if (!(lpCI->dwFlags & CIF_SHOWHELP))
           StandardShowDlgItem(hDlg, ID_OLEUIHELP, SW_HIDE);


     /*
      * 4.  Set text limits and initial control values.  If we're given
      *     an intial label we set it in the edit and static controls.
      *     If we don't, then we copy the default contents of the static
      *     control into the edit control, meaning that only the default
      *     static control string need be localized.
      */

     SendDlgItemMessage(hDlg, ID_LABELEDIT, EM_LIMITTEXT, OLEUI_CCHLABELMAX,
0L);
     SendDlgItemMessage(hDlg, ID_FROMFILEEDIT, EM_LIMITTEXT,
OLEUI_CCHPATHMAX,  0L);
     SetDlgItemText(hDlg, ID_FROMFILEEDIT, lpCI->szFile);

     //Copy the label text into the edit and static controls.
     SetDlgItemText(hDlg, ID_LABELEDIT,   lpCI->szLabel);
     SetDlgItemText(hDlg, ID_RESULTLABEL, lpCI->szLabel);


     lpCI->hDefIcon = NULL;

     if (lpCI->dwFlags & CIF_USEICONEXE)
     {
        lpCI->hDefIcon = ExtractIcon(ghInst, lpCI->lpOCI->szIconExe, 0);

        if (NULL != lpCI->hDefIcon)
        {
            lstrcpy(lpCI->szDefIconFile, lpCI->lpOCI->szIconExe);
            lpCI->iDefIcon = 0;
        }
     }


     if (NULL == lpCI->hDefIcon)
     {
        HGLOBAL hMetaPict;

        hMetaPict = GetIconOfClass(ghInst,
                                                &lpCI->lpOCI->clsid,
                                                NULL,
                                                TRUE);

        lpCI->hDefIcon = OleUIMetafilePictExtractIcon(hMetaPict);

        OleUIMetafilePictExtractIconSource(hMetaPict,
                                                                 lpCI-
>szDefIconFile,
                                                                 &lpCI-
>iDefIcon);

        OleUIMetafilePictIconFree(hMetaPict);
     }


     //Initialize all the icon displays.
     SendDlgItemMessage(hDlg, ID_CURRENTICON, STM_SETICON
           , (WPARAM)lpCI->hCurIcon, 0L);
     SendDlgItemMessage(hDlg, ID_DEFAULTICON, STM_SETICON
           , (WPARAM)lpCI->hDefIcon, 0L);
     SendDlgItemMessage(hDlg, ID_RESULTICON,  STM_SETICON
           , (WPARAM)lpCI->hCurIcon, 0L);


     /*
      * 5.  Since we cannot predict the size of icons on any display,
      *     we have to resize the icon listbox to the size of an icon
      *     (plus padding), a scrollbar, and two borders (top & bottom).
      */
     cyList=GetSystemMetrics(SM_CYICON)+GetSystemMetrics(SM_CYHSCROLL)
              +GetSystemMetrics(SM_CYBORDER)*2+CYICONPAD;

     hList=GetDlgItem(hDlg, ID_ICONLIST);
     GetClientRect(hList, &rc);
     SetWindowPos(hList, NULL, 0, 0, rc.right, cyList
                        , SWP_NOMOVE | SWP_NOZORDER);

     //Set the columns in this multi-column listbox to hold one icon
     SendMessage(hList, LB_SETCOLUMNWIDTH
                       , GetSystemMetrics(SM_CXICON)+CXICONPAD,0L);

     /*
      * 5a.  If the listbox expanded below the group box, then size
      *      the groupbox down, move the label static and exit controls
      *      down, and expand the entire dialog appropriately.
      */

     GetWindowRect(hList, &rc);
     GetWindowRect(GetDlgItem(hDlg, ID_GROUP), &rcG);

     if (rc.bottom > rcG.bottom)
           {
           //Calculate amount to move things down.
           cyList=(rcG.bottom-rcG.top)-(rc.bottom-rc.top-cyList);

           //Expand the group box.
           rcG.right -=rcG.left;
           rcG.bottom-=rcG.top;
           SetWindowPos(GetDlgItem(hDlg, ID_GROUP), NULL, 0, 0
                             , rcG.right, rcG.bottom+cyList
                             , SWP_NOMOVE | SWP_NOZORDER);
           //Expand the dialog box.
           GetClientRect(hDlg, &rc);
           SetWindowPos(hDlg, NULL, 0, 0, rc.right, rc.bottom+cyList
                             , SWP_NOMOVE | SWP_NOZORDER);

           //Move the label and edit controls down.
           GetClientRect(GetDlgItem(hDlg, ID_LABEL), &rc);
           SetWindowPos(GetDlgItem(hDlg, ID_LABEL), NULL, 0, cyList
                             , rc.right, rc.bottom, SWP_NOSIZE |
SWP_NOZORDER);

           GetClientRect(GetDlgItem(hDlg, ID_LABELEDIT), &rc);
           SetWindowPos(GetDlgItem(hDlg, ID_LABELEDIT), NULL, 0, cyList
                             , rc.right, rc.bottom, SWP_NOSIZE |
SWP_NOZORDER);
           }


     /*
      * 6.  Select Current, Default, or From File radiobuttons
appropriately.
      *     The CheckRadioButton call sends WM_COMMANDs which handle
      *     other actions.  Note that if we check From File, which
      *     takes an icon from the list, we better fill the list.
      *     This will also fill the list even if default is selected.
      */

     if (0!=UFillIconList(hDlg, ID_ICONLIST, lpCI->szFile))
           {
           //If szFile worked, then select the source icon in the listbox.
           SendDlgItemMessage(hDlg, ID_ICONLIST, LB_SETCURSEL, lpCI->iIcon,
0L);
           }


     if (lpCI->dwFlags & CIF_SELECTCURRENT)
           CheckRadioButton(hDlg, ID_CURRENT, ID_FROMFILE, ID_CURRENT);
     else
           {
           uID=(lpCI->dwFlags & CIF_SELECTFROMFILE) ? ID_FROMFILE :
ID_DEFAULT;
           CheckRadioButton(hDlg, ID_CURRENT, ID_FROMFILE, uID);
           }

     //7.  Give our parent window access to our hDlg (via a special
SetProp).
     //    The PasteSpecial dialog may need to force our dialog down if the
     //    clipboard contents change underneath it. if so it will send
     //    us a IDCANCEL command.
     SetProp(lpCI->lpOCI->hWndOwner, PROP_HWND_CHGICONDLG, hDlg);

     lpCI->nBrowseHelpID = RegisterWindowMessage(HELPMSGSTRING);

     //8.  Call the hook with lCustData in lParam
     UStandardHook(lpCI, hDlg, WM_INITDIALOG, wParam, lpOCI->lCustData);
     return TRUE;
     }





/*
 * UFillIconList
 *
 * Purpose:
 *  Given a listbox and a filename, attempts to open that file and
 *  read all the icons that exist therein, adding them to the listbox
 *  hList as owner-draw items.  If the file does not exist or has no
 *  icons, then you get no icons and an appropriate warning message.
 *
 * Parameters:
 *  hDlg            HWND of the dialog containing the listbox.
 *  idList          UINT identifier of the listbox to fill.
 *  pszFile         LPSTR of the file from which to extract icons.
 *
 * Return Value:
 *  UINT            Number of items added to the listbox.  0 on failure.
 */

UINT UFillIconList(HWND hDlg, UINT idList, LPSTR pszFile)
     {
     HWND        hList;
     UINT        i;
     UINT        cIcons=0;
     HCURSOR     hCur;
     HICON       hIcon;
     OFSTRUCT    of;

     if (NULL==hDlg || !IsWindow(hDlg) || NULL==pszFile)
           return 0;

     hList=GetDlgItem(hDlg, idList);

     if (NULL==hList)
           return 0;

     //Clean out the listbox.
     SendMessage(hList, LB_RESETCONTENT, 0, 0L);

     //If we have an empty string, just exit leaving the listbox empty as
well
     if (0==lstrlen(pszFile))
           return 0;

     //Turn on the hourglass
     hCur=HourGlassOn();

     //Check if the file is valid.
     if (HFILE_ERROR!=DoesFileExist(pszFile, &of))
           {
        #ifdef EXTRACTICONWORKS
           //Get the icon count for this file.
           cIcons=(UINT)ExtractIcon(ghInst, pszFile, (UINT)-1);
        #else
           /*
            * ExtractIcon in Windows 3.1 with -1 eats a selector, leaving an
            * extra global memory object around for this applciation.  Since
            * changing icons may happen very often with all OLE apps in
            * the system, we have to work around it.  So we'll say we
            * have lots of icons and just call ExtractIcon until it
            * fails.  We check if there's any around by trying to get
            * the first one.
            */
           cIcons=0xFFFF;

           hIcon=ExtractIcon(ghInst, pszFile, 0);

           //Fake a failure with cIcons=0, or cleanup hIcon from this test.
           if (32 > (UINT)hIcon)
                 cIcons=0;
           else
                 DestroyIcon(hIcon);
        #endif

           if (0!=cIcons)
                 {
                 SendMessage(hList, WM_SETREDRAW, FALSE, 0L);

                 for (i=0; i<cIcons; i++)
                       {
                       hIcon=ExtractIcon(ghInst, pszFile, i);

                       if (32 < (UINT)hIcon)
                             SendMessage(hList, LB_ADDSTRING, 0, (LONG)
(UINT)hIcon);
                    #ifndef EXTRACTICONWORKS
                       else
                             {
                             //ExtractIcon failed, so let's leave now.
                             break;
                             }
                    #endif
                       }

                 //Force complete repaint
                 SendMessage(hList, WM_SETREDRAW, TRUE, 0L);
                 InvalidateRect(hList, NULL, TRUE);

                 //Select an icon
                 SendMessage(hList, LB_SETCURSEL, 0, 0L);
                 }
           else
```

```
                 ErrorWithFile(hDlg, ghInst, (UINT)IDS_CINOICONSINFILE,
pszFile, MB_OK);
           }
     else
           OpenFileError(hDlg, of.nErrCode, pszFile);

     HourGlassOff(hCur);
     return cIcons;
     }




/*
 * FDrawListIcon
 *
 * Purpose:
 *  Handles WM_DRAWITEM for the icon listbox.
 *
 * Parameters:
 *  lpDI            LPDRAWITEMSTRUCT from WM_DRAWITEM
 *
 * Return Value:
 *  BOOL            TRUE if we did anything, FALSE if there are no items
 *                  in the list.
 */

BOOL FDrawListIcon(LPDRAWITEMSTRUCT lpDI)
     {
     COLORREF        cr;

     /*
      * If there are no items in the list, then itemID is negative according
      * to the Win3.1 SDK.  Unfortunately DRAWITEMSTRUCT has an unsigned int
      * for this field, so we need the typecast to do a signed comparison.
      */
     if ((int)lpDI->itemID < 0)
           return FALSE;

     /*
      * For selection or draw entire case we just draw the entire item all
      * over again.  For focus cases, we only call DrawFocusRect.
      */

     if (lpDI->itemAction & (ODA_SELECT | ODA_DRAWENTIRE))
           {
           //Clear background and draw the icon.
           if (lpDI->itemState & ODS_SELECTED)
                 cr=SetBkColor(lpDI->hDC, GetSysColor(COLOR_HIGHLIGHT));
           else
                 cr=SetBkColor(lpDI->hDC, GetSysColor(COLOR_WINDOW));

           //Draw a cheap rectangle.
           ExtTextOut(lpDI->hDC, 0, 0, ETO_OPAQUE, &lpDI->rcItem
                          , NULL, 0, NULL);
```

```
            DrawIcon(lpDI->hDC, lpDI->rcItem.left+(CXICONPAD/2)
                          , lpDI->rcItem.top+(CYICONPAD/2)
                          , (HICON)LOWORD(lpDI->itemData));

            //Restore original background for DrawFocusRect
            SetBkColor(lpDI->hDC, cr);
            }

      //Always change focus on the focus action.
      if (lpDI->itemAction & ODA_FOCUS || lpDI->itemState & ODS_FOCUS)
            DrawFocusRect(lpDI->hDC, &lpDI->rcItem);

      return TRUE;
      }





/*
 * UpdateResultIcon
 *
 * Purpose:
 *  Updates the result icon using the current icon in the default display
 *  or the icon listbox depending on fFromDefault.
 *
 * Parameters:
 *  lpCI            LPCHANGEICON containing dialog flags.
 *  hDlg            HWND of the dialog
 *  uID             UINT identifying the radiobutton selected.
 *
 * Return Value:
 *  None
 */

void UpdateResultIcon(LPCHANGEICON lpCI, HWND hDlg, UINT uID)
      {
      UINT        iSel;
      LONG        lTemp=LB_ERR;

      lpCI->dwFlags &= ~(CIF_SELECTCURRENT | CIF_SELECTDEFAULT |
CIF_SELECTFROMFILE);

      switch (uID)
            {
            case ID_CURRENT:
                  lTemp=SendDlgItemMessage(hDlg, ID_CURRENTICON, STM_GETICON,
0, 0L);
                  lpCI->dwFlags |= CIF_SELECTCURRENT;
                  break;

            case ID_DEFAULT:
                  lTemp=SendDlgItemMessage(hDlg, ID_DEFAULTICON, STM_GETICON,
0, 0L);
```

```
                 lpCI->dwFlags |= CIF_SELECTDEFAULT;
                 break;

           case ID_FROMFILE:
                 //Get the selected icon from the list and place it in the
result
                 lpCI->dwFlags |= CIF_SELECTFROMFILE;

                 iSel=(UINT)SendDlgItemMessage(hDlg, ID_ICONLIST,
LB_GETCURSEL, 0, 0L);
                 if ((UINT)LB_ERR==iSel)
                       lTemp=SendDlgItemMessage(hDlg, ID_DEFAULTICON,
STM_GETICON, 0, 0L);
                 else
                       SendDlgItemMessage(hDlg, ID_ICONLIST, LB_GETTEXT,
iSel
                                             , (LPARAM)(LPLONG)&lTemp);

                 break;
           }

     if ((LONG)LB_ERR!=lTemp)
           SendDlgItemMessage(hDlg, ID_RESULTICON, STM_SETICON,
LOWORD(lTemp), 0L);
     return;
     }
```

ICONBOX.H   (WRAPUI Sample)

```
/*
 * ICONBOX.H
 *
 * Structures and definitions for the IconBox control.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */


#ifndef _ICONBOX_H_
#define _ICONBOX_H_

//Function prototypes
BOOL            FIconBoxInitialize(HINSTANCE, HINSTANCE, LPSTR);
void            IconBoxUninitialize(void);
LONG CALLBACK EXPORT IconBoxWndProc(HWND, UINT, WPARAM, LPARAM);


//Window extra bytes contain the bitmap index we deal with currently.
#define CBICONBOXWNDEXTRA               (sizeof(HGLOBAL)+sizeof(BOOL))
#define IBWW_HIMAGE                     0
#define IBWW_FLABEL                     (sizeof(HGLOBAL))

#ifdef WIN32
#define GETHIMAGE(h)                    (HGLOBAL)GetWindowLong(h,
IBWW_HIMAGE)
#define SETHIMAGE(h, hmf)               (HGLOBAL)SetWindowLong(h,
IBWW_HIMAGE, hmf)
#define GETFLABEL(h)                    (BOOL)GetWindowLong(h, IBWW_FLABEL)
#define SETFLABEL(h, b)                 (BOOL)SetWindowLong(h, IBWW_FLABEL,
b)
#else
#define GETHIMAGE(h)                    (HGLOBAL)GetWindowWord(h,
IBWW_HIMAGE)
#define SETHIMAGE(h, hmf)               (HGLOBAL)SetWindowWord(h,
IBWW_HIMAGE, hmf)
#define GETFLABEL(h)                    (BOOL)GetWindowWord(h, IBWW_FLABEL)
#define SETFLABEL(h, b)                 (BOOL)SetWindowWord(h, IBWW_FLABEL,
b)
#endif

//Control messages
#define IBXM_IMAGESET                   (WM_USER+0)
#define IBXM_IMAGEGET                   (WM_USER+1)
#define IBXM_IMAGEFREE                  (WM_USER+2)
#define IBXM_LABELENABLE                (WM_USER+3)


#endif //_ICONBOX_H_
```

ICONBOX.C   (WRAPUI Sample)

```
/*
 * ICONBOX.C
 *
 * Implemenatation of an IconBox control for OLE 2.0 UI dialogs that we'll
 * use wherever a dialog needs an icon/label display.  Through the control's
 * interface we can change the image or control label visibility.
 *
 * The IconBox discusses images in CF_METAFILEPICT format.  When drawing
 * such a metafile, the entire aspect is centered in the IconBox, so long
 * labels are chopped at either end.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */


#define STRICT  1
#include "ole2ui.h"
#include "iconbox.h"


//Flag indicating if we've registered the class
static BOOL     fRegistered=FALSE;


/*
 * FIconBoxInitialize
 *
 * Purpose:
 *  Registers the IconBox control class.
 *
 * Parameters:
 *  hInst           HINSTANCE instance of the DLL.
 *
 *  hPrevInst       HINSTANCE of the previous instance.  Used to
 *                  determine whether to register window classes or not.
 *
 *  lpszClassName   LPSTR containing the class name to register the
 *                  IconBox control class with.
 *
 * Return Value:
 *  BOOL            TRUE if all initialization succeeded, FALSE otherwise.
 */

BOOL FIconBoxInitialize(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR
lpszClassName)
     {
     WNDCLASS        wc;

     // Only register class if we're the first instance
     if (hPrevInst)
           fRegistered = TRUE;
     else
```

```
            {

            // Static flag fRegistered guards against calling this function
more
            // than once

            /* NOTE: the custom control classes for the ICONBOX class and the
            **    RESULTIMAGE class are registered as task specific. this
            **    is done both if the OLE2UI library is built as a static
            **    library or as a DLL. these custom controls are
            **    constructed internally as part of the OLE2UI code and do
            **    NOT need to be registered as global classes.
            */

            if (!fRegistered)
                  {
                  wc.lpfnWndProc   =IconBoxWndProc;
                  wc.cbClsExtra    =0;
                  wc.cbWndExtra    =CBICONBOXWNDEXTRA;
                  wc.hInstance     =hInst;
                  wc.hIcon         =NULL;
                  wc.hCursor       =LoadCursor(NULL, IDC_ARROW);
                  wc.hbrBackground =(HBRUSH)NULL;
                  wc.lpszMenuName  =NULL;
                  wc.lpszClassName =lpszClassName;
                  wc.style         = CS_VREDRAW | CS_HREDRAW;

                  fRegistered=RegisterClass(&wc);
                  }
            }

     return fRegistered;
}


/*
 * IconBoxUninitialize
 *
 * Purpose:
 *  Cleans up anything done in FIconBoxInitialize.  Currently there is
 *  nothing, but we do this for symmetry.
 *
 * Parameters:
 *  None
 *
 * Return Value:
 *  None
 */

void IconBoxUninitialize(void)
     {
     //Nothing to do.
     return;
     }
```

```
/*
 * IconBoxWndProc
 *
 * Purpose:
 *  Window Procedure for the IconBox custom control.  Only handles
 *  WM_CREATE, WM_PAINT, and private messages to manipulate the image.
 *
 * Parameters:
 *  Standard
 *
 * Return Value:
 *  Standard
 */

LONG CALLBACK EXPORT IconBoxWndProc(HWND hWnd, UINT iMsg, WPARAM wParam,
LPARAM lParam)
     {
     HGLOBAL         hMF=NULL;
     PAINTSTRUCT     ps;
     HDC             hDC;
     RECT            rc;


     //Handle standard Windows messages.
     switch (iMsg)
           {
           case WM_CREATE:
            SETHIMAGE(hWnd, 0);
            SETFLABEL(hWnd, TRUE);
                 return 0L;


           case WM_ERASEBKGND:
           {
           HWND     hWndP=GetParent(hWnd);
              HBRUSH   hBrush;
              RECT     rc;
           HDC      hDC=(HDC)wParam;

           SaveDC(hDC);

          #ifdef WIN32
              hBrush = (HBRUSH)SendMessage(hWndP, WM_CTLCOLORDLG, wParam,
                   (LPARAM)hWndP);
          #else
              hBrush = (HBRUSH)SendMessage(hWndP, WM_CTLCOLOR, wParam,
              MAKELPARAM(hWndP, CTLCOLOR_DLG));
          #endif

           RestoreDC(hDC, -1);
```

```
              if (!hBrush)
                  return FALSE;

              SetBrushOrgEx(hDC, 0, 0, NULL);

              GetClientRect(hWnd, &rc);
              FillRect(hDC, &rc, hBrush);

              return TRUE;
           }


           case WM_PAINT:
            hMF=GETHIMAGE(hWnd);

                 //BeginPaint and EndPaint clear us even if hMF is NULL.
                 hDC=BeginPaint(hWnd, &ps);

                 if (NULL!=hMF)
                       {
                       BOOL            fLabel;

                       //Now we get to paint the metafile, centered in our
rect.
                       GetClientRect(hWnd, &rc);

                       /*
                        * If we're doing icon only, then place the metafile
                        * at the center of our box minus half the icon
width.
                        * Top is top.
                        */

                       fLabel=GETFLABEL(hWnd);

                       //Go draw where we decided to place it.
                       OleUIMetafilePictIconDraw(hDC, &rc, hMF, !fLabel);
                       }

                 EndPaint(hWnd, &ps);
                 break;


           case IBXM_IMAGESET:
                 /*
                  * wParam contains the new handle.
                  * lParam is a flag to delete the old or not.
                  */
            hMF=SETHIMAGE(hWnd, (LONG)wParam);
                 InvalidateRect(hWnd, NULL, TRUE);
                 UpdateWindow(hWnd);

                 //Delete the old handle if requested
                 if (0L!=lParam)
```

```
                    {
                    OleUIMetafilePictIconFree(hMF);
                    hMF=NULL;
                    }

              return (LONG)(UINT)hMF;


        case IBXM_IMAGEGET:
              //Return the current index.
         hMF=GETHIMAGE(hWnd);
              return (LONG)(UINT)hMF;


        case IBXM_IMAGEFREE:
              //Free up whatever we're holding.
         hMF=GETHIMAGE(hWnd);
              OleUIMetafilePictIconFree(hMF);
              return 1L;


        case IBXM_LABELENABLE:
              //wParam has the new flag, returns the previous flag.
              return (LONG)SETFLABEL(hWnd, (BOOL)wParam);


        default:
              return DefWindowProc(hWnd, iMsg, wParam, lParam);
        }

  return 0L;
  }
```

INSOBJ.H   (WRAPUI Sample)

```
/*
 * INSOBJ.H
 *
 * Internal definitions, structures, and function prototypes for the
 * OLE 2.0 UI Insert Object dialog.
 *
 * Copyright (c)1993 Microsoft Corporation, All Rights Reserved
 */


#ifndef _INSOBJ_H_
#define _INSOBJ_H_

//Internally used structure
typedef struct tagINSERTOBJECT
     {
     LPOLEUIINSERTOBJECT lpOIO;              //Original structure passed.

     /*
      * What we store extra in this structure besides the original caller's
      * pointer are those fields that we need to modify during the life of
      * the dialog but that we don't want to change in the original
structure
      * until the user presses OK.
      */
     DWORD               dwFlags;
     CLSID               clsid;
     char                szFile[OLEUI_CCHPATHMAX];
     BOOL                fFileSelected;      //Enables Display As Icon for
links
     BOOL                fAsIconNew;
     BOOL                fAsIconFile;
     BOOL                fFileDirty;
     BOOL                fFileValid;
     UINT                nErrCode;
     HGLOBAL             hMetaPictFile;
     UINT                nBrowseHelpID;      // Help ID callback for Browse
dlg
     } INSERTOBJECT, *PINSERTOBJECT, FAR *LPINSERTOBJECT;



//Internal function prototypes
//INSOBJ.C
BOOL CALLBACK EXPORT InsertObjectDialogProc(HWND, UINT, WPARAM, LPARAM);
BOOL            FInsertObjectInit(HWND, WPARAM, LPARAM);
UINT            UFillClassList(HWND, UINT, LPCLSID, BOOL);
BOOL            FToggleObjectSource(HWND, LPINSERTOBJECT, DWORD);
void            UpdateClassIcon(HWND, LPINSERTOBJECT, HWND);
void            UpdateClassType(HWND, LPINSERTOBJECT, BOOL);
void            SetInsertObjectResults(HWND, LPINSERTOBJECT);
BOOL            FValidateInsertFile(HWND, BOOL, UINT FAR*);
```

```
void            InsertObjectCleanup(HWND);

#endif //_INSOBJ_H_
```

INSOBJ.C   (WRAPUI Sample)

```
/*
 * INSOBJ.C
 *
 * Implements the OleUIInsertObject function which invokes the complete
 * Insert Object dialog.  Makes use of the OleChangeIcon function in
 * ICON.C.
 *
 * Copyright (c)1993 Microsoft Corporation, All Rights Reserved
 */

#define STRICT  1
#include "ole2ui.h"
#include <commdlg.h>
#include <memory.h>
#include <direct.h>
#include <malloc.h>
#include <dos.h>
#include "common.h"
#include "utility.h"
#include "icon.h"
#include "insobj.h"
#include "resimage.h"
#include "iconbox.h"
#include "geticon.h"

#ifdef WIN32
/* 1/23/94 - define LINK_COMMDLG avoid LoadLibrary("commdlg.dll") */
#define LINK_COMMDLG
#endif

#define IS_FILENAME_DELIM(c)    ( (c) == '\\' || (c) == '/' || (c) == ':' )

/*
 * OleUIInsertObject
 *
 * Purpose:
 *  Invokes the standard OLE Insert Object dialog box allowing the
 *  user to select an object source and classname as well as the option
 *  to display the object as itself or as an icon.
 *
 * Parameters:
 *  lpIO            LPOLEUIINSERTOBJECT pointing to the in-out structure
 *                  for this dialog.
 *
 * Return Value:
 *  UINT            OLEUI_SUCCESS or OLEUI_OK if all is well, otherwise
 *                  an error value.
 */

STDAPI_(UINT) OleUIInsertObject(LPOLEUIINSERTOBJECT lpIO)
     {
     UINT        uRet;
     HGLOBAL     hMemDlg=NULL;
     HRESULT     hrErr;

     uRet=UStandardValidation((LPOLEUISTANDARD)lpIO,
sizeof(OLEUIINSERTOBJECT)
                                          , &hMemDlg);

     if (OLEUI_SUCCESS!=uRet)
           return uRet;

     //Now we can do Insert Object specific validation.


     // NULL is NOT valid for lpszFile
     if (   (NULL == lpIO->lpszFile)
           || (IsBadReadPtr(lpIO->lpszFile, lpIO->cchFile))
           || (IsBadWritePtr(lpIO->lpszFile, lpIO->cchFile)) )
      uRet=OLEUI_IOERR_LPSZFILEINVALID;

     if (NULL != lpIO->lpszFile
           && (lpIO->cchFile <= 0 || lpIO->cchFile > OLEUI_CCHPATHMAX))
      uRet=OLEUI_IOERR_CCHFILEINVALID;

     if (0!=lpIO->cClsidExclude)
           {
           if (NULL!=lpIO->lpClsidExclude && IsBadReadPtr(lpIO-
>lpClsidExclude
                 , lpIO->cClsidExclude*sizeof(CLSID)))
           uRet=OLEUI_IOERR_LPCLSIDEXCLUDEINVALID;
           }

     //If we have flags to create any object, validate necessary data.
     if (lpIO->dwFlags & (IOF_CREATENEWOBJECT | IOF_CREATEFILEOBJECT |
IOF_CREATELINKOBJECT))
           {
           if (NULL!=lpIO->lpFormatEtc
                 && IsBadReadPtr(lpIO->lpFormatEtc, sizeof(FORMATETC)))
                 uRet=OLEUI_IOERR_LPFORMATETCINVALID;

           if (NULL!=lpIO->ppvObj && IsBadWritePtr(lpIO->ppvObj,
sizeof(LPVOID)))
                 uRet=OLEUI_IOERR_PPVOBJINVALID;

           if (NULL!=lpIO->lpIOleClientSite
                 && IsBadReadPtr(lpIO->lpIOleClientSite->lpVtbl,
sizeof(IOleClientSiteVtbl)))
                 uRet=OLEUI_IOERR_LPIOLECLIENTSITEINVALID;

           if (NULL!=lpIO->lpIStorage
                 && IsBadReadPtr(lpIO->lpIStorage->lpVtbl,
sizeof(IStorageVtbl)))
                 uRet=OLEUI_IOERR_LPISTORAGEINVALID;
           }

     if (OLEUI_ERR_STANDARDMIN <= uRet)
```

```
            {
            if (NULL!=hMemDlg)
                  FreeResource(hMemDlg);

            return uRet;
            }

      //Now that we've validated everything, we can invoke the dialog.
      uRet=UStandardInvocation(InsertObjectDialogProc, (LPOLEUISTANDARD)lpIO
                                          , hMemDlg,
MAKEINTRESOURCE(IDD_INSERTOBJECT));


      //Stop here if we cancelled or had an error.
      if (OLEUI_SUCCESS !=uRet && OLEUI_OK!=uRet)
            return uRet;


      /*
       * If any of the flags specify that we're to create objects on return
       * from this dialog, then do so.  If we encounter an error in this
       * processing, we return OLEUI_IOERR_SCODEHASERROR.  Since the
       * three select flags are mutually exclusive, we don't have to
       * if...else here, just if each case (keeps things cleaner that way).
       */

      lpIO->sc=S_OK;

      //Check if Create New was selected and we have IOF_CREATENEWOBJECT
      if ((lpIO->dwFlags & IOF_SELECTCREATENEW) && (lpIO->dwFlags &
IOF_CREATENEWOBJECT))
            {
            hrErr=OleCreate(&lpIO->clsid, &lpIO->iid, lpIO->oleRender
                  , lpIO->lpFormatEtc, lpIO->lpIOleClientSite, lpIO-
>lpIStorage
                  , lpIO->ppvObj);
            lpIO->sc = GetScode(hrErr);
            }

      //Try Create From File
      if ((lpIO->dwFlags & IOF_SELECTCREATEFROMFILE))
            {
            if (!(lpIO->dwFlags & IOF_CHECKLINK) && (lpIO->dwFlags &
IOF_CREATEFILEOBJECT))
                  {
                  hrErr=OleCreateFromFile(&CLSID_NULL, lpIO->lpszFile, &lpIO-
>iid
                        , lpIO->oleRender, lpIO->lpFormatEtc, lpIO-
>lpIOleClientSite
                        , lpIO->lpIStorage, lpIO->ppvObj);
                  lpIO->sc = GetScode(hrErr);
                  }

            if ((lpIO->dwFlags & IOF_CHECKLINK) && (lpIO->dwFlags &
IOF_CREATELINKOBJECT))
```

```
                 {
                 hrErr=OleCreateLinkToFile(lpIO->lpszFile, &lpIO->iid
                       , lpIO->oleRender, lpIO->lpFormatEtc, lpIO-
>lpIOleClientSite
                       , lpIO->lpIStorage, lpIO->ppvObj);
                 lpIO->sc = GetScode(hrErr);
                 }
           }

     //If we tried but failed a create option, then return the appropriate
error
     if (S_OK!=lpIO->sc)
           uRet=OLEUI_IOERR_SCODEHASERROR;

     return uRet;
     }





/* wGetFileTitle
** -------------
**    Dynamically link to COMMDLG.DLL to call the GetFileTitle API
*/
static int wGetFileTitle(LPCSTR lpszFile, LPSTR lpszTitle, UINT cbBuf)
{
#if !defined( LINK_COMMDLG )
     HINSTANCE hinstCommDlg = NULL;
     typedef int (WINAPI* LPFNGETFILETITLE) (LPCSTR, LPSTR, UINT);
     LPFNGETFILETITLE lpfnGetFileTitle;
     int iResult = -1;

     // Load the COMMDLG.DLL library module
     hinstCommDlg = LoadLibrary("COMMDLG.DLL");
     if ((DWORD)hinstCommDlg > HINSTANCE_ERROR) {  /* loaded successfully */
           // Retrieve the address of the GetFileTitle function
           lpfnGetFileTitle = (LPFNGETFILETITLE)
                       GetProcAddress(hinstCommDlg, "GetFileTitle");
           if (lpfnGetFileTitle != NULL) {
                 // Call the GetFileTitle function
                 iResult = (*lpfnGetFileTitle) ((LPCSTR)lpszFile,
                             (LPSTR)lpszTitle, cbBuf);
           }
           FreeLibrary(hinstCommDlg);
           return iResult;
     }
#else
     return GetFileTitle((LPCSTR)lpszFile, (LPSTR)lpszTitle, (WORD) cbBuf);
#endif
}



/*
```

```
 * InsertObjectDialogProc
 *
 * Purpose:
 *  Implements the OLE Insert Object dialog as invoked through the
 *  OleUIInsertObject function.
 */

BOOL CALLBACK EXPORT InsertObjectDialogProc(HWND hDlg, UINT iMsg
     , WPARAM wParam, LPARAM lParam)
     {
     LPOLEUIINSERTOBJECT     lpOIO;
     LPINSERTOBJECT          lpIO;
     OLEUICHANGEICON         ci;
     UINT                    i;
     BOOL                    fCheck=FALSE;
     UINT                    uRet=0;

     //Declare Win16/Win32 compatible WM_COMMAND parameters.
     COMMANDPARAMS(wID, wCode, hWndMsg);

     //This will fail under WM_INITDIALOG, where we allocate it.
     lpIO=(LPINSERTOBJECT)LpvStandardEntry(hDlg, iMsg, wParam, lParam,
&uRet);

     //If the hook processed the message, we're done.
     if (0!=uRet)
           return (BOOL)uRet;

     //Process help message from Change Icon
     if (iMsg==uMsgHelp)
           {
           PostMessage(lpIO->lpOIO->hWndOwner, uMsgHelp, wParam, lParam);
           return FALSE;
           }

     //Process the temination message
     if (iMsg==uMsgEndDialog)
           {
           InsertObjectCleanup(hDlg);
           StandardCleanup(lpIO, hDlg);
           EndDialog(hDlg, wParam);
           return TRUE;
           }

     switch (iMsg)
           {
           case WM_INITDIALOG:
                 return FInsertObjectInit(hDlg, wParam, lParam);

           case WM_COMMAND:
                 switch (wID)
                       {
                       case ID_IO_CREATENEW:
                             FToggleObjectSource(hDlg, lpIO,
IOF_SELECTCREATENEW);
```

```
                              break;

                        case ID_IO_CREATEFROMFILE:
                              FToggleObjectSource(hDlg, lpIO,
IOF_SELECTCREATEFROMFILE);
                              break;

                        case ID_IO_LINKFILE:
                              fCheck=IsDlgButtonChecked(hDlg, wID);

                              if (fCheck)
                                    lpIO->dwFlags |=IOF_CHECKLINK;
                              else
                                    lpIO->dwFlags &=~IOF_CHECKLINK;

                              //Results change here, so be sure to update it.
                              SetInsertObjectResults(hDlg, lpIO);
                              UpdateClassIcon(hDlg, lpIO, NULL);
                              break;

                        case ID_IO_OBJECTTYPELIST:
                              switch (wCode)
                                    {
                                    case LBN_SELCHANGE:
                                          UpdateClassIcon(hDlg, lpIO,
hWndMsg);
                                          SetInsertObjectResults(hDlg, lpIO);
                                          break;

                                    case LBN_DBLCLK:
                                          //Same as pressing OK.
                                          SendCommand(hDlg, IDOK, BN_CLICKED,
hWndMsg);
                                          break;
                                    }
                              break;


                        case ID_IO_FILEDISPLAY:
                              //If there are characters, enable OK and
Display As Icon
                              if (EN_CHANGE==wCode)
                              {
                                    lpIO->fFileDirty = TRUE;
                                    lpIO->fFileValid = FALSE;

                                    lpIO->fFileSelected=
                                          (0L!=SendMessage(hWndMsg,
EM_LINELENGTH, 0, 0L));
                                    EnableWindow(GetDlgItem(hDlg,
ID_IO_LINKFILE), lpIO->fFileSelected);
                                    EnableWindow(GetDlgItem(hDlg,
ID_IO_DISPLAYASICON), lpIO->fFileSelected);
                                    EnableWindow(GetDlgItem(hDlg,
ID_IO_CHANGEICON), lpIO->fFileSelected);
```

```
                                   EnableWindow(GetDlgItem(hDlg, IDOK),
lpIO->fFileSelected);
                             }

                             if (EN_KILLFOCUS==wCode && NULL!=lpIO)
                             {
                                   if (FValidateInsertFile(hDlg,FALSE,&lpIO-
>nErrCode)) {
                                         lpIO->fFileDirty = FALSE;
                                         lpIO->fFileValid = TRUE;
                                         UpdateClassIcon(hDlg, lpIO, NULL);
                                         UpdateClassType(hDlg, lpIO, TRUE);
                                   } else {
                                         lpIO->fFileDirty = FALSE;
                                         lpIO->fFileValid = FALSE;
                                         UpdateClassType(hDlg, lpIO, FALSE);
                                   }
                             }
                             break;


                       case ID_IO_DISPLAYASICON:
                             fCheck=IsDlgButtonChecked(hDlg, wID);
                             EnableWindow(GetDlgItem(hDlg,
ID_IO_CHANGEICON), fCheck);

                             if (fCheck)
                                   lpIO->dwFlags |=IOF_CHECKDISPLAYASICON;
                             else
                                   lpIO->dwFlags &=~IOF_CHECKDISPLAYASICON;

                             //Update the internal flag based on this
checking
                             if (lpIO->dwFlags & IOF_SELECTCREATENEW)
                                   lpIO->fAsIconNew=fCheck;
                             else
                                   lpIO->fAsIconFile=fCheck;

                             //Re-read the class icon on Display checked
                             if (fCheck)
                             {
                                   if (lpIO->dwFlags &
IOF_SELECTCREATEFROMFILE)
                                   {
                                     if (FValidateInsertFile(hDlg,
TRUE,&lpIO->nErrCode))
                                     {
                                         lpIO->fFileDirty = FALSE;
                                         lpIO->fFileValid = TRUE;
                                         UpdateClassIcon(hDlg, lpIO,

     GetDlgItem(hDlg, ID_IO_OBJECTTYPELIST));

                                         UpdateClassType(hDlg, lpIO, TRUE);
                                     }
                                     else
                                     {
                                         HWND hWndEC;

                                         lpIO->fAsIconFile= FALSE;
                                         lpIO->fFileDirty = FALSE;
                                         lpIO->fFileValid = FALSE;
                                         SendDlgItemMessage(hDlg,
ID_IO_ICONDISPLAY, IBXM_IMAGESET, 0, 0L);
                                         UpdateClassType(hDlg, lpIO, FALSE);

                                         lpIO->dwFlags
&=~IOF_CHECKDISPLAYASICON;
                                         CheckDlgButton(hDlg,
ID_IO_DISPLAYASICON, 0);

                                         hWndEC = GetDlgItem(hDlg,
ID_IO_FILEDISPLAY);
                                         SetFocus(hWndEC);
                            SendSetSel (hWndEC, 0, -1);
                                         return TRUE;
                                     }
                                   }
                                   else
                                     UpdateClassIcon(hDlg, lpIO,

GetDlgItem(hDlg, ID_IO_OBJECTTYPELIST));
                             }


                             //Results change here, so be sure to update it.
                             SetInsertObjectResults(hDlg, lpIO);


                             /*
                              * Show or hide controls as appropriate.  Do
the icon
                              * display last because it will take some time
to repaint.
                              * If we do it first then the dialog looks too
sluggish.
                              */
                             i=(fCheck) ? SW_SHOWNORMAL : SW_HIDE;
                             StandardShowDlgItem(hDlg, ID_IO_CHANGEICON, i);
                             StandardShowDlgItem(hDlg, ID_IO_ICONDISPLAY,
i);

                             break;


                       case ID_IO_CHANGEICON:
                       {

                             LPMALLOC  pIMalloc;
```

```
                              HWND      hList;
                              LPSTR     pszString, pszCLSID;

                              int       iCurSel;

                              // if we're in SELECTCREATEFROMFILE mode, then
we need to Validate
                              // the contents of the edit control first.

                              if (lpIO->dwFlags & IOF_SELECTCREATEFROMFILE)
                              {
                                 if (   lpIO->fFileDirty
                                        && !FValidateInsertFile(hDlg, TRUE,
&lpIO->nErrCode) )
                                 {
                                      HWND hWndEC;

                                      lpIO->fFileDirty = TRUE;
                                      hWndEC = GetDlgItem(hDlg,
ID_IO_FILEDISPLAY);
                                      SetFocus(hWndEC);
                           SendSetSel (hWndEC, 0, -1);
                                      return TRUE;
                                 }
                                 else
                                      lpIO->fFileDirty = FALSE;
                              }



                              //Initialize the structure for the hook.
                              _fmemset((LPOLEUICHANGEICON)&ci, 0,
sizeof(ci));

                              ci.hMetaPict=(HGLOBAL)SendDlgItemMessage(hDlg
                                    , ID_IO_ICONDISPLAY, IBXM_IMAGEGET, 0,
0L);

                              ci.cbStruct =sizeof(ci);
                              ci.hWndOwner=hDlg;
                              ci.dwFlags  =CIF_SELECTCURRENT;

                              if (lpIO->dwFlags & IOF_SHOWHELP)
                                    ci.dwFlags |= CIF_SHOWHELP;




                              if (lpIO->dwFlags & IOF_SELECTCREATENEW)
                              {
                                 // Initialize clsid...
                                 if (NOERROR != CoGetMalloc(MEMCTX_TASK,
&pIMalloc))
                                     return FALSE;
```

```
                                pszString = (LPSTR)pIMalloc->lpVtbl-
>Alloc(pIMalloc, OLEUI_CCHKEYMAX + OLEUI_CCHCLSIDSTRING);


                                hList = GetDlgItem(hDlg,
ID_IO_OBJECTTYPELIST);
                                iCurSel = (int)SendMessage(hList,
LB_GETCURSEL, 0, 0L);
                                SendMessage(hList, LB_GETTEXT, iCurSel,
(LONG)pszString);

                                pszCLSID = PointerToNthField(pszString, 2,
'\t');

                                CLSIDFromString((LPSTR)pszCLSID,
(LPCLSID)&(ci.clsid));

                                pIMalloc->lpVtbl->Free(pIMalloc,
(LPVOID)pszString);
                                pIMalloc->lpVtbl->Release(pIMalloc);
                             }
                             else  // IOF_SELECTCREATEFROMFILE
                             {

                                char  szFileName[OLEUI_CCHPATHMAX];

                                GetDlgItemText(hDlg, ID_IO_FILEDISPLAY,
(LPSTR)szFileName, OLEUI_CCHPATHMAX);

                                if (NOERROR !=
GetClassFile((LPSTR)szFileName, (LPCLSID)&(ci.clsid)))
                                {
                                     LPSTR lpszExtension;
                                     int   istrlen;

                                     istrlen = lstrlen(szFileName);

                                     lpszExtension = (LPSTR)szFileName +
istrlen -1;

                                     while ( (lpszExtension > szFileName) &&
                                             (*lpszExtension != '.') )
                                         lpszExtension--;

                                     GetAssociatedExecutable(lpszExtension,
(LPSTR)ci.szIconExe);
                                     ci.cchIconExe = lstrlen(ci.szIconExe);
                                     ci.dwFlags |= CIF_USEICONEXE;

                                }
                             }


                             //Let the hook in to customize Change Icon if
desired.
```

```
                             uRet=UStandardHook(lpIO, hDlg, uMsgChangeIcon
                                   , 0, (LONG)(LPSTR)&ci);

                             if (0==uRet)
                                   uRet=(UINT)
(OLEUI_OK==OleUIChangeIcon(&ci));

                             //Update the display and itemdata if necessary.
                             if (0!=uRet)
                             {

                                   /*
                                    * OleUIChangeIcon will have already
freed our
                                    * current hMetaPict that we passed in
when OK is
                                    * pressed in that dialog.  So we use 0L
as lParam
                                    * here so the IconBox doesn't try to
free the
                                    * metafilepict again.
                                    */
                                   SendDlgItemMessage(hDlg,
ID_IO_ICONDISPLAY, IBXM_IMAGESET
                                         , (WPARAM)ci.hMetaPict, 0L);

                                   if (lpIO->dwFlags & IOF_SELECTCREATENEW)
                                     SendMessage(hList, LB_SETITEMDATA,
iCurSel, MAKELPARAM(ci.hMetaPict, 0));
                             }
                       }
                             break;


                       case ID_IO_FILE:
                             {
                             /*
                              * To allow the hook to customize the browse
dialog, we
                              * send OLEUI_MSG_BROWSE.  If the hook returns
FALSE
                              * we use the default, otherwise we trust that
it retrieved
                              * a filename for us.  This mechanism prevents
hooks from
                              * trapping ID_IO_BROWSE to customize the
dialog and from
                              * trying to figure out what we do after we
have the name.
                              */

                             char    szTemp[OLEUI_CCHPATHMAX];
                             char    szInitialDir[OLEUI_CCHPATHMAX];
                             DWORD   dwOfnFlags;
                             int     nChars;
                             BOOL    fUseInitialDir = FALSE;


                             nChars = GetDlgItemText(hDlg,
ID_IO_FILEDISPLAY, (LPSTR)szTemp, OLEUI_CCHPATHMAX);

                             if (FValidateInsertFile(hDlg, FALSE, &lpIO-
>nErrCode))
                             {

                                   int istrlen;

                                   wGetFileTitle((LPSTR)szTemp, lpIO-
>szFile, OLEUI_CCHPATHMAX);

                                   istrlen = lstrlen(lpIO->szFile);

                                   LSTRCPYN((LPSTR)szInitialDir, szTemp,
nChars - istrlen);
                                   fUseInitialDir = TRUE;

                             }
                             else  // file name isn't valid...lop off end of
szTemp to get a
                                     // valid directory
                             {
#if defined( WIN32 )
                                   char szBuffer[OLEUI_CCHPATHMAX];
                                   DWORD Attribs;

                                   LSTRCPYN(szBuffer, szTemp,
sizeof(szBuffer)-1);
                                   szBuffer[sizeof(szBuffer)-1] = '\0';

                                   if ('\\' == szBuffer[nChars-1])
                                      szBuffer[nChars-1] = '\0';

                                   Attribs = GetFileAttributes(szBuffer);
                                   if (Attribs != 0xffffffff &&
                                      (Attribs & FILE_ATTRIBUTE_DIRECTORY) )
                                   {
                                      lstrcpy(szInitialDir,
(LPSTR)szBuffer);
                                      fUseInitialDir = TRUE;
                                   }
#else
                                   static char szBuffer[OLEUI_CCHPATHMAX];
                                   static int  attrib ;

                                   LSTRCPYN(szBuffer, szTemp,
sizeof(szBuffer)-1);
                                   szBuffer[sizeof(szBuffer)-1] = '\0';

                                   AnsiToOem(szBuffer, szBuffer);
#if defined( OBSOLETE )      // fix bug# 3575
```

```
                                   if ('\\' == szBuffer[nChars-1])
                                      szBuffer[nChars-1] = '\0';

                                   if(0 == _dos_getfileattr(szBuffer,
&attrib))
#endif  // OBSOLETE
                                   {
                                      lstrcpy(szInitialDir,
(LPSTR)szBuffer);
                                      fUseInitialDir = TRUE;
                                   }
#endif
                                   *lpIO->szFile = '\0';
                             }

                             uRet=UStandardHook(lpIO, hDlg, uMsgBrowse
                                   , OLEUI_CCHPATHMAX, (LPARAM)(LPSTR)lpIO-
>szFile);

                             dwOfnFlags = OFN_FILEMUSTEXIST;

                             if (lpIO->lpOIO->dwFlags & IOF_SHOWHELP)
                                dwOfnFlags |= OFN_SHOWHELP;

                             if (0==uRet)
                                   uRet=(UINT)Browse(hDlg,
                                                              lpIO->szFile,

fUseInitialDir ? (LPSTR)szInitialDir : NULL,

OLEUI_CCHPATHMAX,

(UINT)IDS_FILTERS,

(UINT)IDS_BROWSE,
                                                              dwOfnFlags);

                             //Only update if the file changed.
                             if (0!=uRet && 0!=lstrcmpi(szTemp, lpIO-
>szFile))
                             {
                                   SetDlgItemText(hDlg, ID_IO_FILEDISPLAY,
lpIO->szFile);
                                   lpIO->fFileSelected=TRUE;

                                   if (FValidateInsertFile(hDlg, TRUE,
&lpIO->nErrCode))
                                   {
                                     lpIO->fFileDirty = FALSE;
                                     lpIO->fFileValid = TRUE;
                                     UpdateClassIcon(hDlg, lpIO, NULL);
                                     UpdateClassType(hDlg, lpIO, TRUE);
                                     // auto set OK to be default button if
valid file
                                     SendMessage(hDlg, DM_SETDEFID,
                                                 (WPARAM)GetDlgItem(hDlg,
IDOK), 0L);
                                     SetFocus(GetDlgItem(hDlg, IDOK));
                                   }
                                   else  // filename is invalid - set focus
back to ec
                                   {
                                     HWND hWnd;

                                     lpIO->fFileDirty = FALSE;
                                     lpIO->fFileValid = FALSE;
                                     hWnd = GetDlgItem(hDlg,
ID_IO_FILEDISPLAY);
                                     SetFocus(hWnd);
                          SendSetSel (hWnd, 0, -1);
                                   }

                                   //Once we have a file, Display As Icon is
always enabled
                                   EnableWindow(GetDlgItem(hDlg,
ID_IO_DISPLAYASICON), TRUE);

                                   //As well as OK
                                   EnableWindow(GetDlgItem(hDlg, IDOK),
TRUE);

                             }
                       }
                       break;


                       case IDOK:
                       {
                             HWND    hListBox;
                             WORD    iCurSel;
                             char    szBuffer[OLEUI_CCHKEYMAX +
OLEUI_CCHCLSIDSTRING];
                             LPSTR   lpszCLSID;

                             if ((HWND)(LOWORD(lParam)) != GetFocus())
                               SetFocus((HWND)(LOWORD(lParam)));



                             // If the file name is clean (already
validated), or
                             // if Create New is selected, then we can skip
this part.

                             if (  (lpIO->dwFlags &
IOF_SELECTCREATEFROMFILE)
                                && (TRUE == lpIO->fFileDirty) )
                             {
                                   if (FValidateInsertFile(hDlg, TRUE,
&lpIO->nErrCode))
                                   {
                                     lpIO->fFileDirty = FALSE;
                                     lpIO->fFileValid = TRUE;
                                     UpdateClassIcon(hDlg, lpIO, NULL);
                                     UpdateClassType(hDlg, lpIO, TRUE);
                                   }
                                   else  // filename is invalid - set focus
back to ec
                                   {
                                     HWND hWnd;

                                     lpIO->fFileDirty = FALSE;
                                     lpIO->fFileValid = FALSE;
                                     hWnd = GetDlgItem(hDlg,
ID_IO_FILEDISPLAY);
                                     SetFocus(hWnd);
                          SendSetSel (hWnd, 0, -1);
                                     UpdateClassType(hDlg, lpIO, FALSE);
                                   }

                                   return TRUE;  // eat this message
                             }
                             else if (  (lpIO->dwFlags &
IOF_SELECTCREATEFROMFILE)
                                && (FALSE == lpIO->fFileValid) )
                             {
                                   // filename is invalid - set focus back
to ec
                                   HWND hWnd;
                                   char        szFile[OLEUI_CCHPATHMAX];

                                   if (0!=GetDlgItemText(hDlg,
ID_IO_FILEDISPLAY,
                                                                  szFile,
sizeof(szFile)))
                                   {
                                         OpenFileError(hDlg, lpIO->nErrCode,
szFile);
                                   }
                                   lpIO->fFileDirty = FALSE;
                                   lpIO->fFileValid = FALSE;
                                   hWnd = GetDlgItem(hDlg,
ID_IO_FILEDISPLAY);
                                   SetFocus(hWnd);
                        SendSetSel (hWnd, 0, -1);
                                   UpdateClassType(hDlg, lpIO, FALSE);
                                   return TRUE;  // eat this message
                             }

                             //Copy the necessary information back to the
original struct
                             lpOIO=lpIO->lpOIO;
                             lpOIO->dwFlags=lpIO->dwFlags;
                             if (lpIO->dwFlags & IOF_SELECTCREATENEW)
                             {
                                hListBox=GetDlgItem(hDlg,
ID_IO_OBJECTTYPELIST);
                                iCurSel=(UINT)SendMessage(hListBox,
LB_GETCURSEL, 0, 0);

                                if (lpIO->dwFlags & IOF_CHECKDISPLAYASICON)
                                {
                                      lpOIO-
>hMetaPict=(HGLOBAL)SendMessage(hListBox,
                                            LB_GETITEMDATA, iCurSel, 0L);

                                      /*
                                         * Set the item data to 0 here so
that the cleanup
                                         * code doesn't delete the metafile.
                                         */
                                      SendMessage(hListBox, LB_SETITEMDATA,
iCurSel, 0L);
                                }
                                else
                                      lpOIO->hMetaPict = (HGLOBAL)NULL;

                                SendMessage(hListBox, LB_GETTEXT, iCurSel
                                      , (LPARAM)(LPSTR)szBuffer);

                                lpszCLSID=PointerToNthField((LPSTR)szBuffer,
2, '\t');
                                CLSIDFromString(lpszCLSID, &lpOIO->clsid);

                             }
                             else  // IOF_SELECTCREATEFROMFILE
                             {
                                if (lpIO->dwFlags & IOF_CHECKDISPLAYASICON)
                                {
                                          // get metafile here
                                     lpOIO->hMetaPict =
(HGLOBAL)SendDlgItemMessage(hDlg,

                                    ID_IO_ICONDISPLAY,

                                    IBXM_IMAGEGET,

                                    0, 0L);


                                }
                                else
                                    lpOIO->hMetaPict = (HGLOBAL)NULL;

                             }

                                GetDlgItemText(hDlg, ID_IO_FILEDISPLAY,
                                                       lpIO->szFile, lpOIO-
>cchFile);

                                LSTRCPYN(lpOIO->lpszFile, lpIO->szFile,
lpOIO->cchFile);

                                SendMessage(hDlg, uMsgEndDialog, OLEUI_OK,
0L);
                       }
                       break;

                       case IDCANCEL:
                             SendMessage(hDlg, uMsgEndDialog, OLEUI_CANCEL,
0L);
                             break;

                       case ID_OLEUIHELP:
                             PostMessage(lpIO->lpOIO->hWndOwner, uMsgHelp
                                                  , (WPARAM)hDlg,
MAKELPARAM(IDD_INSERTOBJECT, 0));
                             break;
                       }
                 break;

           default:
           {
                 if (lpIO && iMsg == lpIO->nBrowseHelpID) {
                       PostMessage(lpIO->lpOIO->hWndOwner, uMsgHelp,
                                   (WPARAM)hDlg,
MAKELPARAM(IDD_INSERTFILEBROWSE, 0));
                 }
           }
           break;
           }

     return FALSE;
     }


/*
 * FInsertObjectInit
 *
 * Purpose:
 *  WM_INITIDIALOG handler for the Insert Object dialog box.
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  wParam          WPARAM of the message
 *  lParam          LPARAM of the message
 *
 * Return Value:
 *  BOOL            Value to return for WM_INITDIALOG.
 */

BOOL FInsertObjectInit(HWND hDlg, WPARAM wParam, LPARAM lParam)
```

```
     {
     LPOLEUIINSERTOBJECT     lpOIO;
     LPINSERTOBJECT          lpIO;
     RECT                    rc;
     DWORD                   dw;
     HFONT                   hFont;
     HWND                    hList;
     UINT                    u;
     BOOL                    fCheck;
     char                   *pch;

     //1.  Copy the structure at lParam into our instance memory.
     lpIO=(LPINSERTOBJECT)LpvStandardInit(hDlg, sizeof(INSERTOBJECT), TRUE,
&hFont);

     //PvStandardInit send a termination to us already.
     if (NULL==lpIO)
           return FALSE;

     lpOIO=(LPOLEUIINSERTOBJECT)lParam;

     //2.  Save the original pointer and copy necessary information.
     lpIO->lpOIO  =lpOIO;
     lpIO->dwFlags=lpOIO->dwFlags;
     lpIO->clsid  =lpOIO->clsid;

     if ( (lpOIO->lpszFile) && ('\0' != *lpOIO->lpszFile) )
           LSTRCPYN((LPSTR)lpIO->szFile, lpOIO->lpszFile, OLEUI_CCHPATHMAX);
     else
           *(lpIO->szFile) = '\0';

     lpIO->hMetaPictFile = (HGLOBAL)NULL;

     //3.  If we got a font, send it to the necessary controls.
     if (NULL!=hFont)
           {
           SendDlgItemMessage(hDlg, ID_IO_RESULTTEXT,  WM_SETFONT,
(WPARAM)hFont, 0L);
           SendDlgItemMessage(hDlg, ID_IO_FILETYPE,  WM_SETFONT,
(WPARAM)hFont, 0L);
           }


     //4.  Fill the Object Type listbox with entries from the reg DB.
     hList=GetDlgItem(hDlg, ID_IO_OBJECTTYPELIST);
     UFillClassList(hList, lpOIO->cClsidExclude, lpOIO->lpClsidExclude
           , (BOOL)(lpOIO->dwFlags & IOF_VERIFYSERVERSEXIST));

     //Set the tab width in the list to push all the tabs off the side.
     GetClientRect(hList, &rc);
     dw=GetDialogBaseUnits();
     rc.right =(8*rc.right)/LOWORD(dw);  //Convert pixels to 2x dlg units.
     SendMessage(hList, LB_SETTABSTOPS, 1, (LPARAM)(LPINT)&rc.right);
     //5. Initilize the file name display to cwd if we don't have any name.
     if ('\0' == *(lpIO->szFile))
     {
            if ((pch=_getcwd(NULL, OLEUI_CCHPATHMAX)) != NULL) {

                  if (*(pch+lstrlen(pch)-1) != '\\')
                        lstrcat(pch, "\\");  // put slash on end of cwd
                  SetDlgItemText(hDlg, ID_IO_FILEDISPLAY, pch);
                  lpIO->fFileDirty = TRUE;  // cwd is not a valid filename
#ifndef __TURBOC__
                  free(pch);
#endif
            }else{
                  OutputDebugString("*** Please install VC++ Patch #1
***\n\r");
            }
     }
     else
     {
           SetDlgItemText(hDlg, ID_IO_FILEDISPLAY, lpIO->szFile);

           if (FValidateInsertFile(hDlg, FALSE, &lpIO->nErrCode))
             lpIO->fFileDirty = FALSE;
           else
             lpIO->fFileDirty = TRUE;
     }


     //6.  Initialize the selected type radiobutton.
     if (lpIO->dwFlags & IOF_SELECTCREATENEW)
     {
           StandardShowDlgItem(hDlg, ID_IO_FILETEXT, SW_HIDE);
           StandardShowDlgItem(hDlg, ID_IO_FILETYPE, SW_HIDE);
           StandardShowDlgItem(hDlg, ID_IO_FILEDISPLAY, SW_HIDE);
           StandardShowDlgItem(hDlg, ID_IO_FILE, SW_HIDE);
           StandardShowDlgItem(hDlg, ID_IO_LINKFILE, SW_HIDE);

           CheckRadioButton(hDlg, ID_IO_CREATENEW, ID_IO_CREATEFROMFILE,
ID_IO_CREATENEW);

           lpIO->fAsIconNew=(0L!=(lpIO->dwFlags & IOF_CHECKDISPLAYASICON));
           SetFocus(hList);
     }
     else
     {
           /*
            * Use pszType as the initial File.  If there's no initial
            * file then we have to remove any check from Display As
            * Icon.  We also check Link if so indicated for this option.
            */
           StandardShowDlgItem(hDlg, ID_IO_OBJECTTYPELIST, SW_HIDE);
           StandardShowDlgItem(hDlg, ID_IO_OBJECTTYPETEXT, SW_HIDE);

           // Don't preselect display as icon if the filename isn't valid
           if (TRUE == lpIO->fFileDirty)
```

```
                lpIO->dwFlags &= ~(IOF_CHECKDISPLAYASICON);

          if (IOF_DISABLELINK & lpIO->dwFlags)
                StandardShowDlgItem(hDlg, ID_IO_LINKFILE, SW_HIDE);
          else
          {
                CheckDlgButton(hDlg, ID_IO_LINKFILE
                      , (BOOL)(0L!=(lpIO->dwFlags & IOF_CHECKLINK)));
          }

          CheckRadioButton(hDlg, ID_IO_CREATENEW, ID_IO_CREATEFROMFILE,
ID_IO_CREATEFROMFILE);

          lpIO->fAsIconFile=(0L!=(lpIO->dwFlags & IOF_CHECKDISPLAYASICON));
          SetFocus(GetDlgItem(hDlg, ID_IO_FILEDISPLAY));
    }


    //7.  Initialize the Display as Icon state
    fCheck=(BOOL)(lpIO->dwFlags & IOF_CHECKDISPLAYASICON);
    u=fCheck ? SW_SHOWNORMAL : SW_HIDE;

    StandardShowDlgItem(hDlg, ID_IO_CHANGEICON, u);
    StandardShowDlgItem(hDlg, ID_IO_ICONDISPLAY, u);

    CheckDlgButton(hDlg, ID_IO_DISPLAYASICON, fCheck);


    //8.  Show or hide the help button
    if (!(lpIO->dwFlags & IOF_SHOWHELP))
          StandardShowDlgItem(hDlg, ID_OLEUIHELP, SW_HIDE);


    //9.  Initialize the result display
    UpdateClassIcon(hDlg, lpIO, GetDlgItem(hDlg, ID_IO_OBJECTTYPELIST));
    SetInsertObjectResults(hDlg, lpIO);

    //10.  Change the caption
    if (NULL!=lpOIO->lpszCaption)
          SetWindowText(hDlg, lpOIO->lpszCaption);

    //11. Hide all DisplayAsIcon related controls if it should be disabled
    if ( lpIO->dwFlags & IOF_DISABLEDISPLAYASICON ) {
            StandardShowDlgItem(hDlg, ID_IO_DISPLAYASICON, SW_HIDE);
            StandardShowDlgItem(hDlg, ID_IO_CHANGEICON, SW_HIDE);
            StandardShowDlgItem(hDlg, ID_IO_ICONDISPLAY, SW_HIDE);
    }

    lpIO->nBrowseHelpID = RegisterWindowMessage(HELPMSGSTRING);

    //All Done:  call the hook with lCustData
    UStandardHook(lpIO, hDlg, WM_INITDIALOG, wParam, lpOIO->lCustData);

    /*
     * We either set focus to the listbox or the edit control.  In either
```

```
      * case we don't want Windows to do any SetFocus, so we return FALSE.
      */
     return FALSE;
     }





/*
 * UFillClassList
 *
 * Purpose:
 *  Enumerates available OLE object classes from the registration
 *  database and fills a listbox with those names.
 *
 *  Note that this function removes any prior contents of the listbox.
 *
 * Parameters:
 *  hList           HWND to the listbox to fill.
 *  cIDEx           UINT number of CLSIDs to exclude in lpIDEx
 *  lpIDEx          LPCLSID to CLSIDs to leave out of the listbox.
 *  fVerify         BOOL indicating if we are to validate existence of
 *                  servers before putting them in the list.
 *
 * Return Value:
 *  UINT            Number of strings added to the listbox, -1 on failure.
 */

UINT UFillClassList(HWND hList, UINT cIDEx, LPCLSID lpIDEx, BOOL fVerify)
     {
     DWORD       dw;
     UINT        cStrings=0;
     UINT        i;
     UINT        cch;
     HKEY        hKey;
     LONG        lRet;
     HFILE       hFile;
     OFSTRUCT    of;
     BOOL        fExclude;
     LPMALLOC    pIMalloc;
     LPSTR       pszExec;
     LPSTR       pszClass;
     LPSTR       pszKey;
     LPSTR       pszID = NULL;
     CLSID       clsid;

     //Get some work buffers
     if (NOERROR!=CoGetMalloc(MEMCTX_TASK, &pIMalloc))
           return (UINT)-1;

     pszExec=(LPSTR)pIMalloc->lpVtbl->Alloc(pIMalloc, OLEUI_CCHKEYMAX*3);

     if (NULL==pszExec)
```

```
        {
        pIMalloc->lpVtbl->Release(pIMalloc);
        return (UINT)-1;
        }

    pszClass=pszExec+OLEUI_CCHKEYMAX;
    pszKey=pszClass+OLEUI_CCHKEYMAX;

    //Open up the root key.
    lRet=RegOpenKey(HKEY_CLASSES_ROOT, NULL, &hKey);

    if ((LONG)ERROR_SUCCESS!=lRet)
        {
        pIMalloc->lpVtbl->Free(pIMalloc, (LPVOID)pszExec);
        pIMalloc->lpVtbl->Release(pIMalloc);
        return (UINT)-1;
        }

    //Clean out the existing strings.
    SendMessage(hList, LB_RESETCONTENT, 0, 0L);

    cStrings=0;

    while (TRUE)
        {

        if (NULL!=pszID)
            {
            pIMalloc->lpVtbl->Free(pIMalloc, (LPVOID)pszID);
            pszID = NULL;
            }

        lRet=RegEnumKey(hKey, cStrings++, pszClass, OLEUI_CCHKEYMAX);

        if ((LONG)ERROR_SUCCESS!=lRet)
            break;

        //Cheat on lstrcat by using lstrcpy after this string, saving
time
        cch=lstrlen(pszClass);

        // Check for \NotInsertable. if this is found then this overrides
        // all other keys; this class will NOT be added to the
InsertObject
        // list.

        lstrcpy(pszClass+cch, "\\NotInsertable");

        dw=OLEUI_CCHKEYMAX;
        lRet=RegQueryValue(hKey, pszClass, pszKey, &dw);

        if ((LONG)ERROR_SUCCESS==lRet)
            continue;   // NotInsertable IS found--skip this class

        //Check for a \protocol\StdFileEditing\server entry.
```

```
            lstrcpy(pszClass+cch, "\\protocol\\StdFileEditing\\server");

            dw=OLEUI_CCHKEYMAX;
            lRet=RegQueryValue(hKey, pszClass, pszKey, &dw);

            if ((LONG)ERROR_SUCCESS==lRet)
                  {
                  /*
                  * Check if the EXE actually exists.  By default we don't do
this
                  * to bring up the dialog faster.  If an application wants
to be
                  * stringent, they can provide IOF_VERIFYSERVERSEXIST.
                  */

                  hFile = !HFILE_ERROR;

                  if (fVerify)
                        hFile=DoesFileExist(pszKey, &of);

                  if (HFILE_ERROR!=hFile)
                        {
                        dw=OLEUI_CCHKEYMAX;
                        *(pszClass+cch)=0;  // set back to rootkey
                        // Get full user type name
                        lRet=RegQueryValue(hKey, pszClass, pszKey, &dw);

                        if ((LONG)ERROR_SUCCESS!=lRet)
                              continue;   // error getting type name--skip
this class
                        }
                  }
            else
                  {
                  /*
                   * No \protocol\StdFileEditing\server entry.  Look to see
if
                   * there's an Insertable entry.  If there is, then use the
                   * Clsid to look at CLSID\clsid\LocalServer and
\InprocServer
                   */

                  lstrcpy(pszClass+cch, "\\Insertable");

                  dw=OLEUI_CCHKEYMAX;
                  lRet=RegQueryValue(hKey, pszClass, pszKey, &dw);

                  if ((LONG)ERROR_SUCCESS!=lRet)
                        continue;   // Insertable NOT found--skip this class

                  //Get memory for pszID
                  pszID=pIMalloc->lpVtbl->Alloc(pIMalloc, OLEUI_CCHKEYMAX);

                  if (NULL==pszID)
                        continue;
```

```
                 *(pszClass+cch)=0;  // set back to rootkey
                 lstrcat(pszClass+cch, "\\CLSID");

                 dw=OLEUI_CCHKEYMAX;
                 lRet=RegQueryValue(hKey, pszClass, pszID, &dw);

                 if ((LONG)ERROR_SUCCESS!=lRet)
                       continue;   // CLSID subkey not found

                 lstrcpy(pszExec, "CLSID\\");
                 lstrcat(pszExec, pszID);

                 //CLSID\ is 6, dw contains pszID length (incl. null
terminator)
            // NOTE: we must subrtract 1 for the null terminator.
            //       (this was a bug in OLE 2.01)
                 cch=6+(UINT)dw-1;

#ifdef WIN32
            lstrcpy(pszExec+cch, "\\LocalServer32");
#else
            lstrcpy(pszExec+cch, "\\LocalServer");
#endif
                 dw=OLEUI_CCHKEYMAX;
                 lRet=RegQueryValue(hKey, pszExec, pszKey, &dw);

                 if ((LONG)ERROR_SUCCESS!=lRet)
                       {
                       //Try InprocServer
#ifdef WIN32
                lstrcpy(pszExec+cch, "\\InProcServer32");
#else
                lstrcpy(pszExec+cch, "\\InProcServer");
#endif
                       dw=OLEUI_CCHKEYMAX;
                       lRet=RegQueryValue(hKey, pszExec, pszKey, &dw);

                       if ((LONG)ERROR_SUCCESS!=lRet)
                             continue;
                       }

                 if (fVerify)
                       {
                       if (HFILE_ERROR==DoesFileExist(pszKey, &of))
                             continue;
                       }

                 // Refetch the object name
                 *(pszExec+cch)=0;   //Remove final keyword.
                 dw=OLEUI_CCHKEYMAX;
                 lRet=RegQueryValue(hKey, pszExec, pszKey, &dw);

                 if ((LONG)ERROR_SUCCESS!=lRet)
                       continue;
                 }

           //Get CLSID to add to listbox.
           if (NULL==pszID)
                 {
                 CLSIDFromProgID(pszClass, &clsid);
                 StringFromCLSID(&clsid, &pszID);
                 }
           else
                 CLSIDFromString(pszID, &clsid);

           if (NULL==pszID || IsEqualCLSID(&clsid, &CLSID_NULL))
                 continue;

           //Check if this CLSID is in the exclusion list.
           fExclude=FALSE;

           for (i=0; i < cIDEx; i++)
                 {
                 if (IsEqualCLSID(&clsid, (LPCLSID)(lpIDEx+i)))
                       {
                       fExclude=TRUE;
                       break;
                       }
                 }

           if (fExclude)
                 continue;

           //We go through all the conditions, add the string.
           lstrcat(pszKey, "\t");

           // only add to listbox if not a duplicate
           if (LB_ERR==SendMessage(hList,LB_FINDSTRING,0,(LPARAM)pszKey))
                 {
                 lstrcat(pszKey, pszID);
                 SendMessage(hList, LB_ADDSTRING, 0, (LPARAM)pszKey);
                 }
           }

     if (NULL!=pszID)
           {
           pIMalloc->lpVtbl->Free(pIMalloc, (LPVOID)pszID);
           pszID = NULL;
           }

     //Select the first item by default
     SendMessage(hList, LB_SETCURSEL, 0, 0L);
     RegCloseKey(hKey);

     pIMalloc->lpVtbl->Free(pIMalloc, (LPVOID)pszExec);
     pIMalloc->lpVtbl->Release(pIMalloc);

     return cStrings;
     }
```

```
/*
 * FToggleObjectSource
 *
 * Purpose:
 *  Handles enabling, disabling, showing, and flag manipulation when the
 *  user changes between Create New, Insert File, and Link File in the
 *  Insert Object dialog.
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  lpIO            LPINSERTOBJECT pointing to the dialog structure
 *  dwOption        DWORD flag indicating the option just selected:
 *                  IOF_SELECTCREATENEW or IOF_SELECTCREATEFROMFILE
 *
 * Return Value:
 *  BOOL            TRUE if the option was already selected, FALSE
otherwise.
 */

BOOL FToggleObjectSource(HWND hDlg, LPINSERTOBJECT lpIO, DWORD dwOption)
     {
     BOOL        fTemp;
     UINT        uTemp;
     DWORD       dwTemp;
     int         i;

     //Skip all of this if we're already selected.
     if (lpIO->dwFlags & dwOption)
           return TRUE;


     // if we're switching from "from file" to "create new" and we've got
     // an icon for "from file", then we need to save it so that we can
     // show it if the user reselects "from file".

     if ( (IOF_SELECTCREATENEW == dwOption) &&
            (lpIO->dwFlags & IOF_CHECKDISPLAYASICON) )
       lpIO->hMetaPictFile = (HGLOBAL)SendDlgItemMessage(hDlg,
ID_IO_ICONDISPLAY, IBXM_IMAGEGET, 0, 0L);

     /*
      * 1.  Change the Display As Icon checked state to reflect the
      *     selection for this option, stored in the fAsIcon* flags.
      */
     fTemp=(IOF_SELECTCREATENEW==dwOption) ? lpIO->fAsIconNew : lpIO-
>fAsIconFile;

     if (fTemp)
           lpIO->dwFlags |=IOF_CHECKDISPLAYASICON;
     else
```

```
      lpIO->dwFlags &=~IOF_CHECKDISPLAYASICON;

CheckDlgButton(hDlg, ID_IO_DISPLAYASICON
       , (BOOL)(0L!=(lpIO->dwFlags & IOF_CHECKDISPLAYASICON)));

EnableWindow(GetDlgItem(hDlg, ID_IO_CHANGEICON), fTemp);

/*
 * 2.  Display Icon:  Enabled on Create New or on Create from File if
 *     there is a selected file.
 */
fTemp=(IOF_SELECTCREATENEW==dwOption) ? TRUE : lpIO->fFileSelected;
EnableWindow(GetDlgItem(hDlg, ID_IO_DISPLAYASICON), fTemp);

//OK and Link follow the same enabling as Display As Icon.
EnableWindow(GetDlgItem(hDlg, IDOK), fTemp);
EnableWindow(GetDlgItem(hDlg, ID_IO_LINKFILE), fTemp);

//3.  Enable Browse... when Create from File is selected.
fTemp=(IOF_SELECTCREATENEW==dwOption);
EnableWindow(GetDlgItem(hDlg, ID_IO_FILE),        !fTemp);
EnableWindow(GetDlgItem(hDlg, ID_IO_FILEDISPLAY), !fTemp);

/*
 * 4.  Switch between Object Type listbox on Create New and
 *     file buttons on others.
 */
uTemp=(fTemp) ? SW_SHOWNORMAL : SW_HIDE;
StandardShowDlgItem(hDlg, ID_IO_OBJECTTYPELIST, uTemp);
StandardShowDlgItem(hDlg, ID_IO_OBJECTTYPETEXT, uTemp);

uTemp=(fTemp) ? SW_HIDE : SW_SHOWNORMAL;
StandardShowDlgItem(hDlg, ID_IO_FILETEXT, uTemp);
StandardShowDlgItem(hDlg, ID_IO_FILETYPE, uTemp);
StandardShowDlgItem(hDlg, ID_IO_FILEDISPLAY, uTemp);
StandardShowDlgItem(hDlg, ID_IO_FILE, uTemp);

//Link is always hidden if IOF_DISABLELINK is set.
if (IOF_DISABLELINK & lpIO->dwFlags)
      uTemp=SW_HIDE;

StandardShowDlgItem(hDlg, ID_IO_LINKFILE, uTemp);  //last use of uTemp


//5.  Clear out existing any flags selection and set the new one
dwTemp=IOF_SELECTCREATENEW | IOF_SELECTCREATEFROMFILE;
lpIO->dwFlags=(lpIO->dwFlags & ~dwTemp) | dwOption;


/*
 * Show or hide controls as appropriate.  Do the icon
 * display last because it will take some time to repaint.
 * If we do it first then the dialog looks too sluggish.
 */
```

```
     i=(lpIO->dwFlags & IOF_CHECKDISPLAYASICON) ? SW_SHOWNORMAL : SW_HIDE;
     StandardShowDlgItem(hDlg, ID_IO_CHANGEICON, i);
     StandardShowDlgItem(hDlg, ID_IO_ICONDISPLAY, i);


     //6.Change result display
     SetInsertObjectResults(hDlg, lpIO);

     /*
      * 7.  For Create New, twiddle the listbox to think we selected it
      *     so it updates the icon from the object type. set the focus
      *     to the list box.
      *
      *     For Insert or Link file, set the focus to the filename button
      *     and update the icon if necessary.
      */
     if (fTemp) {
           UpdateClassIcon(hDlg, lpIO, GetDlgItem(hDlg,
ID_IO_OBJECTTYPELIST));
           SetFocus(GetDlgItem(hDlg, ID_IO_OBJECTTYPELIST));
     }
     else
     {
           if (lpIO->fAsIconFile && (NULL != lpIO->hMetaPictFile) )
           {
              SendDlgItemMessage(hDlg, ID_IO_ICONDISPLAY, IBXM_IMAGESET,
(WPARAM)lpIO->hMetaPictFile, 0L);
              lpIO->hMetaPictFile = 0;
           }
           else
              UpdateClassIcon(hDlg, lpIO, NULL);

           SetFocus(GetDlgItem(hDlg, ID_IO_FILE));
     }

     return FALSE;
     }


/*
 * UpdateClassType
 *
 * Purpose:
 *  Updates static text control to reflect current file type.  Assumes
 *  a valid filename.
 *
 * Parameters
 *  hDlg            HWND of the dialog box.
 *  lpIO            LPINSERTOBJECT pointing to the dialog structure
 *  fSet            TRUE to set the text, FALSE to explicitly clear it
 *
 * Return Value:
 *  None
 */
```

```
void UpdateClassType(HWND hDlg, LPINSERTOBJECT lpIO, BOOL fSet)
{

   CLSID clsid;
   char  szFileName[OLEUI_CCHPATHMAX];
   char  szFileType[OLEUI_CCHLABELMAX];

   *szFileType = '\0';

   if (fSet)
   {
       GetDlgItemText(hDlg, ID_IO_FILEDISPLAY, (LPSTR)szFileName,
OLEUI_CCHPATHMAX);

       if (NOERROR == GetClassFile((LPSTR)szFileName, &clsid) )
           OleStdGetUserTypeOfClass(&clsid, szFileType, OLEUI_CCHLABELMAX,
NULL);

   }

   SetDlgItemText(hDlg, ID_IO_FILETYPE, (LPSTR)szFileType);

   return;
}


/*
 * UpdateClassIcon
 *
 * Purpose:
 *  Handles LBN_SELCHANGE for the Object Type listbox.  On a selection
 *  change, we extract an icon from the server handling the currently
 *  selected object type using the utility function HIconFromClass.
 *  Note that we depend on the behavior of FillClassList to stuff the
 *  object class after a tab in the listbox string that we hide from
 *  view (see WM_INITDIALOG).
 *
 * Parameters
 *  hDlg            HWND of the dialog box.
 *  lpIO            LPINSERTOBJECT pointing to the dialog structure
 *  hList           HWND of the Object Type listbox.
 *
 * Return Value:
 *  None
 */

void UpdateClassIcon(HWND hDlg, LPINSERTOBJECT lpIO, HWND hList)
     {
     UINT        iSel;
     DWORD       cb;
     LPMALLOC    pIMalloc;
     LPSTR       pszName, pszCLSID, pszTemp;
     HGLOBAL     hMetaPict;

     LRESULT     dwRet;
     //If Display as Icon is not selected, exit
     if (!(lpIO->dwFlags & IOF_CHECKDISPLAYASICON))
           return;

     /*
      * When we change object type selection, get the new icon for that
      * type into our structure and update it in the display.  We use the
      * class in the listbox when Create New is selected or the association
      * with the extension in Create From File.
      */

     if (lpIO->dwFlags & IOF_SELECTCREATENEW)
           {
           iSel=(UINT)SendMessage(hList, LB_GETCURSEL, 0, 0L);

           if (LB_ERR==(int)iSel)
                 return;

           //Check to see if we've already got the hMetaPict for this item
           dwRet=SendMessage(hList, LB_GETITEMDATA, (WPARAM)iSel, 0L);

           hMetaPict=(HGLOBAL)(UINT)dwRet;

           if (hMetaPict)
                 {
                 //Yep, we've already got it, so just display it and return.
                 SendDlgItemMessage(hDlg, ID_IO_ICONDISPLAY, IBXM_IMAGESET,
(WPARAM)hMetaPict, 0L);
                 return;
                 }

           iSel=(UINT)SendMessage(hList, LB_GETCURSEL, 0, 0L);

           if (LB_ERR==(int)iSel)
                 return;

           //Allocate a string to hold the entire listbox string
           cb=SendMessage(hList, LB_GETTEXTLEN, iSel, 0L);
           }
     else
           cb=OLEUI_CCHPATHMAX;

     if (NOERROR!=CoGetMalloc(MEMCTX_TASK, &pIMalloc))
           return;

     pszName=(LPSTR)pIMalloc->lpVtbl->Alloc(pIMalloc, cb+1);

     if (NULL==pszName)
           {
           pIMalloc->lpVtbl->Release(pIMalloc);
           return;
           }
     *pszName=0;

     //Get the clsid we want.
     if (lpIO->dwFlags & IOF_SELECTCREATENEW)
     {
           //Grab the classname string from the list
           SendMessage(hList, LB_GETTEXT, iSel, (LONG)pszName);

           //Set pointer to CLSID (string)
           pszCLSID=PointerToNthField(pszName, 2, '\t');

           //Null terminate pszName string
           pszTemp=AnsiPrev(pszName, pszCLSID);

           *pszTemp='\0';
           CLSIDFromString(pszCLSID, &lpIO->clsid);

           hMetaPict = GetIconOfClass(ghInst, &lpIO->clsid, NULL, TRUE);
     }

     else
     {
           CLSID clsid;

           //Get the class from the filename
           GetDlgItemText(hDlg, ID_IO_FILEDISPLAY, pszName,
OLEUI_CCHPATHMAX);

           if (lpIO->dwFlags & IOF_CHECKLINK ||
                 NOERROR != GetClassFile((LPSTR)pszName, &clsid) ) {

                 // get icon using filename as label (if creating a link)
                 hMetaPict = GetIconOfFile(ghInst,
                             pszName, lpIO->dwFlags & IOF_CHECKLINK ? TRUE :
FALSE);
           } else {
                 // not creating a link and we can get CLSID from the
filename,
                 // therefore generate an icon with short type name as label
                 hMetaPict = GetIconOfClass(ghInst, &clsid, NULL, TRUE);
           }
     }


     //Replace the current display with this new one.
     SendDlgItemMessage(hDlg, ID_IO_ICONDISPLAY, IBXM_IMAGESET,
(WPARAM)hMetaPict, 0L);

     //Enable or disable "Change Icon" button depending on whether
     //we've got a valid filename or not.
     EnableWindow(GetDlgItem(hDlg, ID_IO_CHANGEICON), hMetaPict ? TRUE :
FALSE);

     //Save the hMetaPict so that we won't have to re-create
     if (lpIO->dwFlags & IOF_SELECTCREATENEW)
```

```
            SendMessage(hList, LB_SETITEMDATA, (WPARAM)iSel,
MAKELPARAM(hMetaPict,0));

     pIMalloc->lpVtbl->Free(pIMalloc, (LPVOID)pszName);
     pIMalloc->lpVtbl->Release(pIMalloc);
     return;
     }




/*
 * SetInsertObjectResults
 *
 * Purpose:
 *  Centralizes setting of the Result and icon displays in the Insert Object
 *  dialog.  Handles loading the appropriate string from the module's
 *  resources and setting the text, displaying the proper result picture,
 *  and showing the proper icon.
 *
 * Parameters:
 *  hDlg            HWND of the dialog box so we can access controls.
 *  lpIO            LPINSERTOBJECT in which we assume that the
 *                  current radiobutton and Display as Icon selections
 *                  are set.  We use the state of those variables to
 *                  determine which string we use.
 *
 * Return Value:
 *  None
 */

void SetInsertObjectResults(HWND hDlg, LPINSERTOBJECT lpIO)
     {
     LPSTR       pszT, psz1, psz2, psz3, psz4, pszTemp;
     UINT        i, iString1, iString2, iImage, cch;
     LPMALLOC    pIMalloc;
     BOOL        fAsIcon;

     /*
      * We need scratch memory for loading the stringtable string, loading
      * the object type from the listbox, and constructing the final string.
      * We therefore allocate three buffers as large as the maximum message
      * length (512) plus the object type, guaranteeing that we have enough
      * in all cases.
      */
     i=(UINT)SendDlgItemMessage(hDlg, ID_IO_OBJECTTYPELIST, LB_GETCURSEL, 0,
0L);
     cch = 512 + (i != LB_ERR
                  ? (UINT)SendDlgItemMessage(hDlg, ID_IO_OBJECTTYPELIST,
LB_GETTEXTLEN, i, 0L)
                               : 0);

     if (NOERROR!=CoGetMalloc(MEMCTX_TASK, &pIMalloc))
          return;
     pszTemp=(LPSTR)pIMalloc->lpVtbl->Alloc(pIMalloc, (DWORD)(4*cch));

     if (NULL==pszTemp)
           {
           pIMalloc->lpVtbl->Release(pIMalloc);
           return;
           }

     psz1=pszTemp;
     psz2=psz1+cch;
     psz3=psz2+cch;
     psz4=psz3+cch;

     //Default is an empty string.
     iString1 = 0;
     iString2 = 0;
     iImage   = RESULTIMAGE_NONE;
     *psz1    = 0;
     fAsIcon=(0L!=(lpIO->dwFlags & IOF_CHECKDISPLAYASICON));

     if (i != LB_ERR && lpIO->dwFlags & IOF_SELECTCREATENEW)
           {
           iString1 = (UINT)(fAsIcon ? IDS_IORESULTNEWICON :
IDS_IORESULTNEW);
           iString2 = (UINT)0;
           iImage   = (UINT)(fAsIcon ? RESULTIMAGE_EMBEDICON :
RESULTIMAGE_EMBED);
           }

     else if (i != LB_ERR && lpIO->dwFlags & IOF_SELECTCREATEFROMFILE)
           {
           //Pay attention to Link checkbox
           if (lpIO->dwFlags & IOF_CHECKLINK)
                 {
                 iString1 = (UINT)(fAsIcon ? IDS_IORESULTLINKFILEICON1 :
IDS_IORESULTLINKFILE1);
                 iString2 = (UINT)(fAsIcon ? IDS_IORESULTLINKFILEICON2 :
IDS_IORESULTLINKFILE2);
                 iImage = (UINT)(fAsIcon ? RESULTIMAGE_LINKICON :
RESULTIMAGE_LINK);
                 }
           else
                 {
                 iString1 = (UINT)IDS_IORESULTFROMFILE1;
                 iString2 = (UINT)(fAsIcon ? IDS_IORESULTFROMFILEICON2 :
IDS_IORESULTFROMFILE2);
                 iImage = (UINT)(fAsIcon ? RESULTIMAGE_EMBEDICON :
RESULTIMAGE_EMBED);
                 }
           }

// REVIEW CDH
     // Load and build the display strings
     if (0!=LoadString(ghInst, iString1, psz1, cch))
     {
```

```
              // Load second string, if necessary
           if (   (0 != iString2)
                 && (0 != LoadString(ghInst, iString2, psz4, cch)) )
           {
             lstrcat(psz1, psz4);  // concatenate strings together.
           }



           //In Create New, do the extra step of inserting the object type
string
           if (lpIO->dwFlags & IOF_SELECTCREATENEW)
           {
                 SendDlgItemMessage(hDlg, ID_IO_OBJECTTYPELIST, LB_GETTEXT
                                           , i, (LONG)psz2);

                 //Null terminate at any tab (before the classname)
                 pszT=psz2;
                 while ('\t'!=*pszT && 0!=*pszT)
                       pszT++;
                 *pszT=0;

                 //Build the string and point psz1 to it.
                 wsprintf(psz3, psz1, psz2);
                 psz1=psz3;
           }
     }

     //If LoadString failed, we simply clear out the results (*psz1=0 above)
     SetDlgItemText(hDlg, ID_IO_RESULTTEXT, psz1);

     //Go change the image and Presto!  There you have it.
     SendDlgItemMessage(hDlg, ID_IO_RESULTIMAGE, RIM_IMAGESET, iImage, 0L);

     pIMalloc->lpVtbl->Free(pIMalloc, (LPVOID)pszTemp);
     pIMalloc->lpVtbl->Release(pIMalloc);
     return;
     }



/*
 * FValidateInsertFile
 *
 * Purpose:
 *  Given a possibly partial pathname from the file edit control,
 *  attempt to locate the file and if found, store the full path
 *  in the edit control ID_IO_FILEDISPLAY.
 *
 * Parameters:
 *  hDlg            HWND of the dialog box.
 *  fTellUser       BOOL TRUE if function should tell user, FALSE if
 *                   function should validate silently.
 *
 * Return Value:
 *  BOOL           TRUE if the file is acceptable, FALSE otherwise.
 */

BOOL FValidateInsertFile(HWND hDlg, BOOL fTellUser, UINT FAR* lpnErrCode)
     {
     OFSTRUCT    of;
     HFILE       hFile;
     char        szFile[OLEUI_CCHPATHMAX];

     *lpnErrCode = 0;
     /*
      * To validate we attempt OpenFile on the string.  If OpenFile
      * fails then we display an error.  If not, OpenFile will store
      * the complete path to that file in the OFSTRUCT which we can
      * then stuff in the edit control.
      */

     if (0==GetDlgItemText(hDlg, ID_IO_FILEDISPLAY, szFile, sizeof(szFile)))
           return FALSE;   // #4569 : return FALSE when there is no text in
ctl

     hFile=DoesFileExist(szFile, &of);

     // if file is currently open (ie. sharing violation) OleCreateFromFile
     // and OleCreateLinkToFile can still succeed; do not consider it an
     // error.
     if (HFILE_ERROR==hFile && 0x0020/*sharing violation*/!=of.nErrCode)
     {
        *lpnErrCode = of.nErrCode;
        if (fTellUser)
              OpenFileError(hDlg, of.nErrCode, szFile);
        return FALSE;
     }

     //OFSTRUCT contains an OEM name, not ANSI as we need for the edit box.
     OemToAnsi(of.szPathName, of.szPathName);
     SetDlgItemText(hDlg, ID_IO_FILEDISPLAY, of.szPathName);
     return TRUE;
     }


/*
 * InsertObjectCleanup
 *
 * Purpose:
 *  Clears cached icon metafiles from those stored in the listbox.
 *
 * Parameters:
 *  hDlg           HWND of the dialog.
 *
 * Return Value:
 *  None
 */
```

```
void InsertObjectCleanup(HWND hDlg)
     {
     HWND      hList;
     UINT      iItems;
     HGLOBAL   hMetaPict;
     LRESULT   dwRet;
     UINT      i;

     hList=GetDlgItem(hDlg, ID_IO_OBJECTTYPELIST);
     iItems=(UINT)SendMessage(hList, LB_GETCOUNT, 0, 0L);

     for (i=0; i < iItems; i++)
           {
           dwRet=SendMessage(hList, LB_GETITEMDATA, (WPARAM)i, 0L);

           //Cast of LRESULT to UINT to HGLOBAL portable to Win32.
           hMetaPict=(HGLOBAL)(UINT)dwRet;

           if (hMetaPict)
                 OleUIMetafilePictIconFree(hMetaPict);
           }

     return;
     }
```

LINKS.C (WRAPUI Sample)

```
/*
 * links.c
 *
 * Implements the OleUIEditLinks function which invokes the complete
 * Edit Links dialog.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */

#define STRICT  1
#include "ole2ui.h"
#include "common.h"
#include "edlinks.h"
#include "utility.h"
#include <commdlg.h>
#include <dlgs.h>
#include <stdlib.h>

#ifdef WIN32
/* 1/23/94 - define LINK_COMMDLG avoid LoadLibrary ("commdlg.dll")*/
#define LINK_COMMDLG
#endif

OLEDBGDATA

/*
* OleUIEditLinks
*
* Purpose:
*  Invokes the standard OLE Edit Links dialog box allowing the user
*  to manipulate ole links (delete, update, change source, etc).
*
* Parameters:
*  lpEL            LPOLEUIEditLinks pointing to the in-out structure
*                  for this dialog.
*
* Return Value:
*  UINT            One of the following codes, indicating success or error:
*                      OLEUI_SUCCESS           Success
*                      OLEUI_ERR_STRUCTSIZE    The dwStructSize value is
wrong
*/

STDAPI_(UINT) OleUIEditLinks(LPOLEUIEDITLINKS lpEL)
{
     UINT        uRet;
     HGLOBAL     hMemDlg=NULL;

     uRet=UStandardValidation((LPOLEUISTANDARD)lpEL, sizeof(OLEUIEDITLINKS)
           , &hMemDlg);

     if (OLEUI_SUCCESS!=uRet)
```

```
            return uRet;

        /*
        * PERFORM ANY STRUCTURE-SPECIFIC VALIDATION HERE!
        * ON FAILURE:
        *  {
        *  if (NULL!=hMemDlg)
        *      FreeResource(hMemDlg)
        *
        *  return OLEUI_<ABBREV>ERR_<ERROR>
        *  }
        */

        //Now that we've validated everything, we can invoke the dialog.
        uRet=UStandardInvocation(EditLinksDialogProc, (LPOLEUISTANDARD)lpEL,
        hMemDlg, MAKEINTRESOURCE(IDD_EDITLINKS));

        /*
        * IF YOU ARE CREATING ANYTHING BASED ON THE RESULTS, DO IT HERE.
        */


        return uRet;
}



/*
* EditLinksDialogProc
*
* Purpose:
*  Implements the OLE Edit Links dialog as invoked through the
*  OleUIEditLinks function.
*
* Parameters:
*  Standard
*
* Return Value:
*  Standard
*/

BOOL CALLBACK EXPORT EditLinksDialogProc(HWND hDlg, UINT iMsg, WPARAM
wParam, LPARAM lParam)
{
        LPEDITLINKS             lpEL = NULL;
        BOOL                    fHook=FALSE;
        UINT                    uRet=0;
        HRESULT                 hErr;
        static int              nColPos[3];

        //Declare Win16/Win32 compatible WM_COMMAND parameters.
        COMMANDPARAMS(wID, wCode, hWndMsg);

        //This will fail under WM_INITDIALOG, where we allocate it.
        lpEL=(LPEDITLINKS)LpvStandardEntry(hDlg, iMsg, wParam, lParam, &uRet);
```

```
      //If the hook processed the message, we're done.
      if (0!=uRet)
            return (BOOL)uRet;

            // Process help message from secondary dialog
      if (iMsg == uMsgHelp) {

            PostMessage(lpEL->lpOEL->hWndOwner, uMsgHelp, wParam, lParam);
            return FALSE;

      }


      //Process the temination message
      if (iMsg==uMsgEndDialog) {

            //Free any specific allocations before calling StandardCleanup

            StandardCleanup(lpEL, hDlg);

            EndDialog(hDlg, wParam);
            return TRUE;
    }

      switch (iMsg) {
            static int    nHeightLine = -1;
            static int    nMaxCharWidth = -1;

            case WM_INITDIALOG:
            {
                  RECT rc;
                  int  nStart;

                  /* calculate the column positions relative to the listbox
*/
                  GetWindowRect(GetDlgItem(hDlg, ID_EL_LINKSLISTBOX),
(LPRECT)&rc);
                  nStart = rc.left;
                  GetWindowRect(GetDlgItem(hDlg, ID_EL_COL1), (LPRECT)&rc);
                  nColPos[0] = rc.left - nStart;
                  GetWindowRect(GetDlgItem(hDlg, ID_EL_COL2), (LPRECT)&rc);
                  nColPos[1] = rc.left - nStart;
                  GetWindowRect(GetDlgItem(hDlg, ID_EL_COL3), (LPRECT)&rc);
                  nColPos[2] = rc.left - nStart;

                  return FEditLinksInit(hDlg, wParam, lParam);
            }
            break;

            case WM_MEASUREITEM:
            {
                  LPMEASUREITEMSTRUCT   lpMIS;

                  lpMIS = (LPMEASUREITEMSTRUCT)lParam;
                  if (nHeightLine == -1) {
                        HFONT hFont;
                        HDC   hDC;
                        TEXTMETRIC  tm;

                        /*  Attempt to get font dialog.  If that fails,
                        use system font
                        */

                        hFont = (HANDLE)(UINT)SendMessage(hDlg, WM_GETFONT,
0, 0L);

                        if (hFont == NULL)
                              hFont = GetStockObject(SYSTEM_FONT);

                        hDC = GetDC(hDlg);
                        hFont = SelectObject(hDC, hFont);

                        GetTextMetrics(hDC, &tm);
                        nHeightLine = tm.tmHeight;
                        nMaxCharWidth = tm.tmMaxCharWidth;

                        ReleaseDC(hDlg, hDC);
                  }

                  lpMIS->itemHeight = nHeightLine;
            }
            break;

            case WM_DRAWITEM:
            {
                  LPDRAWITEMSTRUCT    lpDIS;
                  COLORREF            crText;
                  LPLINKINFO          lpLI;
                  HBRUSH              hbr;
                  int                 nOldBkMode;
                  char                sz[OLEUI_CCHPATHMAX];
                  LPSTR               lpsz;
                  RECT                rcClip;

                  lpDIS = (LPDRAWITEMSTRUCT)lParam;
                  lpLI = (LPLINKINFO)lpDIS->itemData;

                  if ((int)lpDIS->itemID < 0)
                        break;

                  if ((ODA_DRAWENTIRE | ODA_SELECT) & lpDIS->itemAction) {

                        if (ODS_SELECTED & lpDIS->itemState) {
                              /*Get proper txt colors */
                              crText = SetTextColor(lpDIS->hDC,
                                          GetSysColor(COLOR_HIGHLIGHTTEXT));
                              hbr =
CreateSolidBrush(GetSysColor(COLOR_HIGHLIGHT));
```

```
                             lpLI->fIsSelected = TRUE;
                       }
                       else {
                             hbr =
CreateSolidBrush(GetSysColor(COLOR_WINDOW));
                             lpLI->fIsSelected = FALSE;
                       }

                       FillRect(lpDIS->hDC, &lpDIS->rcItem, hbr);
                       DeleteObject(hbr);

                       nOldBkMode = SetBkMode(lpDIS->hDC, TRANSPARENT);

                       if (lpLI->lpszDisplayName) {
                             lstrcpy((LPSTR)sz, lpLI->lpszDisplayName);
                             lpsz = ChopText(
                                         lpDIS->hwndItem,
                                         nColPos[1] - nColPos[0]
                                                 - (nMaxCharWidth > 0 ?
nMaxCharWidth : 5),
                                         sz
                             );
                             rcClip.left = lpDIS->rcItem.left + nColPos[0];
                             rcClip.top = lpDIS->rcItem.top;
                             rcClip.right = lpDIS->rcItem.left + nColPos[1]
                                                     - (nMaxCharWidth > 0 ?
nMaxCharWidth : 5);
                             rcClip.bottom = lpDIS->rcItem.bottom;
                             ExtTextOut(
                                         lpDIS->hDC,
                                         rcClip.left,
                                         rcClip.top,
                                         ETO_CLIPPED,
                                         (LPRECT)&rcClip,
                                         lpsz,
                                         lstrlen(lpsz),
                                         NULL
                             );
                       }
                       if (lpLI->lpszShortLinkType) {
                             rcClip.left = lpDIS->rcItem.left + nColPos[1];
                             rcClip.top = lpDIS->rcItem.top;
                             rcClip.right = lpDIS->rcItem.left + nColPos[2]
                                                     - (nMaxCharWidth > 0 ?
nMaxCharWidth : 5);

                             rcClip.bottom = lpDIS->rcItem.bottom;
                             ExtTextOut(
                                         lpDIS->hDC,
                                         rcClip.left,
                                         rcClip.top,
                                         ETO_CLIPPED,
                                         (LPRECT)&rcClip,
                                         lpLI->lpszShortLinkType,
                                         lstrlen(lpLI->lpszShortLinkType),
                                         NULL
                             );
                       }
                       if (lpLI->lpszAMX) {
                             rcClip.left = lpDIS->rcItem.left + nColPos[2];
                             rcClip.top = lpDIS->rcItem.top;
                             rcClip.right = lpDIS->rcItem.right;
                             rcClip.bottom = lpDIS->rcItem.bottom;
                             ExtTextOut(
                                         lpDIS->hDC,
                                         rcClip.left,
                                         rcClip.top,
                                         ETO_CLIPPED,
                                         (LPRECT)&rcClip,
                                         lpLI->lpszAMX,
                                         lstrlen(lpLI->lpszAMX),
                                         NULL
                             );
                       }

                       SetBkMode(lpDIS->hDC, nOldBkMode);

                       // restore orig colors if we changed them
                       if (ODS_SELECTED & lpDIS->itemState)
                             SetTextColor(lpDIS->hDC, crText);

                 }

                 InitControls(hDlg, lpEL);

                 if (ODA_FOCUS & lpDIS->itemAction)
                       DrawFocusRect(lpDIS->hDC, &lpDIS->rcItem);

           }
           return TRUE;


           case WM_DELETEITEM:
           {
                 UINT  idCtl;
                 LPDELETEITEMSTRUCT  lpDIS;
                 LPLINKINFO  lpLI;

                 lpDIS = (LPDELETEITEMSTRUCT)lParam;
                 idCtl = wParam;
                 lpLI = (LPLINKINFO)lpDIS->itemData;

                 if (lpLI->lpszDisplayName)
                       OleStdFree((LPVOID)lpLI->lpszDisplayName);
                 if (lpLI->lpszShortLinkType)
                       OleStdFree((LPVOID)lpLI->lpszShortLinkType);
                 if (lpLI->lpszFullLinkType)
                       OleStdFree((LPVOID)lpLI->lpszFullLinkType);

                 /* The ChangeSource processing reuses allocated space for
```

```
                **    links that have been modified.
                */
                // Don't free the LINKINFO for the changed links
                if (lpLI->fDontFree)
                      lpLI->fDontFree = FALSE;
                else {
                      if (lpLI->lpszAMX)
                            OleStdFree((LPVOID)lpLI->lpszAMX);
                      OleStdFree((LPVOID)lpLI);
                }

                return TRUE;
          }

          case WM_COMPAREITEM:
          {
                LPCOMPAREITEMSTRUCT lpCIS = (LPCOMPAREITEMSTRUCT)lParam;
                LPLINKINFO lpLI1 = (LPLINKINFO)lpCIS->itemData1;
                LPLINKINFO lpLI2 = (LPLINKINFO)lpCIS->itemData2;

                // Sort list entries by DisplayName
                return lstrcmp(lpLI1->lpszDisplayName,lpLI2-
>lpszDisplayName);
          }

          case WM_COMMAND:
                switch (wID) {

                case ID_EL_CHANGESOURCE:
                {
                      BOOL fRet = FALSE;

                      /* This will bring up the file open dlg with one
                      edit field containing the whole link name.  The file
part
                      will (eventually) be highlighted to indicate where
the
                      file part is.  We need to hook on OK here to be able
to
                      send the changed string to the Parse function */

                      fRet = Container_ChangeSource(hDlg, lpEL);
                      if (!fRet)
                            {
#if defined( OBSOLETE )
                            PopupMessage(hDlg, (UINT)IDS_LINKS,
(UINT)IDS_FAILED,
                                          MB_ICONEXCLAMATION | MB_OK);
#endif
                            MessageBeep(MB_ICONEXCLAMATION);  // beep to
show failure.
                            }

                      InitControls(hDlg, lpEL);
                }
                break;

                case ID_EL_AUTOMATIC:
                {
                      /*  This is available for single or multi-select.
There is
                      a flag in the structure that is set initially
indicating
                      whether the link is auto or manual so that we need
not
                      query the link each time we want to find out.

                      This command will make the link automatic if not
already.
                      It will have no effect on links already set to auto.
                      */
                      // turn the button ON

                      CheckDlgButton(hDlg, ID_EL_AUTOMATIC, 1);
                      CheckDlgButton(hDlg, ID_EL_MANUAL, 0);

                      hErr = Container_AutomaticManual(hDlg, TRUE, lpEL);
                      if (hErr != NOERROR)
#if defined( OBSOLETE )
                            PopupMessage(hDlg, (UINT)IDS_LINKS,
(UINT)IDS_FAILED,
                                        MB_ICONEXCLAMATION | MB_OK);
#endif
                            // in order to avoid a recursion bug where
poping up
                            // and then closing the message box causes the
auto-radio
                            // button to re-take focus which causes it to
                            // automatically send another ID_EL_AUTOMATIC
message, we
                            // will only beep to indicate failure.
                            MessageBeep(MB_ICONEXCLAMATION);  // beep to
show failure.

                      InitControls(hDlg, lpEL);
                }
                break;

                case ID_EL_MANUAL:
                {
                      /* Same rules apply here as they do to auto link.
                      Additional note - just because something is changed
does
                      not mean that it updates at the moment.  It simply
                      reflects what kind of updating it does while it lives
in
                      the document.
                      */
                      // turn the button ON
```

```
                       CheckDlgButton(hDlg, ID_EL_MANUAL, 1);
                       CheckDlgButton(hDlg, ID_EL_AUTOMATIC, 0);

                       hErr = Container_AutomaticManual(hDlg, FALSE, lpEL);
                       if (hErr != NOERROR)
#if defined( OBSOLETE )
                             PopupMessage(hDlg, (UINT)IDS_LINKS,
(UINT)IDS_FAILED,
                                         MB_ICONEXCLAMATION | MB_OK);
#endif
                             // in order to avoid a recursion bug where
poping up
                             // and then closing the message box causes the
auto-radio
                             // button to re-take focus which causes it to
                             // automatically send another ID_EL_MANUAL
message, we
                             // will only beep to indicate failure.
                             MessageBeep(MB_ICONEXCLAMATION);  // beep to
show failure.

                       InitControls(hDlg, lpEL);
                 }
                 break;

                 case ID_EL_CANCELLINK:
                 {
                       /*  This is Break Link now in the dlg.  This sets the
                       moniker to null, thereby effectively breaking the
link.
                       The object still has its data effective at the time
of
                       breaking, but becomes a static object.
                       *****It will need to be deleted from the listbox
                       */

                       hErr = CancelLink(hDlg,lpEL);
                       if (hErr != NOERROR)
                             MessageBeep(MB_ICONEXCLAMATION);  // beep to
show failure.

                       InitControls(hDlg, lpEL);
                 }
                 break;

                 case ID_EL_UPDATENOW:
                 {
                       /*  This forces an immediate update of the selected
                       links.  This will start the server etc, so this is a
very
                       expensive operation.
                       */
                       HCURSOR hCur = HourGlassOn();
                       hErr = Container_UpdateNow(hDlg, lpEL);
                       if (hErr != NOERROR)
```

```
                                    MessageBeep(MB_ICONEXCLAMATION);  // beep to
show failure.

                              InitControls(hDlg, lpEL);
                              HourGlassOff(hCur);
                        }
                        break;

                        case ID_EL_OPENSOURCE:
                        {
                              /*  This will only work on single selection.  It
makes no
                              sense to open multiple sources at the same time,
since
                              the one opened will try to show itself and become the
                              primary operating target, so to speak.  Button is
greyed
                              if multi select.

                              Here we do not add the break because we want to exit
the
                              dlg in this case.
                              */
                              hErr = Container_OpenSource(hDlg, lpEL);
                              if (hErr != NOERROR) {
                                    MessageBeep(MB_ICONEXCLAMATION);  // beep to
show failure.
                                    InitControls(hDlg, lpEL);
                                    break;      // don't close dialog
                              }
                        }       // fall through

                        case ID_EL_CLOSE:
                        {
                              /* The user is finished with their editing - they now
                              return to their container document.

                              */
                              SendMessage(hDlg, uMsgEndDialog, OLEUI_OK, 0L);
                        }
                        break;

                        case IDCANCEL:
                        {
                              /*  This changes to CLOSE after the user does even
ONE
                              thing in the dlg.  Nothing can really effectively be
undone.
                              */
                              SendMessage(hDlg, uMsgEndDialog, OLEUI_CANCEL, 0L);
                        }
                        break;

                        case ID_OLEUIHELP:
                        {
```

```
                        PostMessage(lpEL->lpOEL->hWndOwner, uMsgHelp
                              , (WPARAM)hDlg, MAKELPARAM(IDD_EDITLINKS, 0));
                        break;
                  }
                  break;

/*                case EL_RESETLBOX:
                  {
                  HWND hListBox = GetDlgItem(hDlg, ID_EL_LINKSLISTBOX);
                  UINT cLinks;
                  SendMessage(hListBox, LB_RESETCONTENT, 0, 0L);
                  cLinks = LoadLinkLB(hListBox, lpEL->lpOEL-
>lpOleUILinkContainer);
                  lpEL->fItemsExist = (BOOL)cLinks;
                  InitControls(hDlg, lpEL);
                  }
                  break;  */
            }
            break;

            default:
            {
                  if (lpEL && iMsg == lpEL->nChgSrcHelpID) {
                        PostMessage(lpEL->lpOEL->hWndOwner, uMsgHelp,
                                    (WPARAM)hDlg,
MAKELPARAM(IDD_CHANGESOURCE, 0));
                  }
            }
            break;
      }

      return FALSE;
}




/*
 * FEditLinksInit
 *
 * Purpose:
 *  WM_INITIDIALOG handler for the Edit Links dialog box.
 *
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  wParam          WPARAM of the message
 *  lParam          LPARAM of the message
 *
 * Return Value:
 *  BOOL            Value to return for WM_INITDIALOG.
 */

BOOL FEditLinksInit(HWND hDlg, WPARAM wParam, LPARAM lParam)
{
     LPEDITLINKS              lpEL;
     LPOLEUIEDITLINKS         lpOEL;
     HFONT                    hFont;
     BOOL                     fDlgItem = FALSE;
     DWORD                    dwLink = 0;
     ULONG                    cLinks;
     LPOLEUILINKCONTAINER     lpOleUILinkCntr;
     int                      n;
     HWND                     hListBox = GetDlgItem(hDlg,
ID_EL_LINKSLISTBOX);



     //1.  Copy the structure at lParam into our instance memory.
     lpEL = (LPEDITLINKS)LpvStandardInit(hDlg, sizeof(OLEUIEDITLINKS), TRUE,
                  &hFont);

     //PvStandardInit send a termination to us already.
     if (NULL==lpEL)
           return FALSE;

     lpOEL=(LPOLEUIEDITLINKS)lParam;

     lpEL->lpOEL=lpOEL;

     lpOleUILinkCntr = lpEL->lpOEL->lpOleUILinkContainer;

     cLinks = LoadLinkLB(hListBox, lpOleUILinkCntr);
     if (cLinks < 0)
           return FALSE;

     fDlgItem = (BOOL)cLinks;
     lpEL->fItemsExist = (BOOL)cLinks;


     InitControls(hDlg, lpEL);

     //Copy other information from lpOEL that we might modify.

     //2.  If we got a font, send it to the necessary controls.
     if (NULL != hFont) {
           // Do this for as many controls as you need it for.
           // SendDlgItemMessage(hDlg, ID_<UFILL>, WM_SETFONT,
(WPARAM)hFont, 0L);
     }


     //3.  Show or hide the help button
     if (!(lpEL->lpOEL->dwFlags & ELF_SHOWHELP))
           StandardShowDlgItem(hDlg, ID_OLEUIHELP, SW_HIDE);

     /*
      * PERFORM OTHER INITIALIZATION HERE.  ON ANY LoadString
      * FAILURE POST OLEUI_MSG_ENDDIALOG WITH OLEUI_ERR_LOADSTRING.
      */
     //4.  If requested disable UpdateNow button
     if ((lpEL->lpOEL->dwFlags & ELF_DISABLEUPDATENOW))
           StandardShowDlgItem(hDlg, ID_EL_UPDATENOW, SW_HIDE);

     //5.  If requested disable OpenSource button
     if ((lpEL->lpOEL->dwFlags & ELF_DISABLEOPENSOURCE))
           StandardShowDlgItem(hDlg, ID_EL_OPENSOURCE, SW_HIDE);

     //6.  If requested disable UpdateNow button
     if ((lpEL->lpOEL->dwFlags & ELF_DISABLECHANGESOURCE))
           StandardShowDlgItem(hDlg, ID_EL_CHANGESOURCE, SW_HIDE);

     //7.  If requested disable CancelLink button
     if ((lpEL->lpOEL->dwFlags & ELF_DISABLECANCELLINK))
           StandardShowDlgItem(hDlg, ID_EL_CANCELLINK, SW_HIDE);

     //8. Load 'Close' string used to rename Cancel button
     n = LoadString(ghInst, (UINT)IDS_CLOSE, lpEL->szClose, sizeof(lpEL-
>szClose));
     if (!n)
     {
           PostMessage(hDlg, uMsgEndDialog, OLEUI_ERR_LOADSTRING, 0L);
           return FALSE;
     }

     if (cLinks > 0)
           SetFocus(hListBox);
     else
           SetFocus(GetDlgItem(hDlg, IDCANCEL));

     lpEL->nChgSrcHelpID = RegisterWindowMessage(HELPMSGSTRING);

      //n.  Call the hook with lCustData in lParam
     UStandardHook(lpEL, hDlg, WM_INITDIALOG, wParam, lpOEL->lCustData);

     return FALSE;
}



/*
 * ChangeSourceHook
 *
 * Purpose:
 *  Hooks the ChangeSource dialog to attempt to validate link source changes
 *  specified by the user.
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  uMsg            UINT Message
 *  wParam          WPARAM of the message
 *  lParam          LPARAM of the message
 *
 * Return Value:
```

```
 *  UNIT            Zero = Do default processing;
 *                  Non Zero = Don't do default processing.
 */

UINT CALLBACK EXPORT ChangeSourceHook(HWND hDlg, UINT uMsg, WPARAM wParam,
LPARAM lParam)
{
     LPCHANGESOURCEHOOKDATA lpCshData = NULL;
     LPLINKINFO lpLI = NULL;
     LPEDITLINKS lpEL = NULL;
     LPOLEUILINKCONTAINER lpOleUILinkCntr;
     HRESULT hErr;
     UINT uRet;
     ULONG ulChEaten;
     HGLOBAL gh;

     //This will fail under WM_INITDIALOG, where we allocate it.
     if (NULL!=(gh = GetProp(hDlg, STRUCTUREPROP)))
           {
           // gh was locked previously, lock and unlock to get lpv
           lpCshData=(LPCHANGESOURCEHOOKDATA)GlobalLock(gh);
           GlobalUnlock(gh);
           if (lpCshData)
                 {
                 lpLI = lpCshData->lpOCshData->lpLI;
                 lpEL = lpCshData->lpOCshData->lpEL;
                 }
           }

     //Process the temination message
     if (uMsg==uMsgEndDialog)
           {
           if (NULL!=(gh = RemoveProp(hDlg, STRUCTUREPROP)))
                 {
                 GlobalUnlock(gh);
                 GlobalFree(gh);
                 }
           return TRUE;
           }

     // User pressed the OK button
     if (uMsg == uMsgFileOKString) {
           lpOleUILinkCntr = lpEL->lpOEL->lpOleUILinkContainer;

           /* NOTE: trigger the focus lost of the edit control. This is
           **    not necessary if the user click OK with the mouse but is
           **    needed when the user just press <Enter>. If the mouse was
           **    used, no extra is done as the MODIFY flag of the edit
control
           **    has been cleared.
           */
           SendMessage(hDlg, WM_COMMAND, edt1,
                       MAKELPARAM(GetDlgItem(hDlg, edt1), EN_KILLFOCUS));
           if (lpCshData->bItemNameStored) {
                 lpCshData->nFileLength = lstrlen((LPSTR)lpCshData->szEdit)
-
                             lstrlen((LPSTR)lpCshData->szItemName);
           }
           else {
                 lpCshData->nFileLength = lstrlen((LPSTR)lpCshData->szEdit);
           }

           // Try to validate link source change
           OLEDBG_BEGIN2("IOleUILinkContainer::SetLinkSource called\r\n");
           hErr = lpOleUILinkCntr->lpVtbl->SetLinkSource(
                       lpOleUILinkCntr,
                       lpLI->dwLink,
                       (LPSTR)lpCshData->szEdit,
                       (ULONG)lpCshData->nFileLength,
                       &ulChEaten,
                       TRUE
           );
           OLEDBG_END2

           // Link source change  not validated
           if (hErr != NOERROR) {
                 uRet =PopupMessage(hDlg, (UINT)IDS_CHANGESOURCE,
(UINT)IDS_INVALIDSOURCE,
                             MB_ICONQUESTION | MB_YESNO);

                 if (uRet == IDYES) {
                       /* User wants to correct invalid link. Set the edit
                       **    control selection to the invalid part of the
contents.
                       */
                       SetFocus(GetDlgItem(hDlg, edt1));
#ifdef WIN32
                       SendDlgItemMessage(hDlg, edt1, EM_SETSEL, ulChEaten,
(DWORD)-1);
#else
                       SendDlgItemMessage(hDlg, edt1, EM_SETSEL, 0,
                                   MAKELPARAM(ulChEaten, -1));
#endif
                       return 1; // Don't close ChangeSource dialog
                 }
                 else {
                       /* User does not want to correct invalid link. So set
                       **    the link source but don't validate the link.
                       */
                       OLEDBG_BEGIN2("IOleUILinkContainer::SetLinkSource
called\r\n");
                       hErr = lpOleUILinkCntr->lpVtbl->SetLinkSource(
                                   lpOleUILinkCntr,
                                   lpLI->dwLink,
                                   (LPSTR)lpCshData->szEdit,
                                   (ULONG)lpCshData->nFileLength,
                                   &ulChEaten,
                                   FALSE
                       );
```

```
                OLEDBG_END2
                lpCshData->fValidLink = FALSE;
            }
        }
        else {     // Link source change validated
            lpCshData->fValidLink = TRUE;
        }

        if (lpCshData->bItemNameStored && lpCshData->bFileNameStored) {
            HWND  hListBox = GetDlgItem(lpCshData->lpOfn->hwndOwner,
ID_EL_LINKSLISTBOX);

            DiffPrefix(
                    lpLI->lpszDisplayName,
                    (LPSTR)lpCshData->szEdit,
                    (char FAR* FAR*)&lpCshData->lpszFrom,
                    (char FAR* FAR*)&lpCshData->lpszTo
            );

            /* we keep the strings if there is a difference between the
            **    lpszFrom and lpszTo strings AND if the change is only
            **    in the file portion otherwise free them and other
            **    links won't be compared.
            */
            if ( (lstrcmp(lpCshData->lpszTo, lpCshData->lpszFrom)==0)
                    || (lstrlen(lpCshData->lpszTo)>lpCshData-
>nFileLength)) {
                if (lpCshData->lpszFrom) {
                    OleStdFree(lpCshData->lpszFrom);
                    lpCshData->lpszFrom = NULL;
                }
                if (lpCshData->lpszTo) {
                    OleStdFree(lpCshData->lpszTo);
                    lpCshData->lpszTo = NULL;
                }
            }
        }

        // Copy OUT results to original structure
        lpCshData->lpOCshData->lpszFrom = lpCshData->lpszFrom;
        lpCshData->lpOCshData->lpszTo = lpCshData->lpszTo;
        lpCshData->lpOCshData->fValidLink = lpCshData->fValidLink;

        SendMessage(hDlg, uMsgEndDialog, 0, 0L);    // do cleanup
        return 0;       // Close ChangeSource dialog
    }

    switch (uMsg) {
        case WM_INITDIALOG:
        {
            LPOPENFILENAME lpOfn = (LPOPENFILENAME)lParam;
            LPOLEUICHANGESOURCEHOOKDATA lpOCshData =
                    (LPOLEUICHANGESOURCEHOOKDATA)lpOfn->lCustData;

            gh=GlobalAlloc(
                             GMEM_MOVEABLE|
GMEM_ZEROINIT,sizeof(CHANGESOURCEHOOKDATA));
                 if (NULL==gh)
                       {
                       // Memory allocation error; fail to bring up dialog
                       PostMessage(hDlg, uMsgEndDialog, 0, 0L);
                       return 0;
                       }
                 lpCshData = GlobalLock(gh);
                 SetProp(hDlg, STRUCTUREPROP, gh);

                 lpCshData->lpOCshData = lpOCshData;
                 lpCshData->lpOfn = lpOfn;
                 lpLI = lpCshData->lpOCshData->lpLI;

                 lpCshData->bFileNameStored = TRUE;
                 lpCshData->bItemNameStored = TRUE;
                 lpCshData->nFileLength = (int)lpLI->clenFileName;
                 if (lpLI->lpszDisplayName) {
                       LSTRCPYN((LPSTR)lpCshData->szFileName, lpLI-
>lpszDisplayName,
                                   lpCshData->nFileLength + 1);
                       lstrcpy((LPSTR)lpCshData->szEdit, lpLI-
>lpszDisplayName);
                 } else {
                       lpCshData->szFileName[0] = '\0';
                       lpCshData->szEdit[0] = '\0';
                 }
                 if (lpLI->lpszItemName)
                       lstrcpy((LPSTR)lpCshData->szItemName, lpLI-
>lpszItemName);
                 else
                       lpCshData->szItemName[0] = '\0';

                 return 0;
           }

           case WM_COMMAND:

                 // User pressed the CANCEL button
                 if (wParam == IDCANCEL)
                       {
                       /* If the focus is in the textbox, and the ESC key
was hit, we
                             will get the WM_COMMAND/IDCANCEL message BEFORE
the
                             EN_KILLFOCUS message.  We clear the MODIFY flag
here to
                             make sure that the EN_KILLFOCUS message doesn't
try to
                             access any structures that are freed by
uMsgEndDialog */
                       SendMessage(GetDlgItem(hDlg, edt1), EM_SETMODIFY,
(WPARAM)FALSE, 0L);
```

```
                       if (lpCshData->lpszFrom)
                             {
                             OleStdFree(lpCshData->lpszFrom);
                             lpCshData->lpszFrom = NULL;
                             }
                       if (lpCshData->lpszTo)
                             {
                             OleStdFree(lpCshData->lpszTo);
                             lpCshData->lpszTo = NULL;
                             }

                       // Copy OUT results to original structure
                       lpCshData->lpOCshData->lpszFrom = NULL;
                       lpCshData->lpOCshData->lpszTo = NULL;
                       lpCshData->lpOCshData->fValidLink = FALSE;

                       SendMessage(hDlg, uMsgEndDialog, 0, 0L);    // do
cleanup
                       return 0;       // Close ChangeSource dialog
                       }

#ifdef WIN32
                 if ((LOWORD(wParam) == lst1) &&
                       (HIWORD(wParam) == LBN_SELCHANGE)) {
#else
                 if ((wParam == lst1) &&
                       (HIWORD(lParam) == LBN_SELCHANGE)) {
#endif

                       int     nIndex;
                       HWND    hListBox = (HWND)LOWORD(lParam);
                       static char szFileNameBuf[OLEUI_CCHPATHMAX];
                       static char szEditBuf[OLEUI_CCHPATHMAX];

                       nIndex = (int)SendMessage(hListBox, LB_GETCURSEL, 0,
0L);
                       SendMessage(hListBox, LB_GETTEXT,
                                   (WPARAM)nIndex, (LPARAM)
(LPSTR)szFileNameBuf);

                       /* need to build the full path filename for the
moniker */
                       AnsiToOem(szFileNameBuf, szFileNameBuf);
                       _fullpath(szEditBuf, szFileNameBuf,
sizeof(szEditBuf));
                       OemToAnsi(szEditBuf, szEditBuf);

                       /* convert filename to lower case as it appears in
the
                       **    listbox
                       */
                       AnsiLower((LPSTR)szEditBuf);
                       LSTRCPYN((LPSTR)lpCshData->szEdit, (LPSTR)szEditBuf,
                                   sizeof(lpCshData->szEdit));
                       LSTRCPYN((LPSTR)lpCshData->szFileName,
                                   (LPSTR)lpCshData->szEdit,
                                   sizeof(lpCshData->szFileName));
                       lpCshData->nFileLength = lstrlen((LPSTR)lpCshData-
>szEdit);
                       if (lpCshData->bItemNameStored)
                             lstrcat((LPSTR)lpCshData->szEdit, lpCshData-
>szItemName);

                       SetDlgItemText(hDlg, edt1, (LPSTR)lpCshData->szEdit);
                       lpCshData->nEditLength = lstrlen((LPSTR)lpCshData-
>szEdit);
                       lpCshData->bFileNameStored = TRUE;

                       return 1;
                 }

#ifdef WIN32
                 if ((LOWORD(wParam) == lst2) &&
                       (HIWORD(wParam) == LBN_SELCHANGE)) {
#else
                 if ((LOWORD(wParam) == lst2) &&
                       (HIWORD(lParam) == LBN_SELCHANGE)) {
#endif

                       if (lpCshData->bItemNameStored)
                             SetDlgItemText(hDlg, edt1, (LPSTR)lpCshData-
>szItemName);

                       return 1;
                 }

#ifdef WIN32
                 if ((LOWORD(wParam) == cmb2) &&
                       (HIWORD(wParam) == CBN_SELCHANGE)) {
#else
                 if ((LOWORD(wParam) == cmb2) &&
                       (HIWORD(lParam) == CBN_SELCHANGE)) {
#endif

                       if (lpCshData->bItemNameStored)
                             SetDlgItemText(hDlg, edt1, (LPSTR)lpCshData-
>szItemName);

                       return 1;
                 }

                 if (LOWORD(wParam) == edt1) {
#ifdef WIN32
                       HWND hEdit = (HWND)lParam;
#else
                       HWND hEdit = (HWND)LOWORD(lParam);
#endif


#ifdef WIN32
```

```
                       switch (HIWORD(wParam)) {
#else
                       switch (HIWORD(lParam)) {
#endif
                             case EN_SETFOCUS:
#ifdef WIN32
                                   SendMessage(hEdit, EM_SETSEL, 0,
lpCshData->nFileLength);
#else
                                   SendMessage(hEdit, EM_SETSEL, 0,
                                               MAKELPARAM(0, lpCshData-
>nFileLength));
#endif
                                   return 1;

                             case EN_KILLFOCUS:
                                   if (SendMessage(hEdit, EM_GETMODIFY, 0,
0L)) {
                                         char szTmp[OLEUI_CCHPATHMAX];
                                         int nItemLength =
lstrlen((LPSTR)lpCshData->szItemName);

                                         *(LPWORD)lpCshData->szEdit =
sizeof(lpCshData->szEdit) - 1;
                                         lpCshData->nEditLength =
(int)SendMessage(hEdit,
                                                     EM_GETLINE, 0, (LPARAM)
(LPSTR)lpCshData->szEdit);
                                         lpCshData->szEdit[lpCshData-
>nEditLength] = '\0';
                                         LSTRCPYN((LPSTR)szTmp,
(LPSTR)lpCshData->szEdit,
                                                     lpCshData->nFileLength
+ 1);

                                         if (lpCshData->bFileNameStored &&
                                               !lstrcmp((LPSTR)lpCshData-
>szFileName, (LPSTR)szTmp)) {

     lstrcpy((LPSTR)lpCshData->szItemName,

     (LPSTR)lpCshData->szEdit + lpCshData->nFileLength);
                                                     lpCshData-
>bItemNameStored = TRUE;
                                         }
                                         else if (lpCshData->bItemNameStored
&&
                                                     !
lstrcmp((LPSTR)lpCshData->szItemName,

     (LPSTR)lpCshData->szEdit +

     lpCshData->nEditLength -

     nItemLength)) {
```

```
                                                     if (lpCshData-
>nEditLength==nItemLength) {

                                                           lpCshData-
>bFileNameStored = FALSE;
                                                     } else {

     LSTRCPYN((LPSTR)lpCshData->szFileName,

     (LPSTR)lpCshData->szEdit,

     lpCshData->nEditLength -

     nItemLength+1);
                                                           lpCshData-
>bFileNameStored = TRUE;
                                                     }
                                         }
                                         else {
                                               lpCshData->bItemNameStored =
FALSE;
                                               lpCshData->bFileNameStored =
FALSE;
                                         }

                                         SendMessage(hEdit, EM_SETMODIFY,
FALSE, 0L);
                                   }
                                   return 0;
                       }
                 }
                 return 0;

           default:
                 return 0;
     }
}

/*
* ChangeSource
*
* Purpose:
*  Displays the standard GetOpenFileName dialog with a customized template
and
*  hook.
*
* Parameters:
*  hWndOwner       HWND owning the dialog
*  lpszFile        LPSTR specifying the initial file.  If there is no
*                  initial file the first character of this string should
*                  be a null.
*  cchFile         UINT length of pszFile
*  iFilterString   UINT index into the stringtable for the filter string.
*  lpfnBrowseHook  COMMDLGHOOKPROC hook to process link source information
when user
```

```
*                                   presses OK
*  lpCshData       LPCHANGESOURCEHOOKDATA custom data that is accessible to
the hook
*
* Return Value:
*  BOOL            TRUE if the user selected a file and pressed OK.
*                  FALSE otherwise, such as on pressing Cancel.
*/

BOOL WINAPI ChangeSource(
           HWND hWndOwner,
           LPSTR lpszFile,
           UINT cchFile,
           UINT iFilterString,
           COMMDLGHOOKPROC lpfnBrowseHook,
           LPOLEUICHANGESOURCEHOOKDATA lpCshData
)
{
     UINT            cch;
     char            szFilters[OLEUI_CCHPATHMAX];
     char            szDir[OLEUI_CCHPATHMAX];
     char            szTitle[OLEUI_CCHPATHMAX];
     OPENFILENAME    ofn;
     BOOL            fStatus;
     LPSTR           lpszFileBuffer;
#if !defined( LINK_COMMDLG )
        HINSTANCE hinstCommDlg = NULL;
        typedef BOOL (WINAPI* LPFNGETOPENFILENAME) (OPENFILENAME FAR*);
        LPFNGETOPENFILENAME lpfnGetOpenFileName;
#endif

     if (NULL==lpszFile || 0==cchFile)
           return FALSE;

     lpszFileBuffer = (LPSTR)OleStdMalloc(cchFile * sizeof(char));
     if (!lpszFileBuffer)
           return FALSE;

     lstrcpy(lpszFileBuffer, lpszFile);

     // Get filters
     if (0!=iFilterString)
           cch = LoadString(ghInst, iFilterString, (LPSTR)szFilters,
                       sizeof(szFilters));
     else
     {
           szFilters[0]=0;
           cch=1;
     }
     if (0==cch) {
           fStatus = FALSE;
           goto cleanup;
     }

     ReplaceCharWithNull(szFilters, szFilters[cch-1]);
     LSTRCPYN((LPSTR)szDir, lpszFile, sizeof(szDir));
     for (cch = lstrlen((LPSTR)szDir) - 1; cch >= 0; cch--)
           {
           if ((szDir[cch]=='\\') || (szDir[cch]==':') || (szDir[cch]=='/'))
                 break;
           }
     if (cch < 0)
           cch = 0;

     szDir[cch] = '\0';

     LoadString(ghInst, (UINT)IDS_CHANGESOURCE, (LPSTR)szTitle,
sizeof(szTitle));
     _fmemset((LPOPENFILENAME)&ofn, 0, sizeof(ofn));
     ofn.lStructSize     = sizeof(ofn);
     ofn.hwndOwner       = hWndOwner;
     ofn.lpstrFile       = lpszFileBuffer;
     ofn.nMaxFile        = cchFile;
     ofn.lpstrFilter     = (LPSTR)szFilters;
     ofn.nFilterIndex    = 1;
     ofn.lpstrTitle      = (LPSTR)szTitle;
     ofn.lpstrInitialDir = (LPSTR)szDir;
     ofn.lpTemplateName  = MAKEINTRESOURCE(IDD_FILEOPEN);
     ofn.lpfnHook        = lpfnBrowseHook;
     ofn.hInstance       = ghInst;
     ofn.lCustData       = (LPARAM)lpCshData;
     ofn.Flags           = OFN_NOVALIDATE | OFN_HIDEREADONLY |
                                         OFN_ENABLETEMPLATE |
OFN_ENABLEHOOK;

   // Only show help button if edit links dialog shows it.
   if (lpCshData->lpEL->lpOEL->dwFlags & ELF_SHOWHELP)
      ofn.Flags |= OFN_SHOWHELP;

#if !defined( LINK_COMMDLG )
     // Load the COMMDLG.DLL library module
     hinstCommDlg = LoadLibrary("COMMDLG.DLL");
     if ((DWORD)hinstCommDlg <= HINSTANCE_ERROR)   /* load failed */
           return FALSE;

     // Retrieve the address of GetOpenFileName function
     lpfnGetOpenFileName = (LPFNGETOPENFILENAME)
                 GetProcAddress(hinstCommDlg, "GetOpenFileName");

     if (lpfnGetOpenFileName == NULL)
           {
           FreeLibrary(hinstCommDlg);
           return FALSE;
           }

     fStatus = (*lpfnGetOpenFileName) ((LPOPENFILENAME)&ofn);

     FreeLibrary(hinstCommDlg);
#else
```

```
     fStatus = GetOpenFileName((LPOPENFILENAME)&ofn);
#endif

cleanup:
     OleStdFree((LPVOID)lpszFileBuffer);
     return fStatus;

}

/*
* Container_ChangeSource
*
* Purpose:
*  Tunnel to File Open type dlg and allow user to select new file
*  for file based monikers, OR to change the whole moniker to what
*  the user types into the editable field.
*
* Parameters:
*  hDlg            HWND of the dialog
*  LPEDITLINKS     Pointer to EditLinks structure (contains all nec.
*              info)
*
* Return Value:
*  BOOL          for now, because we are not using any ole functions
*                to return an HRESULT.
*  HRESULT       HRESULT value indicating success or failure of
*              changing the moniker value
*/

BOOL Container_ChangeSource(HWND hDlg, LPEDITLINKS lpEL)
{
     UINT        uRet;
     int         cSelItems;
     int FAR*    rgIndex;
     int         i = 0;
     LPLINKINFO  lpLI;
     HWND        hListBox = GetDlgItem(hDlg, ID_EL_LINKSLISTBOX);
     LPOLEUILINKCONTAINER lpOleUILinkCntr = lpEL->lpOEL-
>lpOleUILinkContainer;
     OLEUICHANGESOURCEHOOKDATA  cshData;     // Data that needs to be
accessed
                                                                 // by the
ChangeSource dialog hook
     BOOL fUpdated = FALSE;

     cSelItems = GetSelectedItems(hListBox, &rgIndex);

     if (cSelItems < 0)
          return FALSE;

     if (!cSelItems)
          return TRUE;

     if (!lpEL->fClose) {
          SetWindowText(GetDlgItem(hDlg, IDCANCEL), (LPSTR)lpEL->szClose);
```

```
            lpEL->fClose = TRUE;
      }

      _fmemset((LPOLEUICHANGESOURCEHOOKDATA)&cshData, 0, sizeof(cshData));
      cshData.cbStruct=sizeof(cshData);
      cshData.hWndOwner=hDlg;
      cshData.lpEL = (LPEDITLINKS)lpEL;
      cshData.lpszFrom = NULL;
      cshData.lpszTo = NULL;

      for (i = cSelItems-1; i >=0; i--) {
            SendMessage(hListBox, LB_GETTEXT, rgIndex[i],
                        (LPARAM) (LPLINKINFO FAR*) &lpLI);

            uRet = UStandardHook(lpEL, hDlg, uMsgBrowse,
                        OLEUI_CCHPATHMAX, (LONG)(LPSTR)lpLI-
>lpszDisplayName);

            if (!uRet) {
                  cshData.lpLI = lpLI;
                  /* Bring up the ChangeSource dialog after hooking it so
                  **    that the user specified link source is verified
                  **    when OK is pressed.
                  */
                  uRet = (UINT)ChangeSource(hDlg, lpLI->lpszDisplayName,
                              OLEUI_CCHPATHMAX, (UINT)IDS_FILTERS,
ChangeSourceHook,
                              &cshData);
            }

            /* If Cancel is pressed in any ChangeSource dialog, stop
            **    the ChangeSource processing for all links.
            */
            if (!uRet) {
                  if (rgIndex)
                        OleStdFree(rgIndex);
                  return TRUE;
            }

            if (cshData.lpszFrom && cshData.lpszTo &&
                  (CFindLinks(hListBox, cshData.lpszFrom) > 1)) {
                  fUpdated = FChangeAllLinks(hListBox, lpOleUILinkCntr,
                              cshData.lpszFrom, cshData.lpszTo);
            }

            if (cshData.lpszFrom)
                  OleStdFree(cshData.lpszFrom);
            if (cshData.lpszTo)
                  OleStdFree(cshData.lpszTo);

            /* If the link was not updated by FChangeAllLinks, then change it
now */
            if (!fUpdated)
                  UpdateLinkLBItem(hListBox, rgIndex[i], lpEL, TRUE);
```

```
     } // end FOR


     if (rgIndex)
          OleStdFree(rgIndex);

     return TRUE;

}


/*
* Container_AutomaticManual
*
* Purpose:
*   To change the selected moniker to manual or automatic update.
*
* Parameters:
*  hDlg            HWND of the dialog
*  FAutoMan        Flag indicating AUTO (TRUE/1) or MANUAL(FALSE/0)
*  LPEDITLINKS     Pointer to EditLinks structure (contains all nec.
*              info)
*            * this may change - don't know how the linked list
*            * of multi-selected items will work.
* Return Value:
*  HRESULT       HRESULT value indicating success or failure of
*              changing the moniker value
*/

HRESULT Container_AutomaticManual(HWND hDlg, BOOL fAutoMan, LPEDITLINKS
lpEL)
{

     HRESULT hErr = NOERROR;
     int cSelItems;
     int FAR* rgIndex;
     int i = 0;
     LPLINKINFO  lpLI;
     LPOLEUILINKCONTAINER lpOleUILinkCntr = lpEL->lpOEL-
>lpOleUILinkContainer;
     HWND        hListBox = GetDlgItem(hDlg, ID_EL_LINKSLISTBOX);
     BOOL        bUpdate = FALSE;

     OleDbgAssert(lpOleUILinkCntr);

     /* Change so looks at flag in structure.  Only update those that
     need to be updated.  Make sure to change flag if status changes.
     */

     cSelItems = GetSelectedItems(hListBox, &rgIndex);

     if (cSelItems < 0)
          return ResultFromScode(E_FAIL);

     if (!cSelItems)
```

```
        return NOERROR;

    if (!lpEL->fClose)
        SetDlgItemText(hDlg, IDCANCEL, (LPSTR)lpEL->szClose);

    for (i = 0; i < cSelItems; i++) {
        SendMessage(hListBox, LB_GETTEXT, (WPARAM)rgIndex[i],
                (LPARAM) (LPLINKINFO FAR*) &lpLI);

        if (fAutoMan) {   // If switching to AUTOMATIC
            if (!lpLI->fIsAuto) {   // Only change MANUAL links

    OLEDBG_BEGIN2("IOleUILinkContainer::SetLinkUpdateOptions called\r\n");
                hErr=lpOleUILinkCntr->lpVtbl->SetLinkUpdateOptions(
                        lpOleUILinkCntr,
                        lpLI->dwLink,
                        OLEUPDATE_ALWAYS
                );
                OLEDBG_END2

                lpLI->fIsAuto=TRUE;
                lpLI->fIsMarked = TRUE;
                bUpdate = TRUE;
            }
        }
        else {   // If switching to MANUAL
            if (lpLI->fIsAuto) {  // Only do AUTOMATIC Links

    OLEDBG_BEGIN2("IOleUILinkContainer::SetLinkUpdateOptions called\r\n");
                hErr=lpOleUILinkCntr->lpVtbl->SetLinkUpdateOptions(
                        lpOleUILinkCntr,
                        lpLI->dwLink,
                        OLEUPDATE_ONCALL
                );
                OLEDBG_END2

                lpLI->fIsAuto = FALSE;
                lpLI->fIsMarked = TRUE;
                bUpdate = TRUE;
            }
        }

        if (hErr != NOERROR) {
            OleDbgOutHResult("WARNING:
IOleUILinkContainer::SetLinkUpdateOptions returned",hErr);
            break;
        }

    }

    if (bUpdate)
        RefreshLinkLB(hListBox, lpOleUILinkCntr);

    if (rgIndex)
        OleStdFree((LPVOID)rgIndex);
```

```
     return hErr;
}


HRESULT CancelLink(HWND hDlg, LPEDITLINKS lpEL)
{
     HRESULT hErr;
     LPMONIKER lpmk;
     int cSelItems;
     int FAR* rgIndex;
     int i = 0;
     LPLINKINFO  lpLI;
     LPOLEUILINKCONTAINER lpOleUILinkCntr = lpEL->lpOEL-
>lpOleUILinkContainer;
     HWND        hListBox = GetDlgItem(hDlg, ID_EL_LINKSLISTBOX);
     BOOL        bUpdate = FALSE;

     OleDbgAssert(lpOleUILinkCntr);

     lpmk = NULL;

     cSelItems = GetSelectedItems(hListBox, &rgIndex);

     if (cSelItems < 0)
           return ResultFromScode(E_FAIL);

     if (!cSelItems)
           return NOERROR;

     if (!lpEL->fClose) {
           SetWindowText(GetDlgItem(hDlg, IDCANCEL), (LPSTR)lpEL->szClose);
           lpEL->fClose = TRUE;
     }

     for (i = (cSelItems - 1); i >= 0; i--) {
           SendMessage(hListBox, LB_GETTEXT, (WPARAM)rgIndex[i],
                       (LPARAM)(LPLINKINFO FAR*) &lpLI);

           OLEDBG_BEGIN2("IOleUILinkContainer::CancelLink called\r\n");
           hErr = lpOleUILinkCntr->lpVtbl->CancelLink(
                       lpOleUILinkCntr,
                       lpLI->dwLink
           );
           OLEDBG_END2

           if (hErr != NOERROR) {
                 OleDbgOutHResult("WARNING: IOleUILinkContainer::CancelLink
returned",hErr);
                 lpLI->fIsMarked = TRUE;
                 bUpdate = TRUE;
           }
           else
                 // Delete links that we make null from listbox
                 SendMessage(hListBox, LB_DELETESTRING, (WPARAM) rgIndex[i],
0L);

     }

     if (bUpdate)
           RefreshLinkLB(hListBox, lpOleUILinkCntr);
     SetFocus(hListBox);

     if (rgIndex)
           OleStdFree((LPVOID)rgIndex);

     return hErr;

}


/*
 * Container_UpdateNow
 *
 * Purpose:
 *   Immediately force an update for all (manual) links
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  LPEDITLINKS     Pointer to EditLinks structure (contains all nec. info)
 *            * this may change - don't know how the linked list
 *            * of multi-selected items will work.
 * Return Value:
 *  HRESULT       HRESULT value indicating success or failure of
 *              changing the moniker value
 */

HRESULT Container_UpdateNow(HWND hDlg, LPEDITLINKS lpEL)
{
     HRESULT         hErr;
     LPLINKINFO      lpLI;
     int cSelItems;
     int FAR* rgIndex;
     int i = 0;
     LPOLEUILINKCONTAINER lpOleUILinkCntr = lpEL->lpOEL-
>lpOleUILinkContainer;
     HWND        hListBox = GetDlgItem(hDlg, ID_EL_LINKSLISTBOX);
     BOOL        bUpdate = FALSE;

     OleDbgAssert(lpOleUILinkCntr);

     cSelItems = GetSelectedItems(hListBox, &rgIndex);

     if (cSelItems < 0)
           return ResultFromScode(E_FAIL);

     if (!cSelItems)
           return NOERROR;
     if (!lpEL->fClose) {
           SetWindowText(GetDlgItem(hDlg, IDCANCEL), (LPSTR)lpEL->szClose);
           lpEL->fClose = TRUE;
     }

     for (i = 0; i < cSelItems; i++) {
           SendMessage(hListBox, LB_GETTEXT,
                       (WPARAM)rgIndex[i], (LPARAM)(LPLINKINFO FAR*)&lpLI);

           OLEDBG_BEGIN2("IOleUILinkContainer::UpdateLink called\r\n");
           hErr = lpOleUILinkCntr->lpVtbl->UpdateLink(
                       lpOleUILinkCntr,
                       lpLI->dwLink,
                       TRUE,
                       FALSE
           );
           OLEDBG_END2
           bUpdate = TRUE;
           lpLI->fIsMarked = TRUE;

           if (hErr != NOERROR) {
                 OleDbgOutHResult("WARNING: IOleUILinkContainer::UpdateLink
returned",hErr);
                 break;
           }

     }

     if (bUpdate)
           RefreshLinkLB(hListBox, lpOleUILinkCntr);

     if (rgIndex)
           OleStdFree((LPVOID)rgIndex);

     return hErr;

}

/*
 * Container_OpenSource
 *
 * Purpose:
 *   Immediately force an update for all (manual) links
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  LPEDITLINKS     Pointer to EditLinks structure (contains all nec.
 *              info)
 *
 * Return Value:
 *  HRESULT       HRESULT value indicating success or failure of
 *              changing the moniker value
 */

HRESULT Container_OpenSource(HWND hDlg, LPEDITLINKS lpEL)
```

```
{
     HRESULT         hErr;
     int             cSelItems;
     int FAR*        rgIndex;
     LPLINKINFO      lpLI;
     RECT            rcPosRect;
     LPOLEUILINKCONTAINER lpOleUILinkCntr = lpEL->lpOEL-
>lpOleUILinkContainer;
     HWND            hListBox = GetDlgItem(hDlg, ID_EL_LINKSLISTBOX);

     OleDbgAssert(lpOleUILinkCntr);

     rcPosRect.top = 0;
     rcPosRect.left = 0;
     rcPosRect.right = 0;
     rcPosRect.bottom = 0;

     cSelItems = GetSelectedItems(hListBox, &rgIndex);

     if (cSelItems < 0)
           return ResultFromScode(E_FAIL);

     if (cSelItems != 1)     // can't open source for multiple items
           return NOERROR;

     if (!lpEL->fClose) {
           SetWindowText(GetDlgItem(hDlg, IDCANCEL), (LPSTR)lpEL->szClose);
           lpEL->fClose = TRUE;
     }

     SendMessage(hListBox, LB_GETTEXT, (WPARAM)rgIndex[0],
                 (LPARAM)(LPLINKINFO FAR*)&lpLI);

     OLEDBG_BEGIN2("IOleUILinkContainer::OpenLinkSource called\r\n");
     hErr = lpOleUILinkCntr->lpVtbl->OpenLinkSource(
                 lpOleUILinkCntr,
                 lpLI->dwLink
     );
     OLEDBG_END2

     UpdateLinkLBItem(hListBox, rgIndex[0], lpEL, TRUE);
     if (hErr != NOERROR)
           OleDbgOutHResult("WARNING: IOleUILinkContainer::OpenLinkSource
returned",hErr);

     if (rgIndex)
           OleStdFree((LPVOID)rgIndex);

     return hErr;
}



/* AddLinkLBItem
** -------------
```

```
**
**    Add the item pointed to by lpLI to the Link ListBox and return
**    the index of it in the ListBox
*/
int AddLinkLBItem(HWND hListBox, LPOLEUILINKCONTAINER lpOleUILinkCntr,
LPLINKINFO lpLI, BOOL fGetSelected, int nIndex)
{
     HRESULT hErr;
     DWORD dwUpdateOpt;

     OleDbgAssert(lpOleUILinkCntr && hListBox && lpLI);

     lpLI->fDontFree = FALSE;

     OLEDBG_BEGIN2("IOleUILinkContainer::GetLinkSource called\r\n");
     hErr = lpOleUILinkCntr->lpVtbl->GetLinkSource(
                  lpOleUILinkCntr,
                  lpLI->dwLink,
                  (LPSTR FAR*)&lpLI->lpszDisplayName,
                  (ULONG FAR*)&lpLI->clenFileName,
                  (LPSTR FAR*)&lpLI->lpszFullLinkType,
                  (LPSTR FAR*)&lpLI->lpszShortLinkType,
                  (BOOL FAR*)&lpLI->fSourceAvailable,
                  fGetSelected ? (BOOL FAR*)&lpLI->fIsSelected : NULL
     );
     OLEDBG_END2

     if (hErr != NOERROR) {
          OleDbgOutHResult("WARNING: IOleUILinkContainer::GetLinkSource
returned",hErr);
          PopupMessage(hListBox, (UINT)IDS_LINKS,
(UINT)IDS_ERR_GETLINKSOURCE,
                       MB_ICONEXCLAMATION | MB_OK);

          goto cleanup;
     }

     OLEDBG_BEGIN2("IOleUILinkContainer::GetLinkUpdateOptions called\r\n");
     hErr=lpOleUILinkCntr->lpVtbl->GetLinkUpdateOptions(
                  lpOleUILinkCntr,
                  lpLI->dwLink,
                  (LPDWORD)&dwUpdateOpt
     );
     OLEDBG_END2


     if (hErr != NOERROR) {
          OleDbgOutHResult("WARNING:
IOleUILinkContainer::GetLinkUpdateOptions returned",hErr);
          PopupMessage(hListBox, (UINT)IDS_LINKS,
(UINT)IDS_ERR_GETLINKUPDATEOPTIONS,
                       MB_ICONEXCLAMATION | MB_OK);

          goto cleanup;
     }
     if (lpLI->fSourceAvailable) {
          if (dwUpdateOpt == OLEUPDATE_ALWAYS) {
               lpLI->fIsAuto = TRUE;
               LoadString(ghInst, (UINT)IDS_LINK_AUTO, lpLI->lpszAMX,
                            (int)OleStdGetSize((LPVOID)lpLI->lpszAMX));
          }
          else {
               lpLI->fIsAuto = FALSE;
               LoadString(ghInst, (UINT)IDS_LINK_MANUAL, lpLI->lpszAMX,
                            (int)OleStdGetSize((LPVOID)lpLI->lpszAMX));
          }
     }
     else
          LoadString(ghInst, (UINT)IDS_LINK_UNKNOWN, lpLI->lpszAMX,
                       (int)OleStdGetSize((LPVOID)lpLI->lpszAMX));

     BreakString(lpLI);

     nIndex = (int)SendMessage(hListBox,
                                          (nIndex < 0 ? LB_ADDSTRING :
LB_INSERTSTRING),
                                          (WPARAM)(nIndex < 0 ? 0 : nIndex),
                                          (LPARAM)(DWORD)lpLI);

     if (nIndex == LB_ERR) {
          PopupMessage(hListBox, (UINT)IDS_LINKS, (UINT)IDS_ERR_ADDSTRING,
                       MB_ICONEXCLAMATION | MB_OK);

          goto cleanup;
     }

     return nIndex;

cleanup:
     if (lpLI->lpszDisplayName)
          OleStdFree((LPVOID)lpLI->lpszDisplayName);

     if (lpLI->lpszShortLinkType)
          OleStdFree((LPVOID)lpLI->lpszShortLinkType);

     if (lpLI->lpszFullLinkType)
          OleStdFree((LPVOID)lpLI->lpszFullLinkType);

     return -1;
}


/* BreakString
 * -----------
 *
 *  Purpose:
 *      Break the lpszDisplayName into various parts
 *
 *  Parameters:
```

```
 *      lpLI            pointer to LINKINFO structure
 *
 *  Returns:
 *
 */
VOID BreakString(LPLINKINFO lpLI)
{
     LPSTR lpsz;

     if (!lpLI->clenFileName ||
           (lstrlen(lpLI->lpszDisplayName)==(int)lpLI->clenFileName)) {

           lpLI->lpszItemName = NULL;
     }
     else {
           lpLI->lpszItemName = lpLI->lpszDisplayName + lpLI->clenFileName;
     }

     // search from last character of filename
     lpsz = lpLI->lpszDisplayName + lstrlen(lpLI->lpszDisplayName);
     while (lpsz > lpLI->lpszDisplayName) {
           lpsz = AnsiPrev(lpLI->lpszDisplayName, lpsz);
           if ((*lpsz == '\\') || (*lpsz == '/') || (*lpsz == ':'))
                 break;
     }

     if (lpsz == lpLI->lpszDisplayName)
           lpLI->lpszShortFileName = lpsz;
     else
           lpLI->lpszShortFileName = AnsiNext(lpsz);
}


/* GetSelectedItems
 * ----------------
 *
 *  Purpose:
 *      Retrieve the indices of the selected items in the listbox
 *      Note that *lprgIndex needed to be free after using the function
 *
 *  Parameters:
 *      hListBox        window handle of listbox
 *      lprgIndex       pointer to an integer array to receive the indices
 *                      must be freed afterwards
 *
 *  Returns:
 *      number of indices retrieved, -1 if error
 */
int GetSelectedItems(HWND hListBox, int FAR* FAR* lprgIndex)
{
     DWORD cSelItems;
     DWORD cCheckItems;

     *lprgIndex = NULL;
     cSelItems = SendMessage(hListBox, LB_GETSELCOUNT, 0, 0L);
     if (cSelItems < 0)      // error
           return (int)cSelItems;

     if (!cSelItems)
           return 0;

     *lprgIndex = (int FAR*)OleStdMalloc((int)cSelItems * sizeof(int));

     cCheckItems = SendMessage(hListBox, LB_GETSELITEMS,
                 (WPARAM) cSelItems, (LPARAM) (int FAR*) *lprgIndex);

     if (cCheckItems == cSelItems)
           return (int)cSelItems;
     else {
           if (*lprgIndex)
                 OleStdFree((LPVOID)*lprgIndex);
           *lprgIndex = NULL;
           return 0;
     }
}


/* InitControls
 * ------------
 *
 *  Purpose:
 *      Initialize the state of the Auto/Manual button, Link source/type
 *      static field, etc in the dialogs according to the selection in the
 *      listbox
 *
 *  Parameters:
 *      hDlg        handle to the dialog window
 */
VOID InitControls(HWND hDlg, LPEDITLINKS lpEL)
{
     int         cSelItems;
     HWND        hListBox;
     int         i;
     int FAR*    rgIndex;
     LPLINKINFO  lpLI;
     LPSTR       lpszType = NULL;
     LPSTR       lpszSource = NULL;
     int         cAuto = 0;
     int         cManual = 0;
     BOOL        bSameType = TRUE;
     BOOL        bSameSource = TRUE;
     BOOL        bNoSource = FALSE;
     char        sz[OLEUI_CCHPATHMAX];
     LPSTR       lpsz;


     hListBox = GetDlgItem(hDlg, ID_EL_LINKSLISTBOX);

     cSelItems = GetSelectedItems(hListBox, &rgIndex);
     if (cSelItems < 0)
           return;

     if (lpEL && !(lpEL->lpOEL->dwFlags & ELF_DISABLECANCELLINK))
           EnableWindow(GetDlgItem(hDlg, ID_EL_CANCELLINK),
(BOOL)cSelItems);
     if (lpEL && !(lpEL->lpOEL->dwFlags & ELF_DISABLEOPENSOURCE))
           EnableWindow(GetDlgItem(hDlg, ID_EL_OPENSOURCE), cSelItems == 1);
     if (lpEL && !(lpEL->lpOEL->dwFlags & ELF_DISABLECHANGESOURCE))
           EnableWindow(GetDlgItem(hDlg, ID_EL_CHANGESOURCE), cSelItems ==
1);
     if (lpEL && !(lpEL->lpOEL->dwFlags & ELF_DISABLEUPDATENOW))
           EnableWindow(GetDlgItem(hDlg, ID_EL_UPDATENOW), (BOOL)cSelItems);

     for (i = 0; i < cSelItems; i++) {
           SendDlgItemMessage(
                       hDlg,
                       ID_EL_LINKSLISTBOX,
                       LB_GETTEXT,
                       (WPARAM)rgIndex[i],
                       (LPARAM)(LPLINKINFO FAR*)&lpLI);

           if (lpszSource && lpLI->lpszDisplayName) {
                 if (bSameSource && lstrcmp(lpszSource, lpLI-
>lpszDisplayName)) {
                       bSameSource = FALSE;
                 }
           }
           else
                 lpszSource = lpLI->lpszDisplayName;

           if (lpszType && lpLI->lpszFullLinkType) {
                 if (bSameType && lstrcmp(lpszType, lpLI->lpszFullLinkType))
{
                       bSameType = FALSE;
                 }
           }
           else
                 lpszType = lpLI->lpszFullLinkType;

           if (lpLI->fIsAuto)
                 cAuto++;
           else
                 cManual++;

           if (!lpLI->fSourceAvailable)
                 bNoSource = TRUE;
     }

     EnableWindow(GetDlgItem(hDlg, ID_EL_AUTOMATIC), (BOOL)cSelItems && !
bNoSource);
     EnableWindow(GetDlgItem(hDlg, ID_EL_MANUAL), (BOOL)cSelItems && !
bNoSource);
     CheckDlgButton(hDlg, ID_EL_AUTOMATIC, !bNoSource && cAuto && !cManual);
     CheckDlgButton(hDlg, ID_EL_MANUAL, bNoSource || (!cAuto && cManual));
     /* fill full source in static text box
     **    below list
     */
     if (!bSameSource || !lpszSource)
           lpszSource = szNULL;
     lstrcpy((LPSTR)sz, lpszSource);
     lpsz = ChopText(GetDlgItem(hDlg, ID_EL_LINKSOURCE), 0, sz);
     SetDlgItemText(hDlg, ID_EL_LINKSOURCE, lpsz);

     /* fill full link type name in static
     **    "type" text box
     */
     if (!bSameType || !lpszType)
           lpszType = szNULL;
     SetDlgItemText(hDlg, ID_EL_LINKTYPE, lpszType);

     if (rgIndex)
           OleStdFree((LPVOID)rgIndex);
}


/* UpdateLinkLBItem
 * -----------------
 *
 *  Purpose:
 *      Update the linkinfo struct in the listbox to reflect the changes
 *      made by the last operation. It is done simply by removing the item
 *      from the listbox and add it back.
 *
 *  Parameters:
 *      hListBox        handle of listbox
 *      nIndex          index of listbox item
 *      lpEL            pointer to editlinks structure
 *      bSelect         select the item or not after update
 */
VOID UpdateLinkLBItem(HWND hListBox, int nIndex, LPEDITLINKS lpEL, BOOL
bSelect)
{
     LPLINKINFO lpLI;
     DWORD       dwErr;
     LPOLEUILINKCONTAINER    lpOleUILinkCntr;

     if (!hListBox || (nIndex < 0) || !lpEL)
           return;

     lpOleUILinkCntr = lpEL->lpOEL->lpOleUILinkContainer;

     dwErr = SendMessage(hListBox, LB_GETTEXT, nIndex,
                 (LPARAM)(LPLINKINFO FAR*) &lpLI);

     if ((dwErr == LB_ERR) || !lpLI)
           return;

     /* Don't free the data associated with this listbox item
     **    because we are going to reuse the allocated space for
     **    the modified link. WM_DELETEITEM processing in the
     **    dialog checks this flag before deleting data
     **    associcated with list item.
     */
     lpLI->fDontFree = TRUE;
     SendMessage(hListBox, LB_DELETESTRING, nIndex, 0L);

     nIndex = AddLinkLBItem(hListBox, lpOleUILinkCntr, lpLI, FALSE, nIndex);
     if (bSelect) {
           SendMessage(hListBox, LB_SETSEL, (WPARAM)TRUE, MAKELPARAM(nIndex,
0));
           SendMessage(hListBox, LB_SETCARETINDEX, (WPARAM)nIndex,
MAKELPARAM(TRUE, 0));
     }
}



/* DiffPrefix
 * ----------
 *
 *  Purpose:
 *      Compare (case-insensitive) two strings and return the prefixes of
the
 *      the strings formed by removing the common suffix string from them.
 *      Integrity of tokens (directory name, filename and object names) are
 *      preserved. Note that the prefixes are converted to upper case
 *      characters.
 *
 *  Parameters:
 *      lpsz1           string 1
 *      lpsz2           string 2
 *      lplpszPrefix1   prefix of string 1
 *      lplpszPrefix2   prefix of string 2
 *
 *  Returns:
 *
 */
VOID DiffPrefix(LPCSTR lpsz1, LPCSTR lpsz2, char FAR* FAR* lplpszPrefix1,
char FAR* FAR* lplpszPrefix2)
{
     LPSTR   lpstr1;
     LPSTR   lpstr2;

     OleDbgAssert(lpsz1 && lpsz2 && *lpsz1 && *lpsz2 && lplpszPrefix1 &&
                 lplpszPrefix2);

     *lplpszPrefix1 = NULL;
     *lplpszPrefix2 = NULL;
     *lplpszPrefix1 = OleStdMalloc((lstrlen(lpsz1)+1) * sizeof(BYTE));
     if (!*lplpszPrefix1)
           return;

     *lplpszPrefix2 = OleStdMalloc((lstrlen(lpsz2)+1) * sizeof(BYTE));
     if (!*lplpszPrefix2) {
           OleStdFree(*lplpszPrefix1);
           *lplpszPrefix1 = NULL;
           return;
     }

     lstrcpy(*lplpszPrefix1, lpsz1);
     lstrcpy(*lplpszPrefix2, lpsz2);
//  AnsiLower(*lplpszPrefix1);
//  AnsiLower(*lplpszPrefix2);

     lpstr1 = *lplpszPrefix1 + lstrlen(*lplpszPrefix1);
     lpstr2 = *lplpszPrefix2 + lstrlen(*lplpszPrefix2);

     while ((lpstr1>*lplpszPrefix1) && (lpstr2>*lplpszPrefix2)) {
           lpstr1 = AnsiPrev(*lplpszPrefix1, lpstr1);
           lpstr2 = AnsiPrev(*lplpszPrefix2, lpstr2);
           if (*lpstr1 != *lpstr2) {
                 lpstr1 = AnsiNext(lpstr1);
                 lpstr2 = AnsiNext(lpstr2);
                 break;
           }
     }

     for (; *lpstr1 && *lpstr1!='\\' && *lpstr1!='!';
lpstr1=AnsiNext(lpstr1));
     for (; *lpstr2 && *lpstr2!='\\' && *lpstr2!='!';
lpstr2=AnsiNext(lpstr2));

     *lpstr1 = '\0';
     *lpstr2 = '\0';
}


/* PopupMessage
 * ------------
 *
 *  Purpose:
 *      Popup s messagebox and get some response from the user. It is the
same
 *      as MessageBox() except that the title and message string are loaded
 *      from the resource file.
 *
 *  Parameters:
 *      hwndParent      parent window of message box
 *      idTitle         id of title string
 *      idMessage       id of message string
 *      fuStyle         style of message box
 */
int PopupMessage(HWND hwndParent, UINT idTitle, UINT idMessage, UINT
fuStyle)
{
     char szTitle[256];
     char szMsg[256];
     LoadString(ghInst, idTitle, (LPSTR)szTitle, sizeof(szTitle));
     LoadString(ghInst, idMessage, (LPSTR)szMsg, sizeof(szMsg));
     return MessageBox(hwndParent, szMsg, szTitle, fuStyle);
}

/* CFindLinks
 * --------------
 *
 *  Purpose:
 *      Count all the links in the listbox starting
 *      with lpsz.
 *
 *  Parameters:
 *      hListBox        window handle of
 *      lpOleUILinkCntr pointer to OleUI Link Container
 *      lpszFrom        prefix for matching
 *      lpszTo          prefix to substitution
 *
 *  Returns: TRUE if we attempted to update links
 */

BOOL CFindLinks(HWND hListBox, LPSTR lpsz)
{
     int     cItems;
     int     nIndex;
     int     cch;
     LPLINKINFO  lpLI;
     LPSTR   szTmp[OLEUI_CCHPATHMAX];
     int     cFound;

     cch = lstrlen(lpsz);

     cItems = (int)SendMessage(hListBox, LB_GETCOUNT, 0, 0L);

     cFound = 0;

     for (nIndex=0; nIndex<cItems; nIndex++) {
           SendMessage(hListBox, LB_GETTEXT, nIndex,
                       (LPARAM)(LPLINKINFO FAR*)&lpLI);

           /* if the corresponding position for the end of lpszFrom in the
           **    display name is not a separator. We stop comparing this
           **    link.
           */
           if (!*(lpLI->lpszDisplayName + cch) ||
                 (*(lpLI->lpszDisplayName + cch) == '\\') ||
                 (*(lpLI->lpszDisplayName + cch) == '!')) {
                 lstrcpyn((LPSTR)szTmp, lpLI->lpszDisplayName, cch + 1);
                 if (!lstrcmp((LPSTR)szTmp, lpsz)) {
                       ++cFound;
                 }
           }
     }
return(cFound);
}
```

```
/* FChangeAllLinks
 * --------------
 *
 *  Purpose:
 *      Enumerate all the links in the listbox and change those starting
 *      with lpszFrom to lpszTo.
 *
 *  Parameters:
 *      hListBox        window handle of
 *      lpOleUILinkCntr pointer to OleUI Link Container
 *      lpszFrom        prefix for matching
 *      lpszTo          prefix to substitution
 *
 *  Returns: TRUE if we attempted to update links
 */
BOOL FChangeAllLinks(HWND hListBox, LPOLEUILINKCONTAINER lpOleUILinkCntr,
LPSTR lpszFrom, LPSTR lpszTo)
{
     int     cItems;
     int     nIndex;
     int     cFrom;
     LPLINKINFO  lpLI;
     LPSTR   szTmp[OLEUI_CCHPATHMAX];
     BOOL    bFound;

     cFrom = lstrlen(lpszFrom);

     cItems = (int)SendMessage(hListBox, LB_GETCOUNT, 0, 0L);
     OleDbgAssert(cItems >= 0);

     bFound = FALSE;

     OleDbgPrint(3, "From : ", lpszFrom, 0);
     OleDbgPrint(3, "", "\r\n", 0);
     OleDbgPrint(3, "To   : ", lpszTo, 0);
     OleDbgPrint(3, "", "\r\n", 0);

     for (nIndex=0; nIndex<cItems; nIndex++) {
           SendMessage(hListBox, LB_GETTEXT, nIndex,
                       (LPARAM)(LPLINKINFO FAR*)&lpLI);

           // unmark the item
           lpLI->fIsMarked = FALSE;

           /* if the corresponding position for the end of lpszFrom in the
           **    display name is not a separator. We stop comparing this
           **    link.
           */
           if (!*(lpLI->lpszDisplayName + cFrom) ||
                 (*(lpLI->lpszDisplayName + cFrom) == '\\') ||
                 (*(lpLI->lpszDisplayName + cFrom) == '!')) {

                 LSTRCPYN((LPSTR)szTmp, lpLI->lpszDisplayName, cFrom + 1);
                 if (!lstrcmp((LPSTR)szTmp, lpszFrom)) {
                       HRESULT hErr;
                       int nFileLength;
                       ULONG ulDummy;

                       if (!bFound) {
                             char szTitle[256];
                             char szMsg[256];
                             char szBuf[256];
                             int uRet;

                             LoadString(ghInst, (UINT)IDS_CHANGESOURCE,
(LPSTR)szTitle,
                                         sizeof(szTitle));
                             LoadString(ghInst,
(UINT)IDS_CHANGEADDITIONALLINKS,
                                         (LPSTR)szMsg, sizeof(szMsg));
                             wsprintf((LPSTR)szBuf, (LPSTR)szMsg, lpszFrom);
                             uRet = MessageBox(hListBox, (LPSTR)szBuf,
(LPSTR)szTitle,
                                         MB_ICONQUESTION | MB_YESNO);
                             if (uRet == IDYES)
                                   bFound = TRUE;
                             else
                                   return(FALSE);         // exit function
                       }

                       lstrcpy((LPSTR)szTmp, lpszTo);
                       lstrcat((LPSTR)szTmp, lpLI->lpszDisplayName + cFrom);
                       nFileLength = lstrlen((LPSTR)szTmp) -
                             (lpLI->lpszItemName ? lstrlen(lpLI-
>lpszItemName) : 0);


                       hErr = lpOleUILinkCntr->lpVtbl->SetLinkSource(
                                   lpOleUILinkCntr,
                                   lpLI->dwLink,
                                   (LPSTR)szTmp,
                                   (ULONG)nFileLength,
                                   (ULONG FAR*)&ulDummy,
                                   TRUE
                       );
                       if (hErr != NOERROR)
                             lpOleUILinkCntr->lpVtbl->SetLinkSource(
                                         lpOleUILinkCntr,
                                         lpLI->dwLink,
                                         (LPSTR)szTmp,
                                         (ULONG)nFileLength,
                                         (ULONG FAR*)&ulDummy,
                                         FALSE
                             );
                       lpLI->fIsMarked = TRUE;
                 }
           }
     }
     /* have to do the refreshing after processing all links, otherwise
     **    the item positions will change during the process as the
     **    listbox stores items in order
     */
     if (bFound)
          RefreshLinkLB(hListBox, lpOleUILinkCntr);

     return(TRUE);
}



/* LoadLinkLB
 * ----------
 *
 *  Purpose:
 *      Enumerate all links from the Link Container and build up the Link
 *      ListBox
 *
 *  Parameters:
 *      hListBox        window handle of
 *      lpOleUILinkCntr pointer to OleUI Link Container
 *      lpszFrom        prefix for matching
 *      lpszTo          prefix to substitution
 *
 *  Returns:
 *      number of link items loaded, -1 if error
 */
int LoadLinkLB(HWND hListBox, LPOLEUILINKCONTAINER lpOleUILinkCntr)
{
     DWORD       dwLink = 0;
     LPLINKINFO  lpLI;
     int         nIndex;
     int         cLinks;

     cLinks = 0;

     while ((dwLink = lpOleUILinkCntr->lpVtbl->GetNextLink(lpOleUILinkCntr,
                 dwLink)) != 0) {
          lpLI = (LPLINKINFO)OleStdMalloc(sizeof(LINKINFO));
          if (NULL == lpLI)
                 return -1;

          lpLI->fIsMarked = FALSE;
          lpLI->fIsSelected = FALSE;
          lpLI->fDontFree = FALSE;
          lpLI->lpszAMX =
(LPSTR)OleStdMalloc((LINKTYPELEN+1)*sizeof(BYTE));

          lpLI->dwLink = dwLink;
          cLinks++;
          if ((nIndex =
AddLinkLBItem(hListBox,lpOleUILinkCntr,lpLI,TRUE,-1)) < 0)
                 // can't load list box
```

```
                  return -1;

            if (lpLI->fIsSelected) {
                  SendMessage(hListBox, LB_SETSEL, TRUE, MAKELPARAM(nIndex,
0));
            }
      }
      if (SendMessage(hListBox,LB_GETSELITEMS,(WPARAM)1,(LPARAM)(int
FAR*)&nIndex))
            SendMessage(hListBox, LB_SETCARETINDEX, (WPARAM)nIndex,
MAKELPARAM(TRUE, 0));

      return cLinks;
}


/* RefreshLinkLB
 * -------------
 *
 *  Purpose:
 *      Enumerate all items in the links listbox and update those with
 *      fIsMarked set.
 *      Note that this is a time consuming routine as it keeps iterating
 *      all items in the listbox until all of them are unmarked.
 *
 *  Parameters:
 *      hListBox        window handle of listbox
 *      lpOleUILinkCntr pointer to OleUI Link Container
 *
 *  Returns:
 *
 */
VOID RefreshLinkLB(HWND hListBox, LPOLEUILINKCONTAINER lpOleUILinkCntr)
{
      int cItems;
      int nIndex;
      LPLINKINFO  lpLI;
      BOOL        bStop;

      OleDbgAssert(hListBox);

      // Turn painting updates off.
      SendMessage(hListBox, WM_SETREDRAW, FALSE, 0L);

      cItems = (int)SendMessage(hListBox, LB_GETCOUNT, 0, 0L);
      OleDbgAssert(cItems >= 0);

      do {
            LRESULT nIsSelected;
            bStop = TRUE;
            for (nIndex=0; nIndex<cItems; nIndex++) {
                  SendMessage(hListBox, LB_GETTEXT, (WPARAM)nIndex,
                              (LPARAM)(LPLINKINFO FAR*)&lpLI);
                  nIsSelected = SendMessage(hListBox, LB_GETSEL,
(WPARAM)nIndex,0L);
```

```
                 if (lpLI->fIsMarked) {
                       lpLI->fIsMarked = FALSE;
                       lpLI->fDontFree = TRUE;

                       SendMessage(hListBox, LB_DELETESTRING, nIndex, 0L);
                       nIndex=AddLinkLBItem(hListBox, lpOleUILinkCntr, lpLI,
FALSE, nIndex);
                       if (nIsSelected > 0) {
                             lpLI->fIsSelected = TRUE;
                             SendMessage(hListBox, LB_SETSEL, (WPARAM)TRUE,
                                         MAKELPARAM(nIndex, 0));
                             SendMessage(hListBox, LB_SETCARETINDEX,
(WPARAM)nIndex,
                                         MAKELPARAM(TRUE, 0));
                       } else {
                             lpLI->fIsSelected = FALSE;
                       }
                       bStop = FALSE;
                       break;
                 }
           }
     } while (!bStop);

     // Turn updates back on.
     SendMessage(hListBox, WM_SETREDRAW, TRUE, 0L);
}
```

MSGFILTR.H   (WRAPUI Sample)

```
/*************************************************************************
**
**    OLE 2 Utility Code
**
**    msgfiltr.h
**
**    This file contains Private definitions, structures, types, and
**    function prototypes for the OleStdMessageFilter implementation of
**    the IMessageFilter interface.
**    This file is part of the OLE 2.0 User Interface support library.
**
**    (c) Copyright Microsoft Corp. 1990 - 1992 All Rights Reserved
**
*************************************************************************/

#if !defined( _MSGFILTR_H_ )
#define _MSGFILTR_H_

#ifndef RC_INVOKED
#pragma message ("INCLUDING MSGFILTR.H from " __FILE__)
#endif  /* RC_INVOKED */

// Message Pending callback procedure
typedef BOOL (CALLBACK* MSGPENDINGPROC)(MSG FAR *);

// HandleInComingCall callback procedure
typedef DWORD (CALLBACK* HANDLEINCOMINGCALLBACKPROC)
     (
           DWORD               dwCallType,
           HTASK               htaskCaller,
           DWORD               dwTickCount,
           LPINTERFACEINFO     lpInterfaceInfo
     );

/* PUBLIC FUNCTIONS */
STDAPI_(LPMESSAGEFILTER) OleStdMsgFilter_Create(
           HWND hWndParent,
           LPSTR szAppName,
           MSGPENDINGPROC lpfnCallback,
           LPFNOLEUIHOOK  lpfnOleUIHook         // Busy dialog hook callback
);

STDAPI_(void) OleStdMsgFilter_SetInComingCallStatus(
           LPMESSAGEFILTER lpThis, DWORD dwInComingCallStatus);

STDAPI_(DWORD) OleStdMsgFilter_GetInComingCallStatus(
           LPMESSAGEFILTER lpThis);

STDAPI_(HANDLEINCOMINGCALLBACKPROC)
     OleStdMsgFilter_SetHandleInComingCallbackProc(
           LPMESSAGEFILTER             lpThis,
           HANDLEINCOMINGCALLBACKPROC  lpfnHandleInComingCallback);
```

```
STDAPI_(BOOL) OleStdMsgFilter_EnableBusyDialog(
            LPMESSAGEFILTER lpThis, BOOL fEnable);

STDAPI_(BOOL) OleStdMsgFilter_EnableNotRespondingDialog(
            LPMESSAGEFILTER lpThis, BOOL fEnable);

STDAPI_(HWND) OleStdMsgFilter_SetParentWindow(
            LPMESSAGEFILTER lpThis, HWND hWndParent);


#endif // _MSGFILTR_H_
```

MSGFILTR.C   (WRAPUI Sample)

```
/*
 *    MSGFILTR.C
 *
 *    This file contains a standard implementation of IMessageFilter
 *    interface.
 *    This file is part of the OLE 2.0 User Interface support library.
 *
 *    (c) Copyright Microsoft Corp. 1990 - 1992 All Rights Reserved
 *
 */


#define STRICT  1
#include "ole2ui.h"
#include "msgfiltr.h"

OLEDBGDATA


typedef struct tagOLESTDMESSAGEFILTER {
     IMessageFilterVtbl FAR* m_lpVtbl;
     UINT                    m_cRef;
     HWND                    m_hWndParent;
     DWORD                   m_dwInComingCallStatus; // Status to return
from

     // HandleIncomingCall
     HANDLEINCOMINGCALLBACKPROC m_lpfnHandleInComingCallback;

     // Callback function

     // to selectively handle

     // interface method calls
     BOOL                    m_fEnableBusyDialog;    // enable RetryRejected

     //  Call dialog
     BOOL                    m_fEnableNotRespondingDialog; // enable

     // MessagePending dialog
     MSGPENDINGPROC          m_lpfnMessagePendingCallback; // MessagePending

     // Callback function
     LPFNOLEUIHOOK           m_lpfnBusyDialogHookCallback; // Busy dialog
hook
     LPSTR                   m_lpszAppName;          // Name of application

     // installing filter
     HWND                    m_hWndBusyDialog;       // HWND of busy dialog.
Used

     // to tear down dialog.
     BOOL                    m_bUnblocking;

 }OLESTDMESSAGEFILTER, FAR* LPOLESTDMESSAGEFILTER;

/* interface IMessageFilter implementation */
STDMETHODIMP OleStdMsgFilter_QueryInterface(
           LPMESSAGEFILTER lpThis, REFIID riid, LPVOID FAR* ppvObj);
STDMETHODIMP_(ULONG) OleStdMsgFilter_AddRef(LPMESSAGEFILTER lpThis);
STDMETHODIMP_(ULONG) OleStdMsgFilter_Release(LPMESSAGEFILTER lpThis);
STDMETHODIMP_(DWORD) OleStdMsgFilter_HandleInComingCall (
           LPMESSAGEFILTER     lpThis,
           DWORD               dwCallType,
           HTASK               htaskCaller,
           DWORD               dwTickCount,
           LPINTERFACEINFO     dwReserved
);
STDMETHODIMP_(DWORD) OleStdMsgFilter_RetryRejectedCall (
           LPMESSAGEFILTER     lpThis,
           HTASK               htaskCallee,
           DWORD               dwTickCount,
           DWORD               dwRejectType
);
STDMETHODIMP_(DWORD) OleStdMsgFilter_MessagePending (
           LPMESSAGEFILTER     lpThis,
           HTASK               htaskCallee,
           DWORD               dwTickCount,
           DWORD               dwPendingType
);


static IMessageFilterVtbl g_OleStdMessageFilterVtbl = {
     OleStdMsgFilter_QueryInterface,
     OleStdMsgFilter_AddRef,
     OleStdMsgFilter_Release,
     OleStdMsgFilter_HandleInComingCall,
     OleStdMsgFilter_RetryRejectedCall,
     OleStdMsgFilter_MessagePending
};


/* GetTopWindowInWindowsTask
** -------------------------
**    Get the top most window that has focus in the given task to be
**    used as the parent for the busy dialog. we do this to handle the
**    case where a dialog window is currently up when we need to give
**    the busy dialog. if we use the current assigned parent window
**    (which typically will be the frame window of the app), then the
**    busy dialog will not be modal to the current active dialog
**    window.
*/
static HWND GetTopWindowInWindowsTask(HWND hwnd)
{
     HWND hwndActive = GetActiveWindow();
     if (!hwndActive)
            return hwnd;
     if (GetWindowTask(hwnd) == GetWindowTask(hwndActive))
           return hwndActive;
     else
           return hwnd;
}

STDAPI_(LPMESSAGEFILTER) OleStdMsgFilter_Create(
           HWND            hWndParent,
           LPSTR           szAppName,
           MSGPENDINGPROC  lpfnCallback,
           LPFNOLEUIHOOK   lpfnOleUIHook         // Busy dialog hook
callback
)
{
     LPOLESTDMESSAGEFILTER lpStdMsgFilter;
     LPMALLOC lpMalloc;

     if (CoGetMalloc(MEMCTX_TASK, (LPMALLOC FAR*)&lpMalloc) != NOERROR)
           return NULL;

     lpStdMsgFilter = (LPOLESTDMESSAGEFILTER)lpMalloc->lpVtbl->Alloc(
                 lpMalloc, (sizeof(OLESTDMESSAGEFILTER)));
     lpMalloc->lpVtbl->Release(lpMalloc);
     if (! lpStdMsgFilter) return NULL;

     lpStdMsgFilter->m_lpVtbl = &g_OleStdMessageFilterVtbl;
     lpStdMsgFilter->m_cRef = 1;
     lpStdMsgFilter->m_hWndParent = hWndParent;
     lpStdMsgFilter->m_dwInComingCallStatus = SERVERCALL_ISHANDLED;
     lpStdMsgFilter->m_lpfnHandleInComingCallback = NULL;
     lpStdMsgFilter->m_fEnableBusyDialog = TRUE;
     lpStdMsgFilter->m_fEnableNotRespondingDialog = TRUE;
     lpStdMsgFilter->m_lpszAppName = szAppName;
     lpStdMsgFilter->m_lpfnMessagePendingCallback = lpfnCallback;
     lpStdMsgFilter->m_lpfnBusyDialogHookCallback = lpfnOleUIHook;
     lpStdMsgFilter->m_hWndBusyDialog = NULL;
     lpStdMsgFilter->m_bUnblocking = FALSE;

    g_dwObjectCount ++;

     return (LPMESSAGEFILTER)lpStdMsgFilter;
}


/* OleStdMsgFilter_SetInComingStatus
** ---------------------------------
**    This is a private function that allows the caller to control what
**    value is returned from the IMessageFilter::HandleInComing method.
**
**    if a HandleInComingCallbackProc is installed by a call to
**    OleStdMsgFilter_SetHandleInComingCallbackProc, then this
**    overrides the dwIncomingCallStatus established by a call to
**    OleStdMsgFilter_SetInComingStatus.  Using
**    OleStdMsgFilter_SetInComingStatus allows the app to reject or
**    accept ALL in coming calls. Using a HandleInComingCallbackProc
**    allows the app to selectively handle or reject particular method
**    calls.
*/

STDAPI_(void) OleStdMsgFilter_SetInComingCallStatus(
           LPMESSAGEFILTER lpThis, DWORD dwInComingCallStatus)
{
     LPOLESTDMESSAGEFILTER lpStdMsgFilter = (LPOLESTDMESSAGEFILTER)lpThis;

     if (!IsBadWritePtr((LPVOID)lpStdMsgFilter,
sizeof(OLESTDMESSAGEFILTER)))
           lpStdMsgFilter->m_dwInComingCallStatus = dwInComingCallStatus;
     else
           OleDbgAssert(
                 "OleStdMsgFilter_SetIncomingCallStatus: Invalid
IMessageFilter*");

#if defined( _DEBUG )
     {
     char szBuf[80];
     char *szReturn;

     switch(dwInComingCallStatus) {
           case SERVERCALL_ISHANDLED:
                 szReturn = "SERVERCALL_ISHANDLED";
                 break;
           case SERVERCALL_REJECTED:
                 szReturn = "SERVERCALL_REJECTED";
                 break;
           case SERVERCALL_RETRYLATER:
                 szReturn = "SERVERCALL_RETRYLATER";
                 break;
           default:
                 szReturn = "** ERROR: UNKNOWN **";
                 break;
           }
     wsprintf(
           szBuf,
           "OleStdMsgFilter_SetInComingCallStatus: Status set to %s.\r\n",
           (LPSTR)szReturn
     );
     OleDbgOut3(szBuf);
     }
#endif

}


/* OleStdMsgFilter_SetHandleInComingCallbackProc
** ----------------------------------------------
**    This is a private function that allows the caller to install (or
**    de-install) a special callback function to selectively
**    handle/reject specific incoming method calls on particular
**    interfaces.
```

```
**
**    if a HandleInComingCallbackProc is installed by a call to
**    OleStdMsgFilter_SetHandleInComingCallbackProc, then this
**    overrides the dwIncomingCallStatus established by a call to
**    OleStdMsgFilter_SetInComingStatus.  Using
**    OleStdMsgFilter_SetInComingStatus allows the app to reject or
**    accept ALL in coming calls. Using a HandleInComingCallbackProc
**    allows the app to selectively handle or reject particular method
**    calls.
**
**    to de-install the HandleInComingCallbackProc, call
**          OleStdMsgFilter_SetHandleInComingCallbackProc(NULL);
**
**    Returns previous callback proc in effect.
*/

STDAPI_(HANDLEINCOMINGCALLBACKPROC)
     OleStdMsgFilter_SetHandleInComingCallbackProc(
           LPMESSAGEFILTER             lpThis,
           HANDLEINCOMINGCALLBACKPROC  lpfnHandleInComingCallback)
{
     LPOLESTDMESSAGEFILTER lpStdMsgFilter = (LPOLESTDMESSAGEFILTER)lpThis;
     HANDLEINCOMINGCALLBACKPROC    lpfnPrevCallback =
                 lpStdMsgFilter->m_lpfnHandleInComingCallback;

     if (!IsBadWritePtr((LPVOID)lpStdMsgFilter,
sizeof(OLESTDMESSAGEFILTER))) {
           lpStdMsgFilter->m_lpfnHandleInComingCallback =
                       lpfnHandleInComingCallback;
     } else {
           OleDbgAssert(
                 "OleStdMsgFilter_SetIncomingCallStatus: Invalid
IMessageFilter*");
     }

#if defined( _DEBUG )
     {
           if (lpfnHandleInComingCallback)
                 OleDbgOut3(
                      "OleStdMsgFilter_SetHandleInComingCallbackProc
SET\r\n");
           else
                 OleDbgOut3(
                      "OleStdMsgFilter_SetHandleInComingCallbackProc
CLEARED\r\n");

     }
#endif  // _DEBUG

     return lpfnPrevCallback;
}


/* OleStdMsgFilter_GetInComingStatus
** ---------------------------------
**    This is a private function that returns the current
**    incoming call status.  Can be used to disable/enable options
**    in the calling application.
**
** Returns: one of
**
**    SERVERCALL_ISHANDLED
**    SERVERCALL_REJECTED
**    SERVERCALL_RETRYLATER
**    or -1 for ERROR
**
*/

STDAPI_(DWORD) OleStdMsgFilter_GetInComingCallStatus(
           LPMESSAGEFILTER lpThis)
{
     LPOLESTDMESSAGEFILTER lpStdMsgFilter = (LPOLESTDMESSAGEFILTER)lpThis;
     DWORD dwReturn;

     if (!IsBadReadPtr((LPVOID)lpStdMsgFilter,
sizeof(OLESTDMESSAGEFILTER)))
           dwReturn = lpStdMsgFilter->m_dwInComingCallStatus;
     else
           {
           OleDbgAssert(
                 "OleStdMsgFilter_GetIncomingCallStatus: Invalid
IMessageFilter*");
           dwReturn = (DWORD)-1;
           }

#if defined( _DEBUG )
     {
     char szBuf[80];
     char *szReturn;

     switch(dwReturn) {
           case SERVERCALL_ISHANDLED:
                 szReturn = "SERVERCALL_ISHANDLED";
                 break;
           case SERVERCALL_REJECTED:
                 szReturn = "SERVERCALL_REJECTED";
                 break;
           case SERVERCALL_RETRYLATER:
                 szReturn = "SERVERCALL_RETRYLATER";
                 break;
           default:
                 szReturn = "-1";
                 break;
           }
     wsprintf(
           szBuf,
           "OleStdMsgFilter_GetInComingCallStatus returns %s.\r\n",
           (LPSTR)szReturn
     );
     OleDbgOut3(szBuf);
     }
#endif

     return dwReturn;
}


/* OleStdMsgFilter_EnableBusyDialog
** --------------------------------
**    This function allows the caller to control whether
**    the busy dialog is enabled. this is the dialog put up when
**    IMessageFilter::RetryRejectedCall is called because the server
**    responded SERVERCALL_RETRYLATER or SERVERCALL_REJECTED.
**
**    if the busy dialog is NOT enabled, then the rejected call is
**    immediately canceled WITHOUT prompting the user. in this situation
**    OleStdMsgFilter_RetryRejectedCall always retuns
**    OLESTDCANCELRETRY canceling the outgoing LRPC call.
**    If the busy dialog is enabled, then the user is given the choice
**    of whether to retry, switch to, or cancel.
**
**    Returns previous dialog enable state
*/

STDAPI_(BOOL) OleStdMsgFilter_EnableBusyDialog(
           LPMESSAGEFILTER lpThis, BOOL fEnable)
{
     LPOLESTDMESSAGEFILTER lpStdMsgFilter = (LPOLESTDMESSAGEFILTER)lpThis;
     BOOL fPrevEnable = lpStdMsgFilter->m_fEnableBusyDialog;

     if (!IsBadWritePtr((LPVOID)lpStdMsgFilter,
sizeof(OLESTDMESSAGEFILTER)))
           lpStdMsgFilter->m_fEnableBusyDialog = fEnable;
     else
           OleDbgAssert(
                       "OleStdMsgFilter_EnableBusyDialog: Invalid
IMessageFilter*");

#if defined( _DEBUG )
     {
     char szBuf[80];
     wsprintf(
           szBuf,
           "OleStdMsgFilter_EnableBusyDialog: Dialog is %s.\r\n",
           fEnable ? (LPSTR)"ENABLED" : (LPSTR)"DISABLED"
     );
     OleDbgOut3(szBuf);
     }
#endif

     return fPrevEnable;
}


/* OleStdMsgFilter_EnableNotRespondingDialog
```

```
** -----------------------------------------
**    This function allows the caller to control whether
**    the app "NotResponding" (Blocked) dialog is enabled. this is the
**    dialog put up when IMessageFilter::MessagePending is called.
**    If the NotResponding dialog is enabled, then the user is given
**    the choice of whether to retry or switch to, but NOT to cancel.
**
**    Returns previous dialog enable state
*/

STDAPI_(BOOL) OleStdMsgFilter_EnableNotRespondingDialog(
            LPMESSAGEFILTER lpThis, BOOL fEnable)
{
     LPOLESTDMESSAGEFILTER lpStdMsgFilter = (LPOLESTDMESSAGEFILTER)lpThis;
     BOOL fPrevEnable = lpStdMsgFilter->m_fEnableNotRespondingDialog;

     if (!IsBadWritePtr((LPVOID)lpStdMsgFilter,
sizeof(OLESTDMESSAGEFILTER)))
            lpStdMsgFilter->m_fEnableNotRespondingDialog = fEnable;
     else
            OleDbgAssert(
                        "OleStdMsgFilter_EnableNotRespondingDialog: Invalid
IMessageFilter*");

#if defined( _DEBUG )
     {
     char szBuf[80];
     wsprintf(
            szBuf,
            "OleStdMsgFilter_EnableNotRespondingDialog: Dialog is %s.\r\n",
            fEnable ? (LPSTR)"ENABLED" : (LPSTR)"DISABLED"
     );
     OleDbgOut3(szBuf);
     }
#endif

     return fPrevEnable;
}


/* OleStdMsgFilter_SetParentWindow
** -------------------------------
**    This function allows caller to set which window will be used as
**    the parent for the busy dialog.
**
**    OLE2NOTE: it would be inportant for an in-place active server to
**    reset this to its current in-place frame window when in-place
**    activated. if the hWndParent is set to NULL then the dialogs will
**    be parented to the desktop.
**
**    Returns: previous parent window
*/

STDAPI_(HWND) OleStdMsgFilter_SetParentWindow(
            LPMESSAGEFILTER lpThis, HWND hWndParent)
```

```
{
     LPOLESTDMESSAGEFILTER lpStdMsgFilter = (LPOLESTDMESSAGEFILTER)lpThis;
     HWND hWndPrev = lpStdMsgFilter->m_hWndParent;

     lpStdMsgFilter->m_hWndParent = hWndParent;
     return hWndPrev;
}


STDMETHODIMP OleStdMsgFilter_QueryInterface(
           LPMESSAGEFILTER lpThis, REFIID riid, LPVOID FAR* ppvObj)
{
     LPOLESTDMESSAGEFILTER lpStdMsgFilter = (LPOLESTDMESSAGEFILTER)lpThis;
     SCODE scode;

     /* Two interfaces supported: IUnknown, IMessageFilter
     */

     if (IsEqualIID(riid, &IID_IMessageFilter) || IsEqualIID(riid,
&IID_IUnknown)) {
           lpStdMsgFilter->m_cRef++;   // A pointer to this object is
returned
           *ppvObj = lpThis;
           scode = S_OK;
     }
     else {                 // unsupported interface
           *ppvObj = NULL;
           scode = E_NOINTERFACE;
     }

     return ResultFromScode(scode);
}


STDMETHODIMP_(ULONG) OleStdMsgFilter_AddRef(LPMESSAGEFILTER lpThis)
{
     LPOLESTDMESSAGEFILTER lpStdMsgFilter = (LPOLESTDMESSAGEFILTER)lpThis;
     return ++lpStdMsgFilter->m_cRef;
}

STDMETHODIMP_(ULONG) OleStdMsgFilter_Release(LPMESSAGEFILTER lpThis)
{
     LPOLESTDMESSAGEFILTER lpStdMsgFilter = (LPOLESTDMESSAGEFILTER)lpThis;
     LPMALLOC lpMalloc;

     if (--lpStdMsgFilter->m_cRef != 0) // Still used by others
           return lpStdMsgFilter->m_cRef;

     // Free storage
     if (CoGetMalloc(MEMCTX_TASK, (LPMALLOC FAR*)&lpMalloc) != NOERROR)
           return (ULONG)0;

     lpMalloc->lpVtbl->Free(lpMalloc, lpStdMsgFilter);
     lpMalloc->lpVtbl->Release(lpMalloc);
```

```
    g_dwObjectCount --;

     return (ULONG)0;
}


STDMETHODIMP_(DWORD) OleStdMsgFilter_HandleInComingCall (
           LPMESSAGEFILTER     lpThis,
           DWORD               dwCallType,
           HTASK               htaskCaller,
           DWORD               dwTickCount,
           LPINTERFACEINFO     dwReserved
)
{
     LPOLESTDMESSAGEFILTER lpStdMsgFilter = (LPOLESTDMESSAGEFILTER)lpThis;

     /* if a HandleInComingCallbackProc is in effect, then this
     **    overrides dwIncomingCallStatus established by a call to
     **    OleStdMsgFilter_SetInComingStatus.  we will call this
     **    callback to allow the app to selectively handle or reject
     **    incoming method calls. the LPINTERFACEINFO parameter
     **    describes which method is being called.
     */
     if (lpStdMsgFilter->m_lpfnHandleInComingCallback &&
           !IsBadCodePtr((FARPROC)lpStdMsgFilter-
>m_lpfnHandleInComingCallback)){
           return lpStdMsgFilter->m_lpfnHandleInComingCallback(
                       dwCallType,
                       htaskCaller,
                       dwTickCount,
                       (LPINTERFACEINFO)dwReserved
           );
     }

     switch (dwCallType) {
           case CALLTYPE_TOPLEVEL:
                 /* OLE2NOTE: we currently have NO pending outgoing call and
                 **    there is a new toplevel incoming call.
                 **    this call may be rejected.
                 */
                 return lpStdMsgFilter->m_dwInComingCallStatus;

           case CALLTYPE_TOPLEVEL_CALLPENDING:
                 /* OLE2NOTE: we currently HAVE a pending outgoing call and
                 **    there is a new toplevel incoming call.
                 **    this call may be rejected.
                 */
                 return lpStdMsgFilter->m_dwInComingCallStatus;

           case CALLTYPE_NESTED:
                 /* OLE2NOTE: we currently HAVE a pending outgoing call and
                 **    there callback on behalf of the previous outgoing
                 **    call. this type of call should ALWAYS be handled.
                 */
                 return SERVERCALL_ISHANDLED;
           case CALLTYPE_ASYNC:
                 /* OLE2NOTE: we currently have NO pending outgoing call and
                 **    there is a new asyncronis incoming call.
                 **    this call can NEVER be rejected. OLE actually ignores
                 **    the return code in this case and always allows the
                 **    call through.
                 */
                 return SERVERCALL_ISHANDLED;    // value returned does not
matter

           case CALLTYPE_ASYNC_CALLPENDING:
                 /* OLE2NOTE: we currently HAVE a pending outgoing call and
                 **    there is a new asyncronis incoming call.
                 **    this call can NEVER be rejected. OLE ignore the
                 **    return code in this case.
                 */
                 return SERVERCALL_ISHANDLED;    // value returned does not

           default:
                 OleDbgAssert(
                             "OleStdMsgFilter_HandleInComingCall: Invalid
CALLTYPE");
                 return lpStdMsgFilter->m_dwInComingCallStatus;
     }
}

STDMETHODIMP_(DWORD) OleStdMsgFilter_RetryRejectedCall (
           LPMESSAGEFILTER     lpThis,
           HTASK               htaskCallee,
           DWORD               dwTickCount,
           DWORD               dwRejectType
)
{
     LPOLESTDMESSAGEFILTER   lpStdMsgFilter = (LPOLESTDMESSAGEFILTER)lpThis;
     DWORD                   dwRet = 0;
     UINT                    uRet;
#if defined( _DEBUG )
     char                    szBuf[80];
#endif
     OLEDBG_BEGIN2("OleStdMsgFilter_RetryRejectedCall\r\n")

     /* OLE2NOTE: we should only put up the application busy dialog when
     **    the callee has responded SERVERCALL_RETRYLATER. if the
     **    dwRejectType is SERVERCALL_REJECTED then there is something
     **    seriously wrong with the callee (perhaps a severe low memory
     **    situation). we don't want to even try to "Switch To" this app
     **    or even try to "Retry".
     */
     if (dwRejectType == SERVERCALL_RETRYLATER &&
                 lpStdMsgFilter->m_fEnableBusyDialog) {

           OLEUIBUSY bz;

           /* OLE2NOTE: we do not want to put up the Busy dialog immediately
           **    the when an app says RETRYLATER. we should continue
retrying
           **    for a while in case the app can un-busy itself in a
           **    reasonable amount of time.
           */
           if (dwTickCount <= (DWORD)OLESTDRETRYDELAY) {
                 dwRet = 500;                  // Retry after .5 sec
                 OLEDBG_END2
                 return dwRet;
           }

           /*
           ** Set up structure for calling OLEUIBUSY dialog
           */

           bz.cbStruct = sizeof(OLEUIBUSY);
           bz.dwFlags = 0L;
           bz.hWndOwner =GetTopWindowInWindowsTask(lpStdMsgFilter-
>m_hWndParent);
           bz.lpszCaption = lpStdMsgFilter->m_lpszAppName;
           bz.lpfnHook = lpStdMsgFilter->m_lpfnBusyDialogHookCallback;
           bz.lCustData = 0;
           bz.hInstance = NULL;
           bz.lpszTemplate = NULL;
           bz.hResource = 0;
           bz.hTask = htaskCallee;
           bz.lphWndDialog = NULL; // We don't need the hDlg for this call

           uRet = OleUIBusy(&bz);

           switch (uRet) {
                 case OLEUI_BZ_RETRYSELECTED:
                       dwRet = 0;                  // Retry immediately
                       break;

                 case OLEUI_CANCEL:
                       dwRet = OLESTDCANCELRETRY;  // Cancel pending
outgoing call
                       break;

                 case OLEUI_BZERR_HTASKINVALID:
                       // Htask was invalid, return OLESTDRETRYDELAY anyway
                       dwRet = OLESTDRETRYDELAY;   // Retry after <retry
delay> msec

#if defined( _DEBUG )
                       wsprintf(
                                   szBuf,
                                   "OleStdMsgFilter_RetryRejectedCall, HTASK
0x%x invalid\r\n",
                                   htaskCallee
                       );
                       OleDbgOut3(szBuf);
#endif
                       break;
```

```
            }
     } else {
            dwRet = OLESTDCANCELRETRY;  // Cancel pending outgoing call
     }

#if defined( _DEBUG )
     wsprintf(szBuf,
                  "OleStdMsgFilter_RetryRejectedCall returns %d\r\n",
                  dwRet);
     OleDbgOut3(szBuf);
#endif

     OLEDBG_END2
     return dwRet;
}



/* a significant message is consider a mouse click or keyboard input. */
#define IS_SIGNIFICANT_MSG(lpmsg)   \
     (   \
           (PeekMessage((lpmsg), NULL, WM_LBUTTONDOWN, WM_LBUTTONDOWN, \
                        PM_NOREMOVE | PM_NOYIELD)) \
      || (PeekMessage((lpmsg), NULL, WM_LBUTTONDBLCLK, WM_LBUTTONDBLCLK, \
                        PM_NOREMOVE | PM_NOYIELD)) \
      || (PeekMessage((lpmsg), NULL, WM_NCLBUTTONDOWN, WM_NCLBUTTONDOWN, \
                        PM_NOREMOVE | PM_NOYIELD)) \
      || (PeekMessage((lpmsg), NULL, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDBLCLK,
\
                        PM_NOREMOVE | PM_NOYIELD)) \
      || (PeekMessage((lpmsg), NULL, WM_KEYDOWN, WM_KEYDOWN, \
                        PM_NOREMOVE | PM_NOYIELD)) \
      || (PeekMessage((lpmsg), NULL, WM_SYSKEYDOWN, WM_SYSKEYDOWN, \
                        PM_NOREMOVE | PM_NOYIELD)) \
     )

STDMETHODIMP_(DWORD) OleStdMsgFilter_MessagePending (
           LPMESSAGEFILTER     lpThis,
           HTASK               htaskCallee,
           DWORD               dwTickCount,
           DWORD               dwPendingType
)
{
     LPOLESTDMESSAGEFILTER   lpStdMsgFilter = (LPOLESTDMESSAGEFILTER)lpThis;
     DWORD                   dwReturn = PENDINGMSG_WAITDEFPROCESS;
     MSG                     msg;
     BOOL                    fIsSignificantMsg = IS_SIGNIFICANT_MSG(&msg);
     UINT                    uRet;

#if defined( _DEBUG )
     char szBuf[128];
     wsprintf(
                 szBuf,
                 "OleStdMsgFilter_MessagePending, dwTickCount = 0x%lX\r\n",
                 (DWORD)dwTickCount
     );
     OleDbgOut4(szBuf);
#endif

     /* OLE2NOTE: If our tick count for this call exceeds our standard retry
     **      delay, then we need to put up the dialog.  We will only
     **      consider putting up the dialog if the user has issued a
     **      "significant" event (ie. mouse click or keyboard event). a
     **      simple mouse move should NOT trigger this dialog.
     **      Since our call to
     **      OleUIBusy below enters a DialogBox() message loop, there's a
     **      possibility that another call will be initiated during the
dialog,
     **      and this procedure will be re-entered.  Just so we don't put up
     **      two dialogs at a time, we use the m_bUnblocking varable
     **      to keep track of this situation.
     */

     if (dwTickCount > (DWORD)OLESTDRETRYDELAY && fIsSignificantMsg
                 && !lpStdMsgFilter->m_bUnblocking)
     {

           if (lpStdMsgFilter->m_fEnableNotRespondingDialog)
           {
           OLEUIBUSY bz;

           lpStdMsgFilter->m_bUnblocking = TRUE;

           // Eat messages in our queue that we do NOT want to be dispatched
           while (PeekMessage(&msg, NULL, WM_CLOSE, WM_CLOSE, PM_REMOVE |
PM_NOYIELD));

           /* Set up structure for calling OLEUIBUSY dialog,
           ** using the "not responding" variety
           */

           bz.cbStruct = sizeof(OLEUIBUSY);
           bz.dwFlags = BZ_NOTRESPONDINGDIALOG;
           bz.hWndOwner =GetTopWindowInWindowsTask(lpStdMsgFilter-
>m_hWndParent);
           bz.lpszCaption = lpStdMsgFilter->m_lpszAppName;
           bz.lpfnHook = lpStdMsgFilter->m_lpfnBusyDialogHookCallback;
           bz.lCustData = 0;
           bz.hInstance = NULL;
           bz.lpszTemplate = NULL;
           bz.hResource = 0;
           bz.hTask = htaskCallee;

           /* Set up the address to the hWnd in our MsgFilter structure.
The
           ** call to OleUIBusy will fill this in with the hWnd of the busy
           ** dialog box
           */
           bz.lphWndDialog =  (HWND FAR *)&(lpStdMsgFilter-
>m_hWndBusyDialog);
           uRet = OleUIBusy(&bz);

           lpStdMsgFilter->m_bUnblocking = FALSE;

           return PENDINGMSG_WAITNOPROCESS;
           }
#if defined( _DEBUG )
           else {
                 OleDbgOut3("OleStdMsgFilter_MessagePending: BLOCKED but
dialog Disabled\r\n");
           }
#endif
     }

     /* If we're already unblocking, we're being re-entered.  Don't
     ** process message
     */

     if (lpStdMsgFilter->m_bUnblocking)
           return PENDINGMSG_WAITDEFPROCESS;

     /* OLE2NOTE: If we have a callback function set up, call it with the
     ** current message.  If not, tell OLE LPRC mechanism to automatically
     ** handle all messages.
     */
     if (lpStdMsgFilter->m_lpfnMessagePendingCallback &&
           !IsBadCodePtr((FARPROC)lpStdMsgFilter-
>m_lpfnMessagePendingCallback)){
           MSG msg;

           /* OLE2NOTE: the app provided a MessagePendingCallback
           **    function. we will PeekMessage for the first message in
           **    the queue and pass it to the app. the app in its callback
           **    function can decide to Dispatch this message or it can
           **    PeekMessage on its own giving particular message filter
           **    criteria. if the app returns TRUE then we return
           **    PENDINGMSG_WAITNOPROCESS to OLE telling OLE to leave the
           **    message in the queue. If the app returns FALSE, then we
           **    return PENDINGMSG_WAITDEFPROCESS to OLE telling OLE to do
           **    its default processing with the message. by default OLE
           **    dispatches system messages and eats other messages and
           **    beeps.
           */
           if (PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE | PM_NOYIELD)) {

                 if (lpStdMsgFilter->m_lpfnMessagePendingCallback(&msg)) {
                       /* TRUE return means that the app processed message.
                       **
                       ** NOTE: (CHANGE FROM OLE2.0 VERSION) we leave it up
to
                       **    the callback routine to remove the message if
it
                       **    wants.
```

```
                        */
                        dwReturn = PENDINGMSG_WAITNOPROCESS;
                  } else {
                        /* FALSE means that the app did not process the
                        **    message. we will let OLE take its
                        **    default action.
                        **
                        ** NOTE: (CHANGE FROM OLE2.0 VERSION) we used to
return
                        **    PENDINGMSG_WAITNOPROCESS to leave the message
in
                        **    the queue; now we return
PENDINGMSG_WAITDEFPROCESS
                        **    to let OLE do default processing.
                        */
                        dwReturn = PENDINGMSG_WAITDEFPROCESS;

#if defined( _DEBUG )
                        wsprintf(
                                    szBuf,
                                    "Message (0x%x) (wParam=0x%x,
lParam=0x%lx) using WAITDEFPROCESS while blocked\r\n",
                                    msg.message,
                                    msg.lParam,
                                    msg.wParam
                        );
                        OleDbgOut2(szBuf);
#endif  // _DEBUG
                  }
            }
      }

      return dwReturn;
}
```

OBJFDBK.C   (WRAPUI Sample)

```
/*
 * OBJFDBK.C
 *
 * Miscellaneous API's to generate UI feedback effects for OLE objects. This
 * is part of the OLE 2.0 User Interface Support Library.
 * The following feedback effects are supported:
 *      1. Object selection handles (OleUIDrawHandles)
 *      2. Open Object window shading (OleUIDrawShading)
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */

#define STRICT  1
#include "ole2ui.h"

static void DrawHandle(HDC hdc, int x, int y, UINT cSize, BOOL bInvert, BOOL
fDraw);

/*
 * OleUIDrawHandles
 *
 * Purpose:
 *  Draw handles or/and boundary around Container Object when selected
 *
 * Parameters:
 *  lpRect      Dimensions of Container Object
 *  hdc         HDC of Container Object (MM_TEXT mapping mode)
 *  dwFlags-
 *      Exclusive flags
 *          OLEUI_HANDLES_INSIDE    Draw handles on inside of rect
 *          OLEUI_HANDLES_OUTSIDE   Draw handles on outside of rect
 *      Optional flags
 *          OLEUI_HANDLES_NOBORDER  Draw handles only, no rect
 *          OLEUI_HANDLES_USEINVERSE
 *              use invert for handles and rect, o.t. use COLOR_WINDOWTEXT
 *  cSize       size of handle box
 *  fDraw       Draw if TRUE, erase if FALSE
 *
 * Return Value: null
 *
 */
STDAPI_(void) OleUIDrawHandles(
     LPRECT  lpRect,
     HDC     hdc,
     DWORD   dwFlags,
     UINT    cSize,
     BOOL    fDraw
)
{
     HBRUSH  hbr;
     RECT    rc;
     int     bkmodeOld;
     BOOL    bInvert = (BOOL) (dwFlags & OLEUI_HANDLES_USEINVERSE);

     CopyRect((LPRECT)&rc, lpRect);

     bkmodeOld = SetBkMode(hdc, TRANSPARENT);

     if (dwFlags & OLEUI_HANDLES_OUTSIDE)
           InflateRect((LPRECT)&rc, cSize - 1, cSize - 1);

     // Draw the handles inside the rectangle boundary
     DrawHandle(hdc, rc.left, rc.top, cSize, bInvert, fDraw);
     DrawHandle(hdc, rc.left, rc.top+(rc.bottom-rc.top-cSize)/2, cSize,
bInvert, fDraw);
     DrawHandle(hdc, rc.left, rc.bottom-cSize, cSize, bInvert, fDraw);
     DrawHandle(hdc, rc.left+(rc.right-rc.left-cSize)/2, rc.top, cSize,
bInvert, fDraw);
     DrawHandle(hdc, rc.left+(rc.right-rc.left-cSize)/2, rc.bottom-cSize,
cSize, bInvert, fDraw);
     DrawHandle(hdc, rc.right-cSize, rc.top, cSize, bInvert, fDraw);
     DrawHandle(hdc, rc.right-cSize, rc.top+(rc.bottom-rc.top-cSize)/2,
cSize, bInvert, fDraw);
     DrawHandle(hdc, rc.right-cSize, rc.bottom-cSize, cSize, bInvert,
fDraw);

     if (!(dwFlags & OLEUI_HANDLES_NOBORDER)) {
           if (fDraw)
                 hbr = GetStockObject(BLACK_BRUSH);
           else
                 hbr = GetStockObject(WHITE_BRUSH);

           FrameRect(hdc, lpRect, hbr);
     }

     SetBkMode(hdc, bkmodeOld);
}



/*
 * DrawHandle
 *
 * Purpose:
 *  Draw a handle box at the specified coordinate
 *
 * Parameters:
 *  hdc         HDC to be drawn into
 *  x, y        upper left corner coordinate of the handle box
 *  cSize       size of handle box
 *  bInvert     use InvertRect() if TRUE, otherwise use Rectangle()
 *  fDraw       Draw if TRUE, erase if FALSE, ignored if bInvert is TRUE
 *
 * Return Value: null
 *
 */
```

```
static void DrawHandle(HDC hdc, int x, int y, UINT cSize, BOOL bInvert, BOOL
fDraw)
{
     HBRUSH  hbr;
     HBRUSH  hbrOld;
     HPEN    hpen;
     HPEN    hpenOld;
     RECT    rc;


     if (!bInvert) {
           if (fDraw) {
                 hpen = GetStockObject(BLACK_PEN);
                 hbr = GetStockObject(BLACK_BRUSH);
           } else {
                 hpen = GetStockObject(WHITE_PEN);
                 hbr = GetStockObject(WHITE_PEN);
           }

           hpenOld = SelectObject(hdc, hpen);
           hbrOld = SelectObject(hdc, hbr);
           Rectangle(hdc, x, y, x+cSize, y+cSize);
           SelectObject(hdc, hpenOld);
           SelectObject(hdc, hbrOld);
     }
     else {
           rc.left = x;
           rc.top = y;
           rc.right = x + cSize;
           rc.bottom = y + cSize;
           InvertRect(hdc, (LPRECT)&rc);
     }
}


/*
 * OleUIDrawShading
 *
 * Purpose:
 *  Shade the object when it is in in-place editing. Borders are drawn
 *  on the Object rectangle. The right and bottom edge of the rectangle
 *  are excluded in the drawing.
 *
 * Parameters:
 *  lpRect      Dimensions of Container Object
 *  hdc         HDC for drawing
 *  dwFlags-
 *      Exclusive flags
 *          OLEUI_SHADE_FULLRECT    Shade the whole rectangle
 *          OLEUI_SHADE_BORDERIN    Shade cWidth pixels inside rect
 *          OLEUI_SHADE_BORDEROUT   Shade cWidth pixels outside rect
 *      Optional flags
 *          OLEUI_SHADE_USEINVERSE
 *              use PATINVERT instead of the hex value
 *  cWidth      width of border in pixel
 *
 * Return Value: null
 *
 */
STDAPI_(void) OleUIDrawShading(LPRECT lpRect, HDC hdc, DWORD dwFlags, UINT
cWidth)
{
     HBRUSH  hbr;
     HBRUSH  hbrOld;
     HBITMAP hbm;
     RECT    rc;
     UINT    wHatchBmp[] = {0x11, 0x22, 0x44, 0x88, 0x11, 0x22, 0x44, 0x88};
     COLORREF cvText;
     COLORREF cvBk;

     hbm = CreateBitmap(8, 8, 1, 1, wHatchBmp);
     hbr = CreatePatternBrush(hbm);
     hbrOld = SelectObject(hdc, hbr);

     rc = *lpRect;

     if (dwFlags == OLEUI_SHADE_FULLRECT) {
           cvText = SetTextColor(hdc, RGB(255, 255, 255));
           cvBk = SetBkColor(hdc, RGB(0, 0, 0));
           PatBlt(hdc, rc.left, rc.top, rc.right-rc.left, rc.bottom-rc.top,
                 0x00A000C9L /* DPa */ );

     } else {    // either inside or outside rect

           if (dwFlags == OLEUI_SHADE_BORDEROUT)
                 InflateRect((LPRECT)&rc, cWidth - 1, cWidth - 1);

           cvText = SetTextColor(hdc, RGB(255, 255, 255));
           cvBk = SetBkColor(hdc, RGB(0, 0, 0));
           PatBlt(hdc, rc.left, rc.top, rc.right - rc.left,
                 cWidth, 0x00A000C9L /* DPa */);
           PatBlt(hdc, rc.left, rc.top, cWidth, rc.bottom - rc.top,
                 0x00A000C9L /* DPa */);
           PatBlt(hdc, rc.right - cWidth, rc.top, cWidth,
                 rc.bottom - rc.top, 0x00A000C9L /* DPa */);
           PatBlt(hdc, rc.left, rc.bottom - cWidth, rc.right-rc.left,
                 cWidth, 0x00A000C9L /* DPa */);
     }

     SetTextColor(hdc, cvText);
     SetBkColor(hdc, cvBk);
     SelectObject(hdc, hbrOld);
     DeleteObject(hbr);
     DeleteObject(hbm);
}


/*
 * OleUIShowObject
 *
 * Purpose:
 *  Draw the ShowObject effect around the object
 *
 * Parameters:
 *  lprc        rectangle for drawing
 *  hdc         HDC for drawing
 *  fIsLink     linked object (TRUE) or embedding object (FALSE)
 *
 * Return Value: null
 *
 */
STDAPI_(void) OleUIShowObject(LPCRECT lprc, HDC hdc, BOOL fIsLink)
{
     HPEN    hpen;
     HPEN    hpenOld;
     HBRUSH  hbrOld;

     if (!lprc || !hdc)
           return;

     hpen = fIsLink ? CreatePen(PS_DASH, 1, RGB(0,0,0)) :
                          GetStockObject(BLACK_PEN);

     if (!hpen)
           return;

     hpenOld = SelectObject(hdc, hpen);
     hbrOld = SelectObject(hdc, GetStockObject(NULL_BRUSH));

     Rectangle(hdc, lprc->left, lprc->top, lprc->right, lprc->bottom);

     SelectObject(hdc, hpenOld);
     SelectObject(hdc, hbrOld);

     if (fIsLink)
           DeleteObject(hpen);

}
```

OLE2UI.DEF (WRAPUI Sample)

```
;;
;;    ole2ui.def
;;
;;        Definition file for ole2ui.dll
;;
;;        (c) Copyright Microsoft Corp. 1992 - 1993 All Rights Reserved
;;
;;

CODE            MOVEABLE DISCARDABLE
DATA            MOVEABLE SINGLE

HEAPSIZE        16392

EXPORTS         WEP                                 @1 RESIDENTNAME
;;
;; NOTE: DO NOT PLACE ANY EXPORTS HERE -- USE THE
;; __export KEYWORD INSTEAD.


SetDCToAnisotropic
SetDCToDrawInHimetricRect
ResetOrigDC
XformRectInPixelsToHimetric
XformRectInHimetricToPixels
XformSizeInPixelsToHimetric
XformSizeInHimetricToPixels
XformWidthInHimetricToPixels
XformWidthInPixelsToHimetric
XformHeightInHimetricToPixels
XformHeightInPixelsToHimetric

;; converted to a macro
;;   AreRectsEqual
ParseCmdLine
OleStdIsOleLink
OleStdQueryInterface
OleStdCreateRootStorage
OleStdOpenRootStorage
OleStdOpenOrCreateRootStorage
OleStdCreateChildStorage
OleStdOpenChildStorage
OleStdCommitStorage
OleStdCreateStorageOnHGlobal
OleStdCreateTempStorage
OleStdDoConvert
OleStdGetTreatAsFmtUserType
OleStdDoTreatAsClass
OleStdSetupAdvises
OleStdSwitchDisplayAspect
OleStdSetIconInCache
OleStdGetData
```

```
OleStdMarkPasteEntryList
OleStdGetPriorityClipboardFormat
OleStdIsDuplicateFormat
OleStdRegisterAsRunning
OleStdRevokeAsRunning
OleStdNoteFileChangeTime
OleStdNoteObjectChangeTime
OleStdGetOleObjectData
OleStdGetLinkSourceData
OleStdGetObjectDescriptorData
OleStdGetObjectDescriptorDataFromOleObject
OleStdFillObjectDescriptorFromData
;; converted to a macro
;;   OleStdCopyMetafilePict
OleStdGetMetafilePictFromOleObject
;; converted to a macro
;;   OleStdQueryOleObjectData
;; converted to a macro
;;   OleStdQueryLinkSourceData
;; converted to a macro
;;   OleStdQueryObjectDescriptorData
;; converted to a macro
;;   OleStdQueryFormatMedium
;; converted to a macro
;;   OleStdGetDropEffect
OleStdCreateTempFileMoniker
OleStdGetFirstMoniker
OleStdGetLenFilePrefixOfMoniker
OleStdMalloc
OleStdRealloc
OleStdFree
OleStdGetSize
OleStdFreeString
OleStdCopyString
OleStdGetItemToken
OleStdInitVtbl
OleStdCheckVtbl
OleStdVerifyRelease
OleStdRelease
OleStdCreateDC
OleStdCreateIC
OleStdCreateTargetDevice
OleStdDeleteTargetDevice
OleStdCopyTargetDevice
OleStdCopyFormatEtc
OleDbgPrint
OleDbgPrintAlways
OleDbgSetDbgLevel
OleDbgGetDbgLevel
OleDbgIndent
OleDbgPrintRefCnt
OleDbgPrintRefCntAlways
OleDbgPrintRect
OleDbgPrintRectAlways
OleDbgPrintScodeAlways
```

```
OleUIInitialize
OleUIUnInitialize
OleUIAddVerbMenu
;; superceeded by OleMetafilePictFromIconAndLabel
;; OleUIMetafilePictFromIconAndLabel
OleUIMetafilePictIconFree
OleUIMetafilePictIconDraw
OleUIMetafilePictExtractLabel
OleUIMetafilePictExtractIcon
OleUIMetafilePictExtractIconSource
OleUIInsertObject
OleUIPasteSpecial
OleUIEditLinks
OleUIChangeIcon
OleUIConvert
OleUIBusy
OleUIUpdateLinks
OleUIDrawHandles
OleUICanConvertOrActivateAs
OleUIDrawShading
OleUIShowObject
RegisterHatchWindowClass
CreateHatchWindow
GetHatchWidth
GetHatchRect
SetHatchRect
SetHatchWindowSize
;;_OleUIPromptUser
OleUIPromptUser
OleStdEnumFmtEtc_Create
;; superceeded by OleGetIconOfFile
;; GetIconOfFile
;; superceeded by OleGetIconOfClass
;; GetIconOfClass
OleStdGetAuxUserType
OleStdGetUserTypeOfClass
OleStdIconLabelTextOut
OleStdMsgFilter_Create
OleStdMsgFilter_SetInComingCallStatus
OleStdMsgFilter_GetInComingCallStatus
OleStdMsgFilter_EnableBusyDialog
OleStdMsgFilter_EnableNotRespondingDialog
OleStdMsgFilter_SetParentWindow
OleStdGetMiscStatusOfClass
OleStdGetDefaultFileFormatOfClass
OleStdDestroyAllElements
OleStdCreateDbAlloc

OleStdInitSummaryInfo
OleStdFreeSummaryInfo
OleStdClearSummaryInfo
;;OleStdReadSummaryInfo
OleStdWriteSummaryInfo
OleStdGetSecurityProperty
```

```
OleStdSetSecurityProperty
OleStdGetStringProperty
OleStdSetStringProperty
OleStdGetStringZProperty
OleStdGetDocProperty
OleStdSetDocProperty
OleStdGetThumbNailProperty
OleStdSetThumbNailProperty
OleStdGetDateProperty
OleStdSetDateProperty

OleStdMsgFilter_SetHandleInComingCallbackProc
OleStdEnumStatData_Create
OleStdCopyStatData
OleStdCreateStandardPalette
OleStdMkParseDisplayName
OleUILockLibrary
_
```

OLE2UI.H   (WRAPUI Sample)

```
/*
 * OLE2UI.H
 *
 * Published definitions, structures, types, and function prototypes for the
 * OLE 2.0 User Interface support library.
 *
 * Copyright (c)1993 Microsoft Corporation, All Rights Reserved
 */

/* NOTE: All dialog and string resource ID's defined in this file are
 *    in the range:
 *          32248 - 32504   (0x7DF8 - 0x7EF8)
*/


#ifndef _OLE2UI_H_
#define _OLE2UI_H_

#ifndef RC_INVOKED
#pragma message ("Including OLE2UI.H from " __FILE__)
#endif  //RC_INVOKED

#ifdef WIN32
#define _INC_OLE
// #define __RPC_H__
#endif

#if !defined(__cplusplus) && !defined( __TURBOC__)
#define NONAMELESSUNION     // use strict ANSI standard (for DVOBJ.H)
#endif

#include <windows.h>
#include <shellapi.h>
#include <ole2.h>
#include <string.h>
#include <dlgs.h>           //For fileopen dlg; standard include

/*************************************************************************
** DEBUG ASSERTION ROUTINES
*************************************************************************/

#ifdef DBG
#include <assert.h>
#define FnAssert(lpstrExpr, lpstrMsg, lpstrFileName, iLine)     \
        (_assert(lpstrMsg ? lpstrMsg : lpstrExpr,               \
                 lpstrFileName,                                 \
                 iLine), NOERROR)
#endif //DBG

#include "olestd.h"

#ifdef __TURBOC__
```

```
#define _getcwd getcwd
#define _itoa   itoa
#define __max   max
#define _find_t find_t
#endif // __TURBOC__

#ifdef WIN32
#define _fmemset memset
#define _fmemcpy memcpy
#define _fmemcmp memcmp
#define _fstrcpy strcpy
#define _fstrlen strlen
#define _fstrrchr strrchr
#define _fstrtok strtok

// BUGBUG32: isspace function does not seem to work properly
#undef isspace
#define isspace(j) (j==' ' || j=='\t' || j=='\n')
#endif  // WIN32

#if !defined( EXPORT )
#ifdef WIN32
#define EXPORT
#else
#define EXPORT  __export
#endif  // WIN32
#endif  // !EXPORT

/*
 * Initialization / Uninitialization routines.  OleUIInitialize
 * must be called prior to using any functions in OLE2UI, and
OleUIUnInitialize
 * must be called before you app shuts down and when you are done using the
 * library.
 *
 * NOTE:  If you are using the DLL version of this library, these functions
 * are automatically called in the DLL's LibMain and WEP, so you should
 * not call them directly from your application.
 */

// Backward compatibility with older library
#define OleUIUninitialize OleUIUnInitialize

STDAPI_(BOOL) OleUIInitialize(HINSTANCE hInstance, HINSTANCE hPrevInst);
STDAPI_(BOOL) OleUIUninitialize(void);

#if !defined( SZCLASSICONBOX )
#define SZCLASSICONBOX  "ole2uiIBCls"
#endif

#if !defined( SZCLASSRESULTIMAGE )
#define SZCLASSRESULTIMAGE  "ole2uiRICls"
#endif

// object count, used to support DllCanUnloadNow and OleUICanUnloadNow
```

```
extern DWORD g_dwObjectCount;

STDAPI OleUICanUnloadNow(void);
STDAPI OleUILockLibrary(BOOL fLock);


//Dialog Identifiers as passed in Help messages to identify the source.
#define IDD_INSERTOBJECT        32248
#define IDD_CHANGEICON          32249
#define IDD_CONVERT             32250
#define IDD_PASTESPECIAL        32251
#define IDD_EDITLINKS           32252
#define IDD_FILEOPEN            32253
#define IDD_BUSY                32254
#define IDD_UPDATELINKS         32255
#define IDD_CANNOTUPDATELINK    32256
#define IDD_CHANGESOURCE        32257
#define IDD_INSERTFILEBROWSE    32258
#define IDD_CHANGEICONBROWSE    32259

// The following Dialogs are message dialogs used by OleUIPromptUser API
#define IDD_LINKSOURCEUNAVAILABLE   32260
#define IDD_SERVERNOTREG        32261
#define IDD_LINKTYPECHANGED     32262

#define IDD_SERVERNOTFOUND      32263
#define IDD_OUTOFMEMORY         32264

// Stringtable identifers
#define IDS_OLE2UIUNKNOWN       32300
#define IDS_OLE2UILINK          32301
#define IDS_OLE2UIOBJECT        32302
#define IDS_OLE2UIEDIT          32303
#define IDS_OLE2UICONVERT       32304
#define IDS_OLE2UIEDITLINKCMD_1VERB     32305
#define IDS_OLE2UIEDITOBJECTCMD_1VERB   32306
#define IDS_OLE2UIEDITLINKCMD_NVERB     32307
#define IDS_OLE2UIEDITOBJECTCMD_NVERB   32308
#define IDS_OLE2UIEDITNOOBJCMD  32309
// def. icon label (usu. "Document")
#define IDS_DEFICONLABEL        32310
#define IDS_OLE2UIPASTELINKEDTYPE  32311


#define IDS_FILTERS             32320
#define IDS_ICONFILTERS         32321
#define IDS_BROWSE              32322

//Resource identifiers for bitmaps
#define IDB_RESULTSEGA                  32325
#define IDB_RESULTSVGA                  32326
#define IDB_RESULTSHIRESVGA             32327


//Missing from windows.h
```

```
#ifndef PVOID
typedef VOID *PVOID;
#endif


//Hook type used in all structures.
typedef UINT (CALLBACK *LPFNOLEUIHOOK)(HWND, UINT, WPARAM, LPARAM);


//Strings for registered messages
#define SZOLEUI_MSG_HELP                "OLEUI_MSG_HELP"
#define SZOLEUI_MSG_ENDDIALOG           "OLEUI_MSG_ENDDIALOG"
#define SZOLEUI_MSG_BROWSE              "OLEUI_MSG_BROWSE"
#define SZOLEUI_MSG_CHANGEICON          "OLEUI_MSG_CHANGEICON"
#define SZOLEUI_MSG_CLOSEBUSYDIALOG     "OLEUI_MSG_CLOSEBUSYDIALOG"
#define SZOLEUI_MSG_FILEOKSTRING        "OLEUI_MSG_FILEOKSTRING"

//Standard error definitions
#define OLEUI_FALSE                     0
#define OLEUI_SUCCESS                   1     //No error, same as OLEUI_OK
#define OLEUI_OK                        1     //OK button pressed
#define OLEUI_CANCEL                    2     //Cancel button pressed

#define OLEUI_ERR_STANDARDMIN           100
#define OLEUI_ERR_STRUCTURENULL         101   //Standard field validation
#define OLEUI_ERR_STRUCTUREINVALID      102
#define OLEUI_ERR_CBSTRUCTINCORRECT     103
#define OLEUI_ERR_HWNDOWNERINVALID      104
#define OLEUI_ERR_LPSZCAPTIONINVALID    105
#define OLEUI_ERR_LPFNHOOKINVALID       106
#define OLEUI_ERR_HINSTANCEINVALID      107
#define OLEUI_ERR_LPSZTEMPLATEINVALID   108
#define OLEUI_ERR_HRESOURCEINVALID      109

#define OLEUI_ERR_FINDTEMPLATEFAILURE   110   //Initialization errors
#define OLEUI_ERR_LOADTEMPLATEFAILURE   111
#define OLEUI_ERR_DIALOGFAILURE         112
#define OLEUI_ERR_LOCALMEMALLOC         113
#define OLEUI_ERR_GLOBALMEMALLOC        114
#define OLEUI_ERR_LOADSTRING            115

#define OLEUI_ERR_STANDARDMAX           116   //Start here for specific
errors.



//Help Button Identifier
#define ID_OLEUIHELP                    99

// Help button for fileopen.dlg  (need this for resizing) 1038 is pshHelp
#define IDHELP  1038

// Static text control (use this instead of -1 so things work correctly for
// localization
#define  ID_STATIC                      98
```

```
//Maximum key size we read from the RegDB.
#define OLEUI_CCHKEYMAX                 256  // make any changes to this in
geticon.c too

//Maximum verb length and length of Object menu
#define OLEUI_CCHVERBMAX                32
#define OLEUI_OBJECTMENUMAX             256

//Maximum MS-DOS pathname.
#define OLEUI_CCHPATHMAX                256 // make any changes to this in
geticon.c too
#define OLEUI_CCHFILEMAX                13

//Icon label length
#define OLEUI_CCHLABELMAX               40  // make any changes to this in
geticon.c too

//Length of the CLSID string
#define OLEUI_CCHCLSIDSTRING            39


/*
 * What follows here are first function prototypes for general utility
 * functions, then sections laid out by dialog.  Each dialog section
 * defines the dialog structure, the API prototype, flags for the dwFlags
 * field, the dialog-specific error values, and dialog control IDs (for
 * hooks and custom templates.
 */


//Miscellaneous utility functions.
STDAPI_(BOOL) OleUIAddVerbMenu(LPOLEOBJECT lpOleObj,
                                                         LPSTR
lpszShortType,
                                                         HMENU hMenu,
                                                         UINT uPos,
                                                         UINT uIDVerbMin,
                                                         UINT uIDVerbMax,
                                                         BOOL bAddConvert,
                                                         UINT idConvert,
                                                         HMENU FAR
*lphMenu);

//Metafile utility functions
#ifdef OBSOLETE
STDAPI_(HGLOBAL) OleUIMetafilePictFromIconAndLabel(HICON, LPSTR, LPSTR,
UINT);
#endif // OBSOLETE
STDAPI_(void)    OleUIMetafilePictIconFree(HGLOBAL);
STDAPI_(BOOL)    OleUIMetafilePictIconDraw(HDC, LPRECT, HGLOBAL, BOOL);
STDAPI_(UINT)    OleUIMetafilePictExtractLabel(HGLOBAL, LPSTR, UINT,
LPDWORD);
STDAPI_(HICON)   OleUIMetafilePictExtractIcon(HGLOBAL);
STDAPI_(BOOL)    OleUIMetafilePictExtractIconSource(HGLOBAL,LPSTR,UINT FAR
*);




/*************************************************************************
** INSERT OBJECT DIALOG
*************************************************************************/


typedef struct tagOLEUIINSERTOBJECT
        {
        //These IN fields are standard across all OLEUI dialog functions.
        DWORD           cbStruct;         //Structure Size
        DWORD           dwFlags;          //IN-OUT:  Flags
        HWND            hWndOwner;        //Owning window
        LPCSTR          lpszCaption;      //Dialog caption bar contents
        LPFNOLEUIHOOK   lpfnHook;         //Hook callback
        LPARAM          lCustData;        //Custom data to pass to hook
        HINSTANCE       hInstance;        //Instance for customized template
name
        LPCSTR          lpszTemplate;     //Customized template name
        HRSRC           hResource;        //Customized template handle

        //Specifics for OLEUIINSERTOBJECT.  All are IN-OUT unless otherwise
spec.
        CLSID           clsid;            //Return space for class ID
        LPSTR           lpszFile;         //Filename for inserts or links
        UINT            cchFile;          //Size of lpszFile buffer:
OLEUI_CCHPATHMAX
        UINT            cClsidExclude;    //IN only:  CLSIDs in
lpClsidExclude
        LPCLSID         lpClsidExclude;   //List of CLSIDs to exclude from
listing.

        //Specific to create objects if flags say so
        IID             iid;              //Requested interface on creation.
        DWORD           oleRender;        //Rendering option
        LPFORMATETC     lpFormatEtc;      //Desired format
        LPOLECLIENTSITE lpIOleClientSite; //Site to be use for the object.
        LPSTORAGE       lpIStorage;       //Storage used for the object
        LPVOID FAR     *ppvObj;           //Where the object is returned.
        SCODE           sc;               //Result of creation calls.
        HGLOBAL         hMetaPict;        //OUT: METAFILEPICT containing
iconic aspect.
                                                                            //
IFF we couldn't stuff it in the cache.
        } OLEUIINSERTOBJECT, *POLEUIINSERTOBJECT, FAR *LPOLEUIINSERTOBJECT;

//API prototype
STDAPI_(UINT) OleUIInsertObject(LPOLEUIINSERTOBJECT);
```

```
//Insert Object flags
#define IOF_SHOWHELP                0x00000001L
#define IOF_SELECTCREATENEW         0x00000002L
#define IOF_SELECTCREATEFROMFILE    0x00000004L
#define IOF_CHECKLINK               0x00000008L
#define IOF_CHECKDISPLAYASICON      0x00000010L
#define IOF_CREATENEWOBJECT         0x00000020L
#define IOF_CREATEFILEOBJECT        0x00000040L
#define IOF_CREATELINKOBJECT        0x00000080L
#define IOF_DISABLELINK             0x00000100L
#define IOF_VERIFYSERVERSEXIST      0x00000200L
#define IOF_DISABLEDISPLAYASICON    0x00000400L


//Insert Object specific error codes
#define OLEUI_IOERR_LPSZFILEINVALID         (OLEUI_ERR_STANDARDMAX+0)
#define OLEUI_IOERR_LPSZLABELINVALID        (OLEUI_ERR_STANDARDMAX+1)
#define OLEUI_IOERR_HICONINVALID            (OLEUI_ERR_STANDARDMAX+2)
#define OLEUI_IOERR_LPFORMATETCINVALID      (OLEUI_ERR_STANDARDMAX+3)
#define OLEUI_IOERR_PPVOBJINVALID           (OLEUI_ERR_STANDARDMAX+4)
#define OLEUI_IOERR_LPIOLECLIENTSITEINVALID (OLEUI_ERR_STANDARDMAX+5)
#define OLEUI_IOERR_LPISTORAGEINVALID       (OLEUI_ERR_STANDARDMAX+6)
#define OLEUI_IOERR_SCODEHASERROR           (OLEUI_ERR_STANDARDMAX+7)
#define OLEUI_IOERR_LPCLSIDEXCLUDEINVALID   (OLEUI_ERR_STANDARDMAX+8)
#define OLEUI_IOERR_CCHFILEINVALID          (OLEUI_ERR_STANDARDMAX+9)


//Insert Object Dialog identifiers
#define ID_IO_CREATENEW                 2100
#define ID_IO_CREATEFROMFILE            2101
#define ID_IO_LINKFILE                  2102
#define ID_IO_OBJECTTYPELIST            2103
#define ID_IO_DISPLAYASICON             2104
#define ID_IO_CHANGEICON                2105
#define ID_IO_FILE                      2106
#define ID_IO_FILEDISPLAY               2107
#define ID_IO_RESULTIMAGE               2108
#define ID_IO_RESULTTEXT                2109
#define ID_IO_ICONDISPLAY               2110
#define ID_IO_OBJECTTYPETEXT            2111
#define ID_IO_FILETEXT                  2112
#define ID_IO_FILETYPE                  2113

// Strings in OLE2UI resources
#define IDS_IORESULTNEW                 32400
#define IDS_IORESULTNEWICON             32401
#define IDS_IORESULTFROMFILE1           32402
#define IDS_IORESULTFROMFILE2           32403
#define IDS_IORESULTFROMFILEICON2       32404
#define IDS_IORESULTLINKFILE1           32405
#define IDS_IORESULTLINKFILE2           32406
#define IDS_IORESULTLINKFILEICON1       32407
#define IDS_IORESULTLINKFILEICON2       32408
```

```
/*************************************************************************
** PASTE SPECIAL DIALOG
*************************************************************************/

// Maximum number of link types
#define     PS_MAXLINKTYPES  8

//NOTE: OLEUIPASTEENTRY and OLEUIPASTEFLAG structs are defined in OLESTD.H

typedef struct tagOLEUIPASTESPECIAL
        {
        //These IN fields are standard across all OLEUI dialog functions.
        DWORD           cbStruct;       //Structure Size
        DWORD           dwFlags;        //IN-OUT:  Flags
        HWND            hWndOwner;      //Owning window
        LPCSTR          lpszCaption;    //Dialog caption bar contents
        LPFNOLEUIHOOK   lpfnHook;       //Hook callback
        LPARAM          lCustData;      //Custom data to pass to hook
        HINSTANCE       hInstance;      //Instance for customized template
name
        LPCSTR          lpszTemplate;   //Customized template name
        HRSRC           hResource;      //Customized template handle

        //Specifics for OLEUIPASTESPECIAL.

        //IN  fields
        LPDATAOBJECT    lpSrcDataObj;       //Source IDataObject* (on the

// clipboard) for data to paste

        LPOLEUIPASTEENTRY arrPasteEntries;  //OLEUIPASTEENTRY array which

// specifies acceptable formats. See

// OLEUIPASTEENTRY for more info.
        int             cPasteEntries;      //No. of OLEUIPASTEENTRY array
entries

        UINT        FAR *arrLinkTypes;      //List of link types that are

// acceptable. Link types are referred

// to using OLEUIPASTEFLAGS in

// arrPasteEntries
        int             cLinkTypes;         //Number of link types
        UINT            cClsidExclude;      //Number of CLSIDs in
lpClsidExclude
        LPCLSID         lpClsidExclude;     //List of CLSIDs to exclude from
list.

        //OUT fields
        int             nSelectedIndex;     //Index of arrPasteEntries[]
that the
// user selected
        BOOL            fLink;              //Indicates if Paste or Paste
Link was

// selected by the user
        HGLOBAL         hMetaPict;          //Handle to Metafile containing
icon

// and icon title selected by the user

// Use the Metafile utility functions

// defined in this header to

// manipulate hMetaPict
        SIZEL           sizel;              // size of object/link in its
source

//  if the display aspect chosen by

//  the user matches the aspect

//  displayed in the source. if

//  different aspect is chosen then

//  sizel.cx=sizel.cy=0 is returned.

//  sizel displayed in source is

//  retrieved from the

//  ObjectDescriptor if fLink is FALSE

//  LinkSrcDescriptor if fLink is TRUE
        } OLEUIPASTESPECIAL, *POLEUIPASTESPECIAL, FAR *LPOLEUIPASTESPECIAL;


//API to bring up PasteSpecial dialog
STDAPI_(UINT) OleUIPasteSpecial(LPOLEUIPASTESPECIAL);


//Paste Special flags
// Show Help button. IN flag.
#define PSF_SHOWHELP                0x00000001L

//Select Paste radio button at dialog startup. This is the default if
// PSF_SELECTPASTE or PSF_SELECTPASTELINK are not specified. Also specifies
// state of button on dialog termination. IN/OUT flag.
#define PSF_SELECTPASTE             0x00000002L

//Select PasteLink radio button at dialog startup. Also specifies state of
// button on dialog termination. IN/OUT flag.
#define PSF_SELECTPASTELINK         0x00000004L
```

```
//Specfies if DisplayAsIcon button was checked on dialog termination. OUT
flag
#define PSF_CHECKDISPLAYASICON      0x00000008L
#define PSF_DISABLEDISPLAYASICON    0x00000010L


//Paste Special specific error codes
#define OLEUI_IOERR_SRCDATAOBJECTINVALID      (OLEUI_ERR_STANDARDMAX+0)
#define OLEUI_IOERR_ARRPASTEENTRIESINVALID    (OLEUI_ERR_STANDARDMAX+1)
#define OLEUI_IOERR_ARRLINKTYPESINVALID       (OLEUI_ERR_STANDARDMAX+2)
#define OLEUI_PSERR_CLIPBOARDCHANGED          (OLEUI_ERR_STANDARDMAX+3)

//Paste Special Dialog identifiers
#define ID_PS_PASTE                     500

#define ID_PS_PASTELINK                 501
#define ID_PS_SOURCETEXT                502
#define ID_PS_PASTELIST                 503
#define ID_PS_PASTELINKLIST             504
#define ID_PS_DISPLAYLIST               505
#define ID_PS_DISPLAYASICON             506
#define ID_PS_ICONDISPLAY               507
#define ID_PS_CHANGEICON                508
#define ID_PS_RESULTIMAGE               509
#define ID_PS_RESULTTEXT                510
#define ID_PS_RESULTGROUP               511
#define ID_PS_STXSOURCE                 512
#define ID_PS_STXAS                     513

// Paste Special String IDs
#define IDS_PSPASTEDATA                 32410
#define IDS_PSPASTEOBJECT               32411
#define IDS_PSPASTEOBJECTASICON         32412
#define IDS_PSPASTELINKDATA             32413
#define IDS_PSPASTELINKOBJECT           32414
#define IDS_PSPASTELINKOBJECTASICON     32415
#define IDS_PSNONOLE                    32416
#define IDS_PSUNKNOWNTYPE               32417
#define IDS_PSUNKNOWNSRC                32418
#define IDS_PSUNKNOWNAPP                32419


/*************************************************************************
** EDIT LINKS DIALOG
*************************************************************************/



/* IOleUILinkContainer Interface
** ------------------------------
**    This interface must be implemented by container applications that
**    want to use the EditLinks dialog. the EditLinks dialog calls back
**    to the container app to perform the OLE functions to manipulate
**    the links within the container.
```

```
*/

#define LPOLEUILINKCONTAINER     IOleUILinkContainer FAR*

#undef  INTERFACE
#define INTERFACE   IOleUILinkContainer

DECLARE_INTERFACE_(IOleUILinkContainer, IUnknown)
{
        //*** IUnknown methods ***/
        STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID FAR* ppvObj)
PURE;
        STDMETHOD_(ULONG,AddRef) (THIS) PURE;
        STDMETHOD_(ULONG,Release) (THIS) PURE;

        STDMETHOD_(DWORD,GetNextLink) (THIS_ DWORD dwLink) PURE;
        STDMETHOD(SetLinkUpdateOptions) (THIS_ DWORD dwLink, DWORD
dwUpdateOpt) PURE;
        STDMETHOD(GetLinkUpdateOptions) (THIS_ DWORD dwLink, DWORD FAR*
lpdwUpdateOpt) PURE;
        STDMETHOD(SetLinkSource) (THIS_
                        DWORD       dwLink,
                        LPSTR       lpszDisplayName,
                        ULONG       lenFileName,
                        ULONG FAR*  pchEaten,
                        BOOL        fValidateSource) PURE;
        STDMETHOD(GetLinkSource) (THIS_
                        DWORD       dwLink,
                        LPSTR FAR*  lplpszDisplayName,
                        ULONG FAR*  lplenFileName,
                        LPSTR FAR*  lplpszFullLinkType,
                        LPSTR FAR*  lplpszShortLinkType,
                        BOOL FAR*   lpfSourceAvailable,
                        BOOL FAR*   lpfIsSelected) PURE;
        STDMETHOD(OpenLinkSource) (THIS_ DWORD dwLink) PURE;
        STDMETHOD(UpdateLink) (THIS_
                        DWORD dwLink,
                        BOOL fErrorMessage,
                        BOOL fErrorAction) PURE;
        STDMETHOD(CancelLink) (THIS_ DWORD dwLink) PURE;
};


typedef struct tagOLEUIEDITLINKS
        {
        //These IN fields are standard across all OLEUI dialog functions.
        DWORD           cbStruct;       //Structure Size
        DWORD           dwFlags;        //IN-OUT:  Flags
        HWND            hWndOwner;      //Owning window
        LPCSTR          lpszCaption;    //Dialog caption bar contents
        LPFNOLEUIHOOK   lpfnHook;       //Hook callback
        LPARAM          lCustData;      //Custom data to pass to hook
        HINSTANCE       hInstance;      //Instance for customized template
name
        LPCSTR          lpszTemplate;   //Customized template name
        HRSRC           hResource;      //Customized template handle

        //Specifics for OLEUI<STRUCT>.  All are IN-OUT unless otherwise
spec.

        LPOLEUILINKCONTAINER lpOleUILinkContainer;  //IN: Interface to
manipulate

//links in the container
        } OLEUIEDITLINKS, *POLEUIEDITLINKS, FAR *LPOLEUIEDITLINKS;


//API Prototype
STDAPI_(UINT) OleUIEditLinks(LPOLEUIEDITLINKS);


// Edit Links flags
#define ELF_SHOWHELP                0x00000001L
#define ELF_DISABLEUPDATENOW        0x00000002L
#define ELF_DISABLEOPENSOURCE       0x00000004L
#define ELF_DISABLECHANGESOURCE     0x00000008L
#define ELF_DISABLECANCELLINK       0x00000010L

// Edit Links Dialog identifiers
#define ID_EL_CHANGESOURCE             201
#define ID_EL_AUTOMATIC                202
#define ID_EL_CLOSE                    208
#define ID_EL_CANCELLINK               209
#define ID_EL_UPDATENOW                210
#define ID_EL_OPENSOURCE               211
#define ID_EL_MANUAL                   212
#define ID_EL_LINKSOURCE               216
#define ID_EL_LINKTYPE                 217
#define ID_EL_UPDATE                   218
#define ID_EL_NULL                     -1
#define ID_EL_LINKSLISTBOX             206
#define ID_EL_COL1                     220
#define ID_EL_COL2                     221
#define ID_EL_COL3                     222



/*************************************************************************
** CHANGE ICON DIALOG
*************************************************************************/

typedef struct tagOLEUICHANGEICON
        {
        //These IN fields are standard across all OLEUI dialog functions.
        DWORD           cbStruct;       //Structure Size
        DWORD           dwFlags;        //IN-OUT:  Flags
        HWND            hWndOwner;      //Owning window
        LPCSTR          lpszCaption;    //Dialog caption bar contents
        LPFNOLEUIHOOK   lpfnHook;       //Hook callback
        LPARAM          lCustData;      //Custom data to pass to hook
        HINSTANCE       hInstance;      //Instance for customized template
name
        LPCSTR          lpszTemplate;   //Customized template name
        HRSRC           hResource;      //Customized template handle

        //Specifics for OLEUICHANGEICON.  All are IN-OUT unless otherwise
spec.
        HGLOBAL         hMetaPict;      //Current and final image.  Source
of the

//icon is embedded in the metafile itself.
        CLSID           clsid;          //IN only: class used to get Default
icon
        char            szIconExe[OLEUI_CCHPATHMAX];
        int             cchIconExe;
        } OLEUICHANGEICON, *POLEUICHANGEICON, FAR *LPOLEUICHANGEICON;


//API prototype
STDAPI_(UINT) OleUIChangeIcon(LPOLEUICHANGEICON);


//Change Icon flags
#define CIF_SHOWHELP                0x00000001L
#define CIF_SELECTCURRENT           0x00000002L
#define CIF_SELECTDEFAULT           0x00000004L
#define CIF_SELECTFROMFILE          0x00000008L
#define CIF_USEICONEXE              0x0000000aL


//Change Icon specific error codes
#define OLEUI_CIERR_MUSTHAVECLSID           (OLEUI_ERR_STANDARDMAX+0)
#define OLEUI_CIERR_MUSTHAVECURRENTMETAFILE (OLEUI_ERR_STANDARDMAX+1)
#define OLEUI_CIERR_SZICONEXEINVALID        (OLEUI_ERR_STANDARDMAX+2)


//Change Icon Dialog identifiers
#define ID_GROUP                    120
#define ID_CURRENT                  121
#define ID_CURRENTICON              122
#define ID_DEFAULT                  123
#define ID_DEFAULTICON              124
#define ID_FROMFILE                 125
#define ID_FROMFILEEDIT             126
#define ID_ICONLIST                 127
#define ID_LABEL                    128
#define ID_LABELEDIT                129
#define ID_BROWSE                   130
#define ID_RESULTICON               132
#define ID_RESULTLABEL              133

// Stringtable defines for Change Icon
#define IDS_CINOICONSINFILE         32430
#define IDS_CIINVALIDFILE           32431
```

```
#define IDS_CIFILEACCESS            32432
#define IDS_CIFILESHARE             32433
#define IDS_CIFILEOPENFAIL          32434



/*************************************************************************
** CONVERT DIALOG
*************************************************************************/

typedef struct tagOLEUICONVERT
        {
        //These IN fields are standard across all OLEUI dialog functions.
        DWORD           cbStruct;         //Structure Size
        DWORD           dwFlags;          //IN-OUT:  Flags
        HWND            hWndOwner;        //Owning window
        LPCSTR          lpszCaption;      //Dialog caption bar contents
        LPFNOLEUIHOOK   lpfnHook;         //Hook callback
        LPARAM          lCustData;        //Custom data to pass to hook
        HINSTANCE       hInstance;        //Instance for customized template
name
        LPCSTR          lpszTemplate;     //Customized template name
        HRSRC           hResource;        //Customized template handle

        //Specifics for OLEUICONVERT.  All are IN-OUT unless otherwise spec.
        CLSID           clsid;            //Class ID sent in to dialog: IN
only
        CLSID           clsidConvertDefault;  //Class ID to use as convert
default: IN only
        CLSID           clsidActivateDefault;  //Class ID to use as activate
default: IN only

        CLSID           clsidNew;         //Selected Class ID: OUT only
        DWORD           dvAspect;         //IN-OUT, either DVASPECT_CONTENT
or
                                                                          //
DVASPECT_ICON
        WORD            wFormat;          //Original data format
        BOOL            fIsLinkedObject;  //IN only; true if object is
linked
        HGLOBAL         hMetaPict;        //IN-OUT: METAFILEPICT containing
iconic aspect.
        LPSTR           lpszUserType;     //IN-OUT: user type name of
original class.
                                                                          //
We'll do lookup if it's NULL.
                                                                          //
This gets freed on exit.
        BOOL            fObjectsIconChanged;  // OUT; TRUE if ChangeIcon was
called (and not cancelled)
        LPSTR           lpszDefLabel;     //IN-OUT: default label to use for
icon.
                                                                          //
if NULL, the short user type name
```

```
                                                                          //
will be used. if the object is a
                                                                          //
link, the caller should pass the
                                                                          //
DisplayName of the link source
                                                                          //
This gets freed on exit.

        UINT            cClsidExclude;    //IN: No. of CLSIDs in
lpClsidExclude
        LPCLSID         lpClsidExclude;   //IN: List of CLSIDs to exclude
from list
        } OLEUICONVERT, *POLEUICONVERT, FAR *LPOLEUICONVERT;


//API prototype
STDAPI_(UINT) OleUIConvert(LPOLEUICONVERT);

// Determine if there is at least one class that can Convert or ActivateAs
// the given clsid.
STDAPI_(BOOL) OleUICanConvertOrActivateAs(
                REFCLSID    rClsid,
                BOOL        fIsLinkedObject,
                WORD        wFormat
);

//Convert Dialog flags

// IN only: Shows "HELP" button
#define CF_SHOWHELPBUTTON          0x00000001L

// IN only: lets you set the convert default object - the one that is
// selected as default in the convert listbox.
#define CF_SETCONVERTDEFAULT       0x00000002L


// IN only: lets you set the activate default object - the one that is
// selected as default in the activate listbox.

#define CF_SETACTIVATEDEFAULT       0x00000004L


// IN/OUT: Selects the "Convert To" radio button, is set on exit if
// this button was selected
#define CF_SELECTCONVERTTO         0x00000008L

// IN/OUT: Selects the "Activate As" radio button, is set on exit if
// this button was selected
#define CF_SELECTACTIVATEAS        0x00000010L
#define CF_DISABLEDISPLAYASICON    0x00000020L
#define CF_DISABLEACTIVATEAS       0x00000040L
```

```
//Convert specific error codes
#define OLEUI_CTERR_CLASSIDINVALID      (OLEUI_ERR_STANDARDMAX+1)
#define OLEUI_CTERR_DVASPECTINVALID     (OLEUI_ERR_STANDARDMAX+2)
#define OLEUI_CTERR_CBFORMATINVALID     (OLEUI_ERR_STANDARDMAX+3)
#define OLEUI_CTERR_HMETAPICTINVALID    (OLEUI_ERR_STANDARDMAX+4)
#define OLEUI_CTERR_STRINGINVALID       (OLEUI_ERR_STANDARDMAX+5)


//Convert Dialog identifiers
#define IDCV_OBJECTTYPE             150
#define IDCV_DISPLAYASICON          152
#define IDCV_CHANGEICON             153
#define IDCV_ACTIVATELIST           154
#define IDCV_CONVERTTO              155
#define IDCV_ACTIVATEAS             156
#define IDCV_RESULTTEXT             157
#define IDCV_CONVERTLIST            158
#define IDCV_ICON                   159
#define IDCV_ICONLABEL1             160
#define IDCV_ICONLABEL2             161
#define IDCV_STXCURTYPE             162
#define IDCV_GRPRESULT              163
#define IDCV_STXCONVERTTO           164

// String IDs for Convert dialog
#define IDS_CVRESULTCONVERTLINK     32440
#define IDS_CVRESULTCONVERTTO       32441
#define IDS_CVRESULTNOCHANGE        32442
#define IDS_CVRESULTDISPLAYASICON   32443
#define IDS_CVRESULTACTIVATEAS      32444
#define IDS_CVRESULTACTIVATEDIFF    32445


/*************************************************************************
** BUSY DIALOG
*************************************************************************/

typedef struct tagOLEUIBUSY
        {
        //These IN fields are standard across all OLEUI dialog functions.
        DWORD           cbStruct;         //Structure Size
        DWORD           dwFlags;          //IN-OUT:  Flags ** NOTE ** this
dialog has no flags
        HWND            hWndOwner;        //Owning window
        LPCSTR          lpszCaption;      //Dialog caption bar contents
        LPFNOLEUIHOOK   lpfnHook;         //Hook callback
        LPARAM          lCustData;        //Custom data to pass to hook
        HINSTANCE       hInstance;        //Instance for customized template
name
        LPCSTR          lpszTemplate;     //Customized template name
        HRSRC           hResource;        //Customized template handle

        //Specifics for OLEUIBUSY.
        HTASK           hTask;            //IN: HTask which is blocking
        HWND FAR *      lphWndDialog;     //IN: Dialog's HWND is placed here
        } OLEUIBUSY, *POLEUIBUSY, FAR *LPOLEUIBUSY;

//API prototype
STDAPI_(UINT) OleUIBusy(LPOLEUIBUSY);

// Flags for this dialog

// IN only: Disables "Cancel" button
#define BZ_DISABLECANCELBUTTON          0x00000001L

// IN only: Disables "Switch To..." button
#define BZ_DISABLESWITCHTOBUTTON        0x00000002L

// IN only: Disables "Retry" button
#define BZ_DISABLERETRYBUTTON           0x00000004L

// IN only: Generates a "Not Responding" dialog as opposed to the
// "Busy" dialog.  The wording in the text is slightly different, and
// the "Cancel" button is grayed out if you set this flag.
#define BZ_NOTRESPONDINGDIALOG          0x00000008L

// Busy specific error/return codes
#define OLEUI_BZERR_HTASKINVALID     (OLEUI_ERR_STANDARDMAX+0)

// SWITCHTOSELECTED is returned when user hit "switch to"
#define OLEUI_BZ_SWITCHTOSELECTED    (OLEUI_ERR_STANDARDMAX+1)

// RETRYSELECTED is returned when user hit "retry"
#define OLEUI_BZ_RETRYSELECTED       (OLEUI_ERR_STANDARDMAX+2)

// CALLUNBLOCKED is returned when call has been unblocked
#define OLEUI_BZ_CALLUNBLOCKED       (OLEUI_ERR_STANDARDMAX+3)

// Busy dialog identifiers
#define IDBZ_RETRY                      600
#define IDBZ_ICON                       601
#define IDBZ_MESSAGE1                   602
#define IDBZ_SWITCHTO                   604

// Busy dialog stringtable defines
#define IDS_BZRESULTTEXTBUSY            32447
#define IDS_BZRESULTTEXTNOTRESPONDING   32448

// Links dialog stringtable defines
#define IDS_LINK_AUTO           32450
#define IDS_LINK_MANUAL         32451
#define IDS_LINK_UNKNOWN        32452
#define IDS_LINKS               32453
#define IDS_FAILED              32454
#define IDS_CHANGESOURCE        32455
#define IDS_INVALIDSOURCE       32456
#define IDS_ERR_GETLINKSOURCE   32457
#define IDS_ERR_GETLINKUPDATEOPTIONS    32458
#define IDS_ERR_ADDSTRING       32459
```

```
#define IDS_CHANGEADDITIONALLINKS   32460
#define IDS_CLOSE                 32461


/*************************************************************************
** PROMPT USER DIALOGS
*************************************************************************/
#define ID_PU_LINKS              900
#define ID_PU_TEXT               901
#define ID_PU_CONVERT            902
#define ID_PU_BROWSE             904
#define ID_PU_METER              905
#define ID_PU_PERCENT            906
#define ID_PU_STOP               907

// used for -1 ids in dialogs:
#define ID_DUMMY    999

/* inside ole2ui.c */
#ifdef __cplusplus
extern "C"
#endif
int EXPORT FAR CDECL OleUIPromptUser(WORD nTemplate, HWND hwndParent, ...);

#define UPDATELINKS_STARTDELAY  2000    // Delay before 1st link updates

//  to give the user a chance to

//  dismiss the dialog before any

//  links update.

STDAPI_(BOOL) OleUIUpdateLinks(
                LPOLEUILINKCONTAINER lpOleUILinkCntr,
                HWND hwndParent,
                LPSTR lpszTitle,
                int cLinks);


/*************************************************************************
** OLE OBJECT FEEDBACK EFFECTS
*************************************************************************/

#define OLEUI_HANDLES_USEINVERSE    0x00000001L
#define OLEUI_HANDLES_NOBORDER      0x00000002L
#define OLEUI_HANDLES_INSIDE        0x00000004L
#define OLEUI_HANDLES_OUTSIDE       0x00000008L


#define OLEUI_SHADE_FULLRECT        1
#define OLEUI_SHADE_BORDERIN        2
#define OLEUI_SHADE_BORDEROUT       3

/* objfdbk.c function prototypes */
```

```
STDAPI_(void) OleUIDrawHandles(LPRECT lpRect, HDC hdc, DWORD dwFlags, UINT
cSize, BOOL fDraw);
STDAPI_(void) OleUIDrawShading(LPRECT lpRect, HDC hdc, DWORD dwFlags, UINT
cWidth);
STDAPI_(void) OleUIShowObject(LPCRECT lprc, HDC hdc, BOOL fIsLink);


/*************************************************************************
** Hatch window definitions and prototypes                              **
*************************************************************************/
#define DEFAULT_HATCHBORDER_WIDTH   4

STDAPI_(BOOL) RegisterHatchWindowClass(HINSTANCE hInst);
STDAPI_(HWND) CreateHatchWindow(HWND hWndParent, HINSTANCE hInst);
STDAPI_(UINT) GetHatchWidth(HWND hWndHatch);
STDAPI_(void) GetHatchRect(HWND hWndHatch, LPRECT lpHatchRect);
STDAPI_(void) SetHatchRect(HWND hWndHatch, LPRECT lprcHatchRect);
STDAPI_(void) SetHatchWindowSize(
                HWND        hWndHatch,
                LPRECT      lprcIPObjRect,
                LPRECT      lprcClipRect,
                LPPOINT     lpptOffset
);



/*************************************************************************
** VERSION VERIFICATION INFORMATION
*************************************************************************/

// The following magic number is used to verify that the resources we bind
// to our EXE are the same "version" as the LIB (or DLL) file which
// contains these routines.  This is not the same as the Version information
// resource that we place in OLE2UI.RC, this is a special ID that we will

// have compiled in to our EXE.  Upon initialization of OLE2UI, we will
// look in our resources for an RCDATA called "VERIFICATION" (see
OLE2UI.RC),
// and make sure that the magic number there equals the magic number below.

#define OLEUI_VERSION_MAGIC 0x4D42

#endif  //_OLE2UI_H_
```

OLE2UI.RC (WRAPUI Sample)

```
/*
 * OLE2UI.RC
 *
 * Icon, menus, strings, and dialogs for the OLE 2.0 UI Support Library.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */

#undef PURE
#include "ole2ui.h"

//Bitmaps for ResultImage control
IDB_RESULTSEGA      BITMAP  egares.bmp
IDB_RESULTSVGA      BITMAP  vgares.bmp
IDB_RESULTSHIRESVGA BITMAP  hivgares.bmp

// Version Verification Resource (see OLE2UI.H)
VERIFICATION RCDATA
     BEGIN
       OLEUI_VERSION_MAGIC
     END

//Include string tables here.
#include "strings.rc"

//Include each dialog template here.
#include "insobj.dlg"
#include "icon.dlg"
#include "links.dlg"
#include "pastespl.dlg"
#include "busy.dlg"
#include "convert.dlg"
#include "fileopen.dlg"
#include "prompt.dlg"

// Only include the version resource if we are compiling the DLL version
#ifdef DLL_VER
//Version Information
#include "RES\OLE2UI.RCV"
#endif
```

OLE2UI.C (WRAPUI Sample)

```
/*
 * OLE2UI.C
 *
 * Contains initialization routines and miscellaneous API implementations
for
 * the OLE 2.0 User Interface Support Library.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */

#define STRICT  1

#include "ole2ui.h"
#include "common.h"
#include "utility.h"
#include "resimage.h"
#include "iconbox.h"
#include <commdlg.h>

#define WINDLL  1           // make far pointer version of stdargs.h
#include <stdarg.h>

// NOTE: If this code is being compiled for a DLL, then we need to define
// our OLE2UI debug symbols here (with the OLEDBGDATA_MAIN macro).  If we're
// compiling for a static LIB, then the application we link to must
// define these symbols -- we just need to make an external reference here
// (with the macro OLEDBGDATA).

#ifdef DLL_VER
OLEDBGDATA_MAIN("OLE2UI")
#else
OLEDBGDATA
#endif

#if defined( _DEBUG ) && !defined( DLL_VER )
// This dummy variable is needed in order for the static link version
// of this library to work correctly.  When we include PRECOMP.OBJ
// in our library (.LIB file), it will only get linked into our
// application IF at least one symbol in precomp.c is referenced from
// either our EXE or LIB.  Therefore, we will define a variable in
// PRECOMP.C and reference it in OLE2UI.C which includes the function
// OleUIInitialize which MUST be called by the static lib user.

extern int near g_nOleUIStaticLibDummy;
int near* g_pnOleUIStaticLibDummy = &g_nOleUIStaticLibDummy;
#endif

//The DLL instance handle shared amongst all dialogs.
HINSTANCE     ghInst;

// object count, used to support DllCanUnloadNow and OleUICanUnloadNow
DWORD g_dwObjectCount=0;
DWORD g_dwLockCount=0;

//Registered messages for use with all the dialogs, registered in LibMain
UINT        uMsgHelp=0;
UINT        uMsgEndDialog=0;
UINT        uMsgBrowse=0;
UINT        uMsgChangeIcon=0;
UINT        uMsgFileOKString=0;
UINT        uMsgCloseBusyDlg=0;

//Clipboard formats used by PasteSpecial
UINT  cfObjectDescriptor;
UINT  cfLinkSrcDescriptor;
UINT  cfEmbedSource;
UINT  cfEmbeddedObject;
UINT  cfLinkSource;
UINT  cfOwnerLink;
UINT  cfFileName;

// local function prototypes
BOOL CALLBACK EXPORT PromptUserDlgProc(HWND hDlg, UINT iMsg, WPARAM wParam,
LPARAM lParam);
BOOL CALLBACK EXPORT UpdateLinksDlgProc(HWND hDlg, UINT iMsg, WPARAM wParam,
LPARAM lParam);


// local definition
#define WM_U_UPDATELINK WM_USER


// local structure definition
typedef struct tagUPDATELINKS
{
     LPOLEUILINKCONTAINER    lpOleUILinkCntr;    // pointer to Link
Container
     UINT                    cLinks;             // total number of links
     UINT                    cUpdated;           // number of links updated
     DWORD                   dwLink;             // pointer to link
     BOOL                    fError;             // error flag
     LPSTR                   lpszTitle;          // caption for dialog box
} UPDATELINKS, *PUPDATELINKS, FAR *LPUPDATELINKS;


/*
 * OleUIInitialize
 *
 * NOTE: This function should only be called by your application IF it is
 * using the static-link version of this library.  If the DLL version is
 * being used, this function is automatically called from the OLE2UI DLL's
 * LibMain.
 *
 * Purpose:
 *   Initializes the OLE UI Library.
 *
 * Parameters:
```

```
 *
 *  hInstance       HINSTANCE of the module where the UI library resources
 *                  and Dialog Procedures are contained.  If you are calling
 *                  this function yourself, this should be the instance
handle
 *                  of your application.
 *
 *  hPrevInst       HINSTANCE of the previous application instance.
 *                  This is the parameter passed in to your WinMain.  For
 *                  the DLL version, this should always be set to zero (for
 *                  WIN16 DLLs).
 *
 * Return Value:
 *  BOOL            TRUE if initialization was successful.
 *                  FALSE if either the "Magic Number" did not verify, or
one of
 *                  the window classes could not be registered.  If the
 *                  "Magic Number" did not verify, then the resources
 *                  in your module are of a different version than the
 *                  ones you compiled with.
 */

STDAPI_(BOOL) OleUIInitialize(HINSTANCE hInstance, HINSTANCE hPrevInst)
{
     HRSRC   hr;
     HGLOBAL hg;
     LPWORD lpdata;

     OleDbgOut1("OleUIInitialize called.\r\n");
     ghInst=hInstance;

     // Verify that we have the correct resources added to our application
     // by checking the "VERIFICATION" resource with the magic number we've
     // compiled into our app.

     if ((hr = FindResource(hInstance, "VERIFICATION", RT_RCDATA)) == NULL)
           goto ResourceLoadError;

     if ((hg = LoadResource(hInstance, hr)) == NULL)
           goto ResourceLoadError;

     if ((lpdata = (LPWORD)LockResource(hg)) == NULL)
           goto ResourceLockError;

     if ((WORD)*lpdata != (WORD)OLEUI_VERSION_MAGIC)
           goto ResourceReadError;

     // OK, resource versions match.  Contine on.
     UnlockResource(hg);
     FreeResource(hg);
     OleDbgOut1("OleUIInitialize: Resource magic number verified.\r\n");

     // Register messages we need for the dialogs.  If
     uMsgHelp      =RegisterWindowMessage(SZOLEUI_MSG_HELP);
     uMsgEndDialog =RegisterWindowMessage(SZOLEUI_MSG_ENDDIALOG);
     uMsgBrowse    =RegisterWindowMessage(SZOLEUI_MSG_BROWSE);
     uMsgChangeIcon=RegisterWindowMessage(SZOLEUI_MSG_CHANGEICON);
     uMsgFileOKString = RegisterWindowMessage(FILEOKSTRING);
     uMsgCloseBusyDlg = RegisterWindowMessage(SZOLEUI_MSG_CLOSEBUSYDIALOG);

     // Register Clipboard Formats used by PasteSpecial dialog.
     cfObjectDescriptor = RegisterClipboardFormat(CF_OBJECTDESCRIPTOR);
     cfLinkSrcDescriptor= RegisterClipboardFormat(CF_LINKSRCDESCRIPTOR);
     cfEmbedSource      = RegisterClipboardFormat(CF_EMBEDSOURCE);
     cfEmbeddedObject   = RegisterClipboardFormat(CF_EMBEDDEDOBJECT);
     cfLinkSource       = RegisterClipboardFormat(CF_LINKSOURCE);
     cfOwnerLink        = RegisterClipboardFormat(CF_OWNERLINK);
     cfFileName         = RegisterClipboardFormat(CF_FILENAME);

     if (!FResultImageInitialize(hInstance, hPrevInst, SZCLASSRESULTIMAGE))
           {
           OleDbgOut1("OleUIInitialize: FResultImageInitialize failed.
Terminating.\r\n");
           return 0;
           }

     if (!FIconBoxInitialize(hInstance, hPrevInst, SZCLASSICONBOX))
           {
           OleDbgOut1("OleUIInitialize: FIconBoxInitialize failed.
Terminating.\r\n");
           return 0;
           }

     return TRUE;

ResourceLoadError:
     OleDbgOut1("OleUIInitialize: ERROR - Unable to find version
verification resource.\r\n");
     return FALSE;

ResourceLockError:
     OleDbgOut1("OleUIInitialize: ERROR - Unable to lock version
verification resource.\r\n");
     FreeResource(hg);
     return FALSE;

ResourceReadError:
     OleDbgOut1("OleUIInitialize: ERROR - Version verification values did
not compare.\r\n");

     {char buf[255];
     wsprintf(buf, "resource read 0x%X, my value is 0x%X\n", (WORD)*lpdata,
(WORD)OLEUI_VERSION_MAGIC);
     OutputDebugString(buf);
     }

     UnlockResource(hg);
     FreeResource(hg);
     return FALSE;
```

```
}


/*
 * OleUIUnInitialize
 *
 * NOTE: This function should only be called by your application IF it is
using
 * the static-link version of this library.  If the DLL version is being
used,
 * this function is automatically called from the DLL's LibMain.
 *
 * Purpose:
 *   Uninitializes OLE UI libraries.  Deletes any resources allocated by the
 *   library.
 *
 * Return Value:
 *   BOOL       TRUE if successful, FALSE if not.  Current implementation
always
 *              returns TRUE.
 */


STDAPI_(BOOL) OleUIUnInitialize()
{
     IconBoxUninitialize();
     ResultImageUninitialize();


     return TRUE;
}


/*
 * OleUICanUnloadNow
 *
 * NOTE: if you statically link to this library and you implement
 *       DllCanUnloadNow, then you must call this routine in your
 *       implementation of DllCanUnloadNow to determine if you can be
 *       unloaded or not.
 *       this function should NOT be used if the OLE2UI library is used
 *       in DLL form.
 *
 * Purpose:
 *   Determines when it is safe to go away
 *   (ie. there are no existing object instances).
 *
 * Return Value:
 *   HRESULT    NOERROR it is safe to go away, S_FALSE this code must stay
 *              loaded.
 *
 * Comments:
 *
 *   If an INPROC server DLL links to the OLE2UI library as a static
 *   library, then the OleUILockLibrary function should NOT be used.
 *   instead the INPROC server DLL should call OleUICanUnloadNow API from
 *   within its own DllCanUnloadNow function. The idea here is, that if
there
 *   are any existing instance of objects created by the OLE2UI library
 *   functions (eg. EnumFORMATETC objects created by
OleStdEnumFmtEtc_Create)
 *   then, the INPROC server DLL must NOT let itself be unloaded.
 *
 *   An EXE based object using the OLE2UI libray need NOT use either the
 *   OleUILockLibrary or OleUICanUnloadNow functions. All objects created
 *   by the OLE2UI library will have LRPC proxies and stubs created to
 *   manage remoting method calls. the worst that can happen when the EXE
 *   exits is that any outstanding proxies for unreleased objects will get
 *   RPC_E_SERVERDIED errors; they will not GPFault.
 */

STDAPI OleUICanUnloadNow()
{
  if (g_dwObjectCount == 0) {
     return NOERROR;
  }

  return ResultFromScode(S_FALSE);
}


/*
 * OleUIAddVerbMenu
 *
 * Purpose:
 *  Add the Verb menu for the specified object to the given menu.  If the
 *  object has one verb, we directly add the verb to the given menu.  If
 *  the object has multiple verbs we create a cascading sub-menu.
 *
 * Parameters:
 *  lpObj           LPOLEOBJECT pointing to the selected object.  If this
 *                  is NULL, then we create a default disabled menu item.
 *
 *  lpszShortType   LPSTR with short type name (AuxName==2) corresponding
 *                  to the lpOleObj. if the string is NOT known, then NULL
 *                  may be passed. if NULL is passed, then
 *                  IOleObject::GetUserType will be called to retrieve it.
 *                  if the caller has the string handy, then it is faster
 *                  to pass it in.
 *
 *  hMenu           HMENU in which to make modifications.
 *
 *  uPos            Position of the menu item
 *
 *  uIDVerbMin      UINT ID value at which to start the verbs.
 *                      verb_0 = wIDMVerbMin + verb_0
 *                      verb_1 = wIDMVerbMin + verb_1
 *                      verb_2 = wIDMVerbMin + verb_2
 *                      etc.
 *  uIDVerbMax      UINT maximum ID value allowed for object verbs.
 *                     if uIDVerbMax==0 then any ID value is allowed
 *
 *  bAddConvert     BOOL specifying whether or not to add a "Convert" item
 *                  to the bottom of the menu (with a separator).
 *
 *  idConvert       UINT ID value to use for the Convert menu item, if
 *                  bAddConvert is TRUE.
 *
 *  lphMenu         HMENU FAR * of the cascading verb menu if it's created.
 *                  If there is only one verb, this will be filled with
NULL.
 *
 *
 * Return Value:
 *  BOOL            TRUE if lpObj was valid and we added at least one verb
 *                  to the menu.  FALSE if lpObj was NULL and we created
 *                  a disabled default menu item
 */

STDAPI_(BOOL) OleUIAddVerbMenu(LPOLEOBJECT lpOleObj,
                                          LPSTR lpszShortType,
                                          HMENU hMenu,
                                          UINT uPos,
                                          UINT uIDVerbMin,
                                          UINT uIDVerbMax,
                                          BOOL bAddConvert,
                                          UINT idConvert,
                                          HMENU FAR *lphMenu)
{
     LPPERSISTSTORAGE    lpPS=NULL;
     LPENUMOLEVERB       lpEnumOleVerb = NULL;
     OLEVERB             oleverb;
     LPUNKNOWN           lpUnk;

     LPSTR               lpszShortTypeName = lpszShortType;
     LPSTR               lpszVerbName = NULL;
     HRESULT             hrErr;
     BOOL                fStatus;
     BOOL                fIsLink = FALSE;
     BOOL                fResult = TRUE;
     BOOL                fAddConvertItem = FALSE;
     int                 cVerbs = 0;
     UINT                uFlags = MF_BYPOSITION;
     static BOOL         fFirstTime = TRUE;
     static char         szBuffer[OLEUI_OBJECTMENUMAX];
     static char         szNoObjectCmd[OLEUI_OBJECTMENUMAX];
     static char         szObjectCmd1Verb[OLEUI_OBJECTMENUMAX];
     static char         szLinkCmd1Verb[OLEUI_OBJECTMENUMAX];
     static char         szObjectCmdNVerb[OLEUI_OBJECTMENUMAX];
     static char         szLinkCmdNVerb[OLEUI_OBJECTMENUMAX];
     static char         szUnknown[OLEUI_OBJECTMENUMAX];
     static char         szEdit[OLEUI_OBJECTMENUMAX];
     static char         szConvert[OLEUI_OBJECTMENUMAX];

     *lphMenu=NULL;
     // Set fAddConvertItem flag
     if (bAddConvert & (idConvert != 0))
        fAddConvertItem = TRUE;

     // only need to load the strings the 1st time
     if (fFirstTime) {
           if (0 == LoadString(ghInst, (UINT)IDS_OLE2UIEDITNOOBJCMD,
                        (LPSTR)szNoObjectCmd, OLEUI_OBJECTMENUMAX))
                 return FALSE;
           if (0 == LoadString(ghInst, (UINT)IDS_OLE2UIEDITLINKCMD_1VERB,
                        (LPSTR)szLinkCmd1Verb, OLEUI_OBJECTMENUMAX))
                 return FALSE;
           if (0 == LoadString(ghInst, (UINT)IDS_OLE2UIEDITOBJECTCMD_1VERB,
                        (LPSTR)szObjectCmd1Verb, OLEUI_OBJECTMENUMAX))
                 return FALSE;

           if (0 == LoadString(ghInst, (UINT)IDS_OLE2UIEDITLINKCMD_NVERB,
                        (LPSTR)szLinkCmdNVerb, OLEUI_OBJECTMENUMAX))
                 return FALSE;
           if (0 == LoadString(ghInst, (UINT)IDS_OLE2UIEDITOBJECTCMD_NVERB,
                        (LPSTR)szObjectCmdNVerb, OLEUI_OBJECTMENUMAX))
                 return FALSE;

           if (0 == LoadString(ghInst, (UINT)IDS_OLE2UIUNKNOWN,
                        (LPSTR)szUnknown, OLEUI_OBJECTMENUMAX))
                 return FALSE;

           if (0 == LoadString(ghInst, (UINT)IDS_OLE2UIEDIT,
                        (LPSTR)szEdit, OLEUI_OBJECTMENUMAX))
                 return FALSE;

           if ( (0 == LoadString(ghInst, (UINT)IDS_OLE2UICONVERT,
                          (LPSTR)szConvert, OLEUI_OBJECTMENUMAX)) &&
fAddConvertItem)
                 return FALSE;

           fFirstTime = FALSE;
     }

     // Delete whatever menu may happen to be here already.
     DeleteMenu(hMenu, uPos, uFlags);

     if (!lpOleObj)
           goto AVMError;

     if (! lpszShortTypeName) {
           // get the Short form of the user type name for the menu
           OLEDBG_BEGIN2("IOleObject::GetUserType called\r\n")
           hrErr = lpOleObj->lpVtbl->GetUserType(
                       lpOleObj,
                       USERCLASSTYPE_SHORT,
                       (LPSTR FAR*)&lpszShortTypeName
           );
           OLEDBG_END2
           if (NOERROR != hrErr) {
                 OleDbgOutHResult("IOleObject::GetUserType returned",
hrErr);
           }
     }

     // check if the object is a link (it is a link if it support IOleLink)
     hrErr = lpOleObj->lpVtbl->QueryInterface(
                 lpOleObj,
                 &IID_IOleLink,
                 (LPVOID FAR*)&lpUnk
     );
     if (NOERROR == hrErr) {
           fIsLink = TRUE;
           OleStdRelease(lpUnk);
     }

     // Get the verb enumerator from the OLE object
     OLEDBG_BEGIN2("IOleObject::EnumVerbs called\r\n")
     hrErr = lpOleObj->lpVtbl->EnumVerbs(
                 lpOleObj,
                 (LPENUMOLEVERB FAR*)&lpEnumOleVerb
     );
     OLEDBG_END2

     if (NOERROR != hrErr) {
           OleDbgOutHResult("IOleObject::EnumVerbs returned", hrErr);
     }

     if (!(*lphMenu = CreatePopupMenu()))
           goto AVMError;

     // loop through all verbs
     while (lpEnumOleVerb != NULL) {         // forever
           hrErr = lpEnumOleVerb->lpVtbl->Next(
                       lpEnumOleVerb,
                       1,
                       (LPOLEVERB)&oleverb,
                       NULL
           );
           if (NOERROR != hrErr)
                 break;              // DONE! no more verbs

           /* OLE2NOTE: negative verb numbers and verbs that do not
           **    indicate ONCONTAINERMENU should NOT be put on the verb menu
           */
           if (oleverb.lVerb < 0 ||
                       ! (oleverb.grfAttribs &
OLEVERBATTRIB_ONCONTAINERMENU)) {

                 /* OLE2NOTE: we must still free the verb name string */
                 if (oleverb.lpszVerbName)
                       OleStdFreeString(oleverb.lpszVerbName, NULL);
                 continue;
```

```
        }

        // we must free the previous verb name string
        if (lpszVerbName)
            OleStdFreeString(lpszVerbName, NULL);

        lpszVerbName = oleverb.lpszVerbName;

        if ( 0 == uIDVerbMax ||
            (uIDVerbMax >= uIDVerbMin+(UINT)oleverb.lVerb) ) {
            fStatus = InsertMenu(
                    *lphMenu,
                    (UINT)-1,
                    MF_BYPOSITION | (UINT)oleverb.fuFlags,
                    uIDVerbMin+(UINT)oleverb.lVerb,
                    (LPSTR)lpszVerbName
            );
            if (! fStatus)
                goto AVMError;

            cVerbs++;
        }
    }

    // Add the separator and "Convert" menu item.
    if (fAddConvertItem) {

        if (0 == cVerbs) {
            LPSTR lpsz;
            LPSTR lpszSrc = (LPSTR)szConvert;

            // if object has no verbs, then use "Convert" as the obj's
verb
            lpsz = lpszVerbName = OleStdMalloc(sizeof(szConvert));
            uIDVerbMin = idConvert;

            // remove "&" and "..." from "&Convert..." string
            if (lpsz) {
                while (*lpszSrc && *lpszSrc != '.') {
                    if (*lpszSrc != '&') {
                        *lpsz = *lpszSrc;
                        if (IsDBCSLeadByte(*lpszSrc)) {
                            // copy second byte of DBCS
character
                            lpsz++;
                            *lpsz = *(lpszSrc+1);
                        }
                        lpsz++;
                    }
                    lpszSrc = AnsiNext(lpszSrc);
                }
                *lpsz = '\0';
            }
        }
```

```
            if (cVerbs > 0) {
                  fStatus = InsertMenu(*lphMenu,
                                    (UINT)-1,
                                    MF_BYPOSITION | MF_SEPARATOR,
                                    (UINT)0,
                                    (LPCSTR)NULL);
                  if (! fStatus)
                        goto AVMError;
            }

            /* add convert menu */
            fStatus = InsertMenu(*lphMenu,
                              (UINT)-1,
                              MF_BYPOSITION,
                              idConvert,
                              (LPCSTR)szConvert);
            if (! fStatus)
                  goto AVMError;

            cVerbs++;
      }


      /*
       * Build the appropriate menu based on the number of verbs found
       *
       * NOTE:  Localized verb menus may require a different format.
       *        to assist in localization of the single verb case, the
       *        szLinkCmd1Verb and szObjectCmd1Verb format strings start
       *        with either a '0' (note: NOT '\0'!) or a '1':
       *           leading '0' -- verb type
       *           leading '1' -- type verb
       */

      if (cVerbs == 0) {

            // there are NO verbs (not even Convert...). set the menu to be
            // "<short type> &Object/Link" and gray it out.
            wsprintf(
                  szBuffer,
                  (fIsLink ? (LPSTR)szLinkCmdNVerb:(LPSTR)szObjectCmdNVerb),
                  (lpszShortTypeName ? lpszShortTypeName : (LPSTR)"")
            );
            uFlags |= MF_GRAYED;

#if defined( OBSOLETE )
            //No verbs.  Create a default using Edit as the verb.
            LPSTR       lpsz = (fIsLink ? szLinkCmd1Verb : szObjectCmd1Verb);

            if (*lpsz == '0') {
                  wsprintf(szBuffer, lpsz+1, (LPSTR)szEdit,
                        (lpszShortTypeName ? lpszShortTypeName : (LPSTR)"")
                  );
            }
            else {
                  wsprintf(szBuffer, lpsz+1,
                        (lpszShortTypeName ? lpszShortTypeName : (LPSTR)""),
                        (LPSTR)szEdit
                  );
            }
#endif

            fResult = FALSE;
            DestroyMenu(*lphMenu);
            *lphMenu = NULL;

      }
      else if (cVerbs == 1) {
            //One verb without Convert, one item.
            LPSTR       lpsz = (fIsLink ? szLinkCmd1Verb : szObjectCmd1Verb);
            LPSTR       lpsz1 = lpszVerbName;

            // remove "&" from the verb name string. the &Object or &Link
            // should be the menu accelerator.
            // if only "verb" is "Convert" then the "&" has already been
stripped
            if (lpsz1 && !fAddConvertItem) {
             LPSTR lpszVerbName2 = OleStdMalloc(lstrlen(lpszVerbName)+1);
             LPSTR lpsz2 = lpszVerbName2;
             if (lpsz2) {
                 while (*lpsz1) {
                     if (*lpsz1 != '&') {

                         *lpsz2 = *lpsz1;
                         if (IsDBCSLeadByte(*lpsz1)) {
                         // copy second byte of DBCS character
                             lpsz2++;
                             *lpsz2 = *(lpsz1+1);
                         }
                         lpsz2++;
                     }
                     lpsz1 = AnsiNext(lpsz1);
                 }
                 *lpsz2 = '\0';
                 OleStdFreeString(lpszVerbName, NULL);
                 lpszVerbName = lpszVerbName2;
             }
        }

            if (*lpsz == '0') {
                  wsprintf(szBuffer, lpsz+1, lpszVerbName,
                        (lpszShortTypeName ? lpszShortTypeName : (LPSTR)"")
                  );
            }
            else {
                  wsprintf(szBuffer, lpsz+1,
                        (lpszShortTypeName ? lpszShortTypeName : (LPSTR)""),
                        lpszVerbName
                  );
            }

            // if only "verb" is "Convert..." then append the ellipses
            if (fAddConvertItem)
                  lstrcat(szBuffer, "...");

            DestroyMenu(*lphMenu);
            *lphMenu=NULL;
      }
      else {

            //Multiple verbs or one verb with Convert, add the cascading menu
            wsprintf(
                  szBuffer,
                  (fIsLink ? (LPSTR)szLinkCmdNVerb:(LPSTR)szObjectCmdNVerb),
                  (lpszShortTypeName ? lpszShortTypeName : (LPSTR)"")
            );
            uFlags |= MF_ENABLED | MF_POPUP;
            uIDVerbMin=(UINT)*lphMenu;
      }

      if (!InsertMenu(hMenu, uPos, uFlags, uIDVerbMin, (LPSTR)szBuffer))

AVMError:
            {
                  InsertMenu(hMenu, uPos, MF_GRAYED | uFlags,
                              uIDVerbMin, (LPSTR)szNoObjectCmd);
#if defined( OBSOLETE )
                  HMENU hmenuDummy = CreatePopupMenu();

                  InsertMenu(hMenu, uPos, MF_GRAYED | MF_POPUP | uFlags,
                              (UINT)hmenuDummy, (LPSTR)szNoObjectCmd);
#endif
                  fResult = FALSE;
            }

      if (lpszVerbName)
            OleStdFreeString(lpszVerbName, NULL);
      if (!lpszShortType && lpszShortTypeName)
            OleStdFreeString(lpszShortTypeName, NULL);
      if (lpEnumOleVerb)
            lpEnumOleVerb->lpVtbl->Release(lpEnumOleVerb);
      return fResult;
}


/* PromptUserDlgProc
 * -----------------
 *
 *  Purpose:
 *      Dialog procedure used by OleUIPromptUser(). Returns when a button is
 *      clicked in the dialog box and the button id is return.
 *
 *  Parameters:
 *      hDlg
 *      iMsg
 *      wParam
 *      lParam
 *
 *  Returns:
 *
 */
BOOL CALLBACK EXPORT PromptUserDlgProc(HWND hDlg, UINT iMsg, WPARAM wParam,
LPARAM lParam)
{
     switch (iMsg) {
           case WM_INITDIALOG:
           {
                 LPSTR lpszTitle;
                 char szBuf[256];
                 char szFormat[256];
                 va_list *pargptr;

                 if (!lParam) {
                       EndDialog(hDlg, -1);
                       return FALSE;
                 }

            //
            //  lParam is really a va_list *.  We called va_start and va_end
in
            //  the function that calls this.
            //

            pargptr = (va_list *) lParam;

                 lpszTitle = va_arg(*pargptr, LPSTR);
                 SetWindowText(hDlg, lpszTitle);

                 GetDlgItemText(hDlg, ID_PU_TEXT,
(LPSTR)szFormat,sizeof(szFormat));
                 wvsprintf((LPSTR)szBuf, (LPSTR)szFormat, *pargptr);
                 SetDlgItemText(hDlg, ID_PU_TEXT, (LPSTR)szBuf);
                 return TRUE;
           }
           case WM_COMMAND:
                 EndDialog(hDlg, wParam);
                 return TRUE;

           default:
                 return FALSE;
     }
}


/* OleUIPromptUser
 * ---------------
 *
 *  Purpose:
 *      Popup a dialog box with the specified template and returned the
 *      response (button id) from the user.
 *
 *  Parameters:
 *      nTemplate       resource number of the dialog
 *      hwndParent      parent of the dialog box
 *      ...             title of the dialog box followed by argument list
 *                      for the format string in the static control
 *                      (ID_PU_TEXT) of the dialog box.
 *                      The caller has to make sure that the correct number
 *                      and type of argument are passed in.
 *
 *  Returns:
 *      button id selected by the user (template dependent)
 *
 *  Comments:
 *      the following message dialog boxes are supported:
 *
 *      IDD_LINKSOURCEUNAVAILABLE -- Link source is unavailable
 *          VARARG Parameters:
 *              None.
 *          Used for the following error codes:
 *              OLE_E_CANT_BINDTOSOURCE
 *              STG_E_PATHNOTFOUND
 *              (sc >= MK_E_FIRST) && (sc <= MK_E_LAST) -- any Moniker error
 *              any unknown error if object is a link
 *
 *      IDD_SERVERNOTFOUND -- server registered but NOT found
 *          VARARG Parameters:
 *              LPSTR lpszUserType -- user type name of object
 *          Used for the following error codes:
 *              CO_E_APPNOTFOUND
 *              CO_E_APPDIDNTREG
 *              any unknown error if object is an embedded object
 *
 *      IDD_SERVERNOTREG -- server NOT registered
 *          VARARG Parameters:
 *              LPSTR lpszUserType -- user type name of object
 *          Used for the following error codes:
 *              REGDB_E_CLASSNOTREG
 *              OLE_E_STATIC -- static object with no server registered
 *
 *      IDD_LINKTYPECHANGED -- class of link source changed since last
binding
 *          VARARG Parameters:
 *              LPSTR lpszUserType -- user type name of ole link source
 *          Used for the following error codes:
 *              OLE_E_CLASSDIFF
 *
 *      IDD_LINKTYPECHANGED -- class of link source changed since last
binding
 *          VARARG Parameters:
 *              LPSTR lpszUserType -- user type name of ole link source
 *          Used for the following error codes:
 *              OLE_E_CLASSDIFF
 *
 *      IDD_OUTOFMEMORY -- out of memory
 *          VARARG Parameters:
 *              None.
 *          Used for the following error codes:
 *              E_OUTOFMEMORY
 *
 */
int EXPORT FAR CDECL OleUIPromptUser(WORD nTemplate, HWND hwndParent, ...)
{
    int         nRet;
    va_list     arglist;
    LPARAM      lParam;

    //
    //  We want to pass the variable list of arguments to PrompUserDlgProc,
    //  but we can't just pass arglist because arglist is not always the
    //  same size as an LPARAM (e.g. on Alpha va_list is a structure).
    //  So, we pass the a pointer to the arglist instead.
    //

    va_start(arglist, hwndParent);
    lParam = (LPARAM) &arglist;

    nRet = DialogBoxParam(ghInst, MAKEINTRESOURCE(nTemplate), hwndParent,
            PromptUserDlgProc, lParam);

    va_end(arglist);

    return nRet;
}



/* UpdateLinksDlgProc
 * ------------------
 *
 *  Purpose:
 *      Dialog procedure used by OleUIUpdateLinks(). It will enumerate all
 *      all links in the container and updates all automatic links.
 *      Returns when the Stop Button is clicked in the dialog box or when
all
 *      links are updated
 *
 *  Parameters:
 *      hDlg
 *      iMsg
 *      wParam
 *      lParam          pointer to the UPDATELINKS structure
 *
 *  Returns:
 *
 */
BOOL CALLBACK EXPORT UpdateLinksDlgProc(HWND hDlg, UINT iMsg, WPARAM wParam,
LPARAM lParam)
```

```
{
     LPUPDATELINKS FAR*      lplpUL = NULL;
     HANDLE                  gh;
     static BOOL             fAbort = FALSE;

     //Process the temination message
     if (iMsg==uMsgEndDialog)
           {
           gh = RemoveProp(hDlg, STRUCTUREPROP);
           if (NULL!=gh) {
                 GlobalUnlock(gh);
                 GlobalFree(gh);
           }
           EndDialog(hDlg, wParam);
           return TRUE;
           }

     switch (iMsg) {

           case WM_INITDIALOG:
           {
                 gh=GlobalAlloc(GMEM_MOVEABLE|
GMEM_ZEROINIT,sizeof(LPUPDATELINKS));
                 SetProp(hDlg, STRUCTUREPROP, gh);

                 if (NULL==gh)
                 {
                       PostMessage(hDlg, uMsgEndDialog,
OLEUI_ERR_GLOBALMEMALLOC,0L);
                       return FALSE;
                 }

                 fAbort = FALSE;
                 lplpUL = (LPUPDATELINKS FAR*)GlobalLock(gh);

                 if (lplpUL) {
                       *lplpUL = (LPUPDATELINKS)lParam;
                       SetWindowText(hDlg, (*lplpUL)->lpszTitle);
                       SetTimer(hDlg, 1, UPDATELINKS_STARTDELAY, NULL);
                       return TRUE;
                 } else {
                       PostMessage(hDlg, uMsgEndDialog,
OLEUI_ERR_GLOBALMEMALLOC,0L);
                       return FALSE;
                 }
           }

           case WM_TIMER:
                 KillTimer(hDlg, 1);
                 gh = GetProp(hDlg, STRUCTUREPROP);

                 if (NULL!=gh) {
                       // gh was locked previously, lock and unlock to get
lplpUL
                       lplpUL = GlobalLock(gh);
```

```
                        GlobalUnlock(gh);
                  }
                  if (! fAbort && lplpUL)
                        PostMessage(hDlg, WM_U_UPDATELINK, 0, (LPARAM)
(*lplpUL));
                  else
                        PostMessage(hDlg,uMsgEndDialog,OLEUI_CANCEL,0L);

                  return 0;

            case WM_COMMAND:    // Stop button
                  fAbort = TRUE;
                  SendMessage(hDlg, uMsgEndDialog, OLEUI_OK, 0L);
                  return TRUE;

            case WM_U_UPDATELINK:
            {
                  HRESULT         hErr;
                  int             nPercent;
                  RECT            rc;
                  char            szPercent[5];       // 0% to 100%
                  HBRUSH          hbr;
                  HDC             hDC;
                  HWND            hwndMeter;
                  MSG             msg;
                  DWORD           dwUpdateOpt;
                  LPUPDATELINKS   lpUL = (LPUPDATELINKS)lParam;

                  lpUL->dwLink=lpUL->lpOleUILinkCntr->lpVtbl->GetNextLink(
                              lpUL->lpOleUILinkCntr,
                              lpUL->dwLink
                  );

                  while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
                        if (! IsDialogMessage(hDlg, &msg)) {
                              TranslateMessage(&msg);
                              DispatchMessage(&msg);
                        }
                  }

                  if (fAbort)
                        return FALSE;

                  if (!lpUL->dwLink) {        // all links processed
                        SendMessage(hDlg, uMsgEndDialog, OLEUI_OK, 0L);
                        return TRUE;
                  }

                  hErr = lpUL->lpOleUILinkCntr->lpVtbl->GetLinkUpdateOptions(
                              lpUL->lpOleUILinkCntr,
                              lpUL->dwLink,
                              (LPDWORD)&dwUpdateOpt
                  );
```

```
                 if ((hErr == NOERROR) && (dwUpdateOpt == OLEUPDATE_ALWAYS))
{

                       hErr = lpUL->lpOleUILinkCntr->lpVtbl->UpdateLink(
                                   lpUL->lpOleUILinkCntr,
                                   lpUL->dwLink,
                                   FALSE,      // fMessage
                                   FALSE       // ignored
                       );
                       lpUL->fError |= (hErr != NOERROR);
                       lpUL->cUpdated++;

                       nPercent = lpUL->cUpdated * 100 / lpUL->cLinks;
                       if (nPercent <= 100) {  // do NOT advance % beyond
100%
                             // update percentage
                             wsprintf((LPSTR)szPercent, "%d%%", nPercent);
                             SetDlgItemText(hDlg, ID_PU_PERCENT,
(LPSTR)szPercent);

                             // update indicator
                             hwndMeter = GetDlgItem(hDlg, ID_PU_METER);
                             GetClientRect(hwndMeter, (LPRECT)&rc);
                             InflateRect((LPRECT)&rc, -1, -1);
                             rc.right = (rc.right - rc.left) * nPercent /
100 + rc.left;
                             hbr =
CreateSolidBrush(GetSysColor(COLOR_HIGHLIGHT));
                             if (hbr) {
                                   hDC = GetDC(hwndMeter);
                                   if (hDC) {
                                         FillRect(hDC, (LPRECT)&rc, hbr);
                                         ReleaseDC(hwndMeter, hDC);
                                   }
                                   DeleteObject(hbr);
                             }
                       }
                 }

                 while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
                       if (! IsDialogMessage(hDlg, &msg)) {
                             TranslateMessage(&msg);
                             DispatchMessage(&msg);
                       }
                 }

                 PostMessage(hDlg, WM_U_UPDATELINK, 0, lParam);

                 return TRUE;
           }

           default:
                 return FALSE;
     }
```

```
}


/* OleUIUpdateLink
 * ---------------
 *
 *  Purpose:
 *      Update all links in the Link Container and popup a dialog box which
 *      shows the progress of the updating.
 *      The process is stopped when the user press Stop button or when all
 *      links are processed.
 *
 *  Parameters:
 *      lpOleUILinkCntr         pointer to Link Container
 *      hwndParent              parent window of the dialog
 *      lpszTitle               title of the dialog box
 *      cLinks                  total number of links
 *
 *  Returns:
 *      TRUE                    all links updated successfully
 *      FALSE                   otherwise
 */
STDAPI_(BOOL) OleUIUpdateLinks(LPOLEUILINKCONTAINER lpOleUILinkCntr, HWND
hwndParent, LPSTR lpszTitle, int cLinks)
{
     LPUPDATELINKS lpUL = (LPUPDATELINKS)OleStdMalloc(sizeof(UPDATELINKS));
     BOOL          fError;

     OleDbgAssert(lpOleUILinkCntr && hwndParent && lpszTitle && (cLinks>0));
     OleDbgAssert(lpUL);

     lpUL->lpOleUILinkCntr = lpOleUILinkCntr;
     lpUL->cLinks           = cLinks;
     lpUL->cUpdated         = 0;
     lpUL->dwLink           = 0;
     lpUL->fError           = FALSE;
     lpUL->lpszTitle    = lpszTitle;

     DialogBoxParam(ghInst, MAKEINTRESOURCE(IDD_UPDATELINKS),
                hwndParent, UpdateLinksDlgProc, (LPARAM)lpUL);

     fError = lpUL->fError;
     OleStdFree((LPVOID)lpUL);

     return !fError;
}
```

OLESTD.H   (WRAPUI Sample)

```
/*************************************************************************
**
**    OLE 2.0 Standard Utilities
**
**    olestd.h
**
**    This file contains file contains data structure defintions,
**    function prototypes, constants, etc. for the common OLE 2.0
**    utilities.
**    These utilities include the following:
**          Debuging Assert/Verify macros
**          HIMETRIC conversion routines
**          reference counting debug support
**          OleStd API's for common compound-document app support
**
**    (c) Copyright Microsoft Corp. 1990 - 1992 All Rights Reserved
**
*************************************************************************/

#if !defined( _OLESTD_H_ )
#define _OLESTD_H_

#ifndef RC_INVOKED
#pragma message ("INCLUDING OLESTD.H from " __FILE__)
#endif  /* RC_INVOKED */

#if defined( __TURBOC__ ) || defined( WIN32 )
#define _based(a)
#endif

#ifndef RC_INVOKED
#include <dos.h>        // needed for filetime
#endif  /* RC_INVOKED */

#include <commdlg.h>    // needed for LPPRINTDLG
#include <shellapi.h>   // needed for HKEY

// String table defines...
#define  IDS_OLESTDNOCREATEFILE   700
#define  IDS_OLESTDNOOPENFILE     701
#define  IDS_OLESTDDISKFULL       702


/*
 * Some C interface declaration stuff
 */

#if ! defined(__cplusplus)
typedef struct tagINTERFACEIMPL {
           IUnknownVtbl FAR*       lpVtbl;
           LPVOID                  lpBack;
           int                     cRef;   // interface specific ref count.
} INTERFACEIMPL, FAR* LPINTERFACEIMPL;

#define INIT_INTERFACEIMPL(lpIFace, pVtbl, pBack)    \
           ((lpIFace)->lpVtbl = pVtbl, \
                 ((LPINTERFACEIMPL)(lpIFace))->lpBack = (LPVOID)pBack,    \
                 ((LPINTERFACEIMPL)(lpIFace))->cRef = 0  \
           )

#if defined( _DEBUG )
#define OleDbgQueryInterfaceMethod(lpUnk)    \
           ((lpUnk) != NULL ? ((LPINTERFACEIMPL)(lpUnk))->cRef++ : 0)
#define OleDbgAddRefMethod(lpThis, iface)    \
           ((LPINTERFACEIMPL)(lpThis))->cRef++

#if _DEBUGLEVEL >= 2
#define OleDbgReleaseMethod(lpThis, iface) \
           (--((LPINTERFACEIMPL)(lpThis))->cRef == 0 ? \
                 OleDbgOut("\t" iface "* RELEASED (cRef == 0)\r\n"),1 : \
                  (((LPINTERFACEIMPL)(lpThis))->cRef < 0) ? \
                       ( \
                             DebugBreak(), \
                             OleDbgOut(  \
                                   "\tERROR: " iface "* RELEASED TOO MANY
TIMES\r\n") \
                       ),1 : \
                       1)

#else       // if _DEBUGLEVEL < 2
#define OleDbgReleaseMethod(lpThis, iface) \
           (--((LPINTERFACEIMPL)(lpThis))->cRef == 0 ? \
                 1 : \
                  (((LPINTERFACEIMPL)(lpThis))->cRef < 0) ? \
                       ( \
                             OleDbgOut(  \
                                   "\tERROR: " iface "* RELEASED TOO MANY
TIMES\r\n") \
           ),1 : \
                       1)

#endif      // if _DEBUGLEVEL < 2

#else       // ! defined (_DEBUG)

#define OleDbgQueryInterfaceMethod(lpUnk)
#define OleDbgAddRefMethod(lpThis, iface)
#define OleDbgReleaseMethod(lpThis, iface)

#endif      // if defined( _DEBUG )

#endif      // ! defined(__cplusplus)

/*
 * Some docfiles stuff
 */
```

```
#define STGM_DFRALL (STGM_READWRITE | STGM_TRANSACTED |
STGM_SHARE_DENY_WRITE)
#define STGM_DFALL (STGM_READWRITE | STGM_TRANSACTED | STGM_SHARE_EXCLUSIVE)
#define STGM_SALL (STGM_READWRITE | STGM_SHARE_EXCLUSIVE)

/*
 * Some moniker stuff
 */

// Delimeter used to separate ItemMoniker pieces of a composite moniker
#if defined( _MAC )
#define OLESTDDELIM ":"
#else
#define OLESTDDELIM "\\"
#endif

/*
 * Some Concurrency stuff
 */

/* standard Delay (in msec) to wait before retrying an LRPC call.
**    this value is returned from IMessageFilter::RetryRejectedCall
*/
#define OLESTDRETRYDELAY    (DWORD)5000

/* Cancel the pending outgoing LRPC call.
**    this value is returned from IMessageFilter::RetryRejectedCall
*/
#define OLESTDCANCELRETRY   (DWORD)-1

/*
 * Some Icon support stuff.
 *
 * The following API's are now OBSOLETE because equivalent API's have been
 * added to the OLE2.DLL library
 *      GetIconOfFile       superceeded by OleGetIconOfFile
 *      GetIconOfClass      superceeded by OleGetIconOfClass
 *      OleUIMetafilePictFromIconAndLabel
 *                          superceeded by OleMetafilePictFromIconAndLabel
 *
 * The following macros are defined for backward compatibility with previous
 * versions of the OLE2UI library. It is recommended that the new Ole* API's
 * should be used instead.
 */
#define GetIconOfFile(hInst, lpszFileName, fUseFileAsLabel) \
     OleGetIconOfFile(lpszFileName, fUseFileAsLabel)

#define GetIconOfClass(hInst, rclsid, lpszLabel, fUseTypeAsLabel) \
     OleGetIconOfClass(rclsid, lpszLabel, fUseTypeAsLabel)

#define
OleUIMetafilePictFromIconAndLabel(hIcon,pszLabel,pszSourceFile,iIcon)\
     OleMetafilePictFromIconAndLabel(hIcon, pszLabel, pszSourceFile, iIcon)
```

```
/*
 * Some Clipboard Copy/Paste & Drag/Drop support stuff
 */

//Macro to set all FormatEtc fields
#define SETFORMATETC(fe, cf, asp, td, med, li)   \
     ((fe).cfFormat=cf, \
      (fe).dwAspect=asp, \
      (fe).ptd=td, \
      (fe).tymed=med, \
      (fe).lindex=li)

//Macro to set interesting FormatEtc fields defaulting the others.
#define SETDEFAULTFORMATETC(fe, cf, med)  \
     ((fe).cfFormat=cf, \
      (fe).dwAspect=DVASPECT_CONTENT, \
      (fe).ptd=NULL, \
      (fe).tymed=med, \
      (fe).lindex=-1)

// Macro to test if two FormatEtc structures are an exact match
#define IsEqualFORMATETC(fe1, fe2)  \
     (OleStdCompareFormatEtc(&(fe1), &(fe2))==0)

// Clipboard format strings
#define CF_EMBEDSOURCE      "Embed Source"
#define CF_EMBEDDEDOBJECT   "Embedded Object"
#define CF_LINKSOURCE       "Link Source"
#define CF_CUSTOMLINKSOURCE "Custom Link Source"
#define CF_OBJECTDESCRIPTOR "Object Descriptor"
#define CF_LINKSRCDESCRIPTOR "Link Source Descriptor"
#define CF_OWNERLINK        "OwnerLink"
#define CF_FILENAME         "FileName"

#define OleStdQueryOleObjectData(lpformatetc)   \
     (((lpformatetc)->tymed & TYMED_ISTORAGE) ?    \
                 NOERROR : ResultFromScode(DV_E_FORMATETC))

#define OleStdQueryLinkSourceData(lpformatetc)   \
     (((lpformatetc)->tymed & TYMED_ISTREAM) ?    \
                 NOERROR : ResultFromScode(DV_E_FORMATETC))

#define OleStdQueryObjectDescriptorData(lpformatetc)    \
     (((lpformatetc)->tymed & TYMED_HGLOBAL) ?    \
                 NOERROR : ResultFromScode(DV_E_FORMATETC))

#define OleStdQueryFormatMedium(lpformatetc, tymd)  \
     (((lpformatetc)->tymed & tymd) ?    \
                 NOERROR : ResultFromScode(DV_E_FORMATETC))

// Make an independent copy of a MetafilePict
#define OleStdCopyMetafilePict(hpictin, phpictout)  \
     (*(phpictout) = OleDuplicateData(hpictin,CF_METAFILEPICT,GHND|
GMEM_SHARE))
```

```
// REVIEW: these need to be added to OLE2.H
#if !defined( DD_DEFSCROLLINTERVAL )
#define DD_DEFSCROLLINTERVAL    50
#endif

#if !defined( DD_DEFDRAGDELAY )
#define DD_DEFDRAGDELAY         200
#endif

#if !defined( DD_DEFDRAGMINDIST )
#define DD_DEFDRAGMINDIST       2
#endif


/* OleStdGetDropEffect
** -------------------
**
** Convert a keyboard state into a DROPEFFECT.
**
** returns the DROPEFFECT value derived from the key state.
**    the following is the standard interpretation:
**          no modifier -- Default Drop     (NULL is returned)
**          CTRL        -- DROPEFFECT_COPY
**          SHIFT       -- DROPEFFECT_MOVE
**          CTRL-SHIFT  -- DROPEFFECT_LINK
**
**    Default Drop: this depends on the type of the target application.
**    this is re-interpretable by each target application. a typical
**    interpretation is if the drag is local to the same document
**    (which is source of the drag) then a MOVE operation is
**    performed. if the drag is not local, then a COPY operation is
**    performed.
*/
#define OleStdGetDropEffect(grfKeyState)    \
     ( (grfKeyState & MK_CONTROL) ?          \
           ( (grfKeyState & MK_SHIFT) ? DROPEFFECT_LINK :
DROPEFFECT_COPY ) :  \
           ( (grfKeyState & MK_SHIFT) ? DROPEFFECT_MOVE : 0 ) )


/* The OLEUIPASTEFLAG enumeration is used by the OLEUIPASTEENTRY structure.
 *
 * OLEUIPASTE_ENABLEICON    If the container does not specify this flag for
the entry in the
 *   OLEUIPASTEENTRY array passed as input to OleUIPasteSpecial, the
DisplayAsIcon button will be
 *   unchecked and disabled when the the user selects the format that
corresponds to the entry.
 *
 * OLEUIPASTE_PASTEONLY     Indicates that the entry in the OLEUIPASTEENTRY
array is valid for pasting only.
```

```
 * OLEUIPASTE_PASTE         Indicates that the entry in the OLEUIPASTEENTRY
array is valid for pasting. It
 *   may also be valid for linking if any of the following linking flags are
specified.
 *
 * If the entry in the OLEUIPASTEENTRY array is valid for linking, the
following flags indicate which link
 * types are acceptable by OR'ing together the appropriate
OLEUIPASTE_LINKTYPE<#> values.
 * These values correspond as follows to the array of link types passed to
OleUIPasteSpecial:
 *   OLEUIPASTE_LINKTYPE1=arrLinkTypes[0]
 *   OLEUIPASTE_LINKTYPE2=arrLinkTypes[1]
 *   OLEUIPASTE_LINKTYPE3=arrLinkTypes[2]
 *   OLEUIPASTE_LINKTYPE4=arrLinkTypes[3]
 *   OLEUIPASTE_LINKTYPE5=arrLinkTypes[4]
 *   OLEUIPASTE_LINKTYPE6=arrLinkTypes[5]
 *   OLEUIPASTE_LINKTYPE7=arrLinkTypes[6]
 *  OLEUIPASTE_LINKTYPE8=arrLinkTypes[7]
 *
 * where,
 *   UINT arrLinkTypes[8] is an array of registered clipboard formats for
linking. A maximium of 8 link
 *   types are allowed.
 */

typedef enum tagOLEUIPASTEFLAG
{
   OLEUIPASTE_ENABLEICON    = 2048,     // enable display as icon
   OLEUIPASTE_PASTEONLY     = 0,
   OLEUIPASTE_PASTE         = 512,
   OLEUIPASTE_LINKANYTYPE   = 1024,
   OLEUIPASTE_LINKTYPE1     = 1,
   OLEUIPASTE_LINKTYPE2     = 2,
   OLEUIPASTE_LINKTYPE3     = 4,
   OLEUIPASTE_LINKTYPE4     = 8,
   OLEUIPASTE_LINKTYPE5     = 16,
   OLEUIPASTE_LINKTYPE6     = 32,
   OLEUIPASTE_LINKTYPE7     = 64,
   OLEUIPASTE_LINKTYPE8     = 128
} OLEUIPASTEFLAG;

/*
 * PasteEntry structure
 * --------------------
 * An array of OLEUIPASTEENTRY entries is specified for the PasteSpecial
dialog
 * box. Each entry includes a FORMATETC which specifies the formats that are
 * acceptable, a string that is to represent the format in the  dialog's
list
 * box, a string to customize the result text of the dialog and a set of
flags
 * from the OLEUIPASTEFLAG enumeration.  The flags indicate if the entry is
 * valid for pasting only, linking only or both pasting and linking. If the
 * entry is valid for linking, the flags indicate which link types are
 * acceptable by OR'ing together the appropriate OLEUIPASTE_LINKTYPE<#>
values.
 * These values correspond to the array of link types as follows:
 *   OLEUIPASTE_LINKTYPE1=arrLinkTypes[0]
 *   OLEUIPASTE_LINKTYPE2=arrLinkTypes[1]
 *   OLEUIPASTE_LINKTYPE3=arrLinkTypes[2]
 *   OLEUIPASTE_LINKTYPE4=arrLinkTypes[3]
 *   OLEUIPASTE_LINKTYPE5=arrLinkTypes[4]
 *   OLEUIPASTE_LINKTYPE6=arrLinkTypes[5]
 *   OLEUIPASTE_LINKTYPE7=arrLinkTypes[6]
 *   OLEUIPASTE_LINKTYPE8=arrLinkTypes[7]
 *   UINT arrLinkTypes[8]; is an array of registered clipboard formats
 *                        for linking. A maximium of 8 link types are
allowed.
 */

typedef struct tagOLEUIPASTEENTRY
{
   FORMATETC        fmtetc;            // Format that is acceptable. The
paste
                                                         //   dialog checks
if this format is
                                                         //   offered by the
object on the
                                                         //   clipboard and
if so offers it for
                                                         //   selection to
the user.
   LPCSTR           lpstrFormatName;   // String that represents the format
to the user. Any %s
                                                         //   in this string
is replaced by the FullUserTypeName
                                                         //   of the object
on the clipboard and the resulting string
                                                         //   is placed in
the list box of the dialog. Atmost
                                                         //   one %s is
allowed. The presence or absence of %s indicates
                                                         //   if the result
text is to indicate that data is
                                                         //   being pasted or
that an object that can be activated by
                                                         //   an application
is being pasted. If %s is
                                                         //   present, the
result-text says that an object is being pasted.
                                                         //   Otherwise it
says that data is being pasted.
   LPCSTR           lpstrResultText;   // String to customize the result
text of the dialog when
                                                         //  the user selects
the format correspoding to this
                                                         //  entry. Any %s in
this string is replaced by the the application
```

```
                                                        //  name or
FullUserTypeName of the object on
                                                        //  the clipboard.
Atmost one %s is allowed.
   DWORD            dwFlags;           // Values from OLEUIPASTEFLAG enum
   DWORD            dwScratchSpace;    // Scratch space available to be used
                                                        //   by routines
which loop through an
                                                        //   IEnumFORMATETC*
to mark if the
                                                        //   PasteEntry
format is available.
                                                        //   this field CAN
be left uninitialized.
} OLEUIPASTEENTRY, *POLEUIPASTEENTRY, FAR *LPOLEUIPASTEENTRY;

#define OLESTDDROP_NONE         0
#define OLESTDDROP_DEFAULT      1
#define OLESTDDROP_NONDEFAULT   2


/*
 * Some misc stuff
 */


#define EMBEDDINGFLAG "Embedding"     // Cmd line switch for launching a
srvr

#define HIMETRIC_PER_INCH   2540      // number HIMETRIC units per inch
#define PTS_PER_INCH        72        // number points (font size) per inch

#define ROUNDED_A_DIV_B(a,b) \
                 ( ((2 * ((a) % (b))) >= (b)) ? (((a)/(b))+1) : ((a)/(b)) )

#define MAP_PIX_TO_LOGHIM(x,ppli)   \
                 ROUNDED_A_DIV_B( (((LONG)HIMETRIC_PER_INCH) * ((LONG)(x))),
\
                                          ((LONG)(ppli)) )
#define MAP_LOGHIM_TO_PIX(x,ppli)   \
                 ROUNDED_A_DIV_B( (((LONG)(ppli)) * ((LONG)(x))), \
                                          ((LONG)HIMETRIC_PER_INCH) )

// Returns TRUE if all fields of the two Rect's are equal, else FALSE.
#define AreRectsEqual(lprc1, lprc2)     \
     (((lprc1->top == lprc2->top) &&     \
       (lprc1->left == lprc2->left) &&   \
       (lprc1->right == lprc2->right) && \
       (lprc1->bottom == lprc2->bottom)) ? TRUE : FALSE)

#ifdef WIN32
#define LSTRCPYN(lpdst, lpsrc, cch) \
(\
     (lpdst)[(cch)-1] = '\0', \
     ((cch)>1 ? strncpy(lpdst, lpsrc, (cch)-1) : 0)\
)
#else
#define LSTRCPYN(lpdst, lpsrc, cch) \
(\
     (lpdst)[(cch)-1] = '\0', \
     ((cch)>1 ? _fstrncpy(lpdst, lpsrc, (cch)-1) : 0)\
)
#endif


/****** DEBUG Stuff *****************************************************/

#ifdef _DEBUG

#if !defined( _DBGTRACE )
#define _DEBUGLEVEL 2
#else
#define _DEBUGLEVEL _DBGTRACE
#endif


#if defined( NOASSERT )

#define OLEDBGASSERTDATA
#define OleDbgAssert(a)
#define OleDbgAssertSz(a, b)
#define OleDbgVerify(a)
#define OleDbgVerifySz(a, b)

#else   // ! NOASSERT

#define OLEDBGASSERTDATA    \
           static char _based(_segname("_CODE")) _szAssertFile[]= __FILE__;

#define OleDbgAssert(a) \
           (!(a) ? FnAssert(#a, NULL, _szAssertFile, __LINE__) : (HRESULT)1)

#define OleDbgAssertSz(a, b)    \
           (!(a) ? FnAssert(#a, b, _szAssertFile, __LINE__) : (HRESULT)1)

#define OleDbgVerify(a) \
           OleDbgAssert(a)

#define OleDbgVerifySz(a, b)    \
           OleDbgAssertSz(a, b)

#endif  // ! NOASSERT


#define OLEDBGDATA_MAIN(szPrefix)   \
           char near g_szDbgPrefix[] = szPrefix;    \
           OLEDBGASSERTDATA
#define OLEDBGDATA  \
           extern char near g_szDbgPrefix[];    \
           OLEDBGASSERTDATA
```

```
#define OLEDBG_BEGIN(lpsz) \
          OleDbgPrintAlways(g_szDbgPrefix,lpsz,1);

#define OLEDBG_END  \
          OleDbgPrintAlways(g_szDbgPrefix,"End\r\n",-1);

#define OleDbgOut(lpsz) \
          OleDbgPrintAlways(g_szDbgPrefix,lpsz,0)

#define OleDbgOutNoPrefix(lpsz) \
          OleDbgPrintAlways("",lpsz,0)

#define OleDbgOutRefCnt(lpsz,lpObj,refcnt)      \
          OleDbgPrintRefCntAlways(g_szDbgPrefix,lpsz,lpObj,(ULONG)refcnt)

#define OleDbgOutRect(lpsz,lpRect)      \
          OleDbgPrintRectAlways(g_szDbgPrefix,lpsz,lpRect)

#define OleDbgOutHResult(lpsz,hr)   \
          OleDbgPrintScodeAlways(g_szDbgPrefix,lpsz,GetScode(hr))

#define OleDbgOutScode(lpsz,sc) \
          OleDbgPrintScodeAlways(g_szDbgPrefix,lpsz,sc)

#define OleDbgOut1(lpsz)    \
          OleDbgPrint(1,g_szDbgPrefix,lpsz,0)

#define OleDbgOutNoPrefix1(lpsz)    \
          OleDbgPrint(1,"",lpsz,0)

#define OLEDBG_BEGIN1(lpsz)    \
          OleDbgPrint(1,g_szDbgPrefix,lpsz,1);

#define OLEDBG_END1 \
          OleDbgPrint(1,g_szDbgPrefix,"End\r\n",-1);

#define OleDbgOutRefCnt1(lpsz,lpObj,refcnt)     \
          OleDbgPrintRefCnt(1,g_szDbgPrefix,lpsz,lpObj,(ULONG)refcnt)

#define OleDbgOutRect1(lpsz,lpRect)     \
          OleDbgPrintRect(1,g_szDbgPrefix,lpsz,lpRect)

#define OleDbgOut2(lpsz)    \
          OleDbgPrint(2,g_szDbgPrefix,lpsz,0)

#define OleDbgOutNoPrefix2(lpsz)    \
          OleDbgPrint(2,"",lpsz,0)

#define OLEDBG_BEGIN2(lpsz)    \
          OleDbgPrint(2,g_szDbgPrefix,lpsz,1);


#define OLEDBG_END2 \
          OleDbgPrint(2,g_szDbgPrefix,"End\r\n",-1);
```

```
#define OleDbgOutRefCnt2(lpsz,lpObj,refcnt)      \
           OleDbgPrintRefCnt(2,g_szDbgPrefix,lpsz,lpObj,(ULONG)refcnt)

#define OleDbgOutRect2(lpsz,lpRect)      \
           OleDbgPrintRect(2,g_szDbgPrefix,lpsz,lpRect)

#define OleDbgOut3(lpsz)     \
           OleDbgPrint(3,g_szDbgPrefix,lpsz,0)

#define OleDbgOutNoPrefix3(lpsz)     \
           OleDbgPrint(3,"",lpsz,0)

#define OLEDBG_BEGIN3(lpsz)     \
           OleDbgPrint(3,g_szDbgPrefix,lpsz,1);

#define OLEDBG_END3 \
           OleDbgPrint(3,g_szDbgPrefix,"End\r\n",-1);

#define OleDbgOutRefCnt3(lpsz,lpObj,refcnt)      \
           OleDbgPrintRefCnt(3,g_szDbgPrefix,lpsz,lpObj,(ULONG)refcnt)

#define OleDbgOutRect3(lpsz,lpRect)      \
           OleDbgPrintRect(3,g_szDbgPrefix,lpsz,lpRect)

#define OleDbgOut4(lpsz)     \
           OleDbgPrint(4,g_szDbgPrefix,lpsz,0)

#define OleDbgOutNoPrefix4(lpsz)     \
           OleDbgPrint(4,"",lpsz,0)

#define OLEDBG_BEGIN4(lpsz)     \
           OleDbgPrint(4,g_szDbgPrefix,lpsz,1);

#define OLEDBG_END4 \
           OleDbgPrint(4,g_szDbgPrefix,"End\r\n",-1);

#define OleDbgOutRefCnt4(lpsz,lpObj,refcnt)      \
           OleDbgPrintRefCnt(4,g_szDbgPrefix,lpsz,lpObj,(ULONG)refcnt)

#define OleDbgOutRect4(lpsz,lpRect)      \
           OleDbgPrintRect(4,g_szDbgPrefix,lpsz,lpRect)

#else   //  !_DEBUG

#define OLEDBGDATA_MAIN(szPrefix)
#define OLEDBGDATA
#define OleDbgAssert(a)
#define OleDbgAssertSz(a, b)
#define OleDbgVerify(a)          (a)
#define OleDbgVerifySz(a, b)     (a)
#define OleDbgOutHResult(lpsz,hr)
#define OleDbgOutScode(lpsz,sc)
#define OLEDBG_BEGIN(lpsz)
#define OLEDBG_END
```

```
#define OleDbgOut(lpsz)
#define OleDbgOut1(lpsz)
#define OleDbgOut2(lpsz)
#define OleDbgOut3(lpsz)
#define OleDbgOut4(lpsz)
#define OleDbgOutNoPrefix(lpsz)
#define OleDbgOutNoPrefix1(lpsz)
#define OleDbgOutNoPrefix2(lpsz)
#define OleDbgOutNoPrefix3(lpsz)
#define OleDbgOutNoPrefix4(lpsz)
#define OLEDBG_BEGIN1(lpsz)
#define OLEDBG_BEGIN2(lpsz)
#define OLEDBG_BEGIN3(lpsz)
#define OLEDBG_BEGIN4(lpsz)
#define OLEDBG_END1
#define OLEDBG_END2
#define OLEDBG_END3
#define OLEDBG_END4
#define OleDbgOutRefCnt(lpsz,lpObj,refcnt)
#define OleDbgOutRefCnt1(lpsz,lpObj,refcnt)
#define OleDbgOutRefCnt2(lpsz,lpObj,refcnt)
#define OleDbgOutRefCnt3(lpsz,lpObj,refcnt)
#define OleDbgOutRefCnt4(lpsz,lpObj,refcnt)
#define OleDbgOutRect(lpsz,lpRect)
#define OleDbgOutRect1(lpsz,lpRect)
#define OleDbgOutRect2(lpsz,lpRect)
#define OleDbgOutRect3(lpsz,lpRect)
#define OleDbgOutRect4(lpsz,lpRect)

#endif  //  _DEBUG


/*************************************************************************
** Function prototypes
*************************************************************************/


//OLESTD.C
STDAPI_(int) SetDCToAnisotropic(HDC hDC, LPRECT lprcPhysical, LPRECT
lprcLogical, LPRECT lprcWindowOld, LPRECT lprcViewportOld);
STDAPI_(int) SetDCToDrawInHimetricRect(HDC, LPRECT, LPRECT, LPRECT, LPRECT);
STDAPI_(int) ResetOrigDC(HDC, int, LPRECT, LPRECT);

STDAPI_(int)        XformWidthInHimetricToPixels(HDC, int);
STDAPI_(int)        XformWidthInPixelsToHimetric(HDC, int);
STDAPI_(int)        XformHeightInHimetricToPixels(HDC, int);
STDAPI_(int)        XformHeightInPixelsToHimetric(HDC, int);

STDAPI_(void) XformRectInPixelsToHimetric(HDC, LPRECT, LPRECT);
STDAPI_(void) XformRectInHimetricToPixels(HDC, LPRECT, LPRECT);
STDAPI_(void) XformSizeInPixelsToHimetric(HDC, LPSIZEL, LPSIZEL);
STDAPI_(void) XformSizeInHimetricToPixels(HDC, LPSIZEL, LPSIZEL);
STDAPI_(int) XformWidthInHimetricToPixels(HDC, int);
STDAPI_(int) XformWidthInPixelsToHimetric(HDC, int);
STDAPI_(int) XformHeightInHimetricToPixels(HDC, int);
STDAPI_(int) XformHeightInPixelsToHimetric(HDC, int);

STDAPI_(void) ParseCmdLine(LPSTR, BOOL FAR *, LPSTR);

STDAPI_(BOOL) OleStdIsOleLink(LPUNKNOWN lpUnk);
STDAPI_(LPUNKNOWN) OleStdQueryInterface(LPUNKNOWN lpUnk, REFIID riid);
STDAPI_(LPSTORAGE) OleStdCreateRootStorage(LPSTR lpszStgName, DWORD
grfMode);
STDAPI_(LPSTORAGE) OleStdOpenRootStorage(LPSTR lpszStgName, DWORD grfMode);
STDAPI_(LPSTORAGE) OleStdOpenOrCreateRootStorage(LPSTR lpszStgName, DWORD
grfMode);
STDAPI_(LPSTORAGE) OleStdCreateChildStorage(LPSTORAGE lpStg, LPSTR
lpszStgName);
STDAPI_(LPSTORAGE) OleStdOpenChildStorage(LPSTORAGE lpStg, LPSTR
lpszStgName, DWORD grfMode);
STDAPI_(BOOL) OleStdCommitStorage(LPSTORAGE lpStg);
STDAPI OleStdDestroyAllElements(LPSTORAGE lpStg);


STDAPI_(LPSTORAGE) OleStdCreateStorageOnHGlobal(
           HANDLE hGlobal,
           BOOL fDeleteOnRelease,
           DWORD dwgrfMode
);
STDAPI_(LPSTORAGE) OleStdCreateTempStorage(BOOL fUseMemory, DWORD grfMode);
STDAPI OleStdDoConvert(LPSTORAGE lpStg, REFCLSID rClsidNew);
STDAPI_(BOOL) OleStdGetTreatAsFmtUserType(
           REFCLSID        rClsidApp,
           LPSTORAGE       lpStg,
           CLSID FAR*      lpclsid,
           CLIPFORMAT FAR* lpcfFmt,
           LPSTR FAR*      lplpszType
);
STDAPI OleStdDoTreatAsClass(LPSTR lpszUserType, REFCLSID rclsid, REFCLSID
rclsidNew);
STDAPI_(BOOL) OleStdSetupAdvises(LPOLEOBJECT lpOleObject, DWORD
dwDrawAspect,
                             LPSTR lpszContainerApp, LPSTR lpszContainerObj,
                             LPADVISESINK lpAdviseSink, BOOL fCreate);
STDAPI OleStdSwitchDisplayAspect(
           LPOLEOBJECT             lpOleObj,
           LPDWORD                 lpdwCurAspect,
           DWORD                   dwNewAspect,
           HGLOBAL                 hMetaPict,
           BOOL                    fDeleteOldAspect,
           BOOL                    fSetupViewAdvise,
           LPADVISESINK            lpAdviseSink,
           BOOL FAR*               lpfMustUpdate
);
STDAPI OleStdSetIconInCache(LPOLEOBJECT lpOleObj, HGLOBAL hMetaPict);
STDAPI_(HGLOBAL) OleStdGetData(
           LPDATAOBJECT        lpDataObj,
           CLIPFORMAT          cfFormat,
           DVTARGETDEVICE FAR* lpTargetDevice,
           DWORD               dwAspect,
            LPSTGMEDIUM         lpMedium
);
STDAPI_(void) OleStdMarkPasteEntryList(
            LPDATAOBJECT        lpSrcDataObj,
            LPOLEUIPASTEENTRY   lpPriorityList,
            int                 cEntries
);
STDAPI_(int) OleStdGetPriorityClipboardFormat(
            LPDATAOBJECT        lpSrcDataObj,
            LPOLEUIPASTEENTRY   lpPriorityList,
            int                 cEntries
);
STDAPI_(BOOL) OleStdIsDuplicateFormat(
            LPFORMATETC         lpFmtEtc,
            LPFORMATETC         arrFmtEtc,
            int                 nFmtEtc
);
STDAPI_(void) OleStdRegisterAsRunning(LPUNKNOWN lpUnk, LPMONIKER lpmkFull,
DWORD FAR* lpdwRegister);
STDAPI_(void) OleStdRevokeAsRunning(DWORD FAR* lpdwRegister);
STDAPI_(void) OleStdNoteFileChangeTime(LPSTR lpszFileName, DWORD
dwRegister);
STDAPI_(void) OleStdNoteObjectChangeTime(DWORD dwRegister);
STDAPI OleStdGetOleObjectData(
            LPPERSISTSTORAGE    lpPStg,
            LPFORMATETC         lpformatetc,
            LPSTGMEDIUM         lpMedium,
            BOOL                fUseMemory
);
STDAPI OleStdGetLinkSourceData(
            LPMONIKER           lpmk,
            LPCLSID             lpClsID,
            LPFORMATETC         lpformatetc,
            LPSTGMEDIUM         lpMedium
);
STDAPI_(HGLOBAL) OleStdGetObjectDescriptorData(
            CLSID               clsid,
            DWORD               dwAspect,
            SIZEL               sizel,
            POINTL              pointl,
            DWORD               dwStatus,
            LPSTR               lpszFullUserTypeName,
            LPSTR               lpszSrcOfCopy
);
STDAPI_(HGLOBAL) OleStdGetObjectDescriptorDataFromOleObject(
            LPOLEOBJECT         lpOleObj,
            LPSTR               lpszSrcOfCopy,
            DWORD               dwAspect,
            POINTL              pointl,
            LPSIZEL             lpSizelHim
);
STDAPI_(HGLOBAL) OleStdFillObjectDescriptorFromData(
            LPDATAOBJECT       lpDataObject,
            LPSTGMEDIUM        lpmedium,
            CLIPFORMAT FAR*    lpcfFmt
```

```
);
STDAPI_(HANDLE) OleStdGetMetafilePictFromOleObject(
           LPOLEOBJECT         lpOleObj,
           DWORD               dwDrawAspect,
           LPSIZEL             lpSizelHim,
           DVTARGETDEVICE FAR* ptd
);

STDAPI_(void) OleStdCreateTempFileMoniker(LPSTR lpszPrefixString, UINT FAR*
lpuUnique, LPSTR lpszName, LPMONIKER FAR* lplpmk);
STDAPI_(LPMONIKER) OleStdGetFirstMoniker(LPMONIKER lpmk);
STDAPI_(ULONG) OleStdGetLenFilePrefixOfMoniker(LPMONIKER lpmk);
STDAPI OleStdMkParseDisplayName(
           REFCLSID        rClsid,
           LPBC            lpbc,
           LPSTR           lpszUserName,
           ULONG FAR*      lpchEaten,
           LPMONIKER FAR*  lplpmk
);
STDAPI_(LPVOID) OleStdMalloc(ULONG ulSize);
STDAPI_(LPVOID) OleStdRealloc(LPVOID pmem, ULONG ulSize);
STDAPI_(void) OleStdFree(LPVOID pmem);
STDAPI_(ULONG) OleStdGetSize(LPVOID pmem);
STDAPI_(void) OleStdFreeString(LPSTR lpsz, LPMALLOC lpMalloc);
STDAPI_(LPSTR) OleStdCopyString(LPSTR lpszSrc, LPMALLOC lpMalloc);
STDAPI_(ULONG) OleStdGetItemToken(LPSTR lpszSrc, LPSTR lpszDst,int
nMaxChars);

STDAPI_(UINT)     OleStdIconLabelTextOut(HDC        hDC,
                                                            HFONT
hFont,
                                                            int
nXStart,
                                                            int
nYStart,
                                                            UINT
fuOptions,
                                                            RECT FAR *
lpRect,
                                                            LPSTR
lpszString,
                                                            UINT
cchString,
                                                            int FAR *
lpDX);

// registration database query functions
STDAPI_(UINT)     OleStdGetAuxUserType(REFCLSID rclsid,
                                                       UINT   wAuxUserType,
                                                       LPSTR
lpszAuxUserType,
                                                       int    cch,
                                                       HKEY   hKey);
```

```
STDAPI_(UINT)     OleStdGetUserTypeOfClass(REFCLSID rclsid,
                                                              LPSTR
lpszUserType,
                                                              UINT cch,
                                                              HKEY hKey);

STDAPI_(BOOL) OleStdGetMiscStatusOfClass(REFCLSID, HKEY, DWORD FAR *);
STDAPI_(CLIPFORMAT) OleStdGetDefaultFileFormatOfClass(
           REFCLSID        rclsid,
           HKEY            hKey
);

STDAPI_(void) OleStdInitVtbl(LPVOID lpVtbl, UINT nSizeOfVtbl);
STDMETHODIMP OleStdNullMethod(LPUNKNOWN lpThis);
STDAPI_(BOOL) OleStdCheckVtbl(LPVOID lpVtbl, UINT nSizeOfVtbl, LPSTR
lpszIface);
STDAPI_(ULONG) OleStdVerifyRelease(LPUNKNOWN lpUnk, LPSTR lpszMsg);
STDAPI_(ULONG) OleStdRelease(LPUNKNOWN lpUnk);

STDAPI_(HDC) OleStdCreateDC(DVTARGETDEVICE FAR* ptd);
STDAPI_(HDC) OleStdCreateIC(DVTARGETDEVICE FAR* ptd);
STDAPI_(DVTARGETDEVICE FAR*) OleStdCreateTargetDevice(LPPRINTDLG
lpPrintDlg);
STDAPI_(BOOL) OleStdDeleteTargetDevice(DVTARGETDEVICE FAR* ptd);
STDAPI_(DVTARGETDEVICE FAR*) OleStdCopyTargetDevice(DVTARGETDEVICE FAR*
ptdSrc);
STDAPI_(BOOL) OleStdCopyFormatEtc(LPFORMATETC petcDest, LPFORMATETC
petcSrc);
STDAPI_(int) OleStdCompareFormatEtc(FORMATETC FAR* pFetcLeft, FORMATETC FAR*
pFetcRight);
STDAPI_(BOOL) OleStdCompareTargetDevice
     (DVTARGETDEVICE FAR* ptdLeft, DVTARGETDEVICE FAR* ptdRight);


STDAPI_(void) OleDbgPrint(
           int     nDbgLvl,
           LPSTR   lpszPrefix,
           LPSTR   lpszMsg,
           int     nIndent
);
STDAPI_(void) OleDbgPrintAlways(LPSTR lpszPrefix, LPSTR lpszMsg, int
nIndent);
STDAPI_(void) OleDbgSetDbgLevel(int nDbgLvl);
STDAPI_(int) OleDbgGetDbgLevel( void );
STDAPI_(void) OleDbgIndent(int n);
STDAPI_(void) OleDbgPrintRefCnt(
           int         nDbgLvl,
           LPSTR       lpszPrefix,
           LPSTR       lpszMsg,
           LPVOID      lpObj,
           ULONG       refcnt
);
STDAPI_(void) OleDbgPrintRefCntAlways(
           LPSTR       lpszPrefix,
           LPSTR       lpszMsg,
           LPVOID      lpObj,
           ULONG       refcnt
);
STDAPI_(void) OleDbgPrintRect(
           int         nDbgLvl,
           LPSTR       lpszPrefix,
           LPSTR       lpszMsg,
           LPRECT      lpRect
);
STDAPI_(void) OleDbgPrintRectAlways(
           LPSTR       lpszPrefix,
           LPSTR       lpszMsg,
           LPRECT      lpRect
);
STDAPI_(void) OleDbgPrintScodeAlways(LPSTR lpszPrefix, LPSTR lpszMsg, SCODE
sc);

// debug implementation of the IMalloc interface.
STDAPI OleStdCreateDbAlloc(ULONG reserved, IMalloc FAR* FAR* ppmalloc);


STDAPI_(LPENUMFORMATETC)
  OleStdEnumFmtEtc_Create(ULONG nCount, LPFORMATETC lpEtc);

STDAPI_(LPENUMSTATDATA)
  OleStdEnumStatData_Create(ULONG nCount, LPSTATDATA lpStat);

STDAPI_(BOOL)
  OleStdCopyStatData(LPSTATDATA pDest, LPSTATDATA pSrc);

STDAPI_(HPALETTE)
  OleStdCreateStandardPalette(void);

#if defined( OBSOLETE )

/*************************************************************************
** The following API's have been converted into macros:
**          OleStdQueryOleObjectData
**          OleStdQueryLinkSourceData
**          OleStdQueryObjectDescriptorData
**          OleStdQueryFormatMedium
**          OleStdCopyMetafilePict
**          AreRectsEqual
**          OleStdGetDropEffect
**
**    These macros are defined above
*************************************************************************/
STDAPI_(BOOL) AreRectsEqual(LPRECT lprc1, LPRECT lprc2);
STDAPI_(BOOL) OleStdCopyMetafilePict(HANDLE hpictin, HANDLE FAR* phpictout);
STDAPI OleStdQueryOleObjectData(LPFORMATETC lpformatetc);
STDAPI OleStdQueryLinkSourceData(LPFORMATETC lpformatetc);
STDAPI OleStdQueryObjectDescriptorData(LPFORMATETC lpformatetc);
STDAPI OleStdQueryFormatMedium(LPFORMATETC lpformatetc, TYMED tymed);
STDAPI_(DWORD) OleStdGetDropEffect ( DWORD grfKeyState );
#endif  // OBSOLETE
```

```
#endif // _OLESTD_H_
```

OLESTD.C (WRAPUI Sample)

```
/*************************************************************************
**
**    OLE 2 Standard Utilities
**
**    olestd.c
**
**    This file contains utilities that are useful for most standard
**        OLE 2.0 compound document type applications.
**
**    (c) Copyright Microsoft Corp. 1992 All Rights Reserved
**
*************************************************************************/

#define NONAMELESSUNION     // use strict ANSI standard (for DVOBJ.H)

#define STRICT  1
#include "ole2ui.h"
#include <stdlib.h>
#include <ctype.h>
#include <shellapi.h>
#include "regdb.h"
#include "geticon.h"
#include "common.h"

OLEDBGDATA

static char szAssertMemAlloc[] = "CoGetMalloc failed";


/* OleStdSetupAdvises
** ------------------
**    Setup the standard View advise required by a standard,
**    compound document-oriented container. Such a container relies on
**    Ole to manage the presentation of the Ole object. The container
**    call IViewObject::Draw to render (display) the object.
**
**    This helper routine performs the following tasks:
**                        setup View advise
**                        Call IOleObject::SetHostNames
**                        Call OleSetContainedObject
**
**    fCreate should be set to TRUE if the object is being created. if
**    an existing object is being loaded, then fCreate should be FALSE.
**    if it is a creation situation, then the ADVF_PRIMEFIRST flag is
**    used when settinp up the IViewObject::Advise. This will result in
**    the immediate sending of the initial picture.
**
**    OLE2NOTE: the standard container does NOT need to set up an OLE
**    Advise (IOleObject::Advise). this routine does NOT set up an OLE
**    Advise (a previous version of this function used to setup this
**    advise, but it was not useful).
*/
```

```
STDAPI_(BOOL) OleStdSetupAdvises(LPOLEOBJECT lpOleObject, DWORD
dwDrawAspect,
                             LPSTR lpszContainerApp, LPSTR lpszContainerObj,
                             LPADVISESINK lpAdviseSink, BOOL fCreate)
{
     LPVIEWOBJECT lpViewObject;
     HRESULT hrErr;
     BOOL fStatus = TRUE;
#if defined( SPECIAL_CONTAINER )
     DWORD dwTemp;
#endif

     hrErr = lpOleObject->lpVtbl->QueryInterface(
                 lpOleObject,
                 &IID_IViewObject,
                 (LPVOID FAR*)&lpViewObject
     );

     /* Setup View advise */
     if (hrErr == NOERROR) {

           OLEDBG_BEGIN2("IViewObject::SetAdvise called\r\n")
           lpViewObject->lpVtbl->SetAdvise(
                       lpViewObject,
                       dwDrawAspect,
                       (fCreate ? ADVF_PRIMEFIRST : 0),
                       lpAdviseSink
           );
           OLEDBG_END2

           OleStdRelease((LPUNKNOWN)lpViewObject);
     } else {
           fStatus = FALSE;
     }

#if defined( SPECIAL_CONTAINER )
     /* Setup OLE advise.
     **    OLE2NOTE: normally containers do NOT need to setup an OLE
     **    advise. this advise connection is only useful for the OLE's
     **    DefHandler and the OleLink object implementation. some
     **    special container's might need to setup this advise for
     **    programatic reasons.
     **
     **    NOTE: this advise will be torn down automatically by the
     **    server when we release the object, therefore we do not need
     **    to store the connection id.
     */
     OLEDBG_BEGIN2("IOleObject::Advise called\r\n")
     hrErr = lpOleObject->lpVtbl->Advise(
                 lpOleObject,
                 lpAdviseSink,
                 (DWORD FAR*)&dwTemp
     );
     OLEDBG_END2
     if (hrErr != NOERROR) fStatus = FALSE;
```

```
#endif

     /* Setup the host names for the OLE object. */
     OLEDBG_BEGIN2("IOleObject::SetHostNames called\r\n")
     hrErr = lpOleObject->lpVtbl->SetHostNames(
                 lpOleObject,
                 lpszContainerApp,
                 lpszContainerObj
     );
     OLEDBG_END2
     if (hrErr != NOERROR) fStatus = FALSE;

     /* Inform the loadded object's handler/inproc-server that it is in
     **    its embedding container's process.
     */
     OLEDBG_BEGIN2("OleSetContainedObject(TRUE) called\r\n")
     OleSetContainedObject((LPUNKNOWN)lpOleObject, TRUE);
     OLEDBG_END2

     return fStatus;
}


/* OleStdSwitchDisplayAspect
** -------------------------
**    Switch the currently cached display aspect between DVASPECT_ICON
**    and DVASPECT_CONTENT.
**
**    NOTE: when setting up icon aspect, any currently cached content
**    cache is discarded and any advise connections for content aspect
**    are broken.
**
**    RETURNS:
**      S_OK -- new display aspect setup successfully
**      E_INVALIDARG -- IOleCache interface is NOT supported (this is
**                  required).
**      <other SCODE> -- any SCODE that can be returned by
**                  IOleCache::Cache method.
**      NOTE: if an error occurs then the current display aspect and
**            cache contents unchanged.
*/
STDAPI OleStdSwitchDisplayAspect(
           LPOLEOBJECT             lpOleObj,
           LPDWORD                 lpdwCurAspect,
           DWORD                   dwNewAspect,
           HGLOBAL                 hMetaPict,
           BOOL                    fDeleteOldAspect,
           BOOL                    fSetupViewAdvise,
           LPADVISESINK            lpAdviseSink,
           BOOL FAR*               lpfMustUpdate
)
{
     LPOLECACHE      lpOleCache = NULL;
     LPVIEWOBJECT    lpViewObj = NULL;
```

```
LPENUMSTATDATA  lpEnumStatData = NULL;
STATDATA        StatData;
FORMATETC       FmtEtc;
STGMEDIUM       Medium;
DWORD           dwAdvf;
DWORD           dwNewConnection;
DWORD           dwOldAspect = *lpdwCurAspect;
HRESULT         hrErr;

if (lpfMustUpdate)
      *lpfMustUpdate = FALSE;

lpOleCache = (LPOLECACHE)OleStdQueryInterface(

(LPUNKNOWN)lpOleObj,&IID_IOleCache);

// if IOleCache* is NOT available, do nothing
if (! lpOleCache)
      return ResultFromScode(E_INVALIDARG);

// Setup new cache with the new aspect
FmtEtc.cfFormat = 0;     // whatever is needed to draw
FmtEtc.ptd      = NULL;
FmtEtc.dwAspect = dwNewAspect;
FmtEtc.lindex   = -1;
FmtEtc.tymed    = TYMED_NULL;

/* OLE2NOTE: if we are setting up Icon aspect
**    then we do not want DataAdvise notifications to ever change
**    the contents of the data cache. thus we set up a NODATA
**    advise connection. otherwise we set up a standard DataAdvise
**    connection.
*/
if (dwNewAspect == DVASPECT_ICON)
      dwAdvf = ADVF_NODATA;
else
      dwAdvf = ADVF_PRIMEFIRST;

OLEDBG_BEGIN2("IOleCache::Cache called\r\n")
hrErr = lpOleCache->lpVtbl->Cache(
            lpOleCache,
            (LPFORMATETC)&FmtEtc,
            dwAdvf,
            (LPDWORD)&dwNewConnection
);
OLEDBG_END2

if (! SUCCEEDED(hrErr)) {
      OleDbgOutHResult("IOleCache::Cache returned", hrErr);
      OleStdRelease((LPUNKNOWN)lpOleCache);
      return hrErr;
}

*lpdwCurAspect = dwNewAspect;
```

```
/* OLE2NOTE: if we are setting up Icon aspect with a custom icon,
**    then stuff the icon into the cache. otherwise the cache must
**    be forced to be updated. set the *lpfMustUpdate flag to tell
**    caller to force the object to Run so that the cache will be
**    updated.
*/
if (dwNewAspect == DVASPECT_ICON && hMetaPict) {

      FmtEtc.cfFormat = CF_METAFILEPICT;
      FmtEtc.ptd      = NULL;
      FmtEtc.dwAspect = DVASPECT_ICON;
      FmtEtc.lindex   = -1;
      FmtEtc.tymed    = TYMED_MFPICT;

      Medium.tymed            = TYMED_MFPICT;
      Medium.u.hGlobal        = hMetaPict;
      Medium.pUnkForRelease   = NULL;

      OLEDBG_BEGIN2("IOleCache::SetData called\r\n")
      hrErr = lpOleCache->lpVtbl->SetData(
                  lpOleCache,
                  (LPFORMATETC)&FmtEtc,
                  (LPSTGMEDIUM)&Medium,
                  FALSE   /* fRelease */
      );
      OLEDBG_END2
} else {
      if (lpfMustUpdate)
            *lpfMustUpdate = TRUE;
}

if (fSetupViewAdvise && lpAdviseSink) {
      /* OLE2NOTE: re-establish the ViewAdvise connection */
      lpViewObj = (LPVIEWOBJECT)OleStdQueryInterface(

(LPUNKNOWN)lpOleObj,&IID_IViewObject);

      if (lpViewObj) {

            OLEDBG_BEGIN2("IViewObject::SetAdvise called\r\n")
            lpViewObj->lpVtbl->SetAdvise(
                        lpViewObj,
                        dwNewAspect,
                        0,

                        lpAdviseSink
            );
            OLEDBG_END2

            OleStdRelease((LPUNKNOWN)lpViewObj);
      }
}

/* OLE2NOTE: remove any existing caches that are set up for the old
**    display aspect. It WOULD be possible to retain the caches set
```

```
    **    up for the old aspect, but this would increase the storage
    **    space required for the object and possibly require additional
    **    overhead to maintain the unused cachaes. For these reasons the
    **    strategy to delete the previous caches is prefered. if it is a
    **    requirement to quickly switch between Icon and Content
    **    display, then it would be better to keep both aspect caches.
    */

    if (fDeleteOldAspect) {
          OLEDBG_BEGIN2("IOleCache::EnumCache called\r\n")
          hrErr = lpOleCache->lpVtbl->EnumCache(
                      lpOleCache,
                      (LPENUMSTATDATA FAR*)&lpEnumStatData
          );
          OLEDBG_END2

          while(hrErr == NOERROR) {
                hrErr = lpEnumStatData->lpVtbl->Next(
                            lpEnumStatData,
                            1,
                            (LPSTATDATA)&StatData,
                            NULL
                );
                if (hrErr != NOERROR)
                      break;              // DONE! no more caches.

                if (StatData.formatetc.dwAspect == dwOldAspect) {

                      // Remove previous cache with old aspect
                      OLEDBG_BEGIN2("IOleCache::Uncache called\r\n")
                      lpOleCache->lpVtbl-
>Uncache(lpOleCache,StatData.dwConnection);
                      OLEDBG_END2
                }
          }

          if (lpEnumStatData) {
                OleStdVerifyRelease(
                            (LPUNKNOWN)lpEnumStatData,
                            "OleStdSwitchDisplayAspect: Cache enumerator
NOT released"
                );
          }
    }

    if (lpOleCache)
          OleStdRelease((LPUNKNOWN)lpOleCache);

    return NOERROR;
}


/* OleStdSetIconInCache
** --------------------
**    SetData a new icon into the existing DVASPECT_ICON cache.
```

```
**
**    RETURNS:
**      HRESULT returned from IOleCache::SetData
*/
STDAPI OleStdSetIconInCache(LPOLEOBJECT lpOleObj, HGLOBAL hMetaPict)
{
     LPOLECACHE      lpOleCache = NULL;
     FORMATETC       FmtEtc;
     STGMEDIUM       Medium;
     HRESULT         hrErr;

     if (! hMetaPict)
           return FALSE;   // invalid icon

     lpOleCache = (LPOLECACHE)OleStdQueryInterface(

     (LPUNKNOWN)lpOleObj,&IID_IOleCache);
     if (! lpOleCache)
           return FALSE;   // if IOleCache* is NOT available, do nothing

     FmtEtc.cfFormat = CF_METAFILEPICT;
     FmtEtc.ptd      = NULL;
     FmtEtc.dwAspect = DVASPECT_ICON;
     FmtEtc.lindex   = -1;
     FmtEtc.tymed    = TYMED_MFPICT;

     // stuff the icon into the cache.
     Medium.tymed            = TYMED_MFPICT;
     Medium.u.hGlobal        = hMetaPict;
     Medium.pUnkForRelease   = NULL;

     OLEDBG_BEGIN2("IOleCache::SetData called\r\n")
     hrErr = lpOleCache->lpVtbl->SetData(
                 lpOleCache,
                 (LPFORMATETC)&FmtEtc,
                 (LPSTGMEDIUM)&Medium,
                 FALSE   /* fRelease */
     );
     OLEDBG_END2

     OleStdRelease((LPUNKNOWN)lpOleCache);

     return hrErr;
}



/* OleStdDoConvert
** ---------------
** Do the container-side responsibilities for converting an object.
**    This function would be used in conjunction with the OleUIConvert
**    dialog. If the user selects to convert an object then the
**    container must do the following:
**          1. unload the object.
**          2. write the NEW CLSID and NEW user type name
**              string into the storage of the object,
**              BUT write the OLD format tag.
**          3. force an update of the object to force the actual
**              conversion of the data bits.
**
**    This function takes care of step 2.
*/
STDAPI OleStdDoConvert(LPSTORAGE lpStg, REFCLSID rClsidNew)
{
     HRESULT error;
     CLSID clsidOld;
     CLIPFORMAT cfOld;
     LPSTR lpszOld = NULL;
     char szNew[OLEUI_CCHKEYMAX];

     if ((error = ReadClassStg(lpStg, &clsidOld)) != NOERROR) {
           clsidOld = CLSID_NULL;
           goto errRtn;
     }

     // read old fmt/old user type; sets out params to NULL on error
     error = ReadFmtUserTypeStg(lpStg, &cfOld, &lpszOld);
     OleDbgAssert(error == NOERROR || (cfOld == 0 && lpszOld == NULL));

     // get new user type name; if error, set to NULL string
     if (OleStdGetUserTypeOfClass(
                 (LPCLSID)rClsidNew, szNew,sizeof(szNew),NULL /* hKey */) ==
0)
           szNew[0] = '\0';

     // write class stg
     if ((error = WriteClassStg(lpStg, rClsidNew)) != NOERROR)
           goto errRtn;

     // write old fmt/new user type;
     if ((error = WriteFmtUserTypeStg(lpStg, cfOld, szNew)) != NOERROR)
           goto errRewriteInfo;

     // set convert bit
     if ((error = SetConvertStg(lpStg, TRUE)) != NOERROR)
           goto errRewriteInfo;

     goto okRtn;

errRewriteInfo:
     (void)WriteClassStg(lpStg, &clsidOld);
     (void)WriteFmtUserTypeStg(lpStg, cfOld, lpszOld);

errRtn:

okRtn:
     OleStdFreeString(lpszOld, NULL);
     return error;
}
```

```
/* OleStdGetTreatAsFmtUserType
** ---------------------------
**    Determine if the application should perform a TreatAs (ActivateAs
**    object or emulation) operation for the object that is stored in
**    the storage.
**
**    if the CLSID written in the storage is not the same as the
**    application's own CLSID (clsidApp), then a TreatAs operation
**    should take place. if so determine the format the data should be
**    written and the user type name of the object the app should
**    emulate (ie. pretend to be). if this information is not written
**    in the storage then it is looked up in the REGDB. if it can not
**    be found in the REGDB, then the TreatAs operation can NOT be
**    executed.
**
**    RETURNS: TRUE -- if TreatAs should be performed.
**               valid lpclsid, lplpszType, lpcfFmt to TreatAs are returned
**                      (NOTE: lplpszType must be freed by caller)
**             FALSE -- NO TreatAs. lpszType will be NULL.
**               lpclsid = CLSID_NULL; lplpszType = lpcfFmt = NULL;
*/
STDAPI_(BOOL) OleStdGetTreatAsFmtUserType(
           REFCLSID        rclsidApp,
           LPSTORAGE       lpStg,
           CLSID FAR*      lpclsid,
           CLIPFORMAT FAR* lpcfFmt,
           LPSTR FAR*      lplpszType
)
{
     HRESULT hrErr;
     HKEY    hKey;
     LONG    lRet;
     UINT    lSize;
     char    szBuf[OLEUI_CCHKEYMAX];

     *lpclsid    = CLSID_NULL;
     *lpcfFmt    = 0;
     *lplpszType = NULL;

     hrErr = ReadClassStg(lpStg, lpclsid);
     if (hrErr == NOERROR &&
           ! IsEqualCLSID(lpclsid, &CLSID_NULL) &&
           ! IsEqualCLSID(lpclsid, rclsidApp)) {

           hrErr = ReadFmtUserTypeStg(lpStg,(CLIPFORMAT
FAR*)lpcfFmt,lplpszType);
           if (hrErr == NOERROR && lplpszType && *lpcfFmt != 0)
                 return TRUE;    // Do TreatAs. info was in lpStg.

           /* read info from REGDB
           **    *lpcfFmt = value of field: CLSID\{...}
\DataFormats\DefaultFile
           **    *lplpszType = value of field: CLSID\{...}
           */
           //Open up the root key.
           lRet=RegOpenKey(HKEY_CLASSES_ROOT, NULL, &hKey);
           if (lRet != (LONG)ERROR_SUCCESS)
                 return FALSE;
           *lpcfFmt = OleStdGetDefaultFileFormatOfClass(lpclsid, hKey);
           if (*lpcfFmt == 0)
                 return FALSE;
           lSize =
OleStdGetUserTypeOfClass(lpclsid,szBuf,sizeof(szBuf),hKey);
           if (lSize == 0)
                 return FALSE;
           *lplpszType = OleStdCopyString(szBuf, NULL);
     } else {
           return FALSE;       // NO TreatAs
     }
}



/* OleStdDoTreatAsClass
** --------------------
** Do the container-side responsibilities for "ActivateAs" (aka.
**    TreatAs) for an object.
**    This function would be used in conjunction with the OleUIConvert
**    dialog. If the user selects to ActivateAs an object then the
**    container must do the following:
**          1. unload ALL objects of the OLD class that app knows about

**          2. add the TreatAs tag in the registration database
**              by calling CoTreatAsClass().
**          3. lazily it can reload the objects; when the objects
**              are reloaded the TreatAs will take effect.
**
**    This function takes care of step 2.
*/
STDAPI OleStdDoTreatAsClass(LPSTR lpszUserType, REFCLSID rclsid, REFCLSID
rclsidNew)
{
     HRESULT hrErr;
     LPSTR   lpszCLSID = NULL;
     LONG    lRet;
     HKEY    hKey;

     OLEDBG_BEGIN2("CoTreatAsClass called\r\n")
     hrErr = CoTreatAsClass(rclsid, rclsidNew);
     OLEDBG_END2

     if ((hrErr != NOERROR) && lpszUserType) {
           lRet = RegOpenKey(HKEY_CLASSES_ROOT, (LPCSTR)"CLSID",
                       (HKEY FAR *)&hKey);
           StringFromCLSID(rclsid, (LPSTR FAR*)&lpszCLSID);
           RegSetValue(hKey, lpszCLSID, REG_SZ, lpszUserType,
                       lstrlen(lpszUserType));
```

```
            if (lpszCLSID)
                  OleStdFreeString(lpszCLSID, NULL);

            hrErr = CoTreatAsClass(rclsid, rclsidNew);
            RegCloseKey(hKey);
      }

      return hrErr;
}



/* OleStdIsOleLink
** ---------------
**    Returns TRUE if the OleObject is infact an OLE link object. this
**    checks if IOleLink interface is supported. if so, the object is a
**    link, otherwise not.
*/
STDAPI_(BOOL) OleStdIsOleLink(LPUNKNOWN lpUnk)
{
      LPOLELINK lpOleLink;

      lpOleLink = (LPOLELINK)OleStdQueryInterface(lpUnk, &IID_IOleLink);

      if (lpOleLink) {
            OleStdRelease((LPUNKNOWN)lpOleLink);
            return TRUE;
      } else
            return FALSE;
}


/* OleStdQueryInterface
** --------------------
**    Returns the desired interface pointer if exposed by the given object.
**    Returns NULL if the interface is not available.
**    eg.:
**      lpDataObj = OleStdQueryInterface(lpOleObj, &IID_DataObject);
*/
STDAPI_(LPUNKNOWN) OleStdQueryInterface(LPUNKNOWN lpUnk, REFIID riid)
{
      LPUNKNOWN lpInterface;
      HRESULT hrErr;

      hrErr = lpUnk->lpVtbl->QueryInterface(
                  lpUnk,
                  riid,
                  (LPVOID FAR*)&lpInterface
      );

      if (hrErr == NOERROR)
            return lpInterface;
      else
            return NULL;
}
```

```
/* OleStdGetData
** -------------
**    Retrieve data from an IDataObject in a specified format on a
**    global memory block. This function ALWAYS returns a private copy
**    of the data to the caller. if necessary a copy is made of the
**    data (ie. if lpMedium->pUnkForRelease != NULL). The caller assumes
**    ownership of the data block in all cases and must free the data
**    when done with it. The caller may directly free the data handle
**    returned (taking care whether it is a simple HGLOBAL or a HANDLE
**    to a MetafilePict) or the caller may call
**    ReleaseStgMedium(lpMedium). this OLE helper function will do the
**    right thing.
**
**    PARAMETERS:
**        LPDATAOBJECT lpDataObj  -- object on which GetData should be
**                                                           called.
**        CLIPFORMAT cfFormat     -- desired clipboard format (eg. CF_TEXT)
**        DVTARGETDEVICE FAR* lpTargetDevice -- target device for which
**                                  the data should be composed. This may
**                                  be NULL. NULL can be used whenever the
**                                  data format is insensitive to target
**                                  device or when the caller does not care
**                                  what device is used.
**        LPSTGMEDIUM lpMedium    -- ptr to STGMEDIUM struct. the
**                                  resultant medium from the
**                                  IDataObject::GetData call is
**                                  returned.
**
**    RETURNS:
**       HGLOBAL -- global memory handle of retrieved data block.
**       NULL    -- if error.
*/
STDAPI_(HGLOBAL) OleStdGetData(
           LPDATAOBJECT        lpDataObj,
           CLIPFORMAT          cfFormat,
           DVTARGETDEVICE FAR* lpTargetDevice,
           DWORD               dwDrawAspect,
           LPSTGMEDIUM         lpMedium
)
{
     HRESULT hrErr;
     FORMATETC formatetc;
     HGLOBAL hGlobal = NULL;
     HGLOBAL hCopy;
     LPVOID  lp;

     formatetc.cfFormat = cfFormat;
     formatetc.ptd = lpTargetDevice;
     formatetc.dwAspect = dwDrawAspect;
     formatetc.lindex = -1;

     switch (cfFormat) {
        case CF_METAFILEPICT:
            formatetc.tymed = TYMED_MFPICT;
            break;

        case CF_BITMAP:
            formatetc.tymed = TYMED_GDI;
            break;

        default:
            formatetc.tymed = TYMED_HGLOBAL;
            break;
    }

    OLEDBG_BEGIN2("IDataObject::GetData called\r\n")
    hrErr = lpDataObj->lpVtbl->GetData(
            lpDataObj,
            (LPFORMATETC)&formatetc,
            lpMedium
    );
    OLEDBG_END2

    if (hrErr != NOERROR)
        return NULL;

    if ((hGlobal = lpMedium->u.hGlobal) == NULL)
        return NULL;

    // Check if hGlobal really points to valid memory
    if ((lp = GlobalLock(hGlobal)) != NULL) {
        if (IsBadReadPtr(lp, 1)) {
            GlobalUnlock(hGlobal);
            return NULL;    // ERROR: memory is NOT valid
        }
        GlobalUnlock(hGlobal);
    }

    if (hGlobal != NULL && lpMedium->pUnkForRelease != NULL) {
        /* OLE2NOTE: the callee wants to retain ownership of the data.
        **    this is indicated by passing a non-NULL pUnkForRelease.
        **    thus, we will make a copy of the data and release the
        **    callee's copy.
        */

        hCopy = OleDuplicateData(hGlobal, cfFormat, GHND|GMEM_SHARE);
        ReleaseStgMedium(lpMedium); // release callee's copy of data

        hGlobal = hCopy;
        lpMedium->u.hGlobal = hCopy;
        lpMedium->pUnkForRelease = NULL;
    }
    return hGlobal;
}


/* OleStdMalloc
```

```
** ------------
**    allocate memory using the currently active IMalloc* allocator
*/
STDAPI_(LPVOID) OleStdMalloc(ULONG ulSize)
{
     LPVOID pout;
     LPMALLOC pmalloc;

     if (CoGetMalloc(MEMCTX_TASK, &pmalloc) != NOERROR) {
           OleDbgAssertSz(0, szAssertMemAlloc);
           return NULL;
     }

     pout = (LPVOID)pmalloc->lpVtbl->Alloc(pmalloc, ulSize);

     if (pmalloc != NULL) {
           ULONG refs = pmalloc->lpVtbl->Release(pmalloc);
     }

     return pout;
}


/* OleStdRealloc
** -------------
**    re-allocate memory using the currently active IMalloc* allocator
*/
STDAPI_(LPVOID) OleStdRealloc(LPVOID pmem, ULONG ulSize)
{
     LPVOID pout;
     LPMALLOC pmalloc;

     if (CoGetMalloc(MEMCTX_TASK, &pmalloc) != NOERROR) {
           OleDbgAssertSz(0, szAssertMemAlloc);
           return NULL;
     }

     pout = (LPVOID)pmalloc->lpVtbl->Realloc(pmalloc, pmem, ulSize);

     if (pmalloc != NULL) {
           ULONG refs = pmalloc->lpVtbl->Release(pmalloc);
     }

     return pout;
}


/* OleStdFree
** ----------
**    free memory using the currently active IMalloc* allocator
*/
STDAPI_(void) OleStdFree(LPVOID pmem)
{
     LPMALLOC pmalloc;
```

```
	if (pmem == NULL)
		return;

	if (CoGetMalloc(MEMCTX_TASK, &pmalloc) != NOERROR) {
		OleDbgAssertSz(0, szAssertMemAlloc);
		return;
	}

	pmalloc->lpVtbl->Free(pmalloc, pmem);

	if (pmalloc != NULL) {
		ULONG refs = pmalloc->lpVtbl->Release(pmalloc);
	}
}



/* OleStdGetSize
** -------------
**    Get the size of a memory block that was allocated using the
**    currently active IMalloc* allocator.
*/
STDAPI_(ULONG) OleStdGetSize(LPVOID pmem)
{
	ULONG ulSize;
	LPMALLOC pmalloc;

	if (CoGetMalloc(MEMCTX_TASK, &pmalloc) != NOERROR) {
		OleDbgAssertSz(0, szAssertMemAlloc);
		return (ULONG)-1;
	}

	ulSize = pmalloc->lpVtbl->GetSize(pmalloc, pmem);

	if (pmalloc != NULL) {
		ULONG refs = pmalloc->lpVtbl->Release(pmalloc);
	}

	return ulSize;
}


/* OleStdFreeString
** ----------------
**    Free a string that was allocated with the currently active
**    IMalloc* allocator.
**
**    if the caller has the current IMalloc* handy, then it can be
**    passed as a argument, otherwise this function will retrieve the
**    active allocator and use it.
*/
STDAPI_(void) OleStdFreeString(LPSTR lpsz, LPMALLOC lpMalloc)
{
	BOOL fMustRelease = FALSE;
```

```
     if (! lpMalloc) {
          if (CoGetMalloc(MEMCTX_TASK, &lpMalloc) != NOERROR)
               return;
          fMustRelease = TRUE;
     }

     lpMalloc->lpVtbl->Free(lpMalloc, lpsz);

     if (fMustRelease)
          lpMalloc->lpVtbl->Release(lpMalloc);
}


/* OleStdCopyString
** ----------------
**    Copy a string into memory allocated with the currently active
**    IMalloc* allocator.
**
**    if the caller has the current IMalloc* handy, then it can be
**    passed as a argument, otherwise this function will retrieve the
**    active allocator and use it.
*/
STDAPI_(LPSTR) OleStdCopyString(LPSTR lpszSrc, LPMALLOC lpMalloc)
{
     LPSTR lpszDest = NULL;
     BOOL fMustRelease = FALSE;
     UINT lSize = lstrlen(lpszSrc);

     if (! lpMalloc) {
          if (CoGetMalloc(MEMCTX_TASK, &lpMalloc) != NOERROR)
               return NULL;
          fMustRelease = TRUE;
     }

     lpszDest = lpMalloc->lpVtbl->Alloc(lpMalloc, lSize+1);

     if (lpszDest)
          lstrcpy(lpszDest, lpszSrc);

     if (fMustRelease)
          lpMalloc->lpVtbl->Release(lpMalloc);
     return lpszDest;
}


/*
 * OleStdCreateStorageOnHGlobal()
 *
 * Purpose:
 *  Create a memory based IStorage*.
 *
 *  OLE2NOTE: if fDeleteOnRelease==TRUE, then the ILockBytes is created
 *            such that it will delete them memory on its last release.
 *            the IStorage on created on top of the ILockBytes in NOT
 *            created with STGM_DELETEONRELEASE. when the IStorage receives
 *             its last release, it will release the ILockBytes which will
 *             in turn free the memory. it is in fact an error to specify
 *             STGM_DELETEONRELEASE in this situation.
 *
 * Parameters:
 *  hGlobal --  handle to MEM_SHARE allocated memory. may be NULL and
 *               memory will be automatically allocated.
 *  fDeleteOnRelease -- controls if the memory is freed on the last release.
 *  grfMode --  flags passed to StgCreateDocfileOnILockBytes
 *
 *  NOTE: if hGlobal is NULL, then a new IStorage is created and
 *               STGM_CREATE flag is passed to StgCreateDocfileOnILockBytes.
 *        if hGlobal is non-NULL, then it is assumed that the hGlobal
already
 *               has an IStorage inside it and STGM_CONVERT flag is passed
 *               to StgCreateDocfileOnILockBytes.
 *
 * Return Value:
 *    SCODE  -  S_OK if successful
 */
STDAPI_(LPSTORAGE) OleStdCreateStorageOnHGlobal(
           HANDLE hGlobal,
           BOOL fDeleteOnRelease,
           DWORD grfMode
)
{
     DWORD grfCreateMode=grfMode | (hGlobal==NULL ?
STGM_CREATE:STGM_CONVERT);
     HRESULT hrErr;
     LPLOCKBYTES lpLockBytes = NULL;
     DWORD reserved = 0;
     LPSTORAGE lpStg = NULL;

     hrErr = CreateILockBytesOnHGlobal(
                 hGlobal,
                 fDeleteOnRelease,
                 (LPLOCKBYTES FAR*)&lpLockBytes
     );
     if (hrErr != NOERROR)
           return NULL;

     hrErr = StgCreateDocfileOnILockBytes(
                 lpLockBytes,
                 grfCreateMode,
                 reserved,
                 (LPSTORAGE FAR*)&lpStg
     );
     if (hrErr != NOERROR) {
           OleStdRelease((LPUNKNOWN)lpLockBytes);
           return NULL;
     }
     OleStdRelease((LPUNKNOWN)lpLockBytes);
     return lpStg;
}
```

```
/*
 * OleStdCreateTempStorage()
 *
 * Purpose:
 *  Create a temporay IStorage* that will DeleteOnRelease.
 *  this can be either memory based or file based.
 *
 * Parameters:
 *  fUseMemory -- controls if memory-based or file-based stg is created
 *  grfMode --  storage mode flags
 *
 * Return Value:
 *    LPSTORAGE  -  if successful, NULL otherwise
 */
STDAPI_(LPSTORAGE) OleStdCreateTempStorage(BOOL fUseMemory, DWORD grfMode)
{
     LPSTORAGE   lpstg;
     HRESULT     hrErr;
     DWORD       reserved = 0;

     if (fUseMemory) {
           lpstg = OleStdCreateStorageOnHGlobal(
                       NULL,  /* auto allocate */
                       TRUE,  /* delete on release */
                       grfMode
           );
     } else {
           /* allocate a temp docfile that will delete on last release */
           hrErr = StgCreateDocfile(
                       NULL,
                       grfMode | STGM_DELETEONRELEASE | STGM_CREATE,
                       reserved,
                       &lpstg
           );
           if (hrErr != NOERROR)
                 return NULL;
     }
     return lpstg;
}


/* OleStdGetOleObjectData
** ----------------------
**    Render CF_EMBEDSOURCE/CF_EMBEDDEDOBJECT data on an TYMED_ISTORAGE
**    medium by asking the object to save into the storage.
**    the object must support IPersistStorage.
**
**    if lpMedium->tymed == TYMED_NULL, then a delete-on-release
**    storage is allocated (either file-based or memory-base depending
**    the value of fUseMemory). this is useful to support an
**    IDataObject::GetData call where the callee must allocate the
**    medium.
```

```
**
**    if lpMedium->tymed == TYMED_ISTORAGE, then the data is writen
**    into the passed in IStorage. this is useful to support an
**    IDataObject::GetDataHere call where the caller has allocated his
**    own IStorage.
*/
STDAPI OleStdGetOleObjectData(
           LPPERSISTSTORAGE        lpPStg,
           LPFORMATETC             lpformatetc,
           LPSTGMEDIUM             lpMedium,
           BOOL                    fUseMemory
)
{
     LPSTORAGE   lpstg = NULL;
     DWORD       reserved = 0;
     SCODE       sc = S_OK;
     HRESULT     hrErr;

     lpMedium->pUnkForRelease = NULL;

     if (lpMedium->tymed == TYMED_NULL) {

           if (lpformatetc->tymed & TYMED_ISTORAGE) {

                 /* allocate a temp docfile that will delete on last release
*/
                 lpstg = OleStdCreateTempStorage(
                             TRUE /*fUseMemory*/,
                             STGM_READWRITE | STGM_TRANSACTED |
STGM_SHARE_EXCLUSIVE
                 );
                 if (!lpstg)
                       return ResultFromScode(E_OUTOFMEMORY);

                 lpMedium->u.pstg = lpstg;
                 lpMedium->tymed = TYMED_ISTORAGE;
                 lpMedium->pUnkForRelease = NULL;
           } else {
                 return ResultFromScode(DATA_E_FORMATETC);
           }
     } else if (lpMedium->tymed == TYMED_ISTORAGE) {
           lpMedium->tymed = TYMED_ISTORAGE;
     } else {
           return ResultFromScode(DATA_E_FORMATETC);
     }

     // OLE2NOTE: even if OleSave returns an error you should still call
     // SaveCompleted.

     OLEDBG_BEGIN2("OleSave called\r\n")
     hrErr = OleSave(lpPStg, lpMedium->u.pstg, FALSE /* fSameAsLoad */);
     OLEDBG_END2

     if (hrErr != NOERROR) {
           OleDbgOutHResult("WARNING: OleSave returned", hrErr);
           sc = GetScode(hrErr);
     }
     OLEDBG_BEGIN2("IPersistStorage::SaveCompleted called\r\n")
     hrErr = lpPStg->lpVtbl->SaveCompleted(lpPStg, NULL);
     OLEDBG_END2

     if (hrErr != NOERROR) {
           OleDbgOutHResult("WARNING: SaveCompleted returned",hrErr);
           if (sc == S_OK)
                 sc = GetScode(hrErr);
     }

     return ResultFromScode(sc);
}


STDAPI OleStdGetLinkSourceData(
           LPMONIKER           lpmk,
           LPCLSID             lpClsID,
           LPFORMATETC         lpformatetc,
           LPSTGMEDIUM         lpMedium
)
{
     LPSTREAM    lpstm = NULL;
     DWORD       reserved = 0;
     HRESULT     hrErr;

     if (lpMedium->tymed == TYMED_NULL) {
           if (lpformatetc->tymed & TYMED_ISTREAM) {
                 hrErr = CreateStreamOnHGlobal(
                             NULL, /* auto allocate */
                             TRUE, /* delete on release */
                             (LPSTREAM FAR*)&lpstm
                 );
                 if (hrErr != NOERROR) {
                       lpMedium->pUnkForRelease = NULL;
                       return ResultFromScode(E_OUTOFMEMORY);
                 }
                 lpMedium->u.pstm = lpstm;
                 lpMedium->tymed = TYMED_ISTREAM;
                 lpMedium->pUnkForRelease = NULL;
           } else {
                 lpMedium->pUnkForRelease = NULL;
                 return ResultFromScode(DATA_E_FORMATETC);
           }
     } else {
           if (lpMedium->tymed == TYMED_ISTREAM) {
                 lpMedium->tymed = TYMED_ISTREAM;
                 lpMedium->u.pstm = lpMedium->u.pstm;
                 lpMedium->pUnkForRelease = NULL;
           } else {
                 lpMedium->pUnkForRelease = NULL;
                 return ResultFromScode(DATA_E_FORMATETC);
           }
     }

     hrErr = OleSaveToStream((LPPERSISTSTREAM)lpmk, lpMedium->u.pstm);
     if (hrErr != NOERROR) return hrErr;
     return WriteClassStm(lpMedium->u.pstm, lpClsID);
}

/*
 * OleStdGetObjectDescriptorData
 *
 * Purpose:
 *  Fills and returns a OBJECTDESCRIPTOR structure.
 *  See OBJECTDESCRIPTOR for more information.
 *
 * Parameters:
 *  clsid           CLSID   CLSID of object being transferred
 *  dwDrawAspect    DWORD   Display Aspect of object
 *  sizel           SIZEL   Size of object in HIMETRIC
 *  pointl          POINTL  Offset from upper-left corner of object where
mouse went
 *                          down for drag. Meaningful only when drag-drop is
used.
 *  dwStatus        DWORD   OLEMISC flags
 *  lpszFullUserTypeName  LPSTR Full User Type Name
 *  lpszSrcOfCopy   LPSTR   Source of Copy
 *
 * Return Value:
 *  HBGLOBAL         Handle to OBJECTDESCRIPTOR structure.
 */
STDAPI_(HGLOBAL) OleStdGetObjectDescriptorData(
     CLSID     clsid,
     DWORD     dwDrawAspect,
     SIZEL     sizel,
     POINTL    pointl,
     DWORD     dwStatus,
     LPSTR     lpszFullUserTypeName,
     LPSTR     lpszSrcOfCopy
)
{
     HGLOBAL            hMem = NULL;
     IBindCtx   FAR    *pbc = NULL;
     LPOBJECTDESCRIPTOR lpOD;
     DWORD              dwObjectDescSize, dwFullUserTypeNameLen,
dwSrcOfCopyLen;

     // Get the length of Full User Type Name; Add 1 for the null terminator
     dwFullUserTypeNameLen = lpszFullUserTypeName ?
lstrlen(lpszFullUserTypeName)+1 : 0;

     // Get the Source of Copy string and it's length; Add 1 for the null
terminator
     if (lpszSrcOfCopy)
        dwSrcOfCopyLen = lstrlen(lpszSrcOfCopy)+1;
     else {
        // No src moniker so use user type name as source string.
        lpszSrcOfCopy =  lpszFullUserTypeName;
        dwSrcOfCopyLen = dwFullUserTypeNameLen;
     }

     // Allocate space for OBJECTDESCRIPTOR and the additional string data
     dwObjectDescSize = sizeof(OBJECTDESCRIPTOR);
     hMem = GlobalAlloc(GMEM_MOVEABLE | GMEM_SHARE,
                    dwObjectDescSize + dwFullUserTypeNameLen +
dwSrcOfCopyLen);
     if (NULL == hMem)
           goto error;

     lpOD = (LPOBJECTDESCRIPTOR)GlobalLock(hMem);

     // Set the FullUserTypeName offset and copy the string
     if (lpszFullUserTypeName)
     {
           lpOD->dwFullUserTypeName = dwObjectDescSize;
           lstrcpy((LPSTR)lpOD+lpOD->dwFullUserTypeName ,
lpszFullUserTypeName);
     }
     else lpOD->dwFullUserTypeName = 0;  // zero offset indicates that
string is not present

     // Set the SrcOfCopy offset and copy the string
     if (lpszSrcOfCopy)
     {
           lpOD->dwSrcOfCopy = dwObjectDescSize + dwFullUserTypeNameLen;
           lstrcpy((LPSTR)lpOD+lpOD->dwSrcOfCopy , lpszSrcOfCopy);
     }
     else lpOD->dwSrcOfCopy = 0;  // zero offset indicates that string is
not present

     // Initialize the rest of the OBJECTDESCRIPTOR
     lpOD->cbSize       = dwObjectDescSize + dwFullUserTypeNameLen +
dwSrcOfCopyLen;
     lpOD->clsid        = clsid;
     lpOD->dwDrawAspect = dwDrawAspect;
     lpOD->sizel        = sizel;
     lpOD->pointl       = pointl;
     lpOD->dwStatus     = dwStatus;

     GlobalUnlock(hMem);
     return hMem;

error:
   if (hMem)
   {
        GlobalUnlock(hMem);
        GlobalFree(hMem);
   }
   return NULL;
}
```

```
/*
 * OleStdGetObjectDescriptorDataFromOleObject
 *
 * Purpose:
 *  Fills and returns a OBJECTDESCRIPTOR structure. Information for the
structure is
 *  obtained from an OLEOBJECT.
 *  See OBJECTDESCRIPTOR for more information.
 *
 * Parameters:
 *  lpOleObj        LPOLEOBJECT OleObject from which ONJECTDESCRIPTOR info
 *                  is obtained.
 *  lpszSrcOfCopy   LPSTR string to identify source of copy.
 *                  May be NULL in which case IOleObject::GetMoniker is
called
 *                  to get the moniker of the object. if the object is
loaded
 *                  as part of a data transfer document, then usually
 *                  lpOleClientSite==NULL is passed to OleLoad when loading
 *                  the object. in this case the IOleObject:GetMoniker call
 *                  will always fail (it tries to call back to the object's
 *                  client site). in this situation a non-NULL lpszSrcOfCopy
 *                  parameter should be passed.
 *  dwDrawAspect    DWORD   Display Aspect of object
 *  pointl          POINTL  Offset from upper-left corner of object where
 *                  mouse went down for drag. Meaningful only when drag-drop
 *                  is used.
 *  lpSizelHim      SIZEL   (optional) If the object is being scaled in its
 *                  container, then the container should pass the extents
 *                  that it is using to display the object.
 *                  May be NULL if the object is NOT being scaled. in this
 *                  case, IViewObject2::GetExtent will be called to get the
 *                  extents from the object.
 *
 * Return Value:
 *  HBGLOBAL         Handle to OBJECTDESCRIPTOR structure.
 */

STDAPI_(HGLOBAL) OleStdGetObjectDescriptorDataFromOleObject(
           LPOLEOBJECT lpOleObj,
           LPSTR       lpszSrcOfCopy,
           DWORD       dwDrawAspect,
           POINTL      pointl,
           LPSIZEL     lpSizelHim
)
{
     CLSID clsid;
     LPSTR lpszFullUserTypeName = NULL;
     LPMONIKER lpSrcMonikerOfCopy = NULL;
     HGLOBAL hObjDesc;
     IBindCtx  FAR  *pbc = NULL;
     HRESULT hrErr;
     SIZEL sizelHim;
     BOOL  fFreeSrcOfCopy = FALSE;
     LPOLELINK lpOleLink = (LPOLELINK)
```

```
              OleStdQueryInterface((LPUNKNOWN)lpOleObj,&IID_IOleLink);
     LPVIEWOBJECT2 lpViewObj2 = (LPVIEWOBJECT2)
                 OleStdQueryInterface((LPUNKNOWN)lpOleObj,
&IID_IViewObject2);
     BOOL  fIsLink = (lpOleLink ? TRUE : FALSE);
     char  szLinkedTypeFmt[80];
     LPSTR lpszBuf = NULL;
     DWORD dwStatus = 0;

     // Get CLSID
     OLEDBG_BEGIN2("IOleObject::GetUserClassID called\r\n")
     hrErr = lpOleObj->lpVtbl->GetUserClassID(lpOleObj, &clsid);
     OLEDBG_END2
     if (hrErr != NOERROR)
           clsid = CLSID_NULL;

     // Get FullUserTypeName
     OLEDBG_BEGIN2("IOleObject::GetUserType called\r\n")
     hrErr = lpOleObj->lpVtbl->GetUserType(
                 lpOleObj,
                 USERCLASSTYPE_FULL,
                 (LPSTR FAR*)&lpszFullUserTypeName
     );
     OLEDBG_END2

// REVIEW: added IDS_OLE2UILINKEDTYPE to strings.rc
     /* if object is a link, then expand usertypename to be "Linked %s" */
     if (fIsLink && lpszFullUserTypeName) {
           if (0 == LoadString(ghInst, (UINT)IDS_OLE2UIPASTELINKEDTYPE,
                                   (LPSTR)szLinkedTypeFmt,
sizeof(szLinkedTypeFmt)))
                 lstrcpy(szLinkedTypeFmt, (LPSTR)"Linked %s");
           lpszBuf = OleStdMalloc(
                       lstrlen(lpszFullUserTypeName)
+lstrlen(szLinkedTypeFmt)+1);
           if (lpszBuf) {
                 wsprintf(lpszBuf, szLinkedTypeFmt, lpszFullUserTypeName);
                 OleStdFreeString(lpszFullUserTypeName, NULL);
                 lpszFullUserTypeName = lpszBuf;
           }
     }

     /* Get Source Of Copy
     **    if the object is an embedding, then get the object's moniker
     **    if the object is a link, then get the link source moniker
     */
     if (fIsLink) {

           OLEDBG_BEGIN2("IOleLink::GetSourceDisplayName called\r\n")
           hrErr = lpOleLink->lpVtbl->GetSourceDisplayName(

                       lpOleLink, &lpszSrcOfCopy );
           OLEDBG_END2
           fFreeSrcOfCopy = TRUE;
```

```
    } else {

        if (lpszSrcOfCopy == NULL) {
            OLEDBG_BEGIN2("IOleObject::GetMoniker called\r\n")
            hrErr = lpOleObj->lpVtbl->GetMoniker(
                    lpOleObj,
                    OLEGETMONIKER_TEMPFORUSER,
                    OLEWHICHMK_OBJFULL,
                    (LPMONIKER FAR*)&lpSrcMonikerOfCopy
            );
            OLEDBG_END2
            if (hrErr == NOERROR)
            {
                CreateBindCtx(0, (LPBC FAR*)&pbc);
                lpSrcMonikerOfCopy->lpVtbl->GetDisplayName(
                        lpSrcMonikerOfCopy, pbc, NULL,
&lpszSrcOfCopy);
                pbc->lpVtbl->Release(pbc);
                fFreeSrcOfCopy = TRUE;
            }
        }
    }

    // Get SIZEL
    if (lpSizelHim) {
        // Use extents passed by the caller
        sizelHim = *lpSizelHim;
    } else if (lpViewObj2) {
        // Get the current extents from the object
        OLEDBG_BEGIN2("IViewObject2::GetExtent called\r\n")
        hrErr = lpViewObj2->lpVtbl->GetExtent(
                lpViewObj2,
                dwDrawAspect,
                -1,     /*lindex*/
                NULL,   /*ptd*/
                (LPSIZEL)&sizelHim
        );
        OLEDBG_END2
        if (hrErr != NOERROR)
            sizelHim.cx = sizelHim.cy = 0;
    } else {
        sizelHim.cx = sizelHim.cy = 0;
    }

    // Get DWSTATUS
    OLEDBG_BEGIN2("IOleObject::GetMiscStatus called\r\n")
    hrErr = lpOleObj->lpVtbl->GetMiscStatus(
                lpOleObj,
                dwDrawAspect,
                &dwStatus
    );
    OLEDBG_END2
    if (hrErr != NOERROR)
        dwStatus = 0;
```

```
     // Get OBJECTDESCRIPTOR
     hObjDesc = OleStdGetObjectDescriptorData(
                 clsid,
                 dwDrawAspect,
                 sizelHim,
                 pointl,
                 dwStatus,
                 lpszFullUserTypeName,
                 lpszSrcOfCopy
     );
     if (! hObjDesc)
           goto error;

     // Clean up
     if (lpszFullUserTypeName)
           OleStdFreeString(lpszFullUserTypeName, NULL);
     if (fFreeSrcOfCopy && lpszSrcOfCopy)
           OleStdFreeString(lpszSrcOfCopy, NULL);
     if (lpSrcMonikerOfCopy)
           OleStdRelease((LPUNKNOWN)lpSrcMonikerOfCopy);
     if (lpOleLink)
           OleStdRelease((LPUNKNOWN)lpOleLink);
     if (lpViewObj2)
           OleStdRelease((LPUNKNOWN)lpViewObj2);

     return hObjDesc;

error:
     if (lpszFullUserTypeName)
           OleStdFreeString(lpszFullUserTypeName, NULL);
     if (fFreeSrcOfCopy && lpszSrcOfCopy)
           OleStdFreeString(lpszSrcOfCopy, NULL);
     if (lpSrcMonikerOfCopy)
           OleStdRelease((LPUNKNOWN)lpSrcMonikerOfCopy);
     if (lpOleLink)
           OleStdRelease((LPUNKNOWN)lpOleLink);
     if (lpViewObj2)
           OleStdRelease((LPUNKNOWN)lpViewObj2);

     return NULL;
}

/*
 * OleStdFillObjectDescriptorFromData
 *
 * Purpose:
 *  Fills and returns a OBJECTDESCRIPTOR structure. The source object will
 *  offer CF_OBJECTDESCRIPTOR if it is an OLE2 object, CF_OWNERLINK if it
 *  is an OLE1 object, or CF_FILENAME if it has been copied to the clipboard
 *  by FileManager.
 *
 * Parameters:
 *  lpDataObject    LPDATAOBJECT Source object
 *  lpmedium        LPSTGMEDIUM  Storage medium
 *  lpcfFmt         CLIPFORMAT FAR * Format offered by lpDataObject
 *                   (OUT parameter)
 *
 * Return Value:
 *  HBGLOBAL         Handle to OBJECTDESCRIPTOR structure.
 */

STDAPI_(HGLOBAL) OleStdFillObjectDescriptorFromData(
           LPDATAOBJECT     lpDataObject,
           LPSTGMEDIUM      lpmedium,
           CLIPFORMAT FAR* lpcfFmt
)
{
     CLSID              clsid;
     SIZEL              sizelHim;
     POINTL             pointl;
     LPSTR              lpsz, szFullUserTypeName, szSrcOfCopy, szClassName,
szDocName, szItemName;
     int                nClassName, nDocName, nItemName, nFullUserTypeName;
     LPSTR              szBuf = NULL;
     HGLOBAL            hMem = NULL;
     HKEY               hKey = NULL;
     LPMALLOC           pIMalloc = NULL;
     DWORD              dw = OLEUI_CCHKEYMAX;
     HGLOBAL            hObjDesc;
     HRESULT            hrErr;


     // GetData CF_OBJECTDESCRIPTOR format from the object on the clipboard.
     // Only OLE 2 objects on the clipboard will offer CF_OBJECTDESCRIPTOR
     if (hMem = OleStdGetData(
                 lpDataObject,
                 (CLIPFORMAT) cfObjectDescriptor,
                 NULL,
                 DVASPECT_CONTENT,
                 lpmedium))
     {
           *lpcfFmt = cfObjectDescriptor;
           return hMem;  // Don't drop to clean up at the end of this
function
     }
     // If CF_OBJECTDESCRIPTOR is not available, i.e. if this is not an OLE2
object,
     //     check if this is an OLE 1 object. OLE 1 objects will offer
CF_OWNERLINK
     else if (hMem = OleStdGetData(
                       lpDataObject,
                       (CLIPFORMAT) cfOwnerLink,
                       NULL,
                       DVASPECT_CONTENT,
                       lpmedium))
     {
           *lpcfFmt = cfOwnerLink;
           // CF_OWNERLINK contains null-terminated strings for class name,
document name
```

```
           // and item name with two null terminating characters at the end
           szClassName = (LPSTR)GlobalLock(hMem);
           nClassName = lstrlen(szClassName);
           szDocName   = szClassName + nClassName + 1;
           nDocName   = lstrlen(szDocName);
           szItemName  = szDocName + nDocName + 1;
           nItemName  =  lstrlen(szItemName);

           hrErr = CoGetMalloc(MEMCTX_TASK, &pIMalloc);
           if (hrErr != NOERROR)
                 goto error;

           // Find FullUserTypeName from Registration database using class
name
           if (RegOpenKey(HKEY_CLASSES_ROOT, NULL, &hKey) != ERROR_SUCCESS)
              goto error;

           // Allocate space for szFullUserTypeName & szSrcOfCopy. Maximum
length of FullUserTypeName
           // is OLEUI_CCHKEYMAX. SrcOfCopy is constructed by concatenating
FullUserTypeName, Document
           // Name and ItemName separated by spaces.
           szBuf = (LPSTR)pIMalloc->lpVtbl->Alloc(pIMalloc,

     (DWORD)2*OLEUI_CCHKEYMAX+nDocName+nItemName+4);
           if (NULL == szBuf)
                 goto error;
           szFullUserTypeName = szBuf;
           szSrcOfCopy = szFullUserTypeName+OLEUI_CCHKEYMAX+1;

           // Get FullUserTypeName
           if (RegQueryValue(hKey, NULL, szFullUserTypeName, &dw) !=
ERROR_SUCCESS)
              goto error;

           // Build up SrcOfCopy string from FullUserTypeName, DocumentName
& ItemName
           lpsz = szSrcOfCopy;
           lstrcpy(lpsz, szFullUserTypeName);
           nFullUserTypeName = lstrlen(szFullUserTypeName);
           lpsz[nFullUserTypeName]=' ';
           lpsz += nFullUserTypeName+1;
           lstrcpy(lpsz, szDocName);
           lpsz[nDocName] = ' ';
           lpsz += nDocName+1;
           lstrcpy(lpsz, szItemName);

           sizelHim.cx = sizelHim.cy = 0;
           pointl.x = pointl.y = 0;

           CLSIDFromProgID(szClassName, &clsid);

           hObjDesc = OleStdGetObjectDescriptorData(
                        clsid,
                        DVASPECT_CONTENT,
                    sizelHim,
                    pointl,
                    0,
                    szFullUserTypeName,
                    szSrcOfCopy
            );
            if (!hObjDesc)
               goto error;
        }
        // Check if object is CF_FILENAME
        else if (hMem = OleStdGetData(
                    lpDataObject,
                    (CLIPFORMAT) cfFileName,
                    NULL,
                    DVASPECT_CONTENT,
                    lpmedium))
        {
             *lpcfFmt = cfFileName;
             lpsz = (LPSTR)GlobalLock(hMem);
             hrErr = GetClassFile(lpsz, &clsid);

             /* OLE2NOTE: if the file does not have an OLE class
             **    associated, then use the OLE 1 Packager as the class of
             **    the object to be created. this is the behavior of
             **    OleCreateFromData API
             */
             if (hrErr != NOERROR)
                  CLSIDFromProgID("Package", &clsid);
             sizelHim.cx = sizelHim.cy = 0;
             pointl.x = pointl.y = 0;

             hrErr = CoGetMalloc(MEMCTX_TASK, &pIMalloc);
             if (hrErr != NOERROR)
                  goto error;
             szBuf = (LPSTR)pIMalloc->lpVtbl->Alloc(pIMalloc,
(DWORD)OLEUI_CCHKEYMAX);
             if (NULL == szBuf)
                  goto error;

             OleStdGetUserTypeOfClass(&clsid, szBuf, OLEUI_CCHKEYMAX, NULL);

             hObjDesc = OleStdGetObjectDescriptorData(
                    clsid,
                    DVASPECT_CONTENT,
                    sizelHim,
                    pointl,
                    0,
                    szBuf,
                    lpsz
            );
            if (!hObjDesc)
               goto error;
        }
        else goto error;
```

```
      // Clean up
      if (szBuf)
            pIMalloc->lpVtbl->Free(pIMalloc, (LPVOID)szBuf);
      if (pIMalloc)
            pIMalloc->lpVtbl->Release(pIMalloc);
      if (hMem)
      {
            GlobalUnlock(hMem);
            GlobalFree(hMem);
      }
      if (hKey)
            RegCloseKey(hKey);
      return hObjDesc;

error:
   if (szBuf)
        pIMalloc->lpVtbl->Free(pIMalloc, (LPVOID)szBuf);
   if (pIMalloc)
        pIMalloc->lpVtbl->Release(pIMalloc);
      if (hMem)
      {
            GlobalUnlock(hMem);
            GlobalFree(hMem);
      }
      if (hKey)
            RegCloseKey(hKey);
      return NULL;
}


#if defined( OBSOLETE )

/*************************************************************************
** The following API's have been converted into macros:
**          OleStdQueryOleObjectData
**          OleStdQueryLinkSourceData
**          OleStdQueryObjectDescriptorData
**          OleStdQueryFormatMedium
**          OleStdCopyMetafilePict
**          OleStdGetDropEffect
**
**    These macros are defined in olestd.h
*************************************************************************/

STDAPI OleStdQueryOleObjectData(LPFORMATETC lpformatetc)
{
     if (lpformatetc->tymed & TYMED_ISTORAGE) {
           return NOERROR;
     } else {
           return ResultFromScode(DATA_E_FORMATETC);
     }
}
```

```
STDAPI OleStdQueryLinkSourceData(LPFORMATETC lpformatetc)
{
     if (lpformatetc->tymed & TYMED_ISTREAM) {
           return NOERROR;
     } else {
           return ResultFromScode(DATA_E_FORMATETC);
     }
}


STDAPI OleStdQueryObjectDescriptorData(LPFORMATETC lpformatetc)
{
     if (lpformatetc->tymed & TYMED_HGLOBAL) {
           return NOERROR;
     } else {
           return ResultFromScode(DATA_E_FORMATETC);
     }
}


STDAPI OleStdQueryFormatMedium(LPFORMATETC lpformatetc, TYMED tymed)
{
     if (lpformatetc->tymed & tymed) {
           return NOERROR;
     } else {
           return ResultFromScode(DATA_E_FORMATETC);
     }
}


/*
 * OleStdCopyMetafilePict()
 *
 * Purpose:
 *    Make an independent copy of a MetafilePict
 * Parameters:
 *
 * Return Value:
 *    TRUE if successful, else FALSE.
 */
STDAPI_(BOOL) OleStdCopyMetafilePict(HANDLE hpictin, HANDLE FAR* phpictout)
{
     HANDLE hpictout;
     LPMETAFILEPICT ppictin, ppictout;

     if (hpictin == NULL || phpictout == NULL) {
           OleDbgAssert(hpictin == NULL || phpictout == NULL);
           return FALSE;
     }

     *phpictout = NULL;

     if ((ppictin = (LPMETAFILEPICT)GlobalLock(hpictin)) == NULL) {
           return FALSE;
     }
     hpictout = GlobalAlloc(GHND|GMEM_SHARE, sizeof(METAFILEPICT));

     if (hpictout && (ppictout = (LPMETAFILEPICT)GlobalLock(hpictout))){
           ppictout->hMF  = CopyMetaFile(ppictin->hMF, NULL);
           ppictout->xExt = ppictin->xExt;
           ppictout->yExt = ppictin->yExt;
           ppictout->mm   = ppictin->mm;
           GlobalUnlock(hpictout);
     }

     *phpictout = hpictout;

     return TRUE;

}


/* OleStdGetDropEffect
** -------------------
**
** Convert a keyboard state into a DROPEFFECT.
**
** returns the DROPEFFECT value derived from the key state.
**    the following is the standard interpretation:
**          no modifier -- Default Drop     (NULL is returned)
**          CTRL        -- DROPEFFECT_COPY
**          SHIFT       -- DROPEFFECT_MOVE
**          CTRL-SHIFT  -- DROPEFFECT_LINK
**
**    Default Drop: this depends on the type of the target application.
**    this is re-interpretable by each target application. a typical
**    interpretation is if the drag is local to the same document
**    (which is source of the drag) then a MOVE operation is
**    performed. if the drag is not local, then a COPY operation is
**    performed.
*/
STDAPI_(DWORD) OleStdGetDropEffect( DWORD grfKeyState )
{

     if (grfKeyState & MK_CONTROL) {

           if (grfKeyState & MK_SHIFT)
                 return DROPEFFECT_LINK;
           else
                 return DROPEFFECT_COPY;

     } else if (grfKeyState & MK_SHIFT)
           return DROPEFFECT_MOVE;

     return 0;    // no modifier -- do default operation
}
#endif  // OBSOLETE
```

```
/*
 * OleStdGetMetafilePictFromOleObject()
 *
 * Purpose:
 *      Generate a MetafilePict by drawing the OLE object.
 * Parameters:
 *  lpOleObj        LPOLEOBJECT pointer to OLE Object
 *  dwDrawAspect    DWORD   Display Aspect of object
 *  lpSizelHim      SIZEL   (optional) If the object is being scaled in its
 *                  container, then the container should pass the extents
 *                  that it is using to display the object.
 *                  May be NULL if the object is NOT being scaled. in this
 *                  case, IViewObject2::GetExtent will be called to get the
 *                  extents from the object.
 *  ptd             TARGETDEVICE FAR*   (optional) target device to render
 *                  metafile for. May be NULL.
 *
 * Return Value:
 *    HANDLE    -- handle of allocated METAFILEPICT
 */
STDAPI_(HANDLE) OleStdGetMetafilePictFromOleObject(
           LPOLEOBJECT         lpOleObj,
           DWORD               dwDrawAspect,
           LPSIZEL             lpSizelHim,
           DVTARGETDEVICE FAR* ptd
)
{
     LPVIEWOBJECT2 lpViewObj2 = NULL;
     HDC hDC;
     HMETAFILE hmf;
     HANDLE hMetaPict;
     LPMETAFILEPICT lpPict;
     RECT rcHim;
     RECTL rclHim;
     SIZEL sizelHim;
     HRESULT hrErr;
     SIZE size;
     POINT point;

     lpViewObj2 = (LPVIEWOBJECT2)OleStdQueryInterface(
                 (LPUNKNOWN)lpOleObj, &IID_IViewObject2);
     if (! lpViewObj2)
           return NULL;

     // Get SIZEL
     if (lpSizelHim) {
           // Use extents passed by the caller
           sizelHim = *lpSizelHim;
     } else {
           // Get the current extents from the object
           OLEDBG_BEGIN2("IViewObject2::GetExtent called\r\n")
           hrErr = lpViewObj2->lpVtbl->GetExtent(
                       lpViewObj2,
                       dwDrawAspect,
                       -1,     /*lindex*/
                       ptd,    /*ptd*/
                       (LPSIZEL)&sizelHim
           );
           OLEDBG_END2
           if (hrErr != NOERROR)
                 sizelHim.cx = sizelHim.cy = 0;
     }

     hDC = CreateMetaFile(NULL);

     rclHim.left     = 0;
     rclHim.top      = 0;
     rclHim.right    = sizelHim.cx;
     rclHim.bottom   = sizelHim.cy;

     rcHim.left      = (int)rclHim.left;
     rcHim.top       = (int)rclHim.top;
     rcHim.right     = (int)rclHim.right;
     rcHim.bottom    = (int)rclHim.bottom;

     SetWindowOrgEx(hDC, rcHim.left, rcHim.top, &point);
     SetWindowExtEx(hDC, rcHim.right-rcHim.left, rcHim.bottom-
rcHim.top,&size);

     OLEDBG_BEGIN2("IViewObject::Draw called\r\n")
     hrErr = lpViewObj2->lpVtbl->Draw(
                 lpViewObj2,
                 dwDrawAspect,
                 -1,
                 NULL,
                 ptd,
                 NULL,
                 hDC,
                 (LPRECTL)&rclHim,
                 (LPRECTL)&rclHim,
                 NULL,
                 0
     );
     OLEDBG_END2

     OleStdRelease((LPUNKNOWN)lpViewObj2);
     if (hrErr != NOERROR) {
           OleDbgOutHResult("IViewObject::Draw returned", hrErr);
     }

     hmf = CloseMetaFile(hDC);

     hMetaPict = GlobalAlloc(GHND|GMEM_SHARE, sizeof(METAFILEPICT));

     if (hMetaPict && (lpPict = (LPMETAFILEPICT)GlobalLock(hMetaPict))){
           lpPict->hMF  = hmf;
           lpPict->xExt = (int)sizelHim.cx ;
           lpPict->yExt = (int)sizelHim.cy ;
           lpPict->mm   = MM_ANISOTROPIC;
            GlobalUnlock(hMetaPict);
      }

      return hMetaPict;
}


/* Call Release on the object that is expected to go away.
**      if the refcnt of the object did no go to 0 then give a debug
message.
*/
STDAPI_(ULONG) OleStdVerifyRelease(LPUNKNOWN lpUnk, LPSTR lpszMsg)
{
      ULONG cRef;

      cRef = lpUnk->lpVtbl->Release(lpUnk);

#if defined( _DEBUG )
      if (cRef != 0) {
            char szBuf[80];
            if (lpszMsg)
                  MessageBox(NULL, lpszMsg, NULL, MB_ICONEXCLAMATION |
MB_OK);
            wsprintf(
                        (LPSTR)szBuf,
                        "refcnt (%ld) != 0 after object (0x%lx) release\n",
                        cRef,
                        lpUnk
            );
            if (lpszMsg)
                  OleDbgOut1(lpszMsg);
            OleDbgOut1((LPSTR)szBuf);
            OleDbgAssertSz(cRef == 0, (LPSTR)szBuf);
      } else {
            char szBuf[80];
            wsprintf(
                        (LPSTR)szBuf,
                        "refcnt = 0 after object (0x%lx) release\n", lpUnk
            );
            OleDbgOut4((LPSTR)szBuf);
      }
#endif
      return cRef;
}


/* Call Release on the object that is NOT necessarily expected to go away.
*/
STDAPI_(ULONG) OleStdRelease(LPUNKNOWN lpUnk)
{
      ULONG cRef;

      cRef = lpUnk->lpVtbl->Release(lpUnk);
```

```
#if defined( _DEBUG )
     {
           char szBuf[80];
           wsprintf(
                       (LPSTR)szBuf,
                       "refcnt = %ld after object (0x%lx) release\n",
                       cRef,
                       lpUnk
           );
           OleDbgOut4((LPSTR)szBuf);
     }
#endif
     return cRef;
}


/* OleStdInitVtbl
** --------------
**
**    Initialize an interface Vtbl to ensure that there are no NULL
**    function pointers in the Vtbl. All entries in the Vtbl are
**    set to a valid funtion pointer (OleStdNullMethod) that issues
**        debug assert message (message box) and returns E_NOTIMPL if
called.
**
**    NOTE: this funtion does not initialize the Vtbl with usefull
**    function pointers, only valid function pointers to avoid the
**    horrible run-time crash when a call is made through the Vtbl with
**    a NULL function pointer. this API is only necessary when
**    initializing the Vtbl's in C. C++ guarantees that all interface
**    functions (in C++ terms -- pure virtual functions) are implemented.
*/

STDAPI_(void) OleStdInitVtbl(LPVOID lpVtbl, UINT nSizeOfVtbl)
{
     LPVOID FAR* lpFuncPtrArr = (LPVOID FAR*)lpVtbl;
     UINT nMethods = nSizeOfVtbl/sizeof(VOID FAR*);
     UINT i;

     for (i = 0; i < nMethods; i++) {
           lpFuncPtrArr[i] = OleStdNullMethod;
     }
}


/* OleStdCheckVtbl
** ---------------
**
**    Check if all entries in the Vtbl are properly initialized with
**    valid function pointers. If any entries are either NULL or
**    OleStdNullMethod, then this function returns FALSE. If compiled
**    for _DEBUG this function reports which function pointers are
**    invalid.
**
**    RETURNS:  TRUE if all entries in Vtbl are valid
**                              FALSE otherwise.
*/

STDAPI_(BOOL) OleStdCheckVtbl(LPVOID lpVtbl, UINT nSizeOfVtbl, LPSTR
lpszIface)
{
     LPVOID FAR* lpFuncPtrArr = (LPVOID FAR*)lpVtbl;
     UINT nMethods = nSizeOfVtbl/sizeof(VOID FAR*);
     UINT i;
     BOOL fStatus = TRUE;
     int nChar = 0;

     for (i = 0; i < nMethods; i++) {
           if (lpFuncPtrArr[i] == NULL || lpFuncPtrArr[i] ==
OleStdNullMethod) {
#if defined( _DEBUG )
                 char szBuf[256];
                 wsprintf(szBuf, "%s::method# %d NOT valid!", lpszIface, i);
                 OleDbgOut1((LPSTR)szBuf);
#endif
                 fStatus = FALSE;
           }
     }
     return fStatus;
}


/* OleStdNullMethod
** ----------------
**    Dummy method used by OleStdInitVtbl to initialize an interface
**    Vtbl to ensure that there are no NULL function pointers in the
**    Vtbl. All entries in the Vtbl are set to this function. this
**    function issues a debug assert message (message box) and returns
**    E_NOTIMPL if called. If all is done properly, this function will
**    NEVER be called!
*/
STDMETHODIMP OleStdNullMethod(LPUNKNOWN lpThis)
{
     MessageBox(
                 NULL,
                 "ERROR: INTERFACE METHOD NOT IMPLEMENTED!\r\n",
                 NULL,
                 MB_SYSTEMMODAL | MB_ICONHAND | MB_OK
     );

     return ResultFromScode(E_NOTIMPL);
}



static BOOL  GetFileTimes(LPSTR lpszFileName, FILETIME FAR* pfiletime)
{
#ifdef WIN32
     WIN32_FIND_DATA fd;
     HANDLE hFind;
     hFind = FindFirstFile(lpszFileName,&fd);
     if (hFind == NULL || hFind == INVALID_HANDLE_VALUE) {
           return FALSE;
     }
     FindClose(hFind);
     *pfiletime = fd.ftLastWriteTime;
     return TRUE;
#else
     static char sz[256];
     static struct _find_t fileinfo;

     LSTRCPYN((LPSTR)sz, lpszFileName, sizeof(sz)-1);
     sz[sizeof(sz)-1]= '\0';
     AnsiToOem(sz, sz);
     return (_dos_findfirst(sz,_A_NORMAL|_A_HIDDEN|_A_SUBDIR|_A_SYSTEM,
                              (struct _find_t *)&fileinfo) == 0 &&

     CoDosDateTimeToFileTime(fileinfo.wr_date,fileinfo.wr_time,pfiletime));
#endif
}



/* OleStdRegisterAsRunning
** -----------------------
**    Register a moniker in the RunningObjectTable.
**    if there is an existing registration (*lpdwRegister!=NULL), then
**    first revoke that registration.
**
**    new dwRegister key is returned via *lpdwRegister parameter.
*/
STDAPI_(void) OleStdRegisterAsRunning(LPUNKNOWN lpUnk, LPMONIKER lpmkFull,
DWORD FAR* lpdwRegister)
{
     LPRUNNINGOBJECTTABLE lpROT;
     HRESULT hrErr;
     DWORD dwOldRegister = *lpdwRegister;

     OLEDBG_BEGIN2("OleStdRegisterAsRunning\r\n")

     OLEDBG_BEGIN2("GetRunningObjectTable called\r\n")
     hrErr = GetRunningObjectTable(0,(LPRUNNINGOBJECTTABLE FAR*)&lpROT);
     OLEDBG_END2

     if (hrErr == NOERROR) {

           /* register as running if a valid moniker is passed
           **
           ** OLE2NOTE: we deliberately register the new moniker BEFORE
           **    revoking the old moniker just in case the object
           **    currently has no external locks. if the object has no
           **    locks then revoking it from the running object table will
           **    cause the object's StubManager to initiate shutdown of
           **    the object.
           */
           if (lpmkFull) {

                 OLEDBG_BEGIN2("IRunningObjectTable::Register called\r\n")
                 lpROT->lpVtbl->Register(lpROT, 0,
lpUnk,lpmkFull,lpdwRegister);
                 OLEDBG_END2

#if defined(_DEBUG)
                 {
                       char szBuf[512];
                       LPSTR lpszDisplay;
                       LPBC lpbc;

                       CreateBindCtx(0, (LPBC FAR*)&lpbc);
                       lpmkFull->lpVtbl->GetDisplayName(
                                   lpmkFull,
                                   lpbc,
                                   NULL,
                                   (LPSTR FAR*)&lpszDisplay
                       );
                       OleStdRelease((LPUNKNOWN)lpbc);
                       wsprintf(
                                   szBuf,
                                   "Moniker '%s' REGISTERED as [0x%lx] in
ROT\r\n",
                                   lpszDisplay,
                                   *lpdwRegister
                       );
                       OleDbgOut2(szBuf);
                       OleStdFreeString(lpszDisplay, NULL);
                 }
#endif  // _DEBUG

           }

           // if already registered, revoke
           if (dwOldRegister != 0) {

#if defined(_DEBUG)
                 {
                       char szBuf[512];

                       wsprintf(
                                   szBuf,
                                   "Moniker [0x%lx] REVOKED from ROT\r\n",
                                   dwOldRegister
                       );
                       OleDbgOut2(szBuf);
                 }
#endif  // _DEBUG

                 OLEDBG_BEGIN2("IRunningObjectTable::Revoke called\r\n")
                 lpROT->lpVtbl->Revoke(lpROT, dwOldRegister);
                 OLEDBG_END2
           }

           OleStdRelease((LPUNKNOWN)lpROT);
     } else {
           OleDbgAssertSz(
                       lpROT != NULL,
                       "OleStdRegisterAsRunning: GetRunningObjectTable
FAILED\r\n"
           );
     }

     OLEDBG_END2
}



/* OleStdRevokeAsRunning
** ---------------------
**    Revoke a moniker from the RunningObjectTable if there is an
**    existing registration (*lpdwRegister!=NULL).
**
**    *lpdwRegister parameter will be set to NULL.
*/
STDAPI_(void) OleStdRevokeAsRunning(DWORD FAR* lpdwRegister)
{
     LPRUNNINGOBJECTTABLE lpROT;
     HRESULT hrErr;

     OLEDBG_BEGIN2("OleStdRevokeAsRunning\r\n")

     // if still registered, then revoke
     if (*lpdwRegister != 0) {

           OLEDBG_BEGIN2("GetRunningObjectTable called\r\n")
           hrErr = GetRunningObjectTable(0,(LPRUNNINGOBJECTTABLE
FAR*)&lpROT);
           OLEDBG_END2

           if (hrErr == NOERROR) {

#if defined(_DEBUG)
                 {
                       char szBuf[512];

                       wsprintf(
                                   szBuf,
                                   "Moniker [0x%lx] REVOKED from ROT\r\n",
                                   *lpdwRegister
                       );
                       OleDbgOut2(szBuf);
                 }
#endif  // _DEBUG

                 OLEDBG_BEGIN2("IRunningObjectTable::Revoke called\r\n")
                 lpROT->lpVtbl->Revoke(lpROT, *lpdwRegister);
                 OLEDBG_END2

                 *lpdwRegister = 0;

                 OleStdRelease((LPUNKNOWN)lpROT);
           } else {
                 OleDbgAssertSz(
                             lpROT != NULL,
                             "OleStdRevokeAsRunning: GetRunningObjectTable
FAILED\r\n"
                 );
           }
     }
     OLEDBG_END2
}


/* OleStdNoteFileChangeTime
** ------------------------
**    Note the time a File-Based object has been saved in the
**    RunningObjectTable. These change times are used as the basis for
**    IOleObject::IsUpToDate.
**    It is important to set the time of the file-based object
**    following a save operation to exactly the time of the saved file.
**    this helps IOleObject::IsUpToDate to give the correct answer
**    after a file has been saved.
*/
STDAPI_(void) OleStdNoteFileChangeTime(LPSTR lpszFileName, DWORD dwRegister)
{
     if (dwRegister != 0) {

           LPRUNNINGOBJECTTABLE lprot;
           FILETIME filetime;

           if (GetFileTimes(lpszFileName, &filetime) &&
                 GetRunningObjectTable(0,&lprot) == NOERROR)
           {
                 lprot->lpVtbl->NoteChangeTime( lprot, dwRegister, &filetime
);
                 lprot->lpVtbl->Release(lprot);

                 OleDbgOut2("IRunningObjectTable::NoteChangeTime
called\r\n");
           }
     }
}


/* OleStdNoteObjectChangeTime
** --------------------------
**    Set the last change time of an object that is registered in the
**    RunningObjectTable. These change times are used as the basis for
**    IOleObject::IsUpToDate.
**
```

```
**    every time the object sends out a OnDataChange notification, it
**    should update the Time of last change in the ROT.
**
**    NOTE: this function set the change time to the current time.
*/
STDAPI_(void) OleStdNoteObjectChangeTime(DWORD dwRegister)
{
     if (dwRegister != 0) {

           LPRUNNINGOBJECTTABLE lprot;
           FILETIME filetime;

           if (GetRunningObjectTable(0,&lprot) == NOERROR)
           {
                 CoFileTimeNow( &filetime );
                 lprot->lpVtbl->NoteChangeTime( lprot, dwRegister, &filetime
);
                 lprot->lpVtbl->Release(lprot);

                 OleDbgOut2("IRunningObjectTable::NoteChangeTime
called\r\n");
           }

     }
}


/* OleStdCreateTempFileMoniker
** ---------------------------
**    return the next available FileMoniker that can be used as the
**    name of an untitled document.
**    the FileMoniker is built of the form:
**          <lpszPrefixString><number>
**      eg. "Outline1", "Outline2", etc.
**
**    The RunningObjectTable (ROT) is consulted to determine if a
**    FileMoniker is in use. If the name is in use then the number is
**    incremented and the ROT is checked again.
**
** Parameters:
**    LPSTR lpszPrefixString    - prefix used to build the name
**    UINT FAR* lpuUnique       - (IN-OUT) last used number.
**                                  this number is used to make the
**                                  name unique. on entry, the input
**                                  number is incremented. on output,
**                                  the number used is returned. this
**                                  number should be passed again
**                                  unchanged on the next call.
**    LPSTR lpszName            - (OUT) buffer used to build string.
**                                  caller must be sure buffer is large
**                                  enough to hold the generated string.
**    LPMONIKER FAR* lplpmk     - (OUT) next unused FileMoniker
**
** Returns:
**    void
```

```
**
** Comments:
**    This function is similar in spirit to the Windows API
**    CreateTempFileName.
*/
STDAPI_(void) OleStdCreateTempFileMoniker(
           LPSTR           lpszPrefixString,
           UINT FAR*       lpuUnique,
           LPSTR           lpszName,
           LPMONIKER FAR*  lplpmk
)
{
     LPRUNNINGOBJECTTABLE lpROT = NULL;
     UINT i = (lpuUnique != NULL ? *lpuUnique : 1);
     HRESULT hrErr;

     wsprintf(lpszName, "%s%d", lpszPrefixString, i++);
     CreateFileMoniker(lpszName, lplpmk);

     OLEDBG_BEGIN2("GetRunningObjectTable called\r\n")
     hrErr = GetRunningObjectTable(0,(LPRUNNINGOBJECTTABLE FAR*)&lpROT);
     OLEDBG_END2

     if (hrErr == NOERROR) {

           while (1) {
                 if (! *lplpmk)
                       break;  // failed to create FileMoniker

                 OLEDBG_BEGIN2("IRunningObjectTable::IsRunning called\r\n")
                 hrErr = lpROT->lpVtbl->IsRunning(lpROT,*lplpmk);
                 OLEDBG_END2

                 if (hrErr != NOERROR)
                       break;  // the Moniker is NOT running; found unused
one!

                 OleStdVerifyRelease(
                             (LPUNKNOWN)*lplpmk,
                             "OleStdCreateTempFileMoniker: Moniker NOT
released"
                       );

                 wsprintf(lpszName, "%s%d", lpszPrefixString, i++);
                 CreateFileMoniker(lpszName, lplpmk);
           }

           OleStdRelease((LPUNKNOWN)lpROT);
     }

     if (lpuUnique != NULL)
           *lpuUnique = i;
}
```

```
/* OleStdGetFirstMoniker
** ---------------------
**    return the first piece of a moniker.
**
**    NOTE: if the given moniker is not a generic composite moniker,
**    then an AddRef'ed pointer to the given moniker is returned.
*/
STDAPI_(LPMONIKER) OleStdGetFirstMoniker(LPMONIKER lpmk)
{
     LPMONIKER       lpmkFirst = NULL;
     LPENUMMONIKER   lpenumMoniker;
     DWORD           dwMksys;
     HRESULT         hrErr;

     if (! lpmk)
           return NULL;

     if (lpmk->lpVtbl->IsSystemMoniker(lpmk, (LPDWORD)&dwMksys) == NOERROR
           && dwMksys == MKSYS_GENERICCOMPOSITE) {

           /* OLE2NOTE: the moniker is a GenericCompositeMoniker.
           **    enumerate the moniker to pull off the first piece.
           */

           hrErr = lpmk->lpVtbl->Enum(
                       lpmk,
                       TRUE /* fForward */,
                       (LPENUMMONIKER FAR*)&lpenumMoniker
           );
           if (hrErr != NOERROR)
                 return NULL;    // ERROR: give up!

           hrErr = lpenumMoniker->lpVtbl->Next(
                       lpenumMoniker,
                       1,
                       (LPMONIKER FAR*)&lpmkFirst,
                       NULL
           );
           lpenumMoniker->lpVtbl->Release(lpenumMoniker);
           return lpmkFirst;


     } else {
           /* OLE2NOTE: the moniker is NOT a GenericCompositeMoniker.
           **    return an AddRef'ed pointer to the input moniker.
           */
           lpmk->lpVtbl->AddRef(lpmk);
           return lpmk;
     }
}


/* OleStdGetLenFilePrefixOfMoniker
** -------------------------------
**    if the first piece of the Moniker is a FileMoniker, then return
**    the length of the filename string.
**
**    lpmk      pointer to moniker
**
**    Returns
**      0       if moniker does NOT start with a FileMoniker
**      uLen    string length of filename prefix of the display name
**              retrieved from the given (lpmk) moniker.
*/
STDAPI_(ULONG) OleStdGetLenFilePrefixOfMoniker(LPMONIKER lpmk)
{
     LPMONIKER       lpmkFirst = NULL;
     DWORD           dwMksys;
     LPSTR           lpsz = NULL;
     LPBC            lpbc = NULL;
     ULONG           uLen = 0;
     HRESULT         hrErr;

     if (! lpmk)
           return 0;

     lpmkFirst = OleStdGetFirstMoniker(lpmk);
     if (lpmkFirst) {
           if ( (lpmkFirst->lpVtbl->IsSystemMoniker(
                                        lpmkFirst, (LPDWORD)&dwMksys) ==
NOERROR)
                      && dwMksys == MKSYS_FILEMONIKER) {

                 hrErr = CreateBindCtx(0, (LPBC FAR*)&lpbc);
                 if (hrErr == NOERROR) {
                       hrErr = lpmkFirst->lpVtbl->GetDisplayName(
                                   lpmkFirst,
                                   lpbc,
                                   NULL,   /* pmkToLeft */
                                   (LPSTR FAR*)&lpsz
                       );
                       if (hrErr == NOERROR && lpsz != NULL) {
                             uLen = lstrlen(lpsz);
                             OleStdFreeString(lpsz, NULL);
                       }
                       OleStdRelease((LPUNKNOWN)lpbc);
                 }
           }
           lpmkFirst->lpVtbl->Release(lpmkFirst);
     }
     return uLen;
}


/* OleStdMkParseDisplayName
**    Parse a string into a Moniker by calling the OLE API
**    MkParseDisplayName. if the original link source class was an OLE1
**    class, then attempt the parsing assuming the same class applies.
**
**    if the class of the previous link source was an OLE1 class,
**    then first attempt to parse a string that it is qualified
**    with the progID associated with the OLE1 class. this more
**    closely matches the semantics of OLE1 where the class of
**    link sources is not expected to change. prefixing the
**    string with "@<ProgID -- OLE1 class name>!" will force the
**    parsing of the string to assume the file is of that
**    class.
**    NOTE: this trick of prepending the string with "@<ProgID>
**    only works for OLE1 classes.
**
**  PARAMETERS:
**    REFCLSID rClsid       -- original class of link source.
**                             CLSID_NULL if class is unknown
**    ... other parameters the same as MkParseDisplayName API ...
**
**  RETURNS
**    NOERROR if string parsed successfully
**    else error code returned by MkParseDisplayName
*/
STDAPI OleStdMkParseDisplayName(
           REFCLSID        rClsid,
           LPBC            lpbc,
           LPSTR           lpszUserName,
           ULONG FAR*      lpchEaten,
           LPMONIKER FAR*  lplpmk
)
{
     HRESULT hrErr;

     if (!IsEqualCLSID(rClsid,&CLSID_NULL) && CoIsOle1Class(rClsid) &&
           lpszUserName[0] != '@') {
           LPSTR lpszBuf;
           LPSTR lpszProgID;

           // Prepend "@<ProgID>!" to the input string
           ProgIDFromCLSID(rClsid, &lpszProgID);
           if (lpszProgID == NULL)
                 goto Cont1;
           lpszBuf = OleStdMalloc(
                       (ULONG)lstrlen(lpszUserName)+lstrlen(lpszProgID)+3);
           if (lpszBuf == NULL) {
                 if (lpszProgID)
                       OleStdFree(lpszProgID);
                 goto Cont1;
           }

           wsprintf(lpszBuf, "@%s!%s", lpszProgID, lpszUserName);
           OLEDBG_BEGIN2("MkParseDisplayName called\r\n")
           hrErr = MkParseDisplayName(lpbc, lpszBuf, lpchEaten, lplpmk);
           OLEDBG_END2

           if (lpszProgID)
                 OleStdFree(lpszProgID);
           if (lpszBuf)
                 OleStdFree(lpszBuf);
```

```
            if (hrErr == NOERROR)
                  return NOERROR;
      }

Cont1:
     OLEDBG_BEGIN2("MkParseDisplayName called\r\n")
     hrErr = MkParseDisplayName(lpbc, lpszUserName, lpchEaten, lplpmk);
     OLEDBG_END2

     return hrErr;
}


/*
 * OleStdMarkPasteEntryList
 *
 * Purpose:
 *  Mark each entry in the PasteEntryList if its format is available from
 *  the source IDataObject*. the dwScratchSpace field of each PasteEntry
 *  is set to TRUE if available, else FALSE.
 *
 * Parameters:
 *  LPOLEUIPASTEENTRY   array of PasteEntry structures
 *  int                 count of elements in PasteEntry array
 *  LPDATAOBJECT        source IDataObject* pointer
 *
 * Return Value:
 *   none
 */
STDAPI_(void) OleStdMarkPasteEntryList(
           LPDATAOBJECT        lpSrcDataObj,
           LPOLEUIPASTEENTRY   lpPriorityList,
           int                 cEntries
)
{
     LPENUMFORMATETC     lpEnumFmtEtc = NULL;
     #define FORMATETC_MAX 20
     FORMATETC           rgfmtetc[FORMATETC_MAX];
     int                 i;
     HRESULT             hrErr;
     long                j, cFetched;

     // Clear all marks
     for (i = 0; i < cEntries; i++) {
           lpPriorityList[i].dwScratchSpace = FALSE;

           if (! lpPriorityList[i].fmtetc.cfFormat) {
                 // caller wants this item always considered available
                 // (by specifying a NULL format)
                 lpPriorityList[i].dwScratchSpace = TRUE;
           } else if (lpPriorityList[i].fmtetc.cfFormat == cfEmbeddedObject
                       || lpPriorityList[i].fmtetc.cfFormat == cfEmbedSource
                       || lpPriorityList[i].fmtetc.cfFormat == cfFileName) {
```

```
            // if there is an OLE object format, then handle it
            // specially by calling OleQueryCreateFromData. the caller
            // need only specify one object type format.
            OLEDBG_BEGIN2("OleQueryCreateFromData called\r\n")
            hrErr = OleQueryCreateFromData(lpSrcDataObj);
            OLEDBG_END2
            if(NOERROR == hrErr) {
                lpPriorityList[i].dwScratchSpace = TRUE;
            }
        } else if (lpPriorityList[i].fmtetc.cfFormat == cfLinkSource) {

            // if there is OLE 2.0 LinkSource format, then handle it
            // specially by calling OleQueryLinkFromData.
            OLEDBG_BEGIN2("OleQueryLinkFromData called\r\n")
            hrErr = OleQueryLinkFromData(lpSrcDataObj);
            OLEDBG_END2
            if(NOERROR == hrErr) {
                lpPriorityList[i].dwScratchSpace = TRUE;
            }
        }
    }

    OLEDBG_BEGIN2("IDataObject::EnumFormatEtc called\r\n")
    hrErr = lpSrcDataObj->lpVtbl->EnumFormatEtc(
            lpSrcDataObj,
            DATADIR_GET,
            (LPENUMFORMATETC FAR*)&lpEnumFmtEtc
    );
    OLEDBG_END2

    if (hrErr != NOERROR)
        return;    // unable to get format enumerator

    // Enumerate the formats offered by the source
    // Loop over all formats offered by the source
    cFetched = 0;
    _fmemset(rgfmtetc,0,sizeof(rgfmtetc[FORMATETC_MAX]) );
    if (lpEnumFmtEtc->lpVtbl->Next(
            lpEnumFmtEtc, FORMATETC_MAX, rgfmtetc, &cFetched) ==
NOERROR
        || (cFetched > 0 && cFetched <= FORMATETC_MAX) )
    {

        for (j = 0; j < cFetched; j++)
        {
            for (i = 0; i < cEntries; i++)
            {
                if (! lpPriorityList[i].dwScratchSpace &&
                    IsEqualFORMATETC(rgfmtetc[j],
lpPriorityList[i].fmtetc))
                {
                    lpPriorityList[i].dwScratchSpace = TRUE;
                }
            }
```

```
           }
     } // endif

     // Clean up
     if (lpEnumFmtEtc)
           OleStdRelease((LPUNKNOWN)lpEnumFmtEtc);
}


/* OleStdGetPriorityClipboardFormat
** --------------------------------
**
**    Retrieve the first clipboard format in a list for which data
**    exists in the source IDataObject*.
**
**    Returns -1 if no acceptable match is found.
**            index of first acceptable match in the priority list.
**
*/
STDAPI_(int) OleStdGetPriorityClipboardFormat(
           LPDATAOBJECT        lpSrcDataObj,
           LPOLEUIPASTEENTRY   lpPriorityList,
           int                 cEntries
)
{
     int i;
     int nFmtEtc = -1;

     // Mark all entries that the Source provides
     OleStdMarkPasteEntryList(lpSrcDataObj, lpPriorityList, cEntries);

     // Loop over the target's priority list of formats
     for (i = 0; i < cEntries; i++)
     {
           if (lpPriorityList[i].dwFlags != OLEUIPASTE_PASTEONLY &&
                       !(lpPriorityList[i].dwFlags & OLEUIPASTE_PASTE))
                 continue;

           // get first marked entry
           if (lpPriorityList[i].dwScratchSpace) {
                 nFmtEtc = i;
                 break;          // Found priority format; DONE
           }
     }

     return nFmtEtc;
}


/* OleStdIsDuplicateFormat
** -----------------------
**    Returns TRUE if the lpFmtEtc->cfFormat is found in the array of
**    FormatEtc structures.
*/
```

```
STDAPI_(BOOL) OleStdIsDuplicateFormat(
           LPFORMATETC         lpFmtEtc,
           LPFORMATETC         arrFmtEtc,
           int                 nFmtEtc
)
{
     int i;

     for (i = 0; i < nFmtEtc; i++) {
           if (IsEqualFORMATETC((*lpFmtEtc), arrFmtEtc[i]))
                 return TRUE;
     }

     return FALSE;
}


/* OleStdGetItemToken
 * ------------------
 *
 * LPSTR lpszSrc - Pointer to a source string
 * LPSTR lpszDst - Pointer to destination buffer
 *
 * Will copy one token from the lpszSrc buffer to the lpszItem buffer.
 * It considers all alpha-numeric and white space characters as valid
 * characters for a token. the first non-valid character delimates the
 * token.
 *
 * returns the number of charaters eaten.
 */
STDAPI_(ULONG) OleStdGetItemToken(LPSTR lpszSrc, LPSTR lpszDst, int
nMaxChars)
{
     ULONG chEaten = 0L;

     // skip leading delimeter characters
     while (*lpszSrc && --nMaxChars > 0
                                            && ((*lpszSrc=='/') ||
(*lpszSrc=='\\') ||
                                                  (*lpszSrc=='!') ||
(*lpszSrc==':'))) {
           *lpszSrc++;
           chEaten++;
     }

     // Extract token string (up to first delimeter char or EOS)
     while (*lpszSrc && --nMaxChars > 0
                                            && !((*lpszSrc=='/') ||
(*lpszSrc=='\\') ||
                                                  (*lpszSrc=='!') ||
(*lpszSrc==':'))) {
           *lpszDst++ = *lpszSrc++;
           chEaten++;
     }
     *lpszDst = '\0';
     return chEaten;
}


/*************************************************************************
** OleStdCreateRootStorage
**    create a root level Storage given a filename that is compatible
**    to be used by a top-level OLE container. if the filename
**    specifies an existing file, then an error is returned.
**    the root storage (Docfile) that is created by this function
**    is suitable to be used to create child storages for embedings.
**    (CreateChildStorage can be used to create child storages.)
**    NOTE: the root-level storage is opened in transacted mode.
*************************************************************************/

STDAPI_(LPSTORAGE) OleStdCreateRootStorage(LPSTR lpszStgName, DWORD grfMode)
{
     HRESULT hr;
     DWORD grfCreateMode = STGM_READWRITE | STGM_TRANSACTED;
     DWORD reserved = 0;
     LPSTORAGE lpRootStg;
     char szMsg[64];

     // if temp file is being created, enable delete-on-release
     if (! lpszStgName)
           grfCreateMode |= STGM_DELETEONRELEASE;

     hr = StgCreateDocfile(
                 lpszStgName,
                 grfMode | grfCreateMode,
                 reserved,
                 (LPSTORAGE FAR*)&lpRootStg
           );

     if (hr == NOERROR)
           return lpRootStg;               // existing file successfully
opened

     OleDbgOutHResult("StgCreateDocfile returned", hr);

     if (0 == LoadString(ghInst, (UINT)IDS_OLESTDNOCREATEFILE, (LPSTR)szMsg,
64))
       return NULL;

     MessageBox(NULL, (LPSTR)szMsg, NULL,MB_ICONEXCLAMATION | MB_OK);
     return NULL;
}


/*************************************************************************
** OleStdOpenRootStorage
**    open a root level Storage given a filename that is compatible
**    to be used by a top-level OLE container. if the file does not
**    exist then an error is returned.
**    the root storage (Docfile) that is opened by this function
**    is suitable to be used to create child storages for embedings.
**    (CreateChildStorage can be used to create child storages.)
**    NOTE: the root-level storage is opened in transacted mode.
*************************************************************************/

STDAPI_(LPSTORAGE) OleStdOpenRootStorage(LPSTR lpszStgName, DWORD grfMode)
{
     HRESULT hr;
     DWORD reserved = 0;
     LPSTORAGE lpRootStg;
     char  szMsg[64];

     if (lpszStgName) {

           hr = StgOpenStorage(
                       lpszStgName,
                       NULL,
                       grfMode | STGM_TRANSACTED,
                       NULL,
                       reserved,
                       (LPSTORAGE FAR*)&lpRootStg
                 );

           if (hr == NOERROR)
                 return lpRootStg;     // existing file successfully opened

           OleDbgOutHResult("StgOpenStorage returned", hr);
     }

     if (0 == LoadString(ghInst, (UINT)IDS_OLESTDNOOPENFILE, szMsg, 64))
       return NULL;

     MessageBox(NULL, (LPSTR)szMsg, NULL,MB_ICONEXCLAMATION | MB_OK);
     return NULL;
}


/*************************************************************************
** OpenOrCreateRootStorage
**    open a root level Storage given a filename that is compatible
**    to be used by a top-level OLE container. if the filename
**    specifies an existing file, then it is open, otherwise a new file
**    with the given name is created.
**    the root storage (Docfile) that is created by this function
**    is suitable to be used to create child storages for embedings.
**    (CreateChildStorage can be used to create child storages.)
**    NOTE: the root-level storage is opened in transacted mode.
*************************************************************************/

STDAPI_(LPSTORAGE) OleStdOpenOrCreateRootStorage(LPSTR lpszStgName, DWORD
grfMode)
{
     HRESULT hrErr;
     SCODE sc;
     DWORD reserved = 0;
     LPSTORAGE lpRootStg;
     char      szMsg[64];

     if (lpszStgName) {

           hrErr = StgOpenStorage(
                       lpszStgName,
                       NULL,
                       grfMode | STGM_READWRITE | STGM_TRANSACTED,
                       NULL,
                       reserved,
                       (LPSTORAGE FAR*)&lpRootStg
           );

           if (hrErr == NOERROR)
                 return lpRootStg;      // existing file successfully opened

           OleDbgOutHResult("StgOpenStorage returned", hrErr);
           sc = GetScode(hrErr);

           if (sc!=STG_E_FILENOTFOUND && sc!=STG_E_FILEALREADYEXISTS) {
                 return NULL;
           }
     }

     /* if file did not already exist, try to create a new one */
     hrErr = StgCreateDocfile(
                 lpszStgName,
                 grfMode | STGM_READWRITE | STGM_TRANSACTED,
                 reserved,
                 (LPSTORAGE FAR*)&lpRootStg
     );

     if (hrErr == NOERROR)
           return lpRootStg;               // existing file successfully
opened

     OleDbgOutHResult("StgCreateDocfile returned", hrErr);

     if (0 == LoadString(ghInst, (UINT)IDS_OLESTDNOCREATEFILE, (LPSTR)szMsg,
64))
       return NULL;

     MessageBox(NULL, (LPSTR)szMsg, NULL, MB_ICONEXCLAMATION | MB_OK);
     return NULL;
}


/*
** OleStdCreateChildStorage
**    create a child Storage inside the given lpStg that is compatible
**    to be used by an embedded OLE object. the return value from this
**    function can be passed to OleCreateXXX functions.
**    NOTE: the child storage is opened in transacted mode.
```

```
*/
STDAPI_(LPSTORAGE) OleStdCreateChildStorage(LPSTORAGE lpStg, LPSTR
lpszStgName)
{
     if (lpStg != NULL) {
           LPSTORAGE lpChildStg;
           DWORD grfMode = (STGM_READWRITE | STGM_TRANSACTED |
                       STGM_SHARE_EXCLUSIVE);
           DWORD reserved = 0;

           HRESULT hrErr = lpStg->lpVtbl->CreateStorage(
                       lpStg,
                       lpszStgName,
                       grfMode,
                       reserved,
                       reserved,
                       (LPSTORAGE FAR*)&lpChildStg
                 );

           if (hrErr == NOERROR)
                 return lpChildStg;

           OleDbgOutHResult("lpStg->lpVtbl->CreateStorage returned", hrErr);
     }
     return NULL;
}


/*
** OleStdOpenChildStorage
**    open a child Storage inside the given lpStg that is compatible
**    to be used by an embedded OLE object. the return value from this
**    function can be passed to OleLoad function.
**    NOTE: the child storage is opened in transacted mode.
*/
STDAPI_(LPSTORAGE) OleStdOpenChildStorage(LPSTORAGE lpStg, LPSTR
lpszStgName, DWORD grfMode)
{
     LPSTORAGE lpChildStg;
     LPSTORAGE lpstgPriority = NULL;
     DWORD reserved = 0;
     HRESULT hrErr;

     if (lpStg  != NULL) {

           hrErr = lpStg->lpVtbl->OpenStorage(
                       lpStg,
                       lpszStgName,
                       lpstgPriority,
                       grfMode | STGM_TRANSACTED | STGM_SHARE_EXCLUSIVE,
                       NULL,
                       reserved,
                       (LPSTORAGE FAR*)&lpChildStg
                 );
```

```
            if (hrErr == NOERROR)
                  return lpChildStg;

            OleDbgOutHResult("lpStg->lpVtbl->OpenStorage returned", hrErr);
      }
      return NULL;
}


/* OleStdCommitStorage
** -------------------
**    Commit the changes to the given IStorage*. This routine can be
**    called on either a root-level storage as used by an OLE-Container
**    or by a child storage as used by an embedded object.
**
**    This routine first attempts to perform this commit in a safe
**    manner. if this fails it then attempts to do the commit in a less
**    robust manner (STGC_OVERWRITE).
*/
STDAPI_(BOOL) OleStdCommitStorage(LPSTORAGE lpStg)
{
      HRESULT hrErr;

      // make the changes permanent
      hrErr = lpStg->lpVtbl->Commit(lpStg, 0);

      if (GetScode(hrErr) == STG_E_MEDIUMFULL) {
            // try to commit changes in less robust manner.
            OleDbgOut("Warning: commiting with STGC_OVERWRITE specified\n");
            hrErr = lpStg->lpVtbl->Commit(lpStg, STGC_OVERWRITE);
      }

      if (hrErr != NOERROR)
      {
            char szMsg[64];

            if (0 == LoadString(ghInst, (UINT)IDS_OLESTDDISKFULL,
(LPSTR)szMsg, 64))
               return FALSE;

            MessageBox(NULL, (LPSTR)szMsg, NULL, MB_ICONEXCLAMATION | MB_OK);
            return FALSE;
      }
      else {
            return TRUE;
      }
}


/* OleStdDestroyAllElements
** ------------------------
**    Destroy all elements within an open storage. this is subject
**    to the current transaction.
*/
```

```
STDAPI OleStdDestroyAllElements(LPSTORAGE lpStg)
{
     IEnumSTATSTG FAR* lpEnum;
     STATSTG sstg;
     HRESULT hrErr;

     hrErr = lpStg->lpVtbl->EnumElements(
                 lpStg, 0, NULL, 0, (IEnumSTATSTG FAR* FAR*)&lpEnum);

     if (hrErr != NOERROR)
           return hrErr;

     while (1) {
           if (lpEnum->lpVtbl->Next(lpEnum, 1, &sstg, NULL) != NOERROR)
                 break;
           lpStg->lpVtbl->DestroyElement(lpStg, sstg.pwcsName);
           OleStdFree(sstg.pwcsName);
     }
     lpEnum->lpVtbl->Release(lpEnum);
     return NOERROR;
}
```

OLEUTL.C (WRAPUI Sample)

```
/*
 * OLEUTL.C
 *
 * Miscellaneous utility functions for OLE 2.0 Applications:
 *
 *  Function                      Purpose
 *  -------------------------------------------------------------------
 *  SetDCToDrawInHimetricRect     Sets up an HIMETRIC mapping mode in a DC.
 *  ResetOrigDC                   Performs the opposite of
 *                                SetDCToDrawInHimetricRect
 *  XformWidthInPixelsToHimetric  Converts an int width into HiMetric units
 *  XformWidthInHimetricToPixels  Converts an int width from HiMetric units
 *  XformHeightInPixelsToHimetric Converts an int height into HiMetric units
 *  XformHeightInHimetricToPixels Converts an int height from HiMetric units
 *  XformRectInPixelsToHimetric   Converts a rect into HiMetric units
 *  XformRectInHimetricToPixels   Converts a rect from HiMetric units
 *  XformSizeInPixelsToHimetric   Converts a SIZEL into HiMetric units
 *  XformSizeInHimetricToPixels   Converts a SIZEL from HiMetric units
 *  AreRectsEqual                 Compares to Rect's
 *
 *  ParseCmdLine                  Determines if -Embedding exists
 *  OpenOrCreateRootStorage       Creates a root docfile for OLE storage
 *  CommitStorage                 Commits all changes in a docfile
 *  CreateChildStorage            Creates child storage in another storage
 *  OpenChildStorage              Opens child storage in another storage
 *
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */


#define STRICT  1
#include "ole2ui.h"
#include <stdlib.h>
#include <ctype.h>

//Internal function to this module
static LPSTR GetWord(LPSTR lpszSrc, LPSTR lpszDst);


/*
 * SetDCToAnisotropic
 *
 * Purpose:
 *  Setup the correspondence between the rect in device unit (Viewport) and
 *  the rect in logical unit (Window) so that the proper scaling of
 *  coordinate systems will be calculated. set up both the Viewport and
 *  the window as follows:
 *
 *      1) ------------------ ( 2
 *      |                     |
 *      |                     |
 *      |                     |
 *      |                     |
 *      |                     |
 *      3) ------------------ ( 4
 *
 *      Origin   = P3
 *      X extent = P2x - P3x
 *      Y extent = P2y - P3y
 *
 * Parameters:
 *  hDC             HDC to affect
 *  lprcPhysical    LPRECT containing the physical (device) extents of DC
 *  lprcLogical     LPRECT containing the logical extents
 *  lprcWindowOld   LPRECT in which to preserve the window for ResetOrigDC
 *  lprcViewportOld LPRECT in which to preserver the viewport for
ResetOrigDC
 *
 * Return Value:
 *  int             The original mapping mode of the DC.
 */

STDAPI_(int) SetDCToAnisotropic(
           HDC hDC,
           LPRECT lprcPhysical, LPRECT lprcLogical,
           LPRECT lprcWindowOld, LPRECT lprcViewportOld)
{
     int     nMapModeOld=SetMapMode(hDC, MM_ANISOTROPIC);

     SetWindowOrgEx(hDC, lprcLogical->left, lprcLogical->bottom,
(LPPOINT)&lprcWindowOld->left);
     SetWindowExtEx(hDC, (lprcLogical->right-lprcLogical->left),
(lprcLogical->top-lprcLogical->bottom), (LPSIZE)&lprcWindowOld->right);
     SetViewportOrgEx(hDC, lprcPhysical->left, lprcPhysical->bottom,
(LPPOINT)&lprcViewportOld->left);
     SetViewportExtEx(hDC, (lprcPhysical->right-lprcPhysical->left),
(lprcPhysical->top-lprcPhysical->bottom), (LPSIZE)&lprcViewportOld->right);

     return nMapModeOld;
}


/*
 * SetDCToDrawInHimetricRect
 *
 * Purpose:
 *  Setup the correspondence between the rect in pixels (Viewport) and
 *  the rect in HIMETRIC (Window) so that the proper scaling of
 *  coordinate systems will be calculated. set up both the Viewport and
 *  the window as follows:
 *
 *      1) ------------------ ( 2
 *      |                     |
 *      |                     |
 *      |                     |
 *      |                     |
 *      |                     |
 *      3) ------------------ ( 4
 *
 *      Origin   = P3
 *      X extent = P2x - P3x
 *      Y extent = P2y - P3y
 *
 * Parameters:
 *  hDC             HDC to affect
 *  lprcPix         LPRECT containing the pixel extents of DC
 *  lprcHiMetric    LPRECT to receive the himetric extents
 *  lprcWindowOld   LPRECT in which to preserve the window for ResetOrigDC
 *  lprcViewportOld LPRECT in which to preserver the viewport for
ResetOrigDC
 *
 * Return Value:
 *  int             The original mapping mode of the DC.
 */
STDAPI_(int) SetDCToDrawInHimetricRect(
     HDC hDC,
     LPRECT lprcPix, LPRECT lprcHiMetric,
     LPRECT lprcWindowOld, LPRECT lprcViewportOld)
     {
     int     nMapModeOld=SetMapMode(hDC, MM_ANISOTROPIC);
     BOOL    fSystemDC  =FALSE;

     if (NULL==hDC)
           {
           hDC=GetDC(NULL);
           fSystemDC=TRUE;
           }

     XformRectInPixelsToHimetric(hDC, lprcPix, lprcHiMetric);

     SetWindowOrgEx(hDC, lprcHiMetric->left, lprcHiMetric->bottom,
(LPPOINT)&lprcWindowOld->left);
     SetWindowExtEx(hDC, (lprcHiMetric->right-lprcHiMetric->left),
(lprcHiMetric->top-lprcHiMetric->bottom), (LPSIZE)&lprcWindowOld->right);
     SetViewportOrgEx(hDC, lprcPix->left, lprcPix->bottom,
(LPPOINT)&lprcViewportOld->left);
     SetViewportExtEx(hDC, (lprcPix->right-lprcPix->left), (lprcPix->top-
lprcPix->bottom), (LPSIZE)&lprcViewportOld->right);

     if (fSystemDC)
           ReleaseDC(NULL, hDC);

     return nMapModeOld;
     }



/*
 * ResetOrigDC
 *
 * Purpose:
 *  Restores a DC set to draw in himetric from SetDCToDrawInHimetricRect.
 *
 * Parameters:
 *  hDC             HDC to restore
 *  nMapModeOld     int original mapping mode of hDC
 *  lprcWindowOld   LPRECT filled in SetDCToDrawInHimetricRect
 *  lprcViewportOld LPRECT filled in SetDCToDrawInHimetricRect
 *
 * Return Value:
 *  int             Same as nMapModeOld.
 */

STDAPI_(int) ResetOrigDC(
     HDC hDC, int nMapModeOld,
     LPRECT lprcWindowOld, LPRECT lprcViewportOld)
     {
     POINT     pOld;

     SetMapMode(hDC, nMapModeOld);

     SetWindowOrgEx(hDC,   lprcWindowOld->left,    lprcWindowOld->top,
(LPPOINT)&pOld);
     SetWindowExtEx(hDC,   lprcWindowOld->right,   lprcWindowOld->bottom,
(LPSIZE)&pOld);
     SetViewportOrgEx(hDC, lprcViewportOld->left,  lprcViewportOld->top,
(LPPOINT)&pOld);
     SetViewportExtEx(hDC, lprcViewportOld->right, lprcViewportOld->bottom,
(LPSIZE)&pOld);

     return nMapModeOld;
     }




/*
 * XformWidthInPixelsToHimetric
 * XformWidthInHimetricToPixels
 * XformHeightInPixelsToHimetric
 * XformHeightInHimetricToPixels
 *
 * Functions to convert an int between a device coordinate system and
 * logical HiMetric units.
 *
 * Parameters:
 *  hDC             HDC providing reference to the pixel mapping.  If
 *                  NULL, a screen DC is used.
 *
 *  Size Functions:
 *  lpSizeSrc       LPSIZEL providing the structure to convert.  This
 *                  contains pixels in XformSizeInPixelsToHimetric and
 *                  logical HiMetric units in the complement function.
 *  lpSizeDst       LPSIZEL providing the structure to receive converted
 *                  units.  This contains pixels in
 *                  XformSizeInPixelsToHimetric and logical HiMetric
 *                  units in the complement function.
 *
 *  Width Functions:
 *  iWidth          int containing the value to convert.
 *
 * Return Value:
 *  Size Functions:     None
 *  Width Functions:    Converted value of the input parameters.
 *
 * NOTE:
 *  When displaying on the screen, Window apps display everything enlarged
 *  from its actual size so that it is easier to read. For example, if an
 *  app wants to display a 1in. horizontal line, that when printed is
 *  actually a 1in. line on the printed page, then it will display the line
 *  on the screen physically larger than 1in. This is described as a line
 *  that is "logically" 1in. along the display width. Windows maintains as
 *  part of the device-specific information about a given display device:
 *      LOGPIXELSX -- no. of pixels per logical in along the display width
 *      LOGPIXELSY -- no. of pixels per logical in along the display height
 *
 *  The following formula converts a distance in pixels into its equivalent
 *  logical HIMETRIC units:
 *
 *      DistInHiMetric = (HIMETRIC_PER_INCH * DistInPix)
 *                       -------------------------------
 *                           PIXELS_PER_LOGICAL_IN
 *
 */
STDAPI_(int) XformWidthInPixelsToHimetric(HDC hDC, int iWidthInPix)
     {
     int     iXppli;     //Pixels per logical inch along width
     int     iWidthInHiMetric;
     BOOL    fSystemDC=FALSE;

     if (NULL==hDC || GetDeviceCaps(hDC, TECHNOLOGY) == DT_METAFILE
           || GetDeviceCaps(hDC, LOGPIXELSX) == 0)
           {
           hDC=GetDC(NULL);
           fSystemDC=TRUE;
           }

     iXppli = GetDeviceCaps (hDC, LOGPIXELSX);

     //We got pixel units, convert them to logical HIMETRIC along the
display
     iWidthInHiMetric = (int)MAP_PIX_TO_LOGHIM(iWidthInPix, iXppli);

     if (fSystemDC)
           ReleaseDC(NULL, hDC);

     return iWidthInHiMetric;
     }
```

```
STDAPI_(int) XformWidthInHimetricToPixels(HDC hDC, int iWidthInHiMetric)
     {
     int     iXppli;     //Pixels per logical inch along width
     int     iWidthInPix;
     BOOL    fSystemDC=FALSE;

     if (NULL==hDC || GetDeviceCaps(hDC, TECHNOLOGY) == DT_METAFILE
           || GetDeviceCaps(hDC, LOGPIXELSX) == 0)
           {
           hDC=GetDC(NULL);
           fSystemDC=TRUE;
           }

     iXppli = GetDeviceCaps (hDC, LOGPIXELSX);

     //We got logical HIMETRIC along the display, convert them to pixel
units
     iWidthInPix = (int)MAP_LOGHIM_TO_PIX(iWidthInHiMetric, iXppli);

     if (fSystemDC)
           ReleaseDC(NULL, hDC);

     return iWidthInPix;
     }


STDAPI_(int) XformHeightInPixelsToHimetric(HDC hDC, int iHeightInPix)
     {
     int     iYppli;     //Pixels per logical inch along height
     int     iHeightInHiMetric;
     BOOL    fSystemDC=FALSE;

     if (NULL==hDC || GetDeviceCaps(hDC, TECHNOLOGY) == DT_METAFILE
           || GetDeviceCaps(hDC, LOGPIXELSY) == 0)
           {
           hDC=GetDC(NULL);
           fSystemDC=TRUE;
           }

     iYppli = GetDeviceCaps (hDC, LOGPIXELSY);

     //* We got pixel units, convert them to logical HIMETRIC along the
display
     iHeightInHiMetric = (int)MAP_PIX_TO_LOGHIM(iHeightInPix, iYppli);

     if (fSystemDC)
           ReleaseDC(NULL, hDC);

     return iHeightInHiMetric;
     }


STDAPI_(int) XformHeightInHimetricToPixels(HDC hDC, int iHeightInHiMetric)
     {
     int     iYppli;     //Pixels per logical inch along height
     int     iHeightInPix;
     BOOL    fSystemDC=FALSE;

     if (NULL==hDC || GetDeviceCaps(hDC, TECHNOLOGY) == DT_METAFILE
           || GetDeviceCaps(hDC, LOGPIXELSY) == 0)
           {
           hDC=GetDC(NULL);
           fSystemDC=TRUE;
           }

     iYppli = GetDeviceCaps (hDC, LOGPIXELSY);

     //* We got logical HIMETRIC along the display, convert them to pixel
units
     iHeightInPix = (int)MAP_LOGHIM_TO_PIX(iHeightInHiMetric, iYppli);

     if (fSystemDC)
           ReleaseDC(NULL, hDC);

     return iHeightInPix;
     }



/*
 * XformRectInPixelsToHimetric
 * XformRectInHimetricToPixels
 *
 * Purpose:
 *  Convert a rectangle between pixels of a given hDC and HIMETRIC units
 *  as manipulated in OLE.  If the hDC is NULL, then a screen DC is used
 *  and assumes the MM_TEXT mapping mode.
 *
 * Parameters:
 *  hDC             HDC providing reference to the pixel mapping.  If
 *                  NULL, a screen DC is used.
 *  lprcSrc         LPRECT providing the rectangle to convert.  This
 *                  contains pixels in XformRectInPixelsToHimetric and
 *                  logical HiMetric units in the complement function.
 *  lprcDst         LPRECT providing the rectangle to receive converted
units.
 *                  This contains pixels in XformRectInPixelsToHimetric and
 *                  logical HiMetric units in the complement function.
 *

 * Return Value:
 *  None
 *
 * NOTE:
 *  When displaying on the screen, Window apps display everything enlarged
 *  from its actual size so that it is easier to read. For example, if an
 *  app wants to display a 1in. horizontal line, that when printed is
 *  actually a 1in. line on the printed page, then it will display the line
 *  on the screen physically larger than 1in. This is described as a line
 *  that is "logically" 1in. along the display width. Windows maintains as
 *  part of the device-specific information about a given display device:
 *      LOGPIXELSX -- no. of pixels per logical in along the display width
 *      LOGPIXELSY -- no. of pixels per logical in along the display height
 *
 *  The following formula converts a distance in pixels into its equivalent
 *  logical HIMETRIC units:
 *
 *      DistInHiMetric = (HIMETRIC_PER_INCH * DistInPix)
 *                      -------------------------------
 *                            PIXELS_PER_LOGICAL_IN
 *
 * Rect in Pixels (MM_TEXT):
 *
 *              0---------- X
 *              |
 *              |       1) ------------------ ( 2   P1 = (rc.left, rc.top)
 *              |       |                     |     P2 = (rc.right, rc.top)
 *              |       |                     |     P3 = (rc.left,
rc.bottom)
 *              |       |                     |     P4 = (rc.right,
rc.bottom)
 *                      |                     |
 *              Y       |                     |
 *                      3) ------------------ ( 4
 *
 *              NOTE:   Origin   = (P1x, P1y)
 *                      X extent = P4x - P1x
 *                      Y extent = P4y - P1y
 *
 *
 * Rect in Himetric (MM_HIMETRIC):
 *
 *
 *                      1) ------------------ ( 2   P1 = (rc.left, rc.top)
 *              Y       |                     |     P2 = (rc.right, rc.top)
 *                      |                     |     P3 = (rc.left,
rc.bottom)
 *              |       |                     |     P4 = (rc.right,
rc.bottom)
 *              |       |                     |
 *              |       |                     |
 *              |       3) ------------------ ( 4
 *              |
 *              0---------- X
 *
 *              NOTE:   Origin   = (P3x, P3y)
 *                      X extent = P2x - P3x
 *                      Y extent = P2y - P3y
 *
 *
 */

STDAPI_(void) XformRectInPixelsToHimetric(
     HDC hDC, LPRECT lprcPix, LPRECT lprcHiMetric)
     {
```

```
     int     iXppli;     //Pixels per logical inch along width
     int     iYppli;     //Pixels per logical inch along height
     int     iXextInPix=(lprcPix->right-lprcPix->left);
     int     iYextInPix=(lprcPix->bottom-lprcPix->top);
     BOOL    fSystemDC=FALSE;

     if (NULL==hDC || GetDeviceCaps(hDC, TECHNOLOGY) == DT_METAFILE
           || GetDeviceCaps(hDC, LOGPIXELSX) == 0)
           {
           hDC=GetDC(NULL);
           fSystemDC=TRUE;
           }

     iXppli = GetDeviceCaps (hDC, LOGPIXELSX);
     iYppli = GetDeviceCaps (hDC, LOGPIXELSY);

     //We got pixel units, convert them to logical HIMETRIC along the
display
     lprcHiMetric->right = (int)MAP_PIX_TO_LOGHIM(iXextInPix, iXppli);
     lprcHiMetric->top   = (int)MAP_PIX_TO_LOGHIM(iYextInPix, iYppli);

     lprcHiMetric->left    = 0;
     lprcHiMetric->bottom  = 0;

     if (fSystemDC)
           ReleaseDC(NULL, hDC);

     return;
     }



STDAPI_(void) XformRectInHimetricToPixels(
     HDC hDC, LPRECT lprcHiMetric, LPRECT lprcPix)
     {
     int     iXppli;     //Pixels per logical inch along width
     int     iYppli;     //Pixels per logical inch along height
     int     iXextInHiMetric=(lprcHiMetric->right-lprcHiMetric->left);
     int     iYextInHiMetric=(lprcHiMetric->bottom-lprcHiMetric->top);
     BOOL    fSystemDC=FALSE;

     if (NULL==hDC || GetDeviceCaps(hDC, TECHNOLOGY) == DT_METAFILE
           || GetDeviceCaps(hDC, LOGPIXELSX) == 0)
           {
           hDC=GetDC(NULL);
           fSystemDC=TRUE;
           }

     iXppli = GetDeviceCaps (hDC, LOGPIXELSX);
     iYppli = GetDeviceCaps (hDC, LOGPIXELSY);

     //We got pixel units, convert them to logical HIMETRIC along the
display
     lprcPix->right = (int)MAP_LOGHIM_TO_PIX(iXextInHiMetric, iXppli);
     lprcPix->top   = (int)MAP_LOGHIM_TO_PIX(iYextInHiMetric, iYppli);
     lprcPix->left  = 0;
     lprcPix->bottom= 0;

     if (fSystemDC)
           ReleaseDC(NULL, hDC);

     return;
     }




/*
 * XformSizeInPixelsToHimetric
 * XformSizeInHimetricToPixels
 *
 * Functions to convert a SIZEL structure (Size functions) or
 * an int (Width functions) between a device coordinate system and
 * logical HiMetric units.
 *
 * Parameters:
 *  hDC             HDC providing reference to the pixel mapping.  If
 *                  NULL, a screen DC is used.
 *
 *  Size Functions:
 *  lpSizeSrc       LPSIZEL providing the structure to convert.  This
 *                  contains pixels in XformSizeInPixelsToHimetric and
 *                  logical HiMetric units in the complement function.
 *  lpSizeDst       LPSIZEL providing the structure to receive converted
 *                  units.  This contains pixels in
 *                  XformSizeInPixelsToHimetric and logical HiMetric
 *                  units in the complement function.
 *
 *  Width Functions:
 *  iWidth          int containing the value to convert.
 *
 * Return Value:
 *  Size Functions:     None
 *  Width Functions:    Converted value of the input parameters.
 *
 * NOTE:
 *  When displaying on the screen, Window apps display everything enlarged
 *  from its actual size so that it is easier to read. For example, if an
 *  app wants to display a 1in. horizontal line, that when printed is
 *  actually a 1in. line on the printed page, then it will display the line
 *  on the screen physically larger than 1in. This is described as a line
 *  that is "logically" 1in. along the display width. Windows maintains as
 *  part of the device-specific information about a given display device:
 *      LOGPIXELSX -- no. of pixels per logical in along the display width
 *      LOGPIXELSY -- no. of pixels per logical in along the display height
 *
 *  The following formula converts a distance in pixels into its equivalent
 *  logical HIMETRIC units:
```

```
 *
 *      DistInHiMetric = (HIMETRIC_PER_INCH * DistInPix)
 *                       -------------------------------
 *                           PIXELS_PER_LOGICAL_IN
 *
 */

STDAPI_(void) XformSizeInPixelsToHimetric(
     HDC hDC, LPSIZEL lpSizeInPix, LPSIZEL lpSizeInHiMetric)
     {
     int     iXppli;     //Pixels per logical inch along width
     int     iYppli;     //Pixels per logical inch along height
     BOOL    fSystemDC=FALSE;

     if (NULL==hDC || GetDeviceCaps(hDC, TECHNOLOGY) == DT_METAFILE
           || GetDeviceCaps(hDC, LOGPIXELSX) == 0)
           {
           hDC=GetDC(NULL);
           fSystemDC=TRUE;
           }

     iXppli = GetDeviceCaps (hDC, LOGPIXELSX);
     iYppli = GetDeviceCaps (hDC, LOGPIXELSY);

     //We got pixel units, convert them to logical HIMETRIC along the
display
     lpSizeInHiMetric->cx = (LONG)MAP_PIX_TO_LOGHIM(lpSizeInPix->cx,
iXppli);
     lpSizeInHiMetric->cy = (LONG)MAP_PIX_TO_LOGHIM(lpSizeInPix->cy,
iYppli);

     if (fSystemDC)
           ReleaseDC(NULL, hDC);

     return;
     }


STDAPI_(void) XformSizeInHimetricToPixels(
     HDC hDC, LPSIZEL lpSizeInHiMetric, LPSIZEL lpSizeInPix)
     {
     int     iXppli;     //Pixels per logical inch along width
     int     iYppli;     //Pixels per logical inch along height
     BOOL    fSystemDC=FALSE;

     if (NULL==hDC || GetDeviceCaps(hDC, TECHNOLOGY) == DT_METAFILE
           || GetDeviceCaps(hDC, LOGPIXELSX) == 0)
           {
           hDC=GetDC(NULL);
           fSystemDC=TRUE;
           }

     iXppli = GetDeviceCaps (hDC, LOGPIXELSX);
     iYppli = GetDeviceCaps (hDC, LOGPIXELSY);
     //We got logical HIMETRIC along the display, convert them to pixel
units
     lpSizeInPix->cx = (LONG)MAP_LOGHIM_TO_PIX(lpSizeInHiMetric->cx,
iXppli);
     lpSizeInPix->cy = (LONG)MAP_LOGHIM_TO_PIX(lpSizeInHiMetric->cy,
iYppli);

     if (fSystemDC)
           ReleaseDC(NULL, hDC);

     return;
     }


#if defined( OBSOLETE )
// This function has been converted to a macro

/* AreRectsEqual
** -------------
*/
STDAPI_(BOOL) AreRectsEqual(LPRECT lprc1, LPRECT lprc2)
{
     if ((lprc1->top == lprc2->top) &&
           (lprc1->left == lprc2->left) &&
           (lprc1->right == lprc2->right) &&
           (lprc1->bottom == lprc2->bottom))
           return TRUE;

     return FALSE;
}
#endif  // OBSOLETE


/*
 * ParseCmdLine
 *
 * Parses the Windows command line which was passed to WinMain.
 * This function determines if the -Embedding switch has been given.
 *
 */

STDAPI_(void) ParseCmdLine(
     LPSTR lpszLine,
     BOOL FAR* lpfEmbedFlag,
     LPSTR szFileName)
{
     int i=0;
     char szBuf[256];

     if(lpfEmbedFlag)
           *lpfEmbedFlag = FALSE;
     szFileName[0]='\0';             // NULL string

     // skip blanks
```

```
    while(isspace(*lpszLine)) lpszLine++;

    if(!*lpszLine)   // No filename or options, so start a fresh document.
        return;

    // Check for "-Embedding" or "/Embedding" and set fEmbedding.
    if(lpfEmbedFlag && (*lpszLine == '-' || *lpszLine == '/')) {
        lpszLine++;
        lpszLine = GetWord(lpszLine, szBuf);
        *lpfEmbedFlag = !lstrcmp(szBuf, EMBEDDINGFLAG);
    }

    // skip blanks
    while(isspace(*lpszLine)) lpszLine++;

    // set szFileName to argument
    while(lpszLine[i]) {
        szFileName[i]=lpszLine[i];
        i++;
    }
    szFileName[i]='\0';
}


/* GetWord
 * -------
 *
 * LPSTR lpszSrc - Pointer to a source string
 * LPSTR lpszDst - Pointer to destination buffer
 *
 * Will copy one space-terminated or null-terminated word from the source
 * string to the destination buffer.
 * returns: pointer to next character following the word.
 */
static LPSTR GetWord(LPSTR lpszSrc, LPSTR lpszDst)
{
    while (*lpszSrc && !isspace(*lpszSrc))
        *lpszDst++ = *lpszSrc++;

    *lpszDst = '\0';
    return lpszSrc;
}
```

PASTESPL.H   (WRAPUI Sample)

```
/*
 * PASTESPL.H
 *
 * Internal definitions, structures, and function prototypes for the
 * OLE 2.0 UI Paste Special dialog.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */

#ifndef _PASTESPL_H_
#define _PASTESPL_H_

#ifndef RC_INVOKED
#pragma message ("INCLUDING PASTESPL.H from " __FILE__)
#endif  /* RC_INVOKED */


// Length of buffers to hold the strings 'Unknown Type', Unknown Source'
//   and 'the application which created it'
#define PS_UNKNOWNSTRLEN               100

//Property label used to store clipboard viewer chain information
#define NEXTCBVIEWER        "NextCBViewer"

//Internally used structure
typedef struct tagPASTESPECIAL
{
     //Keep this item first as the Standard* functions depend on it here.
     LPOLEUIPASTESPECIAL  lpOPS;                //Original structure passed.

     /*
      * What we store extra in this structure besides the original caller's
      * pointer are those fields that we need to modify during the life of
      * the dialog but that we don't want to change in the original
structure
      * until the user presses OK.
      */

     DWORD                dwFlags;              // Local copy of paste
special flags

     int                  nPasteListCurSel;     // Save the selection the
user made last
     int                  nPasteLinkListCurSel; //    in the paste and
pastelink lists
     int                  nSelectedIndex;       // Index in
arrPasteEntries[] corresponding to user selection
     BOOL                 fLink;                // Indicates if Paste or
PasteLink was selected by user

     HGLOBAL              hBuff;                // Scratch Buffer for
building up strings
```

```
     char                 szUnknownType[PS_UNKNOWNSTRLEN];    // Buffer for
'Unknown Type' string
     char                 szUnknownSource[PS_UNKNOWNSTRLEN];  // Buffer for
'Unknown Source' string
     char                 szAppName[OLEUI_CCHKEYMAX]; // Application name of
Source. Used in the result text

      //   when Paste is selected. Obtained using clsidOD.

     // Information obtained from OBJECTDESCRIPTOR. This information is
accessed when the Paste
     //    radio button is selected.
     CLSID                clsidOD;              // ClassID of source
     SIZEL                sizelOD;              // sizel transfered in
                                                                    //
ObjectDescriptor
     LPSTR                szFullUserTypeNameOD; // Full User Type Name
     LPSTR                szSourceOfDataOD;     // Source of Data
     BOOL                 fSrcAspectIconOD;     // Does Source specify
DVASPECT_ICON?
     BOOL                 fSrcOnlyIconicOD;     // Does Source specify
OLEMISC_ONLYICONIC?
     HGLOBAL              hMetaPictOD;          // Metafile containing icon
and icon title
     HGLOBAL              hObjDesc;             // Handle to
OBJECTDESCRIPTOR structure from which the
                                                                    //
above information is obtained

     // Information obtained from LINKSRCDESCRIPTOR. This infomation is
accessed when the PasteLink
     //   radio button is selected.
     CLSID                clsidLSD;             // ClassID of source
     SIZEL                sizelLSD;             // sizel transfered in
                                                                    //
LinkSrcDescriptor
     LPSTR                szFullUserTypeNameLSD;// Full User Type Name
     LPSTR                szSourceOfDataLSD;    // Source of Data
     BOOL                 fSrcAspectIconLSD;    // Does Source specify
DVASPECT_ICON?
     BOOL                 fSrcOnlyIconicLSD;    // Does Source specify
OLEMISC_ONLYICONIC?
     HGLOBAL              hMetaPictLSD;         // Metafile containing icon
and icon title
     HGLOBAL              hLinkSrcDesc;         // Handle to
LINKSRCDESCRIPTOR structure from which the
                                                                    //
above information is obtained

     BOOL                 fClipboardChanged;    // Has clipboard content
changed
                                                                    //   if
so bring down dlg after
                                                                    //
ChangeIcon dlg returns.
```

```
} PASTESPECIAL, *PPASTESPECIAL, FAR *LPPASTESPECIAL;

// Data corresponding to each list item. A pointer to this structure is
attached to each
//   Paste\PasteLink list box item using LB_SETITEMDATA
typedef struct tagPASTELISTITEMDATA
{
   int                  nPasteEntriesIndex;   // Index of arrPasteEntries[]
corresponding to list item
   BOOL                 fCntrEnableIcon;      // Does calling application
(called container here)
                                                                    //
specify OLEUIPASTE_ENABLEICON for this item?
} PASTELISTITEMDATA, *PPASTELISTITEMDATA, FAR *LPPASTELISTITEMDATA;


//Internal function prototypes
//PASTESPL.C
BOOL CALLBACK EXPORT PasteSpecialDialogProc(HWND, UINT, WPARAM, LPARAM);
BOOL            FPasteSpecialInit(HWND hDlg, WPARAM, LPARAM);
BOOL            FTogglePasteType(HWND, LPPASTESPECIAL, DWORD);
void            ChangeListSelection(HWND, LPPASTESPECIAL, HWND);
void            EnableDisplayAsIcon(HWND, LPPASTESPECIAL);

void            ToggleDisplayAsIcon(HWND, LPPASTESPECIAL);
void            ChangeIcon(HWND, LPPASTESPECIAL);
void            SetPasteSpecialHelpResults(HWND, LPPASTESPECIAL);
BOOL            FAddPasteListItem(HWND, BOOL, int, LPPASTESPECIAL, LPMALLOC,
LPSTR, LPSTR);
BOOL            FFillPasteList(HWND, LPPASTESPECIAL);
BOOL            FFillPasteLinkList(HWND, LPPASTESPECIAL);
BOOL            FHasPercentS(LPCSTR, LPPASTESPECIAL);
HGLOBAL         AllocateScratchMem(LPPASTESPECIAL);
void            FreeListData(HWND);

#endif  //_PASTESPL_H_
```

PASTESPL.C   (WRAPUI Sample)

```
/*
 * PASTESPL.C
 *
 * Implements the OleUIPasteSpecial function which invokes the complete
 * Paste Special dialog.
 *
 * Copyright (c)1992 Microsoft Corporation, All Rights Reserved
 */

#define STRICT  1
#include "ole2ui.h"
#include "pastespl.h"
#include "common.h"
#include "utility.h"
#include "resimage.h"
#include "iconbox.h"
#include "geticon.h"
#include "icon.h"
#include "regdb.h"
#include <stdlib.h>

OLEDBGDATA

/*
 * OleUIPasteSpecial
 *
 * Purpose:
 *  Invokes the standard OLE Paste Special dialog box which allows the user
 *  to select the format of the clipboard object to be pasted or paste
linked.
 *
 * Parameters:
 *  lpPS         LPOLEUIPasteSpecial pointing to the in-out structure
 *               for this dialog.
 *
 * Return Value:
 *  UINT        One of the following codes or one of the standard error
codes (OLEUI_ERR_*)
 *              defined in OLE2UI.H, indicating success or error:
 *              OLEUI_OK                           User selected OK
 *              OLEUI_CANCEL                       User cancelled the dialog
 *              OLEUI_IOERR_SRCDATAOBJECTINVALID   lpSrcDataObject field of
OLEUIPASTESPECIAL invalid
 *              OLEUI_IOERR_ARRPASTEENTRIESINVALID arrPasteEntries field of
OLEUIPASTESPECIAL invalid
 *              OLEUI_IOERR_ARRLINKTYPESINVALID    arrLinkTypes field of
OLEUIPASTESPECIAL invalid
 *              OLEUI_PSERR_CLIPBOARDCHANGED       Clipboard contents
changed while dialog was up
 */

STDAPI_(UINT) OleUIPasteSpecial(LPOLEUIPASTESPECIAL lpPS)
```

```
{
     UINT        uRet;
     HGLOBAL     hMemDlg=NULL;

     uRet=UStandardValidation((LPOLEUISTANDARD)lpPS,
sizeof(OLEUIPASTESPECIAL)
           , &hMemDlg);

     if (uRet != OLEUI_SUCCESS)
           return uRet;

     //Validate PasteSpecial specific fields
     if (NULL == lpPS->lpSrcDataObj || IsBadReadPtr(lpPS->lpSrcDataObj,
sizeof(IDataObject)))
           uRet = OLEUI_IOERR_SRCDATAOBJECTINVALID;
     if (NULL == lpPS->arrPasteEntries || IsBadReadPtr(lpPS-
>arrPasteEntries,  sizeof(OLEUIPASTEENTRY)))
           uRet = OLEUI_IOERR_ARRPASTEENTRIESINVALID;
     if (NULL != lpPS->arrLinkTypes && IsBadReadPtr(lpPS->arrLinkTypes,
sizeof(UINT)))
           uRet = OLEUI_IOERR_ARRLINKTYPESINVALID;

     if (0!=lpPS->cClsidExclude)
           {
           if (NULL!=lpPS->lpClsidExclude && IsBadReadPtr(lpPS-
>lpClsidExclude
                 , lpPS->cClsidExclude*sizeof(CLSID)))
           uRet=OLEUI_IOERR_LPCLSIDEXCLUDEINVALID;
           }

     if (uRet >= OLEUI_ERR_STANDARDMIN)
     {
           if (NULL != hMemDlg)
                 FreeResource(hMemDlg);
           return uRet;
     }

     //Now that we've validated everything, we can invoke the dialog.
     uRet = UStandardInvocation(PasteSpecialDialogProc,
(LPOLEUISTANDARD)lpPS
           , hMemDlg, MAKEINTRESOURCE(IDD_PASTESPECIAL));

     /*
     * IF YOU ARE CREATING ANYTHING BASED ON THE RESULTS, DO IT HERE.
     */

     return uRet;
}


/*
 * PasteSpecialDialogProc
 *
 * Purpose:
 *  Implements the OLE Paste Special dialog as invoked through the
 *  OleUIPasteSpecial function.
 *
 * Parameters:
 *  Standard
 *
 * Return Value:
 *  Standard
 */

BOOL CALLBACK EXPORT PasteSpecialDialogProc(HWND hDlg, UINT iMsg, WPARAM
wParam, LPARAM lParam)
{
     LPOLEUIPASTESPECIAL     lpOPS;
     LPPASTESPECIAL          lpPS;
     BOOL                    fHook=FALSE;
     HCURSOR                 hCursorOld;

     //Declare Win16/Win32 compatible WM_COMMAND parameters.
     COMMANDPARAMS(wID, wCode, hWndMsg);

     //This will fail under WM_INITDIALOG, where we allocate it.
     lpPS=(LPPASTESPECIAL)LpvStandardEntry(hDlg, iMsg, wParam, lParam,
&fHook);

     //If the hook processed the message, we're done.
     if (0!=fHook)
           return fHook;

     // Process help message from Change Icon

     if (iMsg == uMsgHelp)
     {
           PostMessage(lpPS->lpOPS->hWndOwner, uMsgHelp, wParam, lParam);
           return FALSE;
     }

     //Process the temination message
     if (iMsg==uMsgEndDialog)
     {
           HWND    hwndNextViewer;

           // Free the icon/icon-title metafile corresponding to
Paste/PasteList option which is not selected
           if (lpPS->fLink)
                 OleUIMetafilePictIconFree(lpPS->hMetaPictOD);
           else OleUIMetafilePictIconFree(lpPS->hMetaPictLSD);

           // Free data associated with each list box entry
           FreeListData(GetDlgItem(hDlg, ID_PS_PASTELIST));
           FreeListData(GetDlgItem(hDlg, ID_PS_PASTELINKLIST));

           //Free any specific allocations before calling StandardCleanup
           if (lpPS->hObjDesc) GlobalFree(lpPS->hObjDesc);
           if (lpPS->hLinkSrcDesc) GlobalFree(lpPS->hLinkSrcDesc);
           if (lpPS->hBuff) GlobalFree(lpPS->hBuff);
           // Change the clipboard notification chain
           hwndNextViewer = GetProp(hDlg, NEXTCBVIEWER);
           if (hwndNextViewer != HWND_BROADCAST)
           {
                 SetProp(hDlg, NEXTCBVIEWER, HWND_BROADCAST);
                 ChangeClipboardChain(hDlg, hwndNextViewer);
           }
           RemoveProp(hDlg, NEXTCBVIEWER);

           StandardCleanup(lpPS, hDlg);
           EndDialog(hDlg, wParam);
           return TRUE;
     }

     switch (iMsg)
     {
           case WM_INITDIALOG:
                 hCursorOld = HourGlassOn();
                 FPasteSpecialInit(hDlg, wParam, lParam);
                 HourGlassOff(hCursorOld);
                 return FALSE;

           case WM_DRAWCLIPBOARD:
           {
                 HWND    hwndNextViewer = GetProp(hDlg, NEXTCBVIEWER);
                 HWND    hDlg_ChgIcon;

                 if (hwndNextViewer == HWND_BROADCAST)
                       break;

                 if (hwndNextViewer)
                 {
                       SendMessage(hwndNextViewer, iMsg, wParam, lParam);
                       // Refresh next viewer in case it got modified
                       //    by the SendMessage() (likely if multiple
                       //    PasteSpecial dialogs are up simultaneously)
                       hwndNextViewer = GetProp(hDlg, NEXTCBVIEWER);
                 }
                 SetProp(hDlg, NEXTCBVIEWER, HWND_BROADCAST);
                 ChangeClipboardChain(hDlg, hwndNextViewer);

                 /* OLE2NOTE: if the ChangeIcon dialog is currently up, then
                 **    we need to defer bringing down PasteSpecial dialog
                 **    until after ChangeIcon dialog returns. if the
                 **    ChangeIcon dialog is NOT up, then we can bring down
                 **    the PasteSpecial dialog immediately.
                 */
                 if
((hDlg_ChgIcon=(HWND)GetProp(hDlg,PROP_HWND_CHGICONDLG))!=NULL)
                 {
                       // ChangeIcon dialog is UP
                       lpPS->fClipboardChanged = TRUE;
                 } else {
                       // ChangeIcon dialog is NOT up
```

```
                        //  Free icon and icon title metafile
                        SendDlgItemMessage(
                                    hDlg, ID_PS_ICONDISPLAY, IBXM_IMAGEFREE,
0, 0L);

                        SendMessage(
                                    hDlg, uMsgEndDialog,
OLEUI_PSERR_CLIPBOARDCHANGED,0L);
                  }
                  break;
            }

            case WM_CHANGECBCHAIN:
            {
                  HWND    hwndNextViewer = GetProp(hDlg, NEXTCBVIEWER);

                  if (wParam == (UINT)hwndNextViewer)
#ifdef WIN32
                        SetProp(hDlg, NEXTCBVIEWER, (hwndNextViewer =
(HWND)lParam));
#else
                        SetProp(hDlg, NEXTCBVIEWER, (hwndNextViewer =
(HWND)LOWORD(lParam)));
#endif
                  else if (hwndNextViewer && hwndNextViewer !=
HWND_BROADCAST)
                        SendMessage(hwndNextViewer, iMsg, wParam, lParam);
                  break;
            }

            case WM_COMMAND:
                  switch (wID)
                  {
                        case ID_PS_PASTE:
                              FTogglePasteType(hDlg, lpPS, PSF_SELECTPASTE);
                              break;

                        case ID_PS_PASTELINK:
                              FTogglePasteType(hDlg, lpPS,
PSF_SELECTPASTELINK);
                              break;

                        case ID_PS_DISPLAYLIST:
                              switch (wCode)
                              {
                                    case LBN_SELCHANGE:
                                          ChangeListSelection(hDlg, lpPS,
hWndMsg);
                                          break;

                                    case LBN_DBLCLK:
                                          // Same as pressing OK
                                          SendCommand(hDlg, IDOK, BN_CLICKED,
hWndMsg);
```

```
                                         break;
                             }
                             break;

                       case ID_PS_DISPLAYASICON:
                             ToggleDisplayAsIcon(hDlg, lpPS);
                             break;

                       case ID_PS_CHANGEICON:
                             ChangeIcon(hDlg, lpPS);
                             if (lpPS->fClipboardChanged) {
                                   // Free icon and icon title metafile
                                   SendDlgItemMessage(
                                               hDlg, ID_PS_ICONDISPLAY,
IBXM_IMAGEFREE,0,0L);
                                   SendMessage(
                                               hDlg, uMsgEndDialog,
                                               OLEUI_PSERR_CLIPBOARDCHANGED,
0L);
                             }
                             break;

                       case IDOK:
                       {
                             BOOL fDestAspectIcon =
                                         ((lpPS->dwFlags &
PSF_CHECKDISPLAYASICON) ?
                                                     TRUE : FALSE);
                             lpOPS = lpPS->lpOPS;
                             // Return current flags
                             lpOPS->dwFlags = lpPS->dwFlags;
                             // Return index of arrPasteEntries[]
corresponding to format selected by user
                             lpOPS->nSelectedIndex = lpPS->nSelectedIndex;
                             // Return if user selected Paste or PasteLink
                             lpOPS->fLink = lpPS->fLink;

                             /* if user selected same ASPECT as displayed in
the
                             **    source, then sizel passed in the
                             **    ObjectDescriptor/LinkSrcDescriptor is
                             **    applicable. otherwise, the sizel does not
apply.
                             */
                             if (lpPS->fLink) {
                                   if (lpPS->fSrcAspectIconLSD ==
fDestAspectIcon)
                                         lpOPS->sizel = lpPS->sizelLSD;
                                   else
                                         lpOPS->sizel.cx = lpOPS->sizel.cy =
0;
                             } else {
                                   if (lpPS->fSrcAspectIconOD ==
fDestAspectIcon)
```

```
                                         lpOPS->sizel = lpPS->sizelOD;
                                   else
                                         lpOPS->sizel.cx = lpOPS->sizel.cy =
0;
                             }
                             // Return metafile with icon and icon title
that the user selected
                             lpOPS-
>hMetaPict=(HGLOBAL)SendDlgItemMessage(hDlg, ID_PS_ICONDISPLAY,

     IBXM_IMAGEGET, 0, 0L);
                             SendMessage(hDlg, uMsgEndDialog, OLEUI_OK, 0L);
                             break;
                       }
                       case IDCANCEL:
                             // Free icon and icon title metafile
                             SendDlgItemMessage(
                                         hDlg, ID_PS_ICONDISPLAY,
IBXM_IMAGEFREE, 0, 0L);
                             SendMessage(hDlg, uMsgEndDialog, OLEUI_CANCEL,
0L);

                             break;

                       case ID_OLEUIHELP:
                             PostMessage(lpPS->lpOPS->hWndOwner, uMsgHelp,
                                   (WPARAM)hDlg,
MAKELPARAM(IDD_PASTESPECIAL, 0));
                             break;
                 }
                 break;
     }
     return FALSE;
}


/*
 * FPasteSpecialInit
 *
 * Purpose:
 *  WM_INITIDIALOG handler for the Paste Special dialog box.
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  wParam          WPARAM of the message
 *  lParam          LPARAM of the message
 *
 * Return Value:
 *  BOOL            Value to return for WM_INITDIALOG.
 */

BOOL FPasteSpecialInit(HWND hDlg, WPARAM wParam, LPARAM lParam)
{
     LPPASTESPECIAL              lpPS;
     LPOLEUIPASTESPECIAL         lpOPS;
     HFONT                       hFont;
     BOOL                        fPasteAvailable, fPasteLinkAvailable;
     STGMEDIUM                   medium;
     LPOBJECTDESCRIPTOR          lpOD;
     LPLINKSRCDESCRIPTOR         lpLSD;
     int                         n;
     CLIPFORMAT                  cfFormat;

     // Copy the structure at lParam into our instance memory.
     lpPS = (LPPASTESPECIAL)LpvStandardInit(hDlg, sizeof(PASTESPECIAL),
TRUE, &hFont);

     // PvStandardInit sent a termination to us already.
     if (NULL == lpPS)
           return FALSE;

     lpOPS=(LPOLEUIPASTESPECIAL)lParam;

     // Copy other information from lpOPS that we might modify.
     lpPS->lpOPS = lpOPS;
     lpPS->dwFlags = lpOPS->dwFlags;

     // Initialize user selections in the Paste and PasteLink listboxes
     lpPS->nPasteListCurSel = 0;
     lpPS->nPasteLinkListCurSel = 0;

     // If we got a font, send it to the necessary controls.
     if (NULL!=hFont)
     {
           SendDlgItemMessage(hDlg, ID_PS_SOURCETEXT, WM_SETFONT,
(WPARAM)hFont, 0L);
           SendDlgItemMessage(hDlg, ID_PS_RESULTTEXT, WM_SETFONT,
(WPARAM)hFont, 0L);
     }

     // Hide the help button if required
     if (!(lpPS->lpOPS->dwFlags & PSF_SHOWHELP))
           StandardShowDlgItem(hDlg, ID_OLEUIHELP, SW_HIDE);


     // Hide all DisplayAsIcon related controls if it should be disabled
     if ( lpPS->dwFlags & PSF_DISABLEDISPLAYASICON ) {
             StandardShowDlgItem(hDlg, ID_PS_DISPLAYASICON, SW_HIDE);
             StandardShowDlgItem(hDlg, ID_PS_CHANGEICON, SW_HIDE);
             StandardShowDlgItem(hDlg, ID_PS_ICONDISPLAY, SW_HIDE);
     }

     // PSF_CHECKDISPLAYASICON is an OUT flag. Clear it if has been set on
the way in.
     lpPS->dwFlags = lpPS->dwFlags & ~PSF_CHECKDISPLAYASICON;

     //  Change the caption if required
     if (NULL != lpOPS->lpszCaption)
           SetWindowText(hDlg, lpOPS->lpszCaption);

     // Load 'Unknown Source' and 'Unknown Type' strings
     n = LoadString(ghInst, (UINT)IDS_PSUNKNOWNTYPE, lpPS->szUnknownType,
PS_UNKNOWNSTRLEN);
     if (n)
           n = LoadString(ghInst, (UINT)IDS_PSUNKNOWNSRC, lpPS-
>szUnknownSource, PS_UNKNOWNSTRLEN);
     if (!n)
     {
           PostMessage(hDlg, uMsgEndDialog, OLEUI_ERR_LOADSTRING, 0L);
           return FALSE;
     }
     lpPS->szAppName[0]='\0';

     // GetData CF_OBJECTDESCRIPTOR. If the object on the clipboard in an
OLE1 object (offering CF_OWNERLINK)
     // or has been copied to clipboard by FileMaager (offering
CF_FILENAME), an OBJECTDESCRIPTOR will be
     // created will be created from CF_OWNERLINK or CF_FILENAME. See
OBJECTDESCRIPTOR for more info.

     if (lpPS->hObjDesc = OleStdFillObjectDescriptorFromData(lpOPS-
>lpSrcDataObj, &medium, &cfFormat))
     {
           lpOD = GlobalLock(lpPS->hObjDesc);

           // Get FullUserTypeName, SourceOfCopy and CLSID
           if (lpOD->dwFullUserTypeName)
                 lpPS->szFullUserTypeNameOD = (LPSTR)lpOD+lpOD-
>dwFullUserTypeName;
           else lpPS->szFullUserTypeNameOD = lpPS->szUnknownType;

           if (lpOD->dwSrcOfCopy)
           {
                 lpPS->szSourceOfDataOD = (LPSTR)lpOD+lpOD->dwSrcOfCopy;
                 // If CF_FILENAME was offered, source of copy is a path
name. Fit the path to the
                 // static control that will display it.
                 if (cfFormat == cfFileName)
                       lpPS->szSourceOfDataOD = ChopText(GetDlgItem(hDlg,
ID_PS_SOURCETEXT), 0, lpPS->szSourceOfDataOD);
           }
           else lpPS->szSourceOfDataOD = lpPS->szUnknownSource;

           lpPS->clsidOD = lpOD->clsid;
           lpPS->sizelOD = lpOD->sizel;

           // Does source specify DVASPECT_ICON?
           if (lpOD->dwDrawAspect & DVASPECT_ICON)
              lpPS->fSrcAspectIconOD = TRUE;
           else lpPS->fSrcAspectIconOD = FALSE;

           // Does source specify OLEMISC_ONLYICONIC?
           if (lpOD->dwStatus & OLEMISC_ONLYICONIC)
                 lpPS->fSrcOnlyIconicOD = TRUE;
           else lpPS->fSrcOnlyIconicOD = FALSE;
           // Get application name of source from auxusertype3 in the
registration database
           if (0==OleStdGetAuxUserType(&lpPS->clsidOD, 3, lpPS->szAppName,
OLEUI_CCHKEYMAX, NULL))
           {
                  // Use "the application which created it" as the name of
the application
                  if (0==LoadString(ghInst, (UINT)IDS_PSUNKNOWNAPP, lpPS-
>szAppName, PS_UNKNOWNSTRLEN))
                  {
                        PostMessage(hDlg, uMsgEndDialog,
OLEUI_ERR_LOADSTRING, 0L);
                        return FALSE;
                  }
           }

           // Retrieve an icon from the object
           if (lpPS->fSrcAspectIconOD)
           {
                 lpPS->hMetaPictOD = OleStdGetData(
                       lpOPS->lpSrcDataObj,
                       CF_METAFILEPICT,
                       NULL,
                       DVASPECT_ICON,
                       &medium
                 );

           }
           // If object does not offer icon, obtain it from the CLSID
           if (NULL == lpPS->hMetaPictOD)
           {
                 lpPS->hMetaPictOD = GetIconOfClass(
                             ghInst,
                             &lpPS->clsidOD,
                             NULL,
                             TRUE);   // Use the short user type name
(auxusertype3)


           }
     }

     // Does object offer CF_LINKSRCDESCRIPTOR?
     if (lpPS->hLinkSrcDesc = OleStdGetData(
                 lpOPS->lpSrcDataObj,
                 (CLIPFORMAT)cfLinkSrcDescriptor,
                 NULL,
                 DVASPECT_CONTENT,
                 &medium))
     {
           // Get FullUserTypeName, SourceOfCopy and CLSID
           lpLSD = GlobalLock(lpPS->hLinkSrcDesc);
           if (lpLSD->dwFullUserTypeName)
                 lpPS->szFullUserTypeNameLSD = (LPSTR)lpLSD+lpLSD-
>dwFullUserTypeName;
```

```
           else lpPS->szFullUserTypeNameLSD = lpPS->szUnknownType;

           if (lpLSD->dwSrcOfCopy)
                 lpPS->szSourceOfDataLSD = (LPSTR)lpLSD+lpLSD->dwSrcOfCopy;
           else lpPS->szSourceOfDataLSD = lpPS->szUnknownSource;

           // if no ObjectDescriptor, then use LinkSourceDescriptor source
string
           if (!lpPS->hObjDesc)
                 lpPS->szSourceOfDataOD = lpPS->szSourceOfDataLSD;

           lpPS->clsidLSD = lpLSD->clsid;
           lpPS->sizelLSD = lpLSD->sizel;

           // Does source specify DVASPECT_ICON?
           if (lpLSD->dwDrawAspect & DVASPECT_ICON)
              lpPS->fSrcAspectIconLSD = TRUE;
           else lpPS->fSrcAspectIconLSD = FALSE;

           // Does source specify OLEMISC_ONLYICONIC?
           if (lpLSD->dwStatus & OLEMISC_ONLYICONIC)
                 lpPS->fSrcOnlyIconicLSD = TRUE;
           else lpPS->fSrcOnlyIconicLSD = FALSE;

           // Retrieve an icon from the object
           if (lpPS->fSrcAspectIconLSD)
           {
                 lpPS->hMetaPictLSD = OleStdGetData(
                       lpOPS->lpSrcDataObj,
                       CF_METAFILEPICT,
                       NULL,
                       DVASPECT_ICON,
                       &medium
                 );

           }
           // If object does not offer icon, obtain it from the CLSID
           if (NULL == lpPS->hMetaPictLSD)
           {
                 char szLabel[OLEUI_CCHLABELMAX];
                 HWND hIconWnd;
                 RECT IconRect;
                 int  nWidth;
                 LPSTR lpszLabel;

                 hIconWnd = GetDlgItem(hDlg, ID_PS_ICONDISPLAY);

                 GetClientRect(hIconWnd, &IconRect);

                 nWidth = ((IconRect.right-IconRect.left) * 3) / 2;   //
width is 1.5 times width of iconbox

                 LSTRCPYN(szLabel, lpPS->szSourceOfDataLSD,
sizeof(szLabel));
```

```
                 szLabel[sizeof(szLabel)-1] = '\0';

                 lpszLabel = ChopText(hIconWnd, nWidth, (LPSTR)szLabel);

                 lpPS->hMetaPictLSD = GetIconOfClass(
                             ghInst,
                             &lpPS->clsidLSD,
                             lpszLabel,       /* use chopped source string
as label */
                             FALSE            /* not applicable */
                 );
           }
     }
     else if (lpPS->hObjDesc)     // Does not offer CF_LINKSRCDESCRIPTOR but
offers CF_OBJECTDESCRIPTOR
     {
           // Copy the values of OBJECTDESCRIPTOR
           lpPS->szFullUserTypeNameLSD = lpPS->szFullUserTypeNameOD;
           lpPS->szSourceOfDataLSD = lpPS->szSourceOfDataOD;
           lpPS->clsidLSD = lpPS->clsidOD;
           lpPS->sizelLSD = lpPS->sizelOD;
           lpPS->fSrcAspectIconLSD = lpPS->fSrcAspectIconOD;
           lpPS->fSrcOnlyIconicLSD = lpPS->fSrcOnlyIconicOD;

           // Don't copy the hMetaPict; instead get a separate copy
           if (lpPS->fSrcAspectIconLSD)
           {
                 lpPS->hMetaPictLSD = OleStdGetData(
                       lpOPS->lpSrcDataObj,
                       CF_METAFILEPICT,
                       NULL,
                       DVASPECT_ICON,
                       &medium
                 );
           }
           if (NULL == lpPS->hMetaPictLSD)
           {
                 char szLabel[OLEUI_CCHLABELMAX];
                 HWND hIconWnd;
                 RECT IconRect;
                 int  nWidth;
                 LPSTR lpszLabel;

                 hIconWnd = GetDlgItem(hDlg, ID_PS_ICONDISPLAY);

                 GetClientRect(hIconWnd, &IconRect);

                 nWidth = ((IconRect.right-IconRect.left) * 3) / 2;   //
width is 1.5 times width of iconbox

                 LSTRCPYN(szLabel, lpPS->szSourceOfDataLSD,
sizeof(szLabel));
                 szLabel[sizeof(szLabel)-1] = '\0';

                 lpszLabel = ChopText(hIconWnd, nWidth, (LPSTR)szLabel);
                 lpPS->hMetaPictLSD = GetIconOfClass(
                             ghInst,
                             &lpPS->clsidLSD,
                             lpszLabel,   /* Use chopped source string as
label */
                             FALSE        /* Not applicable */
                 );
           }
     }

     // Not an OLE object
     if (lpPS->hObjDesc == NULL && lpPS->hLinkSrcDesc == NULL)
     {
            lpPS->szFullUserTypeNameLSD = lpPS->szFullUserTypeNameOD = lpPS-
>szUnknownType;
            lpPS->szSourceOfDataLSD = lpPS->szSourceOfDataOD = lpPS-
>szUnknownSource;
            lpPS->hMetaPictLSD = lpPS->hMetaPictOD = NULL;
     }

     // Allocate scratch memory to construct item names in the paste and
pastelink listboxes
     lpPS->hBuff = AllocateScratchMem(lpPS);
     if (lpPS->hBuff == NULL)
     {
        PostMessage(hDlg, uMsgEndDialog, OLEUI_ERR_GLOBALMEMALLOC, 0L);
        return FALSE;
     }

     // Select the Paste Link Button if specified. Otherwise select
     //      Paste Button by default
     if (lpPS->dwFlags & PSF_SELECTPASTELINK)
           lpPS->dwFlags = (lpPS->dwFlags & ~PSF_SELECTPASTE) |
PSF_SELECTPASTELINK;
     else
           lpPS->dwFlags =(lpPS->dwFlags & ~PSF_SELECTPASTELINK) |
PSF_SELECTPASTE;

     // Mark which PasteEntry formats are available from source data object
     OleStdMarkPasteEntryList(
                 lpOPS->lpSrcDataObj,lpOPS->arrPasteEntries,lpOPS-
>cPasteEntries);

     // Check if items are available to be pasted
     fPasteAvailable = FFillPasteList(hDlg, lpPS);
     if (!fPasteAvailable)
     {
           lpPS->dwFlags &= ~PSF_SELECTPASTE;
           EnableWindow(GetDlgItem(hDlg, ID_PS_PASTE), FALSE);
     }

     // Check if items are available to be paste-linked
     fPasteLinkAvailable = FFillPasteLinkList(hDlg, lpPS);
     if (!fPasteLinkAvailable)
     {
           lpPS->dwFlags &= ~PSF_SELECTPASTELINK;
           EnableWindow(GetDlgItem(hDlg, ID_PS_PASTELINK), FALSE);
     }

     // If one of Paste or PasteLink is disabled, select the other one
     //    regardless of what the input flags say
     if (fPasteAvailable && !fPasteLinkAvailable)
           lpPS->dwFlags |= PSF_SELECTPASTE;
     if (fPasteLinkAvailable && !fPasteAvailable)
           lpPS->dwFlags |= PSF_SELECTPASTELINK;

     if (lpPS->dwFlags & PSF_SELECTPASTE)
     {
           // FTogglePaste will set the PSF_SELECTPASTE flag, so clear it.
           lpPS->dwFlags &= ~PSF_SELECTPASTE;
           CheckRadioButton(hDlg, ID_PS_PASTE, ID_PS_PASTELINK,
ID_PS_PASTE);
           FTogglePasteType(hDlg, lpPS, PSF_SELECTPASTE);
     }
     else if (lpPS->dwFlags & PSF_SELECTPASTELINK)
     {
           // FTogglePaste will set the PSF_SELECTPASTELINK flag, so clear
it.
           lpPS->dwFlags &= ~PSF_SELECTPASTELINK;
           CheckRadioButton(hDlg, ID_PS_PASTE, ID_PS_PASTELINK,
ID_PS_PASTELINK);
           FTogglePasteType(hDlg, lpPS, PSF_SELECTPASTELINK);
     }
     else  // Items are not available to be be Pasted or Paste-Linked
     {
           // Enable or disable DisplayAsIcon and set the result text and
image
           EnableDisplayAsIcon(hDlg, lpPS);
           SetPasteSpecialHelpResults(hDlg, lpPS);
     }

     // Give initial focus to the list box
     SetFocus(GetDlgItem(hDlg, ID_PS_DISPLAYLIST));

     // Set property to handle clipboard change notifications
     SetProp(hDlg, NEXTCBVIEWER, HWND_BROADCAST);
     SetProp(hDlg, NEXTCBVIEWER, SetClipboardViewer(hDlg));

     lpPS->fClipboardChanged = FALSE;

     /*
      * PERFORM OTHER INITIALIZATION HERE.
      */

     // Call the hook with lCustData in lParam
     UStandardHook(lpPS, hDlg, WM_INITDIALOG, wParam, lpOPS->lCustData);
     return TRUE;
}
```

```
/*
 * FTogglePasteType
 *
 * Purpose:
 *  Toggles between Paste and Paste Link. The Paste list and PasteLink
 *  list are always invisible. The Display List is filled from either
 *  the Paste list or the PasteLink list depending on which Paste radio
 *  button is selected.
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  lpPS             Paste Special Dialog Structure
 *  dwOption        Paste or PasteSpecial option
 *
 * Return Value:
 *  BOOL            Returns TRUE if the option has already been selected.
 *                  Otherwise the option is selected and FALSE is returned
 */

BOOL FTogglePasteType(HWND hDlg, LPPASTESPECIAL lpPS, DWORD dwOption)
{
     DWORD dwTemp;
     HWND hList, hListDisplay;
     DWORD dwData;
     int i, nItems;
     LPSTR lpsz;

     // Skip all this if the button is already selected
     if (lpPS->dwFlags & dwOption)
           return TRUE;

     dwTemp = PSF_SELECTPASTE | PSF_SELECTPASTELINK;
     lpPS->dwFlags = (lpPS->dwFlags & ~dwTemp) | dwOption;

     // Hide IconDisplay. This prevents flashing if the icon display is
changed
     StandardShowDlgItem(hDlg, ID_PS_ICONDISPLAY, SW_HIDE);

     hListDisplay = GetDlgItem(hDlg, ID_PS_DISPLAYLIST);

     // If Paste was selected
     if (lpPS->dwFlags & PSF_SELECTPASTE)
     {
           // Set the Source of the object in the clipboard
           SetDlgItemText(hDlg, ID_PS_SOURCETEXT, lpPS->szSourceOfDataOD);

           // If an icon is available
           if (lpPS->hMetaPictOD)
                 // Set the icon display
                 SendDlgItemMessage(hDlg, ID_PS_ICONDISPLAY, IBXM_IMAGESET,
                         (WPARAM)lpPS->hMetaPictOD, 0L);


           hList = GetDlgItem(hDlg, ID_PS_PASTELIST);
           // We are switching from PasteLink to Paste. Remember current
selection
           //    in PasteLink list so it can be restored.
           lpPS->nPasteLinkListCurSel = (int)SendMessage(hListDisplay,
LB_GETCURSEL, 0, 0L);
           if (lpPS->nPasteLinkListCurSel == LB_ERR)
                 lpPS->nPasteLinkListCurSel = 0;
           // Remember if user selected Paste or PasteLink
           lpPS->fLink = FALSE;
     }
     else    // If PasteLink was selected
     {
           // Set the Source of the object in the clipboard
           SetDlgItemText(hDlg, ID_PS_SOURCETEXT, lpPS->szSourceOfDataLSD);

           // If an icon is available
           if (lpPS->hMetaPictLSD)
                 // Set the icon display
                 SendDlgItemMessage(hDlg, ID_PS_ICONDISPLAY, IBXM_IMAGESET,
                         (WPARAM)lpPS->hMetaPictLSD, 0L);


           hList = GetDlgItem(hDlg, ID_PS_PASTELINKLIST);
           // We are switching from Paste to PasteLink. Remember current
selection
           //    in Paste list so it can be restored.
           lpPS->nPasteListCurSel = (int)SendMessage(hListDisplay,
LB_GETCURSEL, 0, 0L);
           if (lpPS->nPasteListCurSel == LB_ERR)
                 lpPS->nPasteListCurSel = 0;
           // Remember if user selected Paste or PasteLink
           lpPS->fLink = TRUE;
     }

     // Turn drawing off while the Display List is being filled
     SendMessage(hListDisplay, WM_SETREDRAW, (WPARAM)FALSE, 0L);

     // Move data to Display list box
     SendMessage(hListDisplay, LB_RESETCONTENT, 0, 0L);
     nItems = (int) SendMessage(hList, LB_GETCOUNT, 0, 0L);
     lpsz = (LPSTR)GlobalLock(lpPS->hBuff);
     for (i = 0; i < nItems; i++)
     {
           SendMessage(hList, LB_GETTEXT, (WPARAM)i, (LPARAM)lpsz);
           dwData = SendMessage(hList, LB_GETITEMDATA, (WPARAM)i, 0L);
           SendMessage(hListDisplay, LB_INSERTSTRING, (WPARAM)i,
(LPARAM)lpsz);
           SendMessage(hListDisplay, LB_SETITEMDATA, (WPARAM)i, dwData);
     }
     GlobalUnlock(lpPS->hBuff);

     // Restore the selection in the Display List from user's last selection
     if (lpPS->dwFlags & PSF_SELECTPASTE)
```

```
           SendMessage(hListDisplay, LB_SETCURSEL, lpPS->nPasteListCurSel,
0L);
     else
           SendMessage(hListDisplay, LB_SETCURSEL, lpPS-
>nPasteLinkListCurSel, 0L);

     // Paint Display List
     SendMessage(hListDisplay, WM_SETREDRAW, (WPARAM)TRUE, 0L);
     InvalidateRect(hListDisplay, NULL, TRUE);
     UpdateWindow(hListDisplay);

     // Auto give the focus to the Display List
     SetFocus(hListDisplay);

     // Enable/Disable DisplayAsIcon and set the help result text and bitmap
corresponding to
     //    the current selection
     ChangeListSelection(hDlg, lpPS, hListDisplay);

     return FALSE;
}


/*
 * ChangeListSelection
 *
 * Purpose:
 *  When the user changes the selection in the list, DisplayAsIcon is
enabled or disabled,
 *  Result text and bitmap are updated and the index of the
arrPasteEntries[] corresponding
 *  to the current format selection is saved.
 *
 * Parameters:
 *  hDlg               HWND of the dialog
 *  lpPS                Paste Special Dialog Structure
 *  hList              HWND of the List
 *
 * Return Value:
 *  No return value
 */

void ChangeListSelection(HWND hDlg, LPPASTESPECIAL lpPS, HWND hList)
{
     LPPASTELISTITEMDATA lpItemData;
     int nCurSel;

     EnableDisplayAsIcon(hDlg, lpPS);
     SetPasteSpecialHelpResults(hDlg, lpPS);

     // Remember index of arrPasteEntries[] corresponding to the current
selection
     nCurSel = (int)SendMessage(hList, LB_GETCURSEL, 0, 0L);
     if (nCurSel == LB_ERR) return;
     lpItemData = (LPPASTELISTITEMDATA) SendMessage(hList, LB_GETITEMDATA,
                        (WPARAM)nCurSel, 0L);
     if ((LRESULT)lpItemData == LB_ERR) return;
     lpPS->nSelectedIndex = lpItemData->nPasteEntriesIndex;
}

/*
 * EnableDisplayAsIcon
 *
 * Purpose:
 *  Enable or disable the DisplayAsIcon button depending on whether
 *  the current selection can be displayed as an icon or not. The following
table describes
 *  the state of DisplayAsIcon. The calling application is termed CONTAINER,
the source
 *  of data on the clipboard is termed SOURCE.
 *  Y = Yes; N = No; Blank = State does not matter;
 * =====================================================================
 * SOURCE          SOURCE             CONTAINER             DisplayAsIcon
 * specifies       specifies          specifies             Initial State
 * DVASPECT_ICON   OLEMISC_ONLYICONIC OLEUIPASTE_ENABLEICON
 *
 *                                    N
Unchecked&Disabled
 *                 Y                  Y                     Checked&Disabled
 * Y               N                  Y                     Checked&Enabled
 * N               N                  Y
Unchecked&Enabled
 * =====================================================================
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  lpPS             Paste Special Dialog Structure
 *
 * Return Value:
 *  No return value
 */

void EnableDisplayAsIcon(HWND hDlg, LPPASTESPECIAL lpPS)
{
     int nIndex;
     BOOL fCntrEnableIcon;
     BOOL fSrcOnlyIconic = (lpPS->fLink) ? lpPS->fSrcOnlyIconicLSD : lpPS-
>fSrcOnlyIconicOD;
     BOOL fSrcAspectIcon = (lpPS->fLink) ? lpPS->fSrcAspectIconLSD : lpPS-
>fSrcAspectIconOD;
     HWND hList;
     LPPASTELISTITEMDATA lpItemData;
     HGLOBAL hMetaPict = (lpPS->fLink) ? lpPS->hMetaPictLSD : lpPS-
>hMetaPictOD;

     hList = GetDlgItem(hDlg, ID_PS_DISPLAYLIST);

     // Get data corresponding to the current selection in the listbox
     nIndex = (int)SendMessage(hList, LB_GETCURSEL, 0, 0);
     if (nIndex != LB_ERR)
     {
           lpItemData = (LPPASTELISTITEMDATA) SendMessage(hList,
LB_GETITEMDATA, (WPARAM)nIndex, 0L);
           if ((LRESULT)lpItemData != LB_ERR)
                 fCntrEnableIcon = lpItemData->fCntrEnableIcon;
           else fCntrEnableIcon = FALSE;
     }
     else fCntrEnableIcon = FALSE;

     // If there is an icon available
     if (hMetaPict != NULL)
     {
           if (!fCntrEnableIcon)          // Does CONTAINER specify
OLEUIPASTE_ENABLEICON?
           {
                 // Uncheck & Disable DisplayAsIcon
                 lpPS->dwFlags &= ~PSF_CHECKDISPLAYASICON;
                 CheckDlgButton(hDlg, ID_PS_DISPLAYASICON, FALSE);
                 EnableWindow(GetDlgItem(hDlg, ID_PS_DISPLAYASICON), FALSE);

                 // Hide IconDisplay and ChangeIcon button
                 StandardShowDlgItem(hDlg, ID_PS_ICONDISPLAY, SW_HIDE);
                 StandardShowDlgItem(hDlg, ID_PS_CHANGEICON, SW_HIDE);
           }
           else if (fSrcOnlyIconic)       // Does SOURCE specify
OLEMISC_ONLYICONIC?
           {
                 // Check & Disable DisplayAsIcon
                 lpPS->dwFlags |= PSF_CHECKDISPLAYASICON;
                 CheckDlgButton(hDlg, ID_PS_DISPLAYASICON, TRUE);
                 EnableWindow(GetDlgItem(hDlg, ID_PS_DISPLAYASICON), FALSE);

                 // Show IconDisplay and ChangeIcon button
                 StandardShowDlgItem(hDlg, ID_PS_ICONDISPLAY,
SW_SHOWNORMAL);
                 StandardShowDlgItem(hDlg, ID_PS_CHANGEICON, SW_SHOWNORMAL);
           }
           else if (fSrcAspectIcon)       // Does SOURCE specify
DVASPECT_ICON?
           {
                  // Check & Enable DisplayAsIcon
                  lpPS->dwFlags |= PSF_CHECKDISPLAYASICON;
                  CheckDlgButton(hDlg, ID_PS_DISPLAYASICON, TRUE);
                  EnableWindow(GetDlgItem(hDlg, ID_PS_DISPLAYASICON), TRUE);

                  // Show IconDisplay and ChangeIcon button
                  StandardShowDlgItem(hDlg, ID_PS_ICONDISPLAY,
SW_SHOWNORMAL);
                  StandardShowDlgItem(hDlg, ID_PS_CHANGEICON,
SW_SHOWNORMAL);
           }
           else
           {
                  //Uncheck and Enable DisplayAsIcon
```

```
                    lpPS->dwFlags &= ~PSF_CHECKDISPLAYASICON;
                    CheckDlgButton(hDlg, ID_PS_DISPLAYASICON, FALSE);
                    EnableWindow(GetDlgItem(hDlg, ID_PS_DISPLAYASICON), TRUE);

                    // Hide IconDisplay and ChangeIcon button
                    StandardShowDlgItem(hDlg, ID_PS_ICONDISPLAY, SW_HIDE);
                    StandardShowDlgItem(hDlg, ID_PS_CHANGEICON, SW_HIDE);

            }
     }
     else  // No icon available
     {
            // Unchecked & Disabled
            lpPS->dwFlags &= ~PSF_CHECKDISPLAYASICON;
            CheckDlgButton(hDlg, ID_PS_DISPLAYASICON, FALSE);
            EnableWindow(GetDlgItem(hDlg, ID_PS_DISPLAYASICON), FALSE);

            // Hide IconDisplay and ChangeIcon button
            StandardShowDlgItem(hDlg, ID_PS_ICONDISPLAY, SW_HIDE);
            StandardShowDlgItem(hDlg, ID_PS_CHANGEICON, SW_HIDE);
     }
}

/*
 * ToggleDisplayAsIcon
 *
 * Purpose:
 *  Toggles the DisplayAsIcon button. Hides or shows the Icon Display and
 *  the ChangeIcon button and changes the help result text and bitmap.
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  lpPS             Paste Special Dialog Structure
 *
 * Return Value:
 *  None
 *
 */

void ToggleDisplayAsIcon(HWND hDlg, LPPASTESPECIAL lpPS)
{
     BOOL fCheck;
     int i;

     fCheck = IsDlgButtonChecked(hDlg, ID_PS_DISPLAYASICON);

     if (fCheck)
            lpPS->dwFlags |= PSF_CHECKDISPLAYASICON;
     else lpPS->dwFlags &= ~PSF_CHECKDISPLAYASICON;

     // Set the help result text and bitmap
     SetPasteSpecialHelpResults(hDlg, lpPS);

     // Show or hide the Icon Display and ChangeIcon button depending
     // on the check state
     i = (fCheck) ? SW_SHOWNORMAL : SW_HIDE;
     StandardShowDlgItem(hDlg, ID_PS_ICONDISPLAY, i);

     StandardShowDlgItem(hDlg, ID_PS_CHANGEICON, i);
}

/*
 * ChangeIcon
 *
 * Purpose:
 *  Brings up the ChangeIcon dialog which allows the user to change
 *  the icon and label.
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  lpPS             Paste Special Dialog Structure
 *
 * Return Value:
 *  None
 *
 */

void ChangeIcon(HWND hDlg, LPPASTESPECIAL lpPS)
{
     OLEUICHANGEICON ci;
     UINT uRet;
     CLSID   clsid     = (lpPS->fLink) ? lpPS->clsidLSD : lpPS->clsidOD;

     //Initialize the structure
     _fmemset((LPOLEUICHANGEICON)&ci, 0, sizeof(ci));

     ci.hMetaPict = (HGLOBAL)SendDlgItemMessage(hDlg, ID_PS_ICONDISPLAY,
IBXM_IMAGEGET, 0, 0L);
     ci.cbStruct = sizeof(ci);
     ci.hWndOwner = hDlg;
     ci.clsid = clsid;
     ci.dwFlags  = CIF_SELECTCURRENT;

     // Only show help in the ChangeIcon dialog if we're showing it in this
dialog.
     if (lpPS->dwFlags & PSF_SHOWHELP)
           ci.dwFlags |= CIF_SHOWHELP;

     // Let the hook in to customize Change Icon if desired.
     uRet = UStandardHook(lpPS, hDlg, uMsgChangeIcon, 0, (LONG)(LPSTR)&ci);

     if (0 == uRet)
           uRet=(UINT)(OLEUI_OK==OleUIChangeIcon(&ci));

     // Update the display if necessary.
     if (0!=uRet)
     {
           /*
           * OleUIChangeIcon will have already freed our
           * current hMetaPict that we passed in when OK is
           * pressed in that dialog.  So we use 0L as lParam
           * here so the IconBox doesn't try to free the
           * metafilepict again.
           */
           SendDlgItemMessage(hDlg, ID_PS_ICONDISPLAY, IBXM_IMAGESET,
(WPARAM)ci.hMetaPict, 0L);
           // Remember the new icon chosen by the user. Note that Paste and
PasteLink have separate
           //    icons - changing one does not change the other.
           if (lpPS->fLink)
                 lpPS->hMetaPictLSD = ci.hMetaPict;
           else lpPS->hMetaPictOD = ci.hMetaPict;
     }
}

/*
 *SetPasteSpecialHelpResults
 *
 * Purpose:
 *  Sets the help result text and bitmap according to the current
 *  list selection. The following state table indicates which ResultText
 *  and ResultImage are selected. If %s in the lpstrFormatName is present,
 *  it is assumed that an object is being pasted/paste-linked, otherwise it
 *  is assumed that data is being pasted/paste-linked.
 *  Y = Yes; N = No; Blank = State does not matter;
 *  The numbers in the the ResultText and ResultImage columns refer to the
table
 *  entries that follow.
 * =====================================================================
 * Paste/       lpstrFormatName in               DisplayAsIcon Result
Result
 * PasteLink    arrPasteEntry[]contains %s       checked       Text
Image
 *              (Is Object == Y, Is Data == N)
 * Paste        N                                              1
1
 * Paste        Y                                N             2
2
 * Paste        Y                                Y             3
3
 * PasteLink    N                                              4
4
 * PasteLink    Y                                N             5
4
 * PasteLink    Y                                Y             6
5
 * =====================================================================
 * Result Text:
 *
 * 1. "Inserts the contents of the Clipboard into your document as <native
type name,
 *     and optionally an additional help sentence>"
 * 2. "Inserts the contents of the Clipboard into your document so that you
may
 *     activate it using <object app name>"
 * 3. "Inserts the contents of the Clipboard into your document so that you
may
 *     activate it using <object app name>.  It will be displayed as an
icon."
 * 4. "Inserts the contents of the Clipboard into your document as <native
type name>.
 *     Paste Link creates a link to the source file so that changes to the
source file
 *     will be reflected in your document."
 * 5. "Inserts a picture of the Clipboard contents into your document.
Paste Link
 *     creates a link to the source file so that changes to the source file
will be
 *     reflected in your document."
 * 6. "Inserts an icon into your document which represents the Clipboard
contents.
 *     Paste Link creates a link to the source file so that changes to the
source file
 *     will be reflected in your document."
 * =====================================================================
 * Result Image:
 *
 * 1. Clipboard Image
 * 2. Paste image, non-iconic.
 * 3. Paste image, iconic.
 * 4. Paste Link image, non-iconic
 * 5. Paste Link image, iconic
 * =====================================================================
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  lpPS             Paste Special Dialog Structure
 *
 * Return Value:
 *  No return value
 */
void SetPasteSpecialHelpResults(HWND hDlg, LPPASTESPECIAL lpPS)
{
     LPSTR               psz1, psz2, psz3, psz4;
     UINT                i, iString, iImage, cch;
     int                 nPasteEntriesIndex;
     BOOL                fDisplayAsIcon;
     BOOL                fIsObject;
     HWND                hList;
     LPPASTELISTITEMDATA  lpItemData;
     LPOLEUIPASTESPECIAL lpOPS = lpPS->lpOPS;
     LPSTR       szFullUserTypeName = (lpPS->fLink) ?
                             lpPS->szFullUserTypeNameLSD : lpPS-
>szFullUserTypeNameOD;
     LPSTR       szInsert;

     hList = GetDlgItem(hDlg, ID_PS_DISPLAYLIST);

     i=(UINT)SendMessage(hList, LB_GETCURSEL, 0, 0L);
     if (i != LB_ERR)
     {
           lpItemData = (LPPASTELISTITEMDATA)SendMessage(hList,
LB_GETITEMDATA, i, 0L);
           if ((LRESULT)lpItemData == LB_ERR) return;
           nPasteEntriesIndex = lpItemData->nPasteEntriesIndex;
           // Check if there is a '%s' in the lpstrFormatName, then an
object is being
           //   pasted/pastelinked. Otherwise Data is being pasted-
pastelinked.
           fIsObject = FHasPercentS(lpOPS-
>arrPasteEntries[nPasteEntriesIndex].lpstrFormatName,
                                                           lpPS);
     }
     else return;

     // Is DisplayAsIcon checked?
     fDisplayAsIcon=(0L!=(lpPS->dwFlags & PSF_CHECKDISPLAYASICON));

     szInsert = szFullUserTypeName;

     if (lpPS->dwFlags & PSF_SELECTPASTE)     // If user selected Paste
     {
           if (fIsObject)
           {
                 iString = (UINT)(fDisplayAsIcon ? IDS_PSPASTEOBJECTASICON :
IDS_PSPASTEOBJECT);
                 iImage  = (UINT)(fDisplayAsIcon ? RESULTIMAGE_EMBEDICON   :
RESULTIMAGE_EMBED);
                 szInsert = lpPS->szAppName;
           }
           else
           {
                 iString = (UINT)IDS_PSPASTEDATA;
                 iImage  = RESULTIMAGE_PASTE;
           }
     }
     else if (lpPS->dwFlags & PSF_SELECTPASTELINK)   // User selected
PasteLink
     {
           if (fIsObject)
           {
                 iString = (UINT)(fDisplayAsIcon ?
IDS_PSPASTELINKOBJECTASICON : IDS_PSPASTELINKOBJECT);
                 iImage  = (UINT)(fDisplayAsIcon ? RESULTIMAGE_LINKICON :
RESULTIMAGE_LINK);
           }
           else
           {
                 iString = (UINT)IDS_PSPASTELINKDATA;
                 iImage  = (UINT)RESULTIMAGE_LINK;
           }

     }
     else   // Should never occur.
     {
           iString = (UINT)IDS_PSNONOLE;
           iImage = RESULTIMAGE_PASTE;
     }

     // hBuff contains enough space for the 4 buffers required to build up 
the help
     //   result text.
     cch = (UINT)GlobalSize(lpPS->hBuff)/4;

     psz1=(LPSTR)GlobalLock(lpPS->hBuff);
     psz2=psz1+cch;
     psz3=psz2+cch;
     psz4=psz3+cch;

     // Default is an empty string.
     *psz1=0;

     if (0!=LoadString(ghInst, iString, psz1, cch))
     {
           // Insert the FullUserTypeName of the source object into the 
partial result text
           //   specified by the container.
           wsprintf(psz3, lpOPS-
>arrPasteEntries[nPasteEntriesIndex].lpstrResultText,
           (LPSTR)szInsert);
           // Insert the above partial result text into the standard result 
text.
           wsprintf(psz4, psz1, (LPSTR)psz3);
           psz1=psz4;
     }

     // If LoadString failed, we simply clear out the results (*psz1=0 
above)
     SetDlgItemText(hDlg, ID_PS_RESULTTEXT, psz1);

     // Change the result bitmap
     SendDlgItemMessage(hDlg, ID_PS_RESULTIMAGE, RIM_IMAGESET, iImage, 0L);

     GlobalUnlock(lpPS->hBuff);
}

/*
 * FAddPasteListItem
 *
 * Purpose:
 *  Adds an item to the list box
 *
 * Parameters:
 *  hList            HWND List into which item is to be added
 *  fInsertFirst     BOOL Insert in the beginning of the list?
 *  nPasteEntriesIndex int Index of Paste Entry array this list item 
corresponsds to
 *  lpPS             Paste Special Dialog Structure
 *  pIMalloc         LPMALLOC  Memory Allocator
 *  lpszBuf          LPSTR Scratch buffer to build up string for list entry
 *  lpszFullUserTypeName LPSTR full user type name for object entry
 *
 * Return Value:
 *  BOOL            TRUE if sucessful.
 *                  FALSE if unsucessful.
 */
BOOL FAddPasteListItem(
           HWND hList, BOOL fInsertFirst, int nPasteEntriesIndex,
           LPPASTESPECIAL lpPS,
           LPMALLOC pIMalloc, LPSTR lpszBuf, LPSTR lpszFullUserTypeName)
{
     LPOLEUIPASTESPECIAL lpOPS = lpPS->lpOPS;
     LPPASTELISTITEMDATA lpItemData;
     int                 nIndex;

     // Allocate memory for each list box item
     lpItemData = (LPPASTELISTITEMDATA)pIMalloc->lpVtbl->Alloc(
                 pIMalloc, (DWORD)sizeof(PASTELISTITEMDATA));
     if (NULL == lpItemData)
           return FALSE;

     // Fill data associated with each list box item
     lpItemData->nPasteEntriesIndex = nPasteEntriesIndex;
     lpItemData->fCntrEnableIcon = ((lpOPS-
>arrPasteEntries[nPasteEntriesIndex].dwFlags &
                 OLEUIPASTE_ENABLEICON) ? TRUE : FALSE);

     // Build list box entry string, insert the string and add the data the
corresponds to it
     wsprintf(
                 (LPSTR)lpszBuf,
                 lpOPS->arrPasteEntries[nPasteEntriesIndex].lpstrFormatName,
                 (LPSTR)lpszFullUserTypeName
     );

     // only add to listbox if not a duplicate
     if (LB_ERR!=SendMessage(hList,LB_FINDSTRING, 0, (LPARAM)
(LPSTR)lpszBuf)) {
           // item is already in list; SKIP this one
           pIMalloc->lpVtbl->Free(pIMalloc, (LPVOID)lpItemData);
           return TRUE;    // this is NOT an error
     }

     nIndex = (int)SendMessage(
                 hList,
                 (fInsertFirst ? LB_INSERTSTRING : LB_ADDSTRING),
                 0,
                 (LPARAM)(LPSTR)lpszBuf
     );
     SendMessage(
                 hList,
                 LB_SETITEMDATA,
                 nIndex,
                 (LPARAM)(LPPASTELISTITEMDATA)lpItemData
     );
     return TRUE;
}


/*
 * FFillPasteList
 *
 * Purpose:
 *  Fills the invisible paste list with the formats offered by the clipboard
object and
 *  asked for by the container.
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  lpPS             Paste Special Dialog Structure
 *
 * Return Value:
 *  BOOL            TRUE if sucessful and if formats could be found.
 *                  FALSE if unsucessful or if no formats could be found.
 */
BOOL FFillPasteList(HWND hDlg, LPPASTESPECIAL lpPS)
{
     LPOLEUIPASTESPECIAL lpOPS = lpPS->lpOPS;
     LPMALLOC            pIMalloc     = NULL;
     LPSTR               lpszBuf      = (LPSTR)GlobalLock(lpPS->hBuff);
     HWND                hList;
     int                 i, j;
     int                 nItems = 0;
     int                 nDefFormat = -1;
     BOOL                fTryObjFmt = FALSE;
     BOOL                fInsertFirst;
     BOOL                fExclude;
     HRESULT             hrErr;

     hrErr = CoGetMalloc(MEMCTX_TASK, &pIMalloc);
     if (hrErr != NOERROR)
           goto error;

     hList = GetDlgItem(hDlg, ID_PS_PASTELIST);

     // Loop over the target's priority list of formats
     for (i = 0; i < lpOPS->cPasteEntries; i++)
     {
           if (lpOPS->arrPasteEntries[i].dwFlags != OLEUIPASTE_PASTEONLY &&
                       !(lpOPS->arrPasteEntries[i].dwFlags &
OLEUIPASTE_PASTE))
                 continue;

           fInsertFirst = FALSE;

           if (lpOPS->arrPasteEntries[i].fmtetc.cfFormat==cfFileName
                       || lpOPS-
>arrPasteEntries[i].fmtetc.cfFormat==cfEmbeddedObject
```

```
                        || lpOPS-
>arrPasteEntries[i].fmtetc.cfFormat==cfEmbedSource) {
                  if (! fTryObjFmt) {
                        fTryObjFmt = TRUE;      // only use 1st object format
                        fInsertFirst = TRUE;    // OLE obj format should
always be 1st

                        //Check if this CLSID is in the exclusion list.
                        fExclude=FALSE;

                        for (j=0; j < (int)lpOPS->cClsidExclude; j++)
                        {
                              if (IsEqualCLSID(&lpPS->clsidOD,
                                                          (LPCLSID)(lpOPS-
>lpClsidExclude+j)))
                              {
                                    fExclude=TRUE;
                                    break;
                              }
                        }

                        if (fExclude)
                              continue;   // don't add the object entry to
list

                  } else {
                        continue;   // already added an object format to list
                  }
            }

            // add to list if entry is marked TRUE
            if (lpOPS->arrPasteEntries[i].dwScratchSpace) {
                  if (nDefFormat < 0)
                        nDefFormat = (fInsertFirst ? 0 : nItems);
                  else if (fInsertFirst)
                        nDefFormat++;   // adjust for obj fmt inserted 1st in
list

                  if (!FAddPasteListItem(hList, fInsertFirst, i, lpPS,
pIMalloc,
                                    lpszBuf, lpPS->szFullUserTypeNameOD))
                        goto error;
                  nItems++;
            }
      }

      // initialize selection to first format matched in list
      if (nDefFormat >= 0)
            lpPS->nPasteListCurSel = nDefFormat;

      // Clean up
      if (pIMalloc)
            pIMalloc->lpVtbl->Release(pIMalloc);
      if (lpszBuf)
```

```
        GlobalUnlock(lpPS->hBuff);

     // If no items have been added to the list box (none of the formats
     //   offered by the source matched those acceptable to the container),
     //   return FALSE
     if (nItems > 0)
           return TRUE;
     else
           return FALSE;

error:
     if (pIMalloc)
           pIMalloc->lpVtbl->Release(pIMalloc);
     if (lpszBuf)
        GlobalUnlock(lpPS->hBuff);
     FreeListData(hList);

     return FALSE;
}


/*
 * FFillPasteLinkList
 *
 * Purpose:
 *  Fills the invisible paste link list with the formats offered by the
clipboard object and
 *  asked for by the container.
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  lpPS             Paste Special Dialog Structure
 *
 * Return Value:
 *  BOOL            TRUE if sucessful and if formats could be found.
 *                  FALSE if unsucessful or if no formats could be found.
 */
BOOL FFillPasteLinkList(HWND hDlg, LPPASTESPECIAL lpPS)
{
     LPOLEUIPASTESPECIAL lpOPS        = lpPS->lpOPS;
     LPDATAOBJECT        lpSrcDataObj = lpOPS->lpSrcDataObj;
     LPENUMFORMATETC     lpEnumFmtEtc = NULL;
     LPMALLOC            pIMalloc     = NULL;
     LPSTR               lpszBuf      = (LPSTR)GlobalLock(lpPS->hBuff);
     OLEUIPASTEFLAG      pasteFlag;
     UINT arrLinkTypesSupported[PS_MAXLINKTYPES];  // Array of flags that
                                                                         //
indicate which link types
                                                                         //
are supported by source.
     FORMATETC           fmtetc;
     int                 i, j;
     int                 nItems = 0;
     BOOL                fLinkTypeSupported = FALSE;
     HWND                hList;
```

```
    int                 nDefFormat = -1;
    BOOL                fTryObjFmt = FALSE;
    BOOL                fInsertFirst;
    HRESULT             hrErr;

    hrErr = CoGetMalloc(MEMCTX_TASK, &pIMalloc);
    if (hrErr != NOERROR)
          goto error;

    // Remember which link type formats are offered by lpSrcDataObj.
    _fmemset(&fmtetc, 0, sizeof(FORMATETC));
    for (i = 0; i < lpOPS->cLinkTypes; i++)
    {
          if (lpOPS->arrLinkTypes[i] == cfLinkSource) {
                OLEDBG_BEGIN2("OleQueryLinkFromData called\r\n")
                hrErr = OleQueryLinkFromData(lpSrcDataObj);
                OLEDBG_END2
                if(NOERROR == hrErr)
                {
                      arrLinkTypesSupported[i] = 1;
                      fLinkTypeSupported = TRUE;
                }
                else arrLinkTypesSupported[i] = 0;
          }
          else {
                fmtetc.cfFormat = lpOPS->arrLinkTypes[i];
                fmtetc.dwAspect = DVASPECT_CONTENT;
                fmtetc.tymed    = 0xFFFFFFFF;       // All tymed values
                fmtetc.lindex   = -1;
                OLEDBG_BEGIN2("IDataObject::QueryGetData called\r\n")
                hrErr = lpSrcDataObj->lpVtbl-
>QueryGetData(lpSrcDataObj,&fmtetc);
                OLEDBG_END2
                if(NOERROR == hrErr)
                {
                      arrLinkTypesSupported[i] = 1;
                      fLinkTypeSupported = TRUE;
                }
                else arrLinkTypesSupported[i] = 0;
          }
    }
    // No link types are offered by lpSrcDataObj
    if (! fLinkTypeSupported) {
          nItems = 0;
          goto cleanup;
    }

    hList = GetDlgItem(hDlg, ID_PS_PASTELINKLIST);

    // Enumerate the formats acceptable to container
    for (i = 0; i < lpOPS->cPasteEntries; i++)
    {
          fLinkTypeSupported = FALSE;
```

```
           // If container will accept any link type offered by source
object
           if (lpOPS->arrPasteEntries[i].dwFlags & OLEUIPASTE_LINKANYTYPE)
                 fLinkTypeSupported = TRUE;
           else
           {
                 // Check if any of the link types offered by the source
                 //    object are acceptable to the container
                 // This code depends on the LINKTYPE enum values being
powers of 2
                 for (pasteFlag = OLEUIPASTE_LINKTYPE1, j = 0;
                        j < lpOPS->cLinkTypes;
                        pasteFlag*=2, j++)
                 {
                       if ((lpOPS->arrPasteEntries[i].dwFlags & pasteFlag)
&&
                                   arrLinkTypesSupported[j])
                       {
                             fLinkTypeSupported = TRUE;
                             break;
                       }
                 }
           }

           fInsertFirst = FALSE;

           if (lpOPS->arrPasteEntries[i].fmtetc.cfFormat==cfFileName
                       || lpOPS-
>arrPasteEntries[i].fmtetc.cfFormat==cfLinkSource) {
                 if (! fTryObjFmt) {
                       fTryObjFmt = TRUE;      // only use 1st object format
                       fInsertFirst = TRUE;    // OLE obj format should
always be 1st
                 } else {
                       continue;   // already added an object format to list
                 }
           }

           // add to list if entry is marked TRUE
           if (fLinkTypeSupported && lpOPS-
>arrPasteEntries[i].dwScratchSpace) {
                 if (nDefFormat < 0)
                       nDefFormat = (fInsertFirst ? 0 : nItems);
                 else if (fInsertFirst)
                       nDefFormat++;   // adjust for obj fmt inserted 1st in
list

                 if (!FAddPasteListItem(hList, fInsertFirst, i, lpPS,
pIMalloc,
                                   lpszBuf, lpPS->szFullUserTypeNameLSD))
                       goto error;
                 nItems++;
           }
     } // end FOR
     nItems = (int)SendMessage(hList, LB_GETCOUNT, 0, 0L);

     // initialize selection to first format matched in list
     if (nDefFormat >= 0)
           lpPS->nPasteLinkListCurSel = nDefFormat;

cleanup:
     // Clean up
     if (pIMalloc)
           pIMalloc->lpVtbl->Release(pIMalloc);
     if (lpszBuf)
        GlobalUnlock(lpPS->hBuff);

     // If no items have been added to the list box (none of the formats
     //   offered by the source matched those acceptable to the
destination),
     //   return FALSE
     if (nItems > 0)
           return TRUE;
     else
           return FALSE;

error:
     if (pIMalloc)
           pIMalloc->lpVtbl->Release(pIMalloc);
     if (lpszBuf)
        GlobalUnlock(lpPS->hBuff);
     FreeListData(hList);

     return FALSE;
}


/*
 * FreeListData
 *
 * Purpose:
 *  Free the local memory associated with each list box item
 *
 * Parameters:
 *  hList           HWND of the list
 *
 * Return Value:
 *  None
 */
void FreeListData(HWND hList)
{
     int                nItems, i;
     LPPASTELISTITEMDATA lpItemData;
     LPMALLOC           pIMalloc;
     HRESULT            hrErr;

     hrErr = CoGetMalloc(MEMCTX_TASK, &pIMalloc);
     if (hrErr != NOERROR)
```

```
            return;

     nItems = (int) SendMessage(hList, LB_GETCOUNT, 0, 0L);
     for (i = 0; i < nItems; i++)
     {
            lpItemData = (LPPASTELISTITEMDATA)SendMessage(hList,
LB_GETITEMDATA, (WPARAM)i, 0L);
            if ((LRESULT)lpItemData != LB_ERR)
                  pIMalloc->lpVtbl->Free(pIMalloc, (LPVOID)lpItemData);
     }
     pIMalloc->lpVtbl->Release(pIMalloc);
}

/*
 * FHasPercentS
 *
 * Purpose:
 *  Determines if string contains %s.
 *
 * Parameters:
 *  lpsz            LPCSTR string in which occurence of '%s' is looked for
 *
 * Return Value:
 *  BOOL            TRUE if %s is found, else FALSE.
 */

BOOL FHasPercentS(LPCSTR lpsz, LPPASTESPECIAL lpPS)
{
   int n = 0;
   LPSTR lpszTmp;

   if (!lpsz) return FALSE;
   // Copy input string to buffer. This allows caller to pass a
   //   code-based string. Code segments may be swapped out in low memory
situations
   //   and so code-based strings need to be copied before string elements
can be accessed.
   lpszTmp = (LPSTR)GlobalLock(lpPS->hBuff);
   lstrcpy(lpszTmp, lpsz);


   while (*lpszTmp)
   {
        if (*lpszTmp == '%')
        {
              lpszTmp = AnsiNext(lpszTmp);
              if (*lpszTmp == 's')            // If %s, return
              {
                    GlobalUnlock(lpPS->hBuff);
                    return TRUE;
              }
              else if (*lpszTmp == '%')        // if %%, skip to next
character
                    lpszTmp = AnsiNext(lpszTmp);
        }
        else lpszTmp = AnsiNext(lpszTmp);
   }

   GlobalUnlock(lpPS->hBuff);
   return FALSE;
}

/*
 * AllocateScratchMem
 *
 * Purpose:
 *  Allocates scratch memory for use by the PasteSpecial dialog. The memory
is
 *  is used as the buffer for building up strings using wsprintf. Strings
are built up
 *  using the buffer while inserting items into the Paste & PasteLink lists
and while
 *  setting the help result text. It must be big  enough to handle the
string that results after
 *  replacing the %s in the lpstrFormatName and lpstrResultText in
arrPasteEntries[]
 *  by the FullUserTypeName. It must also be big enough to build the
dialog's result text
 *  after %s substitutions by the FullUserTypeName or the ApplicationName.
 *
 * Parameters:
 *  lpPS             Paste Special Dialog Structure
 *
 * Return Value:
 *  HGLOBAL         Handle to allocated global memory
 */

HGLOBAL AllocateScratchMem(LPPASTESPECIAL lpPS)
{
     LPOLEUIPASTESPECIAL lpOPS = lpPS->lpOPS;
     int nLen, i;
     int nSubstitutedText = 0;
     int nAlloc = 0;

     // Get the maximum length of the FullUserTypeNames specified by
OBJECTDESCRIPTOR
     //   and the LINKSRCDESCRIPTOR and the Application Name. Any of these
may be substituted
     //   for %s in the result-text/list entries.
     if (lpPS->szFullUserTypeNameOD)
           nSubstitutedText = lstrlen(lpPS->szFullUserTypeNameOD);
     if (lpPS->szFullUserTypeNameLSD)
           nSubstitutedText = __max(nSubstitutedText, lstrlen(lpPS-
>szFullUserTypeNameLSD));
     if (lpPS->szAppName)
           nSubstitutedText = __max(nSubstitutedText, lstrlen(lpPS-
>szAppName));

     // Get the maximum length of lpstrFormatNames & lpstrResultText in
arrPasteEntries
```

```
     nLen = 0;
     for (i = 0; i < lpOPS->cPasteEntries; i++)
     {
        nLen = __max(nLen, lstrlen(lpOPS-
>arrPasteEntries[i].lpstrFormatName));
        nLen = __max(nLen, lstrlen(lpOPS-
>arrPasteEntries[i].lpstrResultText));
     }

     // Get the maximum length of lpstrFormatNames and lpstrResultText after
%s  has
     //   been substituted (At most one %s can appear in each string).
     //   Add 1 to hold NULL terminator.
     nAlloc = nLen+nSubstitutedText+1;

     // Allocate scratch memory to be used to build strings
     // nAlloc is big enough to hold any of the lpstrResultText or
lpstrFormatName in arrPasteEntries[]
     //   after %s substitution.
     // We also need space to build up the help result text. 512 is the
maximum length of the
     //   standard dialog help text before substitutions. 512+nAlloc is the
maximum length
     //   after %s substition.
     // SetPasteSpecialHelpResults() requires 4 such buffers to build up the
result text
     return GlobalAlloc(GHND, (DWORD)4*(512+nAlloc));
}
```

PRECOMP.C (WRAPUI Sample)

```
/*
 * PRECOMP.C
 *
 * This file is used to precompile the OLE2UI.H header file
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */

#define STRICT  1
#include "ole2ui.h"

#if defined( _DEBUG ) && !defined( DLL_VER )
// This dummy variable is needed in order for the static link version
// of this library to work correctly.  When we include PRECOMP.OBJ
// in our library (.LIB file), it will only get linked into our
// application IF at least one symbol in precomp.c is referenced from
// either our EXE or LIB.  Therefore, we will define a variable in
// PRECOMP.C and reference it in OLE2UI.C which includes the function
// OleUIInitialize which MUST be called by the static lib user.

int near g_nOleUIStaticLibDummy = 0;
#endif
```

REGDB.H (WRAPUI Sample)

```
// This file is now OBSOLETE (include olestd.h instead)
/*
 *  Regdb.h
 *
 *    (c) Copyright Microsoft Corp. 1992 All Rights Reserved
 */

// Function prototypes moved to olestd.h
```

REGDB.C (WRAPUI Sample)

```
/*
 *  REGDB.C
 *
 *  Functions to query the registration database
 *
 *  OleStdGetMiscStatusOfClass
 *  OleStdGetDefaultFileFormatOfClass
 *  OleStdGetAuxUserType
 *  OleStdGetUserTypeOfClass
 *
 *    (c) Copyright Microsoft Corp. 1992-1993 All Rights Reserved
 *
 */

#define STRICT  1
#include "ole2ui.h"
#include "common.h"
#include <ctype.h>

OLEDBGDATA


// Replacement for stdlib atol,
// which didn't work and doesn't take far pointers.
// Must be tolerant of leading spaces.
//
//
static LONG Atol(LPSTR lpsz)
{
     signed int sign = +1;
     UINT base = 10;
     LONG l = 0;

     if (NULL==lpsz)
     {
           OleDbgAssert (0);
           return 0;
     }
     while (isspace(*lpsz))
           lpsz++;

     if (*lpsz=='-')
     {
           lpsz++;
           sign = -1;
     }
     if (lpsz[0]=='0' && lpsz[1]=='x')
     {
           base = 16;
           lpsz+=2;
     }
     if (base==10)
     {
           while (isdigit(*lpsz))
           {
                 l = l * base + *lpsz - '0';
                 lpsz++;
           }
     }
     else
     {
           OleDbgAssert (base==16);
           while (isxdigit(*lpsz))
           {
                 l = l * base + isdigit(*lpsz) ? *lpsz - '0' :
toupper(*lpsz) - 'A' + 10;
                 lpsz++;
           }
     }
     return l * sign;
}



/*
 * OleStdGetUserTypeOfClass(REFCLSID, LPSTR, UINT, HKEY)
 *
 * Purpose:
 *  Returns the user type (human readable class name) of the specified
class.
 *
 * Parameters:
 *  rclsid          pointer to the clsid to retrieve user type of.
 *  lpszUserType    pointer to buffer to return user type in.
 *  cch             length of buffer pointed to by lpszUserType
 *  hKey            hKey for reg db - if this is NULL, then we
 *                   open and close the reg db within this function.  If it
 *                   is non-NULL, then we assume it's a valid key to the
 *                   \ root and use it without closing it. (useful
 *                   if you're doing lots of reg db stuff).
 *
 * Return Value:
 *  UINT            Number of characters in returned string.  0 on error.
 *
 */
STDAPI_(UINT) OleStdGetUserTypeOfClass(REFCLSID rclsid, LPSTR lpszUserType,
UINT cch, HKEY hKey)
{

   LONG     dw;
   LONG     lRet;
   LPSTR    lpszCLSID, lpszProgID;
   BOOL     fFreeProgID = FALSE;
   BOOL     bCloseRegDB = FALSE;
   char     szKey[128];
   LPMALLOC lpIMalloc;
   if (!lpszUserType)
        return 0;

   *lpszUserType = '\0';
   if (hKey == NULL)
   {

      //Open up the root key.
      lRet=RegOpenKey(HKEY_CLASSES_ROOT, NULL, &hKey);

      if ((LONG)ERROR_SUCCESS!=lRet)
        return (UINT)FALSE;

      bCloseRegDB = TRUE;
   }

   // Get a string containing the class name
   StringFromCLSID(rclsid, &lpszCLSID);

   wsprintf(szKey, "CLSID\\%s", lpszCLSID);


   dw=cch;
   lRet = RegQueryValue(hKey, szKey, lpszUserType, &dw);

   if ((LONG)ERROR_SUCCESS!=lRet) {
        // Load 'Unknown Source' and 'Unknown Type' strings
        dw = (LONG)LoadString(ghInst, (UINT)IDS_PSUNKNOWNTYPE, lpszUserType,
cch);
   }

   if ( ((LONG)ERROR_SUCCESS!=lRet) && (CoIsOle1Class(rclsid)) )
   {
       // We've got an OLE 1.0 class, so let's try to get the user type
       // name from the ProgID entry.

       ProgIDFromCLSID(rclsid, &lpszProgID);
       fFreeProgID = TRUE;

       dw = cch;
       lRet = RegQueryValue(hKey, lpszProgID, lpszUserType, &dw);

       if ((LONG)ERROR_SUCCESS != lRet)
           dw = 0;
   }


   if (NOERROR == CoGetMalloc(MEMCTX_TASK, &lpIMalloc))
   {
        if (fFreeProgID)
            lpIMalloc->lpVtbl->Free(lpIMalloc, (LPVOID)lpszProgID);

        lpIMalloc->lpVtbl->Free(lpIMalloc, (LPVOID)lpszCLSID);
        lpIMalloc->lpVtbl->Release(lpIMalloc);
    }

    if (bCloseRegDB)
        RegCloseKey(hKey);

    return (UINT)dw;

}




/*
 * OleStdGetAuxUserType(RCLSID, WORD, LPSTR, int, HKEY)
 *
 * Purpose:
 *  Returns the specified AuxUserType from the reg db.
 *
 * Parameters:
 *  rclsid          pointer to the clsid to retrieve aux user type of.
 *  hKey            hKey for reg db - if this is NULL, then we
 *                   open and close the reg db within this function.  If it
 *                   is non-NULL, then we assume it's a valid key to the
 *                   \ root and use it without closing it. (useful
 *                   if you're doing lots of reg db stuff).
 *  wAuxUserType    which aux user type field to look for.  In 4/93 release
 *                  2 is short name and 3 is exe name.
 *  lpszUserType    pointer to buffer to return user type in.
 *  cch             length of buffer pointed to by lpszUserType
 *
 * Return Value:
 *  UINT            Number of characters in returned string.  0 on error.
 *
 */
STDAPI_(UINT) OleStdGetAuxUserType(REFCLSID rclsid,
                                                  UINT     wAuxUserType,
                                                  LPSTR    lpszAuxUserType,
                                                  int      cch,
                                                  HKEY     hKey)
{
    HKEY     hThisKey;
    BOOL     fCloseRegDB = FALSE;
    LONG     dw;
    LRESULT  lRet;
    LPSTR    lpszCLSID;
    LPMALLOC lpIMalloc;
    char     szKey[OLEUI_CCHKEYMAX];
    char     szTemp[32];

    lpszAuxUserType[0] = '\0';

    if (NULL == hKey)
    {
        lRet = RegOpenKey(HKEY_CLASSES_ROOT, NULL, &hThisKey);
```

```
       if (ERROR_SUCCESS != lRet)
             return 0;
   }
   else
       hThisKey = hKey;

   StringFromCLSID(rclsid, &lpszCLSID);

   lstrcpy(szKey, "CLSID\\");
   lstrcat(szKey, lpszCLSID);
   wsprintf(szTemp, "\\AuxUserType\\%d", wAuxUserType);
   lstrcat(szKey, szTemp);

   dw = cch;

   lRet = RegQueryValue(hThisKey, szKey, lpszAuxUserType, &dw);

   if (ERROR_SUCCESS != lRet) {
      dw = 0;
      lpszAuxUserType[0] = '\0';
   }


   if (fCloseRegDB)
       RegCloseKey(hThisKey);

   if (NOERROR == CoGetMalloc(MEMCTX_TASK, &lpIMalloc))
   {
        lpIMalloc->lpVtbl->Free(lpIMalloc, (LPVOID)lpszCLSID);
        lpIMalloc->lpVtbl->Release(lpIMalloc);
   }

   return (UINT)dw;
}



/*
 * OleStdGetMiscStatusOfClass(REFCLSID, HKEY)
 *
 * Purpose:
 *  Returns the value of the misc status for the given clsid.
 *
 * Parameters:
 *  rclsid          pointer to the clsid to retrieve user type of.
 *  hKey            hKey for reg db - if this is NULL, then we
 *                   open and close the reg db within this function.  If it
 *                   is non-NULL, then we assume it's a valid key to the
 *                   \\CLSID root and use it without closing it. (useful
 *                   if you're doing lots of reg db stuff).
 *
 * Return Value:
 *  BOOL            TRUE on success, FALSE on failure.
 *
 */
STDAPI_(BOOL) OleStdGetMiscStatusOfClass(REFCLSID rclsid, HKEY hKey, DWORD
FAR * lpdwValue)
{
   DWORD dw;
   LONG  lRet;
   LPSTR lpszCLSID;
   char  szKey[64];
   char  szMiscStatus[OLEUI_CCHKEYMAX];
   BOOL  bCloseRegDB = FALSE;

   if (hKey == NULL)
   {

      //Open up the root key.
      lRet=RegOpenKey(HKEY_CLASSES_ROOT, "CLSID", &hKey);

      if ((LONG)ERROR_SUCCESS!=lRet)
        return FALSE;

      bCloseRegDB = TRUE;
   }

   // Get a string containing the class name
   StringFromCLSID(rclsid, &lpszCLSID);

   // Construct key
   lstrcpy(szKey, lpszCLSID);
   lstrcat(szKey, "\\MiscStatus");


   dw=OLEUI_CCHKEYMAX;
   lRet = RegQueryValue(hKey, szKey, (LPSTR)szMiscStatus, &dw);

   if ((LONG)ERROR_SUCCESS!=lRet)
   {
        OleStdFreeString(lpszCLSID, NULL);

        if (bCloseRegDB)
             RegCloseKey(hKey);

        return FALSE;

   }

   *lpdwValue = Atol((LPSTR)szMiscStatus);

   OleStdFreeString(lpszCLSID, NULL);

   if (bCloseRegDB)
       RegCloseKey(hKey);

   return TRUE;
```

```
}


/*
 * CLIPFORMAT OleStdGetDefaultFileFormatOfClass(REFCLSID, HKEY)
 *
 * Purpose:
 *  Returns the default file format of the specified class.
 *  this is entered in REGDB as follows:
 *      CLSID\{...}\DataFormats\DefaultFile = <cfFmt>
 *
 * Parameters:
 *  rclsid          pointer to the clsid to retrieve user type of.
 *  hKey            hKey for reg db- if this is NULL, then we
 *                   open and close the reg db within this function.  If it
 *                   is non-NULL, then we assume it's a valid key to the
 *                   \ root and use it without closing it. (useful
 *                   if you're doing lots of reg db stuff).
 *
 * Return Value:
 *  cfFmt   -- DefaultFile format
 *  NULL    -- failed to get default file format
 *
 */
STDAPI_(CLIPFORMAT) OleStdGetDefaultFileFormatOfClass(
           REFCLSID        rclsid,
           HKEY            hKey
)
{
   CLIPFORMAT cfFmt = 0;
   DWORD dw;
   LONG  lRet;
   LPSTR lpszCLSID;
   BOOL  bCloseRegDB = FALSE;
   char  szKey[128];
   char  szDefaultFile[OLEUI_CCHKEYMAX];
   BOOL  bStatus = TRUE;


   if (hKey == NULL)
   {

      //Open up the root key.
      lRet=RegOpenKey(HKEY_CLASSES_ROOT, NULL, &hKey);

      if ((LONG)ERROR_SUCCESS!=lRet)
        return 0;

      bCloseRegDB = TRUE;
   }


   // Get a string containing the class name
   StringFromCLSID(rclsid, &lpszCLSID);
   // Construct key
   wsprintf(szKey, "CLSID\\%s\\DataFormats\\DefaultFile", lpszCLSID);
   OleStdFreeString(lpszCLSID, NULL);

   dw=OLEUI_CCHKEYMAX;
   lRet = RegQueryValue(hKey, szKey, (LPSTR)szDefaultFile, (LONG FAR *)&dw);

   if ((LONG)ERROR_SUCCESS!=lRet)
        bStatus = FALSE;
   else {
        /* if the format is a number, then it should refer to one of the
        **    standard Windows formats.
        */
        if (isdigit(szDefaultFile[0]))
              cfFmt = (CLIPFORMAT)Atol(szDefaultFile);
        else
              cfFmt = RegisterClipboardFormat(szDefaultFile);
   }

   if (bCloseRegDB)
       RegCloseKey(hKey);


   return cfFmt;
}
```

RESIMAGE.H (WRAPUI Sample)

```
/*
 * RESIMAGE.H
 *
 * Structures and definitions for the ResultImage control.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */


#ifndef _RESIMAGE_H_
#define _RESIMAGE_H_


/*
 * Indices into the bitmaps to extract the right image.  Each bitmap
 * contains five images arranged vertically, so the offset to the correct
 * image is (iImage*cy)
 */

#define RESULTIMAGE_NONE                0xFFFF
#define RESULTIMAGE_PASTE               0
#define RESULTIMAGE_EMBED               1
#define RESULTIMAGE_EMBEDICON           2
#define RESULTIMAGE_LINK                3
#define RESULTIMAGE_LINKICON            4
#define RESULTIMAGE_LINKTOLINK          5
#define RESULTIMAGE_LINKTOLINKICON      6

#define RESULTIMAGE_MIN                 0
#define RESULTIMAGE_MAX                 6


//Total number of images in each bitmap.
#define CIMAGESY                        (RESULTIMAGE_MAX+1)

//The color to use for transparancy (cyan)
#define RGBTRANSPARENT                  RGB(0, 255, 255)


//Function prototypes
BOOL            FResultImageInitialize(HINSTANCE, HINSTANCE, LPSTR);
void            ResultImageUninitialize(void);
LONG CALLBACK EXPORT ResultImageWndProc(HWND, UINT, WPARAM, LPARAM);
void            TransparentBlt(HDC, UINT, UINT, HBITMAP, UINT, UINT, UINT,
UINT, COLORREF);


//Window extra bytes contain the bitmap index we deal with currently.
#define CBRESULTIMAGEWNDEXTRA          sizeof(UINT)
#define RIWW_IMAGEINDEX                0
```

```
//Control messages
#define RIM_IMAGESET                  (WM_USER+0)
#define RIM_IMAGEGET                  (WM_USER+1)


//Special ROP code for TransparentBlt.
#define ROP_DSPDxax  0x00E20746


#endif //_RESIMAGE_H_
```

RESIMAGE.C (WRAPUI Sample)

```
/*
 * RESIMAGE.C
 *
 * Implementation of the Results Image control for OLE 2.0 UI dialogs.
 * We need a separate control for dialogs in order to control the repaints
 * properly and to provide a clean message interface for the dialog
 * implementations.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */


#define STRICT  1
#include "ole2ui.h"
#include "resimage.h"

OLEDBGDATA

//Flag indicating if we've registered the class
static BOOL     fRegistered=FALSE;

//Bitmap and image dimensions for result images.
static HBITMAP  hBmpResults=NULL;
static UINT     cxBmpResult=0;
static UINT     cyBmpResult=0;

/*
 * FResultImageInitialize
 *
 * Purpose:
 *  Attempts to load result bitmaps for the current display driver
 *  for use in OLE 2.0 UI dialogs.  Also registers the ResultImage
 *  control class.
 *
 * Parameters:
 *  hInst           HINSTANCE instance of the DLL.
 *
 *  hPrevInst       HINSTANCE of the previous instance.  Used to
 *                  determine whether to register window classes or not.
 *
 *  lpszClassName   LPSTR containing the class name to register the
 *                  ResultImage control class with.
 *
 * Return Value:
 *  BOOL            TRUE if all initialization succeeded, FALSE otherwise.
 */

BOOL FResultImageInitialize(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR
lpszClassName)
     {
     int         cx, iBmp;
     HDC         hDC;
```

```
BITMAP      bm;
WNDCLASS        wc;


/*
 * Determine the aspect ratio of the display we're currently
 * running on and load the appropriate bitmap into the global
 * hBmpResults (used from the ResultImage control only).
 *
 * By retrieving the logical Y extent of the display driver, you
 * only have limited possibilities:
 *      LOGPIXELSY      Display
 *      ----------------------------------------
 *         48             CGA    (unsupported)
 *         72             EGA
 *         96             VGA
 *        120             8514/a (i.e. HiRes VGA)
 */

hDC=GetDC(NULL);

if (NULL==hDC)
      return FALSE;

cx=GetDeviceCaps(hDC, LOGPIXELSY);
ReleaseDC(NULL, hDC);

/*
 * Instead of single comparisons, check ranges instead, so in case
 * we get something funky, we'll act reasonable.
 */
if (72 >=cx)             iBmp=(UINT)IDB_RESULTSEGA;
if (72 < cx && 120 > cx) iBmp=(UINT)IDB_RESULTSVGA;
if (120 <=cx)            iBmp=(UINT)IDB_RESULTSHIRESVGA;

hBmpResults=LoadBitmap(hInst, MAKEINTRESOURCE(iBmp));

if (NULL==hBmpResults)
      {
      //On error, fail loading the DLL
      OleDbgOut1("FResultImageInitialize:  Failed LoadBitmap.\r\n");
      return FALSE;
      }

OleDbgOut4("FResultImageInitialize:  Loaded hBmpResults\r\n");

//Now that we have the bitmap, calculate image dimensions
GetObject(hBmpResults, sizeof(BITMAP), &bm);
cxBmpResult=bm.bmWidth;
cyBmpResult=bm.bmHeight/CIMAGESY;


// Only register class if we're the first instance
if (hPrevInst)
      fRegistered = TRUE;
```

```
     else
           {
           // Static flag fRegistered guards against calling this function
more
           // than once in the same instance

           /* NOTE: the custom control classes for the ICONBOX class and the
           **    RESULTIMAGE class are registered as task specific. this
           **    is done both if the OLE2UI library is built as a static
           **    library or as a DLL. these custom controls are
           **    constructed internally as part of the OLE2UI code and do
           **    NOT need to be registered as global classes.
           */

           if (!fRegistered)
                 {
                 wc.lpfnWndProc   =ResultImageWndProc;
                 wc.cbClsExtra    =0;
                 wc.cbWndExtra    =CBRESULTIMAGEWNDEXTRA;
                 wc.hInstance     =hInst;
                 wc.hIcon         =NULL;
                 wc.hCursor       =LoadCursor(NULL, IDC_ARROW);
                 wc.hbrBackground =NULL;
                 wc.lpszMenuName  =NULL;
                 wc.lpszClassName =lpszClassName;
                 wc.style         = CS_VREDRAW | CS_HREDRAW;

                 fRegistered = RegisterClass(&wc);
                 }
           }

     return fRegistered;
}





/*
 * ResultImageUninitialize
 *
 * Purpose:
 *  Cleans up anything done in FResultImageInitialize, such as freeing
 *  the bitmaps.  Call from WEP.
 *
 * Parameters:
 *  None
 *
 * Return Value:
 *  None
 */

void ResultImageUninitialize(void)
     {
```

```
     if (NULL!=hBmpResults)
           {
           DeleteObject(hBmpResults);
           }

     return;
     }






/*
 * ResultImageWndProc
 *
 * Purpose:
 *  Window Procedure for the ResultImage custom control.  Only handles
 *  WM_CREATE, WM_PAINT, and private messages to manipulate the bitmap.
 *
 * Parameters:
 *  Standard
 *
 * Return Value:
 *  Standard
 */

LONG CALLBACK EXPORT ResultImageWndProc(HWND hWnd, UINT iMsg
     , WPARAM wParam, LPARAM lParam)
     {
     UINT            iBmp;
     PAINTSTRUCT     ps;
     HDC             hDC;

     //Handle standard Windows messages.
     switch (iMsg)
           {
           case WM_CREATE:
                 SetWindowWord(hWnd, RIWW_IMAGEINDEX, RESULTIMAGE_NONE);
                 return 0L;

           case WM_PAINT:
                 {
                 RECT            rc;
                 UINT            x, y;
                 HDC             hDCDlg;
                 HBRUSH          hBr;
                 LOGBRUSH        lb;
                 HWND            hDlg;

                 iBmp=GetWindowWord(hWnd, RIWW_IMAGEINDEX);

                 hDC=BeginPaint(hWnd, &ps);

                 /*
```

```
                  * Our job before using TransparentBlt is to figure out
                  * where to position the result image.  We place it
centered
                  * on this control, so get our rect's center and subtract
                  * half of the image dimensions.
                  */
                 GetClientRect(hWnd, &rc);
                 x=(rc.right+rc.left-cxBmpResult)/2;
                 y=(rc.bottom+rc.top-cyBmpResult)/2;

                 //Get the backgroup color the dialog is using.
                 hDlg=GetParent(hWnd);
                 hDCDlg=GetDC(hDlg);
#if defined( WIN32 )
                 hBr = (HBRUSH)SendMessage(hDlg,
                                                     WM_CTLCOLORDLG,
                                                     (WPARAM)hDCDlg,
                                                     (LPARAM)hDlg);
#else
                 hBr = (HBRUSH)SendMessage(hDlg,
                                                     WM_CTLCOLOR,
                                                     (WPARAM)hDCDlg,
                                                     MAKELPARAM(hDlg,
CTLCOLOR_DLG));
#endif
                 ReleaseDC(hDlg, hDCDlg);

                 GetObject(hBr, sizeof(LOGBRUSH), &lb);
                 SetBkColor(hDC, lb.lbColor);

                 // Paint the image
                 if (RESULTIMAGE_NONE!=iBmp)
                       TransparentBlt(hDC, x, y, hBmpResults, 0,
iBmp*cyBmpResult
                                            , cxBmpResult, cyBmpResult,
RGBTRANSPARENT);

                 // Or, erase the rectangle if no image is to be displayed
                 else
                       {
                       RECT    rc;

                       rc.top  = y;
                       rc.left = x;
                       rc.bottom = y + cyBmpResult;
                       rc.right  = x + cxBmpResult;
                       FillRect(hDC, &rc, hBr);
                       }

                 EndPaint(hWnd, &ps);
                 }
                 break;

           case RIM_IMAGESET:
                 //wParam contains the new index.
                 iBmp=GetWindowWord(hWnd, RIWW_IMAGEINDEX);

                 //Validate the index before changing it and repainting
                 if (RESULTIMAGE_NONE==wParam ||
                       ((RESULTIMAGE_MIN <= wParam) && (RESULTIMAGE_MAX >=
wParam)))
                       {
                       SetWindowWord(hWnd, RIWW_IMAGEINDEX, (WORD)wParam);
                       InvalidateRect(hWnd, NULL, FALSE);
                       UpdateWindow(hWnd);
                       }

                 //Return the previous index.
                 return iBmp;

           case RIM_IMAGEGET:
                 //Return the current index.
                 iBmp=GetWindowWord(hWnd, RIWW_IMAGEINDEX);
                 return (LONG)iBmp;

           default:
                 return DefWindowProc(hWnd, iMsg, wParam, lParam);
           }

     return 0L;
     }






/*
 * TransparentBlt
 *
 * Purpose:
 *  Given a DC, a bitmap, and a color to assume as transparent in that
 *  bitmap, BitBlts the bitmap to the DC letting the existing background
 *  show in place of the transparent color.
 *
 * Parameters:
 *  hDC             HDC on which to draw.
 *  x, y            UINT location at which to draw the bitmap
 *  hBmp            HBITMIP to draw from
 *  xOrg, yOrg      UINT coordinates from which to draw the bitamp
 *  cx, cy          UINT dimensions of the bitmap to Blt.
 *  cr              COLORREF to consider as transparent.
 *
 * Return Value:
 *  None
 */

void TransparentBlt(HDC hDC, UINT x, UINT y, HBITMAP hBmp, UINT xOrg, UINT
yOrg
```

```
     , UINT cx, UINT cy, COLORREF cr)
     {
     HDC         hDCSrc, hDCMid, hMemDC;
     HBITMAP     hBmpMono, hBmpT;
     HBRUSH      hBr, hBrT;
     COLORREF    crBack, crText;

     if (NULL==hBmp)
           return;

     //Get three intermediate DC's
     hDCSrc=CreateCompatibleDC(hDC);
     hDCMid=CreateCompatibleDC(hDC);
     hMemDC=CreateCompatibleDC(hDC);

     SelectObject(hDCSrc, hBmp);

     //Create a monochrome bitmap for masking
     hBmpMono=CreateCompatibleBitmap(hDCMid, cx, cy);
     SelectObject(hDCMid, hBmpMono);

     //Create a middle bitmap
     hBmpT=CreateCompatibleBitmap(hDC, cx, cy);
     SelectObject(hMemDC, hBmpT);


     //Create a monochrome mask where we have 0's in the image, 1's
elsewhere.
     crBack=SetBkColor(hDCSrc, cr);
     BitBlt(hDCMid, 0, 0, cx, cy, hDCSrc, xOrg, yOrg, SRCCOPY);
     SetBkColor(hDCSrc, crBack);

     //Put the unmodified image in the temporary bitmap
     BitBlt(hMemDC, 0, 0, cx, cy, hDCSrc, xOrg, yOrg, SRCCOPY);

     //Create an select a brush of the background color
     hBr=CreateSolidBrush(GetBkColor(hDC));
     hBrT=SelectObject(hMemDC, hBr);

     //Force conversion of the monochrome to stay black and white.
     crText=SetTextColor(hMemDC, 0L);
     crBack=SetBkColor(hMemDC, RGB(255, 255, 255));

     /*
      * Where the monochrome mask is 1, Blt the brush; where the mono mask
      * is 0, leave the destination untouches.  This results in painting
      * around the image with the background brush.  We do this first
      * in the temporary bitmap, then put the whole thing to the screen.
      */
     BitBlt(hMemDC, 0, 0, cx, cy, hDCMid, 0, 0, ROP_DSPDxax);
     BitBlt(hDC,    x, y, cx, cy, hMemDC, 0, 0, SRCCOPY);


     SetTextColor(hMemDC, crText);
     SetBkColor(hMemDC, crBack);
```

```
SelectObject(hMemDC, hBrT);
DeleteObject(hBr);

DeleteDC(hMemDC);
DeleteDC(hDCSrc);
DeleteDC(hDCMid);
DeleteObject(hBmpT);
DeleteObject(hBmpMono);

return;
}
```

STDPAL.H (WRAPUI Sample)

```
/*-------------------------------------------------------------------------
|   stdpal.h
|
|   Standard App Palette useful for OLE applications.  v 1.01
|
|   #include this file in the same file as HpalCreateAppPalette
|
|   NOTE:  Palette MUST be created with HpalCreateAppPalette
|
|   Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
|
-------------------------------------------------------------------------*/


#ifndef CSCONST
#if (defined(FLAT) || defined(__TURBOC__))
#define CSCONST(type) type const
#else
#define CSCONST(type) type _based(_segname("_CODE")) const
#endif
#endif

CSCONST(unsigned char) palSVGA[256][3] =
     {
     // R     G     B
     {0x00, 0x00, 0x00}, // 0 Sys Black  gray 0
     {0x80, 0x00, 0x00}, // 1 Sys Dk Red
     {0x00, 0x80, 0x00}, // 2 Sys Dk Green
     {0x80, 0x80, 0x00}, // 3 Sys Dk Yellow
     {0x00, 0x00, 0x80}, // 4 Sys Dk Blue
     {0x80, 0x00, 0x80}, // 5 Sys Dk Violet
     {0x00, 0x80, 0x80}, // 6 Sys Dk Cyan
     {0xc0, 0xc0, 0xc0}, // 7 Sys Lt Grey    gray 192
     {0xc0, 0xdc, 0xc0}, // 8 Sys 8
     {0xa6, 0xca, 0xf0}, // 9 Sys 9 (the first 10 are fixed by Windows)

     {0x80, 0x00, 0x00}, // 10 Sys Dk Red repeat
     {0x00, 0x80, 0x00}, // 11 Sys Dk Green repeat
     {0x80, 0x80, 0x00}, // 12 Sys Dk Yellow repeat
     {0x00, 0x00, 0x80}, // 13 Sys Dk Blue repeat
     {0x80, 0x00, 0x80}, // 14 Sys Dk Violet repeat
     {0x00, 0x80, 0x80}, // 15 Sys Dk Cyan repeat
     {0x80, 0x80, 0x80}, // 16 Sys Dk Grey repeat    gray 128
     {0x80, 0x80, 0xff}, // 17 Excel Chart Fill 1
     {0x80, 0x20, 0x60}, // 18 Excel Chart Fill 2
     {0xff, 0xff, 0xc0}, // 19 Excel Chart Fill 3
     {0xa0, 0xe0, 0xe0}, // 20 Excel Chart Fill 4
     {0x60, 0x00, 0x80}, // 21 Excel Chart Fill 4
     {0xff, 0x80, 0x80}, // 22 Excel Chart Fill 6
     {0x00, 0x80, 0xc0}, // 23 Excel Chart Fill 7
     {0xc0, 0xc0, 0xff}, // 24 Excel Chart Fill 8
     {0x00, 0xcf, 0xff}, // 25 Excel clrt entry
```

```
{0x69, 0xff, 0xff}, // 26 Excel clrt entry
{0xe0, 0xff, 0xe0}, // 27 Excel clrt entry
{0xdd, 0x9c, 0xb3}, // 28 Excel clrt entry
{0xb3, 0x8f, 0xee}, // 29 Excel clrt entry
{0x2a, 0x6f, 0xf9}, // 30 Excel clrt entry
{0x3f, 0xb8, 0xcd}, // 31 Excel clrt entry
{0x48, 0x84, 0x36}, // 32 Excel clrt entry
{0x95, 0x8c, 0x41}, // 33 Excel clrt entry
{0x8e, 0x5e, 0x42}, // 34 Excel clrt entry
{0xa0, 0x62, 0x7a}, // 35 Excel clrt entry
{0x62, 0x4f, 0xac}, // 36 Excel clrt entry
{0x1d, 0x2f, 0xbe}, // 37 Excel clrt entry
{0x28, 0x66, 0x76}, // 38 Excel clrt entry
{0x00, 0x45, 0x00}, // 39 Excel clrt entry
{0x45, 0x3e, 0x01}, // 40 Excel clrt entry
{0x6a, 0x28, 0x13}, // 41 Excel clrt entry
{0x85, 0x39, 0x6a}, // 42 Excel clrt entry
{0x4a, 0x32, 0x85}, // 43 Excel clrt entry
{0x04, 0x04, 0x04}, // 44   gray 4
{0x08, 0x08, 0x08}, // 45   gray 8
{0x0c, 0x0c, 0x0c}, // 46   gray 12
{0x11, 0x11, 0x11}, // 47   gray 17
{0x16, 0x16, 0x16}, // 48   gray 22
{0x1c, 0x1c, 0x1c}, // 49   gray 28
{0x22, 0x22, 0x22}, // 50   gray 34
{0x29, 0x29, 0x29}, // 51   gray 41
{0x30, 0x30, 0x30}, // 52   gray 48
{0x5f, 0x5f, 0x5f}, // 53 swapped so inversions look good   gray 95
{0x55, 0x55, 0x55}, // 54 swapped so inversions look good   gray 85
{0x4d, 0x4d, 0x4d}, // 55 swapped so inversions look good   gray 77
{0x42, 0x42, 0x42}, // 56 swapped so inversions look good   gray 66
{0x39, 0x39, 0x39}, // 57 swapped so inversions look good   gray 57
{0x00, 0x07, 0x00}, // 58
{0x0d, 0x00, 0x00}, // 59
{0xb7, 0x99, 0x81}, // 60
{0x84, 0x99, 0xb4}, // 61
{0xbd, 0xbd, 0x90}, // 62
{0x7f, 0x7f, 0x60}, // 63
{0x60, 0x60, 0x7f}, // 64
{0x00, 0x0e, 0x00}, // 65
{0x1b, 0x00, 0x00}, // 66
{0x28, 0x00, 0x00}, // 67
{0x08, 0x09, 0x2b}, // 68
{0x00, 0x1d, 0x00}, // 69
{0x39, 0x00, 0x00}, // 70
{0x00, 0x00, 0x9b}, // 71
{0x00, 0x25, 0x00}, // 72
{0x49, 0x00, 0x00}, // 73
{0x11, 0x11, 0x3b}, // 74
{0x00, 0x2f, 0x00}, // 75
{0x5d, 0x00, 0x00}, // 76
{0x17, 0x17, 0x45}, // 77
{0x00, 0x3a, 0x00}, // 78
{0x49, 0x11, 0x11}, // 79
{0x1c, 0x1c, 0x53}, // 80
```

```
{0x00, 0x16, 0xff}, // 81
{0x2b, 0x00, 0xff}, // 82
{0x21, 0x21, 0x6c}, // 83
{0x59, 0x14, 0x14}, // 84
{0x00, 0x51, 0x00}, // 85
{0x47, 0x1a, 0x6a}, // 86
{0x19, 0x32, 0x67}, // 87
{0x00, 0x61, 0x00}, // 88
{0x00, 0x31, 0xff}, // 89
{0x61, 0x00, 0xff}, // 90
{0x53, 0x20, 0x7b}, // 91
{0x16, 0x43, 0x67}, // 92
{0x2e, 0x2e, 0xe2}, // 93
{0x26, 0x59, 0x16}, // 94
{0x51, 0x46, 0x04}, // 95
{0x68, 0x2e, 0x49}, // 96
{0x07, 0x52, 0x8f}, // 97
{0x6a, 0x18, 0xb8}, // 98
{0x90, 0x23, 0x15}, // 99
{0x00, 0x53, 0xff}, // 100
{0xa3, 0x00, 0xff}, // 101
{0x6a, 0x4a, 0x12}, // 102
{0x75, 0x33, 0x6c}, // 103
{0x4a, 0x41, 0x9a}, // 104
{0x37, 0x65, 0x0b}, // 105
{0xa4, 0x2c, 0x15}, // 106
{0x83, 0x1f, 0xb1}, // 107
{0x4e, 0x2c, 0xff}, // 108
{0x20, 0x51, 0xb6}, // 109
{0x08, 0x64, 0x92}, // 110
{0x6f, 0x56, 0x0b}, // 111
{0x59, 0x43, 0xad}, // 112
{0x36, 0x72, 0x12}, // 113
{0xb0, 0x33, 0x17}, // 114
{0x00, 0xa1, 0x00}, // 115
{0x77, 0x5f, 0x1f}, // 116
{0x89, 0x47, 0x71}, // 117
{0xb0, 0x43, 0x1c}, // 118
{0xb7, 0x2d, 0x7d}, // 119
{0x00, 0x86, 0x95}, // 120
{0x7a, 0x6e, 0x23}, // 121
{0x26, 0x9f, 0x00}, // 122
{0x73, 0xa9, 0x01}, // 123
{0x00, 0x00, 0x00}, // 124 free 0   gray 0
{0x00, 0x00, 0x00}, // 125 free 2   gray 0
{0x00, 0x00, 0x00}, // 126 free 4   gray 0
{0x00, 0x00, 0x00}, // 127 free 6   gray 0
{0x00, 0x00, 0x00}, // 128 free 7   gray 0
{0x00, 0x00, 0x00}, // 129 free 5   gray 0
{0x00, 0x00, 0x00}, // 130 free 3   gray 0
{0x00, 0x00, 0x00}, // 131 free 1   gray 0
{0x00, 0xca, 0x00}, // 132
{0xac, 0x5b, 0x01}, // 133
{0x20, 0x1d, 0xc2}, // 134
```

```
{0x94, 0x52, 0x70}, // 135
{0x24, 0xaa, 0x4c}, // 136
{0x0a, 0x94, 0x89}, // 137
{0x36, 0x6e, 0x7b}, // 138
{0x44, 0x75, 0x90}, // 139
{0xff, 0x00, 0xa8}, // 140
{0x00, 0x71, 0xff}, // 141
{0xdf, 0x00, 0xff}, // 142
{0x56, 0x91, 0x4a}, // 143
{0x34, 0x48, 0xf8}, // 144
{0xcc, 0x32, 0x82}, // 145
{0xe4, 0x41, 0x70}, // 146
{0x68, 0xca, 0x01}, // 147
{0x36, 0xbc, 0x42}, // 148
{0x00, 0x9a, 0xff}, // 149
{0x96, 0x22, 0xb7}, // 150
{0x85, 0x7d, 0x33}, // 151
{0x25, 0xb7, 0x8c}, // 152
{0x36, 0x5a, 0xed}, // 153
{0x5c, 0xff, 0x00}, // 154
{0xff, 0x48, 0x00}, // 155
{0x22, 0x9b, 0xa2}, // 156
{0x42, 0xcf, 0x4d}, // 157
{0xc2, 0x58, 0x52}, // 158
{0x20, 0xd3, 0x95}, // 159
{0xa5, 0x24, 0xe0}, // 160
{0x73, 0x56, 0xb5}, // 161
{0xa9, 0xa9, 0x00}, // 162
{0xd0, 0x6f, 0x3c}, // 163
{0x67, 0x9f, 0x58}, // 164
{0x89, 0xcf, 0x0b}, // 165
{0xff, 0xac, 0x00}, // 166
{0xa7, 0x2e, 0xfe}, // 167
{0xe2, 0x59, 0x7f}, // 168
{0x4c, 0xdc, 0x67}, // 169
{0xff, 0x18, 0xff}, // 170
{0x3a, 0x7d, 0xff}, // 171
{0xb1, 0xd0, 0x18}, // 172
{0xc7, 0xff, 0x00}, // 173
{0xff, 0xe2, 0x00}, // 174
{0xdf, 0x9a, 0x3d}, // 175
{0x56, 0x81, 0x9f}, // 176
{0xc6, 0x43, 0xba}, // 177
{0xaf, 0x71, 0x8b}, // 178
{0x38, 0xa2, 0xc9}, // 179
{0xd1, 0x53, 0xce}, // 180
{0xff, 0x9a, 0x65}, // 181
{0x46, 0xca, 0xdb}, // 182
{0xff, 0x4d, 0xff}, // 183
{0xc8, 0xe9, 0x6a}, // 184
{0x4c, 0xde, 0xe0}, // 185
{0xff, 0x98, 0xff}, // 186
{0xdf, 0xc0, 0x82}, // 187
{0xe9, 0xec, 0xa5}, // 188
{0xf5, 0xf6, 0xcd}, // 189
```

```
{0xff, 0xd0, 0xff}, // 190
{0xb1, 0xac, 0x5a}, // 191
{0x63, 0x91, 0xae}, // 192
{0x22, 0x4c, 0x65}, // 193
{0x8d, 0x4e, 0x3f}, // 194
{0x50, 0x70, 0x70}, // 195
{0xd0, 0xff, 0xff}, // 196
{0xff, 0xe7, 0xff}, // 197
{0x69, 0x69, 0x69}, // 198  gray 105
{0x77, 0x77, 0x77}, // 199  gray 119
{0x86, 0x86, 0x86}, // 200  gray 134
{0x96, 0x96, 0x96}, // 201  gray 150
{0x9d, 0x9d, 0x9d}, // 202  gray 157
{0xa4, 0xa4, 0xa4}, // 203  gray 164
{0xb2, 0xb2, 0xb2}, // 204  gray 178
{0xcb, 0xcb, 0xcb}, // 205  gray 203
{0xd7, 0xd7, 0xd7}, // 206  gray 215
{0xdd, 0xdd, 0xdd}, // 207  gray 221
{0xe3, 0xe3, 0xe3}, // 208  gray 227
{0xea, 0xea, 0xea}, // 209  gray 234
{0xf1, 0xf1, 0xf1}, // 210  gray 241
{0xf8, 0xf8, 0xf8}, // 211  gray 248
{0xb2, 0xc1, 0x66}, // 212
{0x80, 0xbf, 0x78}, // 213
{0xc6, 0xf0, 0xf0}, // 214
{0xb2, 0xa4, 0xff}, // 215
{0xff, 0xb3, 0xff}, // 216
{0xd1, 0x8e, 0xa3}, // 217
{0xc3, 0xdc, 0x37}, // 218
{0xa0, 0x9e, 0x54}, // 219
{0x76, 0xae, 0x70}, // 220
{0x78, 0x9e, 0xc1}, // 221
{0x83, 0x64, 0xbf}, // 222
{0xa4, 0x83, 0xd3}, // 223
{0xd1, 0x3f, 0x32}, // 224
{0xff, 0x7d, 0x00}, // 225
{0x44, 0x78, 0x23}, // 226
{0x24, 0x5f, 0x60}, // 227
{0x0e, 0x0e, 0x2c}, // 228
{0xbe, 0x00, 0x00}, // 229
{0xff, 0x1f, 0x00}, // 230
{0x31, 0x39, 0x00}, // 231
{0xd9, 0x85, 0x3e}, // 232
{0x02, 0x77, 0x85}, // 233
{0xb0, 0xd8, 0x81}, // 234
{0x56, 0x21, 0x1d}, // 235
{0x00, 0x00, 0x30}, // 236
{0x88, 0xc8, 0xb3}, // 237
{0xa0, 0x79, 0x00}, // 238
{0xc0, 0xc0, 0xc0}, // 239 Sys Dk Grey repeat inversion gray 192
{0xea, 0x70, 0x81}, // 240
{0x51, 0xf1, 0x69}, // 241
{0xff, 0xff, 0x80}, // 242
{0x91, 0x74, 0xcd}, // 243
{0xff, 0x7c, 0xff}, // 244
```

```
{0xa2, 0xff, 0xff}, // 245

{0xff, 0xfb, 0xf0}, // 246 Sys Reserved
{0xa0, 0xa0, 0xa4}, // 247 Sys Reserved
{0x80, 0x80, 0x80}, // 248 Sys Lt Gray  gray 128
{0xff, 0x00, 0x00}, // 249 Sys Red
{0x00, 0xff, 0x00}, // 250 Sys Green
{0xff, 0xff, 0x00}, // 251 Sys Yellow
{0x00, 0x00, 0xff}, // 252 Sys Blue
{0xff, 0x00, 0xff}, // 253 Sys Violet
{0x00, 0xff, 0xff}, // 254 Sys Cyan
{0xff, 0xff, 0xff} // 255 Sys White gray 255
};
```

STDPAL.C   (WRAPUI Sample)

```
/*-------------------------------------------------------------------------
|   stdpal.c
|
|   Standard App Palette useful for OLE applications.
|   Use OleStdCreateStandardPalette API to create the palette.
|
|   NOTE:  The OLE2UI sample source code that will ship with Windows
|          "Chicago" will have the "Chicago" standard palette, which
|          we expect to be different than this palette. We recommend
|          using the "Chicago" palette when it becomes available.
|
|   Copyright (c) 1992 - 1993 Microsoft Corporation. All rights reserved.
|
-----------------------------------------------------------------------*/

#ifndef PC_RESERVED
#include <windows.h>
#endif

#include "ole2ui.h"
#include "stdpal.h"

#define cpeAppPal 256  // number of colors in our apps palette
typedef struct
     {
     WORD wVersion;
     WORD cpe;
     PALETTEENTRY rgpe[cpeAppPal];
     } LOGPAL;


/*-------------------------------------------------------------------------
|   OleStdCreateStandardPalette
|
|       Creates the standard Apps palette.  Create one of these for your
|   app, and select/realize it into each DC.
|
|   Arguments:
|       void:
|
|   Returns:
|
|   Keywords:
-----------------------------------------------------------------------*/
STDAPI_(HPALETTE) OleStdCreateStandardPalette(void)
     {
     HDC hdc;
     HPALETTE hpal;

     hpal = (HPALETTE) NULL;
     hdc = GetDC(NULL);
     if (hdc != NULL && GetDeviceCaps(hdc, RASTERCAPS) & RC_PALETTE)
```

```
           {
           int cpeSysPal;
           int cpeReserved;

           cpeSysPal = GetDeviceCaps(hdc, SIZEPALETTE);
           cpeReserved = GetDeviceCaps(hdc, NUMRESERVED);
           if (cpeSysPal > cpeReserved)
                 {
                 int cpeReserved2;
                 unsigned char FAR* lpb;
                 PALETTEENTRY FAR* ppe;
                 PALETTEENTRY FAR* ppeMac;
                 LOGPAL logpal;

                 cpeReserved2 = cpeReserved/2;

                 // Get the system palette entries at the beginning and end.
                 GetSystemPaletteEntries(hdc, 0, cpeReserved2, logpal.rgpe);
                 GetSystemPaletteEntries(hdc, cpeSysPal - cpeReserved2,
cpeReserved2,
                       &logpal.rgpe[cpeAppPal-cpeReserved2]);

                 logpal.cpe = cpeAppPal;
                 logpal.wVersion = 0x300;

                 lpb = (BYTE FAR *) &palSVGA[10];
                 ppe = (PALETTEENTRY FAR*)&logpal.rgpe[cpeReserved2];
                 ppeMac = (PALETTEENTRY FAR*)&logpal.rgpe[cpeAppPal-
cpeReserved2];
                 while (ppe < ppeMac)
                       {
                       ppe->peFlags = PC_NOCOLLAPSE;
                       ppe->peRed   = *lpb++;
                       ppe->peGreen = *lpb++;
                       ppe->peBlue  = *lpb++;
                       ppe++;
                       }
                 hpal = CreatePalette((LOGPALETTE FAR *)&logpal);
                 }
           }
     ReleaseDC(NULL, hdc);
     return hpal;
     }
```

SUMINFO.H (WRAPUI Sample)

```
/*************************************************************************
**
**    OLE 2.0 Property Set Utilities
**
**    suminfo.h
**
**    This file contains file contains data structure defintions,
**    function prototypes, constants, etc. for OLE 2.0 Property Set
**    utilities used to manage the Summary Info property set.
**
**    (c) Copyright Microsoft Corp. 1990 - 1992 All Rights Reserved
**
*************************************************************************/

#ifndef SUMINFO_H
#define SUMINFO_H

#include <ole2.h>
#include <objbase.h>
#include <oleauto.h>

/* A SUMINFO variable is an instance of an abstract data type.  Thus,
**    there can be an arbitrary number of SummaryInfo streams open
**    simultaneously (subject to available memory).  Each variable must
**    be initialized prior to use by calling Init and freed after its
**    last use by calling Free.  The param argument to Init is reserved
**    for future expansion and should be zero initially. Once a SUMINFO
**    variable is allocated (by Init), the user can call the Set
**    procedures to initialize fields.  A copy of the arguments is made
**    in every case except SetThumbnail where control of the storage
**    occupied by the METAFILEPICT is merely transferred.  When the
**    Free routine is called, all storage will be deallocated including
**    that of the thumbnail.  The arguments to SetThumbNail and the
**    return values from GetThumbNail correspond to the OLE2.0 spec.
**    Note that on input, the thumbnail is read on demand but all the
**    other properties are pre-loaded.  The thumbnail is manipulated as
**    a windows handle to a METAFILEPICT structure, which in turn
**    contains a handle to the METAFILE.  The transferClip argument on
**    GetThumbNail, when set to true, transfers responsibility for
**    storage management of the thumbnail to the caller; that is, after
**    Free has been called, the handle is still valid. Clear can be
**    used to free storage for all the properties but then you must
**    call Read to load them again.  All the code is based on FAR
**    pointers.
**    CoInitialize MUST be called PRIOR to calling OleStdInitSummaryInfo.
**    Memory is allocated using the currently active IMalloc*
**    allocator (as is returned by call CoGetMalloc(MEMCTX_TASK) ).
**
** Common scenarios:
**    Read SummaryInfo
**    ----------------
**      OleStdInitSummaryInfo()
**      OleStdReadSummaryInfo()
**      . . . . .
**      call different Get routines
**      . . . . .
**      OleStdFreeSummaryInfo()
**
**    Create SummaryInfo
**    ------------------
**      OleStdInitSummaryInfo()
**      call different Set routines
**      OleStdWriteSummaryInfo()
**      OleStdFreeSummaryInfo()
**
**    Update SummaryInfo
**    ------------------
**      OleStdInitSummaryInfo()
**      OleStdReadSummaryInfo()
**      OleStdGetThumbNailProperty(necessary only if no SetThumb)
**      call different Set routines
**      OleStdWriteSummaryInfo()
**      OleStdFreeSummaryInfo()
*/

#define WORDMAX 256     //current string max for APPS; 255 + null terminator


typedef     union {
          short    iVal;             /* VT_I2                */
          long     lVal;             /* VT_I4                */
          float    fltVal;           /* VT_R4                */
          double       dblVal;       /* VT_R8                */
          DWORD bool;                /* VT_BOOL              */
          SCODE    scodeVal;         /* VT_ERROR             */
          DWORD    systimeVal;       /* VT_SYSTIME           */
          unsigned char bstrVal[WORDMAX]; /* VT_BSTR              */
        } VTUNION;

typedef struct _FMTID
        {
        DWORD dword;
        WORD words[2];
        BYTE bytes[8];
        } FMTID;

typedef struct _PROPSETLIST
        {
        FMTID formatID;
        DWORD byteOffset;
        } PROPSETLIST;

typedef struct _PROPIDLIST
        {
        DWORD propertyID;
        DWORD byteOffset;
        } PROPIDLIST;
```

```
typedef struct _PROPVALUE
        {
        DWORD vtType;
        VTUNION vtValue;
        } PROPVALUE;

typedef struct _SECTION
        {
        DWORD cBytes;
        DWORD cProperties;
        PROPIDLIST rgPropId[1/*cProperties*/];  //variable-length array
        PROPVALUE rgPropValue[1];      //CANNOT BE ACCESSED BY NAME; ONLY BY
POINTER
        } SECTION;

typedef struct _SUMMARYINFO
        {
        WORD byteOrder;
        WORD formatVersion;
        WORD getOSVersion;
        WORD osVersion;
        CLSID classId;  //from compobj.h
        DWORD cSections;
        PROPSETLIST rgPropSet[1/*cSections*/]; //variable-length array
        SECTION rgSections[1/*cSections*/];    //CANNOT BE ACCESSED BY NAME;
ONLY BY POINTER
        } SUMMARYINFO;

#define osWinOnDos 0
#define osMac 1
#define osWinNT 2

#define PID_DICTIONARY 0X00000000
#define PID_CODEPAGE 0X00000001
#define PID_TITLE 0X00000002
#define PID_SUBJECT 0X00000003
#define PID_AUTHOR 0X00000004
#define PID_KEYWORDS 0X00000005
#define PID_COMMENTS 0X00000006
#define PID_TEMPLATE 0X00000007
#define PID_LASTAUTHOR 0X00000008
#define PID_REVNUMBER 0X00000009
#define PID_EDITTIME 0X0000000A
#define PID_LASTPRINTED 0X0000000B
#define PID_CREATE_DTM_RO 0X0000000C
#define PID_LASTSAVE_DTM 0X0000000D
#define PID_PAGECOUNT 0X0000000E
#define PID_WORDCOUNT 0X0000000F
#define PID_CHARCOUNT 0X00000010
#define PID_THUMBNAIL 0X00000011
#define PID_APPNAME 0X00000012
#define PID_SECURITY 0X00000013
#define cPID_STANDARD (PID_SECURITY+1-2)
```

```
#ifndef WIN32
#define MAXWORD 256         //maximum string size for APPS at present
#endif

typedef struct _STDZ
        {
        DWORD vtType;
        union {
        DWORD vtByteCount;
        unsigned char fill[4];  //use last byte as byte count for stz
requests
        };
        unsigned char rgchars[MAXWORD];
        } STDZ;
#define VTCB fill[3]    //used to set/get the count byte when in memory

typedef struct _THUMB
        {
        DWORD vtType;
        DWORD cBytes;       //clip size in memory
        DWORD selector;     //on disk -1,win clip no.  -2,mac clip no.
-3,ole FMTID  0,bytes  nameLength, format name
        DWORD clipFormat;
        unsigned char FAR *lpstzName;
        unsigned char FAR *lpByte;
        } THUMB;

#define VT_CF_BYTES 0
#define VT_CF_WIN (-1)
#define VT_CF_MAC (-2)
#define VT_CF_FMTID (-3)
#define VT_CF_NAME (-4)
#define VT_CF_EMPTY (-5)
#define VT_CF_OOM (-6)      // Out of memory
typedef THUMB FAR *LPTHUMB;

typedef STDZ FAR *LPSTDZ;

typedef struct _TIME
        {
        DWORD vtType;
        FILETIME time;
        } TIME;

typedef struct _INTS
        {
        DWORD vtType;
        DWORD value;
        } INTS;

#define MAXTIME (PID_LASTSAVE_DTM-PID_EDITTIME+1)
#define MAXINTS (PID_CHARCOUNT-PID_PAGECOUNT+1+1)
#define MAXSTDZ (PID_REVNUMBER-PID_TITLE+1+1)
```

```
typedef struct _STANDARDSECINMEM
        {
        DWORD cBytes;
        DWORD cProperties;
        PROPIDLIST rgPropId[cPID_STANDARD/*cProperties*/];  //variable-
length array
        TIME rgTime[MAXTIME];
        INTS rgInts[MAXINTS];
        LPSTDZ rglpsz[MAXSTDZ];
        THUMB thumb;
        } STANDARDSECINMEM;


#define OFFSET_NIL 0X00000000

#define AllSecurityFlagsEqNone 0
#define fSecurityPassworded 1
#define fSecurityRORecommended 2
#define fSecurityRO 4
#define fSecurityLockedForAnnotations 8

#define PropStreamNamePrefixByte '\005'
#define PropStreamName "\005SummaryInformation"
#define cbNewSummaryInfo(nSection) (sizeof(SUMMARYINFO)-sizeof(SECTION)
+sizeof(PROPSETLIST)*((nSection)-1))
#define cbNewSection(nPropIds) (sizeof(SECTION)-sizeof(PROPVALUE)
+sizeof(PROPIDLIST)*((nPropIds)-1))

#define FIntelOrder(prop) ((prop)->byteOrder==0xfffe)
#define SetOs(prop, os) {(prop)->osVersion=os; (prop)-
>getOSVersion=LOWORD(GetVersion());}
#define SetSumInfFMTID(fmtId) {(fmtId)->dword=0XF29F85E0; *(long FAR
*)&(fmtId)->words=0X10684FF9;\
                                *(long FAR *)&(fmtId)->bytes[0]=0X000891AB;
*(long FAR *)&(fmtId)->bytes[4]=0XD9B3272B;}
#define FEqSumInfFMTID(fmtId) ((fmtId)->dword==0XF29F85E0&&*((long FAR
*)&(fmtId)->words)==0X10684FF9&&\
                                *((long FAR *)&(fmtId)-
>bytes[0])==0X000891AB&&*((long FAR *)&(fmtId)->bytes[4])==0XD9B3272B)
#define FSzEqPropStreamName(sz) _fstricmp(sz, PropStreamName)
#define ClearSumInf(lpsuminf, cb) {_fmemset(lpsuminf,0,cb); (lpsuminf)-
>byteOrder=0xfffe;\
                                SetOs(lpsuminf, osWinOnDos);}

typedef void FAR *LPSUMINFO;
typedef LPSTR LPSTZR;
typedef void FAR *THUMBNAIL;  //for VT_CF_WIN this is an unlocked global
handle
#define API __far __pascal


/*************************************************************************
** Public Summary Info Property Set Management API
*************************************************************************/
```

```
extern "C" {
STDAPI_(LPSUMINFO) OleStdInitSummaryInfo(int reserved);
STDAPI_(void) OleStdFreeSummaryInfo(LPSUMINFO FAR *lplp);
STDAPI_(void) OleStdClearSummaryInfo(LPSUMINFO lp);
STDAPI_(int) OleStdReadSummaryInfo(LPSTREAM lpStream, LPSUMINFO lp);
STDAPI_(int) OleStdWriteSummaryInfo(LPSTREAM lpStream, LPSUMINFO lp);
STDAPI_(DWORD) OleStdGetSecurityProperty(LPSUMINFO lp);
STDAPI_(int) OleStdSetSecurityProperty(LPSUMINFO lp, DWORD security);
STDAPI_(LPSTR) OleStdGetStringProperty(LPSUMINFO lp, DWORD pid);
STDAPI_(int) OleStdSetStringProperty(LPSUMINFO lp, DWORD pid, LPSTR lpsz);
STDAPI_(LPSTZR) OleStdGetStringZProperty(LPSUMINFO lp, DWORD pid);
STDAPI_(void) OleStdGetDocProperty(
                LPSUMINFO       lp,
                DWORD FAR*      nPage,
                DWORD FAR*      nWords,
                DWORD FAR*      nChars
);
STDAPI_(int) OleStdSetDocProperty(
                LPSUMINFO       lp,
                DWORD           nPage,
                DWORD           nWords,
                DWORD           nChars
);
STDAPI_(int) OleStdGetThumbNailProperty(
                LPSTREAM        lps,
                LPSUMINFO       lp,
                DWORD FAR*      clipFormatNo,
                LPSTR FAR*      lpszName,
                THUMBNAIL FAR*  clip,
                DWORD FAR*      byteCount,
                BOOL            transferClip
);
STDAPI_(int) OleStdSetThumbNailProperty(
                LPSTREAM        lps,
                LPSUMINFO       lp,
                int             vtcfNo,
                DWORD           clipFormatNo,
                LPSTR           lpszName,
                THUMBNAIL       clip,
                DWORD           byteCount
);
STDAPI_(void) OleStdGetDateProperty(
                LPSUMINFO       lp,
                DWORD           pid,
                int FAR*        yr,
                int FAR*        mo,
                int FAR*        dy,
                DWORD FAR*      sc
);
STDAPI_(int) OleStdSetDateProperty(
                LPSUMINFO       lp,
                DWORD           pid,
                int             yr,
                int             mo,
                int             dy,
                int             hr,
                int             mn,
                int             sc
);

} //END C

#endif  // SUMINFO_H
```

SUMINFO.CPP   (WRAPUI Sample)

```
/*************************************************************************
**
**    OLE 2.0 Property Set Utilities
**
**    suminfo.cpp
**
**    This file contains functions that are useful for the manipulation
**    of OLE 2.0 Property Sets particularly to manage the Summary Info
**    property set.
**
**    (c) Copyright Microsoft Corp. 1990 - 1992 All Rights Reserved
**
*************************************************************************/

// Note: this file is designed to be stand-alone; it includes a
// carefully chosen, minimal set of headers.
//
// For conditional compilation we use the ole2 conventions,
//    _MAC      = mac
//    WIN32     = Win32 (NT really)
//    <nothing> = defaults to Win16

// REVIEW: the following needs to modified to handle _MAC
#define STRICT
#include <windows.h>
#include <string.h>
#include <ole2.h>
#include "ole2ui.h"

/* A LPSUMINFO variable is a pointer to an instance of an abstract data
**    type.  There can be an arbitrary number of SummaryInfo streams open
**    simultaneously (subject to available memory); each must have its
**    own LPSUMINFO instance. Each LPSUMINFO instance must
**    be initialized prior to use by calling Init and freed after its
**    last use by calling Free.  The param argument to Init is reserved
**    for future expansion and should be zero initially. Once a LPSUMINFO
**    instance is allocated (by Init), the user can call the Set
**    procedures to initialize fields.  A copy of the arguments is made
**    in every case except SetThumbnail where control of the storage
**    occupied by the METAFILEPICT is merely transferred.  When the
**    Free routine is called, all storage will be deallocated including
**    that of the thumbnail.  The arguments to SetThumbNail and the
**    return values from GetThumbNail correspond to the OLE2.0 spec.
**    Note that on input, the thumbnail is read on demand but all the
**    other properties are pre-loaded.  The thumbnail is manipulated as
**    a windows handle to a METAFILEPICT structure, which in turn
**    contains a handle to the METAFILE.  The transferClip argument on
**    GetThumbNail, when set to true, transfers responsibility for
**    storage management of the thumbnail to the caller; that is, after
**    Free has been called, the handle is still valid. Clear can be
**    used to free storage for all the properties but then you must
**    call Read to load them again.  All the code is based on FAR
**    pointers.
**    CoInitialize MUST be called PRIOR to calling OleStdInitSummaryInfo.
**    Memory is allocated using the currently active IMalloc*
**    allocator (as is returned by call CoGetMalloc(MEMCTX_TASK) ).
**
** Common scenarios:
**    Read SummaryInfo
**    ----------------
**      OleStdInitSummaryInfo()
**      OleStdReadSummaryInfo()
**      . . . . .
**      call different Get routines
**      . . . . .
**      OleStdFreeSummaryInfo()
**
**    Create SummaryInfo
**    ------------------
**      OleStdInitSummaryInfo()
**      call different Set routines
**      OleStdWriteSummaryInfo()
**      OleStdFreeSummaryInfo()
**
**    Update SummaryInfo
**    ------------------
**      OleStdInitSummaryInfo()
**      OleStdReadSummaryInfo()
**      OleStdGetThumbNailProperty(necessary only if no SetThumb)
**      call different Set routines
**      OleStdWriteSummaryInfo()
**      OleStdFreeSummaryInfo()
*/

#define CHAR unsigned char
#define fTrue 1
#define fFalse 0
#define BYTE unsigned char
#define WORD unsigned short
#define DWORD unsigned long
#define LPVOID void FAR *
#define uchar unsigned char
#define ulong unsigned long
#define BOOL unsigned char
#define BF unsigned int

#include "suminfo.h"
#include "wn_dos.h"

#if defined( _DEBUG )
// following is from compobj.dll (ole2)
#define ASSERT(x) (!(x) ? FnAssert(#x, NULL, __FILE__, __LINE__) : 0)
#else
#define ASSERT(x)
#endif
```

```
typedef struct _RSUMINFO
     {
     WORD byteOrder;
     WORD formatVersion;
     WORD getOSVersion;
     WORD osVersion;
     CLSID classId;  //from compobj.h
     DWORD cSections;
     PROPSETLIST rgPropSet[1/*cSections*/]; //one section in standard
summary info
     STANDARDSECINMEM section;
     ULONG fileOffset;   //offset for thumbnail to support demand read
     } RSUMINFO;

typedef RSUMINFO FAR * LPRSI;

     typedef union _foo{
           ULARGE_INTEGER uli;
           struct {
                 DWORD       dw;
                 DWORD       dwh;
                 };
           struct {
                 WORD    w0;
                 WORD    w1;
                 WORD    w2;
                 WORD    w3;
                 };
           } Foo;



/* MemAlloc
** ---------
**    allocate memory using the currently active IMalloc* allocator
*/
static LPVOID MemAlloc(ULONG ulSize)
{
     LPVOID pout;
     LPMALLOC pmalloc;

     if (CoGetMalloc(MEMCTX_TASK, &pmalloc) != NOERROR) {
           ASSERT(pmalloc);
           return NULL;
     }

     pout = (LPVOID)pmalloc->Alloc(ulSize);

     if (pmalloc != NULL) {
           ULONG refs = pmalloc->Release();
     }

     return pout;
}
```

```
/* MemFree
** -------
**    free memory using the currently active IMalloc* allocator
*/
static void MemFree(LPVOID pmem)
{
     LPMALLOC pmalloc;

     if (pmem == NULL)
           return;

     if (CoGetMalloc(MEMCTX_TASK, &pmalloc) != NOERROR) {
           ASSERT(pmalloc);
           return;
     }

     pmalloc->Free(pmem);

     if (pmalloc != NULL) {
           ULONG refs = pmalloc->Release();
     }
}

// Replace the first argument with the product of itself and the multiplier
static void ulargeMultiply(ULARGE_INTEGER FAR *ul, USHORT m)
{
     Foo out, in;
     in.uli = *ul;
     out.dw = (ULONG)m * in.w0;      in.w0 = out.w0;
     out.dw = (ULONG)m * in.w1 + out.w1; in.w1 = out.w0;
     out.dw = (ULONG)m * in.w2 + out.w1; in.w2 = out.w0;
     out.dw = (ULONG)m * in.w3 + out.w1; in.w3 = out.w0;
     *ul = in.uli;
}

// Replace the first argument with the product of itself and the multiplier
static void ulargeDivide(ULARGE_INTEGER FAR *ul, USHORT m)
{
     Foo out, in;
     DWORD i;
     in.uli = *ul;
     out.dwh = in.dwh/(ULONG)m;
     i = in.dwh%(ULONG)m;
     in.w2 = in.w1;
     in.w3 = (WORD)i;
     out.w1 = (WORD)(in.dwh/(ULONG)m);
     in.w1 = (WORD)(in.dwh%(ULONG)m);
     out.w0 = (WORD)(in.dw/(ULONG)m);
     *ul = out.uli;
}


static void setStandard(LPRSI lprsi)
```

```
{
     int i;
     lprsi->cSections = 1;
     SetSumInfFMTID(&lprsi->rgPropSet[0].formatID);
     _fmemcpy(&lprsi->classId, &lprsi->rgPropSet[0].formatID,
sizeof(FMTID));
     lprsi->rgPropSet[0].byteOffset = cbNewSummaryInfo(1);
     for (i=0; i<cPID_STANDARD; i++)
           lprsi->section.rgPropId[i].propertyID = PID_TITLE+i;
     lprsi->section.cProperties = cPID_STANDARD; //always; do null test to
check validity
}

extern "C" {

/*************************************************************************
**
** OleStdInitSummaryInfo
**
** Purpose:
**    Initialize a Summary Info structure.
**
** Parameters:
**    int reserved             - reserverd for future use. must be 0.
**
** Return Value:
**    LPSUMINFO
**
** Comments:
**    CoInitialize MUST be called PRIOR to calling OleStdInitSummaryInfo.
**    Memory is allocated using the currently active IMalloc*
**    allocator (as is returned by call CoGetMalloc(MEMCTX_TASK) ).
**    Each LPSUMINFO instance must be initialized prior to use by
**    calling OleStdInitSummaryInfo. Once a LPSUMINFO instance is allocated
**    (by OleStdInitSummaryInfo), the user can call the Set procedures to
**    initialize fields.
*************************************************************************/

STDAPI_(LPSUMINFO) OleStdInitSummaryInfo(int reserved)
{
     LPRSI lprsi;

     if ((lprsi = (LPRSI)MemAlloc(sizeof(RSUMINFO))) != NULL)
     {
           ClearSumInf(lprsi, sizeof(RSUMINFO));
     } else return NULL;

     setStandard(lprsi);
     return (LPSUMINFO)lprsi;
}


/*************************************************************************
**
```

```
** OleStdFreeSummaryInfo
**
** Purpose:
**    Free a Summary Info structure.
**
** Parameters:
**    LPSUMINFO FAR *lp           - pointer to open Summary Info struct
**
** Return Value:
**    void
**
** Comments:
**    Memory is freed using the currently active IMalloc*
**    allocator (as is returned by call CoGetMalloc(MEMCTX_TASK) ).
**    Every LPSUMINFO struct must be freed after its last use.
**    When the OleStdFreeSummaryInfo routine is called, all storage will be
**    deallocated including that of the thumbnail (unless ownership of
**    the thumbnail has been transfered to the caller -- see
**    description of transferClip in GetThumbnail API).
**
*************************************************************************/

STDAPI_(void) OleStdFreeSummaryInfo(LPSUMINFO FAR *lplp)
{
     if (lplp==NULL||*lplp==NULL) return;
     OleStdClearSummaryInfo(*lplp);
     MemFree(*lplp);
     *lplp = NULL;
}


/*************************************************************************
**
** OleStdClearSummaryInfo
**
** Purpose:
**    Free storage (memory) for all the properties of the LPSUMINFO.
**
** Parameters:
**    LPSUMINFO FAR *lp             - pointer to open Summary Info struct
**
** Return Value:
**    void
**
** Comments:
**    After calling OleStdClearSummaryInfo you must call
OleStdReadSummaryInfo to
**    load them again.
**
*************************************************************************/

STDAPI_(void) OleStdClearSummaryInfo(LPSUMINFO lp)
{
     OleStdSetStringProperty(lp, PID_TITLE, NULL);
     OleStdSetStringProperty(lp, PID_SUBJECT, NULL);
     OleStdSetStringProperty(lp, PID_AUTHOR, NULL);
     OleStdSetStringProperty(lp, PID_KEYWORDS, NULL);
     OleStdSetStringProperty(lp, PID_COMMENTS, NULL);
     OleStdSetStringProperty(lp, PID_TEMPLATE, NULL);
     OleStdSetStringProperty(lp, PID_REVNUMBER, NULL);
     OleStdSetStringProperty(lp, PID_APPNAME, NULL);
     OleStdSetThumbNailProperty(NULL, lp, VT_CF_EMPTY, 0, NULL, NULL, 0);
     ClearSumInf((LPRSI)lp, sizeof(RSUMINFO));
}


/*************************************************************************
**
** OleStdReadSummaryInfo
**
** Purpose:
**    Read all Summary Info properties into memory (except thumbnail
**    which is demand loaded).
**
** Parameters:
**    LPSTREAM lps                  - open SummaryInfo IStream*
**    LPSUMINFO FAR *lp             - pointer to open Summary Info struct
**
** Return Value:
**    int                           - 1 for success
**                                  - 0 if error occurs
** Comments:
**
*************************************************************************/

STDAPI_(int) OleStdReadSummaryInfo(LPSTREAM lpStream, LPSUMINFO lp)
{
     STANDARDSECINMEM FAR *lpSSIM=(STANDARDSECINMEM FAR*)&((LPRSI)lp)-
>section;
     LPRSI lpSumInfo = (LPRSI)lp;
     SCODE sc;
     ULONG cbRead,i,sectionOffset;
     LARGE_INTEGER a;
     ULARGE_INTEGER b;
     int j,k,l;
     union {
           RSUMINFO rsi;
           STDZ stdz;
     };
     OleStdClearSummaryInfo(lp);
     LISet32(a, 0);
     sc = GetScode(lpStream->Seek(a, STREAM_SEEK_SET, &b));
     if (FAILED(sc)) goto fail;
     sectionOffset = cbNewSummaryInfo(1);
     sc = GetScode(lpStream->Read(&rsi, sectionOffset, &cbRead));
     if (FAILED(sc)||cbRead<sectionOffset) goto fail;

     if (!FIntelOrder(&rsi)||rsi.formatVersion!=0) goto fail;
     j = (int)rsi.cSections;
     while (j-->0) {
           if (FEqSumInfFMTID(&rsi.rgPropSet[0].formatID)) {
                 sectionOffset = rsi.rgPropSet[0].byteOffset;
                 break;
           } else {
                 sc = GetScode(lpStream->Read(&rsi.rgPropSet[0].formatID,
sizeof(PROPSETLIST), &cbRead));
                 if (FAILED(sc)||cbRead!=sizeof(PROPSETLIST)) goto fail;
           }
           if (j<=0) goto fail;
     }

     LISet32(a, sectionOffset);
     sc = GetScode(lpStream->Seek(a, STREAM_SEEK_SET, &b));
     if (FAILED(sc)) goto fail;
     sc = GetScode(lpStream->Read(&rsi.section, cbNewSection(1), &cbRead));
     if (FAILED(sc)||cbRead!=cbNewSection(1)) goto fail;
     i = rsi.section.cBytes+sectionOffset;
     j = (int)rsi.section.cProperties;
     if (j>cPID_STANDARD) goto fail;
     k = 0;
     while (j-->0) {
           k++;
           switch (l=(int)rsi.section.rgPropId[0].propertyID) {
                 case PID_PAGECOUNT:
                 case PID_WORDCOUNT:
                 case PID_CHARCOUNT:
                 case PID_SECURITY:
                       if (l==PID_SECURITY) l=3; else l-=PID_PAGECOUNT;
                       cbRead =
sectionOffset+rsi.section.rgPropId[0].byteOffset;
                       if (cbRead>=i) goto fail;
                       LISet32(a, cbRead);
                       sc = GetScode(lpStream->Seek(a, STREAM_SEEK_SET,
&b));
                       if (FAILED(sc)) goto fail;
                       sc = GetScode(lpStream->Read(&lpSSIM->rgInts[l],
sizeof(INTS), &cbRead));
                       if (FAILED(sc)||cbRead!=sizeof(INTS)) goto fail;
                       if (lpSSIM->rgInts[l].vtType==VT_EMPTY) break;
                       if (lpSSIM->rgInts[l].vtType!=VT_I4) goto fail;
                       break;
                 case PID_EDITTIME:
                 case PID_LASTPRINTED:
                 case PID_CREATE_DTM_RO:
                 case PID_LASTSAVE_DTM:
                       l-=PID_EDITTIME;
                       cbRead =
sectionOffset+rsi.section.rgPropId[0].byteOffset;
                       if (cbRead>=i) goto fail;
                       LISet32(a, cbRead);
                       sc = GetScode(lpStream->Seek(a, STREAM_SEEK_SET,
&b));
                       if (FAILED(sc)) goto fail;
                       sc = GetScode(lpStream->Read(&lpSSIM->rgTime[l],
sizeof(TIME), &cbRead));
```

```
                       if (FAILED(sc)||cbRead!=sizeof(TIME)) goto fail;
                       if (lpSSIM->rgTime[l].vtType==VT_EMPTY) break;
                       if (lpSSIM->rgTime[l].vtType!=VT_FILETIME) goto fail;
                       break;
                 case PID_TITLE:
                 case PID_SUBJECT:
                 case PID_AUTHOR:
                 case PID_KEYWORDS:
                 case PID_COMMENTS:
                 case PID_TEMPLATE:
                 case PID_LASTAUTHOR:
                 case PID_REVNUMBER:
                 case PID_APPNAME:
                       cbRead =
sectionOffset+rsi.section.rgPropId[0].byteOffset;
                       if (cbRead>=i) goto fail;
                       LISet32(a, cbRead);
                       sc = GetScode(lpStream->Seek(a, STREAM_SEEK_SET,
&b));
                       if (FAILED(sc)) goto fail;
                       sc = GetScode(lpStream->Read(&stdz, sizeof(STDZ),
&cbRead));
                       if (FAILED(sc)||cbRead<sizeof(DWORD)*2) goto fail;
                       if (stdz.vtType==VT_EMPTY||stdz.vtByteCount<=1)
break;
                       if (stdz.vtType!=VT_LPSTR||stdz.vtByteCount>WORDMAX)
goto fail;
                       stdz.rgchars[(int)stdz.vtByteCount-1] = '\0';
                       OleStdSetStringProperty(lp, (DWORD)l,
(LPSTR)&stdz.rgchars[0]);
                       break;
                 case PID_THUMBNAIL:
                       cbRead =
sectionOffset+rsi.section.rgPropId[0].byteOffset;
                       if (cbRead>=i) goto fail;
                       LISet32(a, cbRead);
                       sc = GetScode(lpStream->Seek(a, STREAM_SEEK_SET,
&b));
                       if (FAILED(sc)) goto fail;
                       lpSumInfo->fileOffset = cbRead+sizeof(DWORD)*3;
                       sc = GetScode(lpStream->Read(&lpSSIM->thumb,
sizeof(DWORD)*4, &cbRead));
                       if (FAILED(sc)||cbRead!=sizeof(DWORD)*4) {
                             lpSSIM->thumb.vtType = VT_EMPTY;
                             goto fail;
                       }
                       if (lpSSIM->thumb.vtType == VT_EMPTY) {
                             lpSSIM->thumb.cBytes = 0;
                             break;
                       }
                       if (lpSSIM->thumb.vtType != VT_CF) {
                             lpSSIM->thumb.vtType = VT_EMPTY;
                             goto fail;
                       }
                       lpSSIM->thumb.cBytes -= sizeof(DWORD); //for selector
```

```
                        if (lpSSIM->thumb.selector==VT_CF_WIN||lpSSIM-
>thumb.selector==VT_CF_MAC) {
                              lpSumInfo->fileOffset += sizeof(DWORD);
                              lpSSIM->thumb.cBytes -= sizeof(DWORD); //for
format val
                        }
                        break;
                        default: ;
            }
            if (j<=0)
            {
            // We should fail if the document is password-protected.
                  if(OleStdGetSecurityProperty(lp)==fSecurityPassworded)
                        goto fail;
                  return 1;
            }
            LISet32(a, sectionOffset+sizeof(DWORD)*2+k*sizeof(PROPIDLIST));
            sc = GetScode(lpStream->Seek(a, STREAM_SEEK_SET, &b));
            if (FAILED(sc)) goto fail;
            sc = GetScode(lpStream->Read(&rsi.section.rgPropId[0],
sizeof(PROPIDLIST), &cbRead));
            if (FAILED(sc)||cbRead!=sizeof(PROPIDLIST)) goto fail;
      }

fail:
     OleStdClearSummaryInfo(lpSumInfo);
     return 0;
}


/*************************************************************************
**
** OleStdWriteSummaryInfo
**
** Purpose:
**    Write all Summary Info properties to a IStream*
**
** Parameters:
**    LPSTREAM lps                  - open SummaryInfo IStream*
**    LPSUMINFO FAR *lp             - pointer to open Summary Info struct
**
** Return Value:
**    int                           - 1 for success
**                                  - 0 if error occurs
** Comments:
**
*************************************************************************/

STDAPI_(int) OleStdWriteSummaryInfo(LPSTREAM lpStream, LPSUMINFO lp)
{
     STANDARDSECINMEM FAR *lpSSIM=(STANDARDSECINMEM FAR*)&((LPRSI)lp)-
>section;
     // REVIEW: localization issues for propert sets
     //         do we need to include a code page and dictionary?
     LPRSI lpSumInfo = (LPRSI)lp;
     SCODE sc;
     ULONG cbWritten;
     ULONG cBytes, oBytes, k,l,m,n;
     LARGE_INTEGER a;
     ULARGE_INTEGER b;
     CHAR FAR *lps;
     LPMETAFILEPICT lpmfp;
     int i,j,s;

     setStandard(lpSumInfo);
     oBytes = cbNewSection(cPID_STANDARD);  //offsets are relative to the
section
     cBytes = cbNewSection(cPID_STANDARD)+(sizeof(TIME)*MAXTIME)+
(sizeof(INTS)*MAXINTS);

     lpSSIM->rgPropId[PID_EDITTIME-2].byteOffset = oBytes;
     lpSSIM->rgPropId[PID_LASTPRINTED-2].byteOffset = oBytes+sizeof(TIME);
     lpSSIM->rgPropId[PID_CREATE_DTM_RO-2].byteOffset =
oBytes+sizeof(TIME)*2;
     lpSSIM->rgPropId[PID_LASTSAVE_DTM-2].byteOffset =
oBytes+sizeof(TIME)*3;

     lpSSIM->rgPropId[PID_PAGECOUNT-2].byteOffset = oBytes+
(sizeof(TIME)*MAXTIME);
     lpSSIM->rgPropId[PID_WORDCOUNT-2].byteOffset = oBytes+
(sizeof(TIME)*MAXTIME+sizeof(INTS));
     lpSSIM->rgPropId[PID_CHARCOUNT-2].byteOffset = oBytes+
(sizeof(TIME)*MAXTIME+sizeof(INTS)*2);
     lpSSIM->rgPropId[PID_SECURITY-2].byteOffset = oBytes+
(sizeof(TIME)*MAXTIME+sizeof(INTS)*3);
     oBytes += sizeof(TIME)*MAXTIME + sizeof(INTS)*MAXINTS;

     lpSSIM->rgPropId[PID_THUMBNAIL-2].byteOffset = oBytes;
     l = 0;
     if (lpSSIM->thumb.vtType==VT_EMPTY) k = sizeof(DWORD);
     else {
           l = ((lpSSIM->thumb.cBytes+4-1)>>2)<<2;
           if (lpSSIM->thumb.selector==VT_CF_BYTES) k = sizeof(DWORD)*3;
           else if (lpSSIM->thumb.selector==VT_CF_FMTID) {k =
sizeof(DWORD)*3; l += sizeof(FMTID); }
           else if (lpSSIM->thumb.selector==VT_CF_NAME)  {k =
sizeof(DWORD)*3; l += (((*lpSSIM->thumb.lpstzName+1+3)>>2)<<2);}
           else k = sizeof(DWORD)*4;
     }
     cBytes += k+l;
     oBytes += k+l;

     for (i=0; i<MAXSTDZ; i++) {
           j = 0;
           if (lpSSIM->rglpsz[i]!=NULL) {
                 j = lpSSIM->rglpsz[i]->VTCB+1/*null*/;
                 lpSSIM->rglpsz[i]->vtByteCount = j;
                 j = (((j+4-1)>>2)<<2)+sizeof(DWORD);
                 cBytes += j;
           }
           if (i!=MAXSTDZ-1) lpSSIM->rgPropId[i].byteOffset = oBytes;
           else lpSSIM->rgPropId[PID_APPNAME-2].byteOffset = oBytes;
           oBytes += j+sizeof(DWORD);
           cBytes += sizeof(DWORD); //type
     }
     lpSSIM->cBytes = cBytes;


     LISet32(a, 0);
     sc = GetScode(lpStream->Seek(a, STREAM_SEEK_SET, &b));
     if (FAILED(sc)) return 0;
     sc = GetScode(lpStream->Write(lpSumInfo, cbNewSummaryInfo(1),
&cbWritten));
     if (FAILED(sc)||cbWritten!=cbNewSummaryInfo(1)) return 0;
     sc = GetScode(lpStream->Write(lpSSIM, cbNewSection(cPID_STANDARD)
+sizeof(TIME)*MAXTIME+sizeof(INTS)*MAXINTS, &cbWritten));
     if (FAILED(sc)||cbWritten!=cbNewSection(cPID_STANDARD)
+sizeof(TIME)*MAXTIME+sizeof(INTS)*MAXINTS) return 0;

     m = lpSSIM->thumb.cBytes;
     if (lpSSIM->thumb.lpstzName!=NULL) s = *lpSSIM->thumb.lpstzName;
     else s = 0;
     if (m!=0) {
           lpSSIM->thumb.cBytes = (k-sizeof(DWORD)*2)+
                 (((lpSSIM->thumb.cBytes+4-1)>>2)<<2)+(((s+4-1)>>2)<<2);
           n = lpSSIM->thumb.selector;
           lps = lpSSIM->thumb.lpByte;
           ASSERT(lps!=NULL);  //maybe a GetThumbNail here
           ASSERT(n!=VT_CF_NAME);
           if (n==VT_CF_WIN) { //bytes are in global memory
                 lpmfp = (LPMETAFILEPICT)GlobalLock((HANDLE)(DWORD)lps);
                 if (lpmfp==NULL) goto fail;
                 lps = (CHAR FAR*)GlobalLock(lpmfp->hMF);
           }
           if (n==VT_CF_NAME) lpSSIM->thumb.selector = *lpSSIM-
>thumb.lpstzName+1/*null*/;
     }
     sc = GetScode(lpStream->Write(&lpSSIM->thumb, k, &cbWritten));
     if (FAILED(sc)||cbWritten!=k) goto fail;
     if (s!=0) {
           k = ((s+1+4-1)>>2)<<2;
           sc = GetScode(lpStream->Write(lpSSIM->thumb.lpstzName+1, k,
&cbWritten));
           if (FAILED(sc)||cbWritten!=k) goto fail;
     }
     if (m!=0) {
           k = ((m+3)>>2)<<2;
           if (n==VT_CF_WIN||VT_CF_NAME) { //bytes are in global memory
                 sc = GetScode(lpStream->Write(lpmfp, sizeof(METAFILEPICT),
&cbWritten));
                 k -= sizeof(METAFILEPICT);
           }
```

```
            sc = GetScode(lpStream->Write(lps, k, &cbWritten));
            if (FAILED(sc)||cbWritten!=k) goto fail;
            if (n==VT_CF_WIN||VT_CF_NAME) { //bytes are in global memory
                  GlobalUnlock(lpmfp->hMF);
                  GlobalUnlock((HANDLE)(DWORD)lpSSIM->thumb.lpByte);
            }
      }
      lpSSIM->thumb.cBytes = m;   //restore in mem value
      lpSSIM->thumb.selector = n;

      k = VT_EMPTY;
      for (i=0; i<MAXSTDZ; i++) {
            if (lpSSIM->rglpsz[i]!=NULL) {
                  l = lpSSIM->rglpsz[i]->vtByteCount;
                  j = ((((int)l+4-1)/4)*4)+sizeof(DWORD)*2;
                  sc = GetScode(lpStream->Write(lpSSIM->rglpsz[i], j,
&cbWritten));
                  if (FAILED(sc)||cbWritten!=(ULONG)j) return 0;
                  lpSSIM->rglpsz[i]->vtByteCount = 0; //restore stz count
convention
                  lpSSIM->rglpsz[i]->VTCB = (int)l;
            } else {
                  sc = GetScode(lpStream->Write(&k, sizeof(DWORD),
&cbWritten));
                  if (FAILED(sc)||cbWritten!=sizeof(DWORD)) return 0;
            }
      }
      return 1;
fail:
      lpSSIM->thumb.cBytes = m;   //restore in mem value
      lpSSIM->thumb.selector = n;
      if (m!=0&&(n==VT_CF_WIN||VT_CF_NAME)) { //bytes are in global memory
            GlobalUnlock((HANDLE)(DWORD)lps);
      }
      return 0;
}


/*************************************************************************
**
** OleStdGetSecurityProperty
**
** Purpose:
**    Retrieve the Security Property
**
** Parameters:
**    LPSUMINFO FAR *lp           - pointer to open Summary Info struct
**
** Return Value:
**    DWORD                       - security level
**          AllSecurityFlagsEqNone 0    - no security
**          fSecurityPassworded 1       - password required
**          fSecurityRORecommended 2    - read-only is recommended
**          fSecurityRO 4               - read-only is required
**          fSecurityLockedForAnnotations 8 - locked for annotations
```

```
**
** Comments:
**    by noting the (suggested; that is, application-enforced) security
**    level on the document, an application other than the originator
**    of the document can adjust its user interface to the properties
**    appropriately. An application should not display any of the
**    information about a password protected document, and should not
**    allow modifications to enforced read-only or locked for
**    annotations documents. It should warn the user about read-only
**    recommended if the user attempts to modify properties.
**
*************************************************************************/

STDAPI_(DWORD) OleStdGetSecurityProperty(LPSUMINFO lp)
{
STANDARDSECINMEM FAR *lpSSIM=(STANDARDSECINMEM FAR*)&((LPRSI)lp)->section;
     if (lpSSIM->rgInts[3].vtType == VT_I4) return lpSSIM->rgInts[3].value;
     return 0;
}


/*************************************************************************
**
** OleStdSetSecurityProperty
**
** Purpose:
**    Set the Security Property
**
** Parameters:
**    LPSUMINFO FAR *lp           - pointer to open Summary Info struct
**    DWORD security              - security level
**          AllSecurityFlagsEqNone 0    - no security
**          fSecurityPassworded 1       - password required
**          fSecurityRORecommended 2    - read-only is recommended
**          fSecurityRO 4               - read-only is required
**          fSecurityLockedForAnnotations 8 - locked for annotations
**
** Return Value:
**    int                           - 1 for success
**                                  - 0 if error occurs
**                                    (there are no errors)
**
** Comments:
**    by noting the (suggested; that is, application-enforced) security
**    level on the document, an application other than the originator
**    of the document can adjust its user interface to the properties
**    appropriately. An application should not display any of the
**    information about a password protected document, and should not
**    allow modifications to enforced read-only or locked for
**    annotations documents. It should warn the user about read-only
**    recommended if the user attempts to modify properties.
**
*************************************************************************/
```

```
STDAPI_(int) OleStdSetSecurityProperty(LPSUMINFO lp, DWORD security)
{
     STANDARDSECINMEM FAR *lpSSIM=(STANDARDSECINMEM FAR*)&((LPRSI)lp)-
>section;

     // REVIEW: check valid transitions; how do we know APP called us?

     if (security==0) {
           lpSSIM->rgInts[3].vtType = VT_EMPTY;
           return 1;
     }
     lpSSIM->rgInts[3].vtType = VT_I4;
     lpSSIM->rgInts[3].value = security;
     return 1;
}


/*************************************************************************
**
** OleStdGetStringProperty
**
** Purpose:
**    Retrieve a String Propety.
**    (returns zero terminated string -- C string)
**
** Parameters:
**    LPSUMINFO FAR *lp             - pointer to open Summary Info struct
**    DWORD pid                     - ID of String Property
**
** Return Value:
**    LPSTR                         - value of String Property
**                                      (zero terminated string--C string)
**
** Comments:
**    String should NOT be freed by caller. Memory for string will be
**    freed when OleStdFreeSummaryInfo is called.
*************************************************************************/

STDAPI_(LPSTR) OleStdGetStringProperty(LPSUMINFO lp, DWORD pid)
{
     LPSTR l = OleStdGetStringZProperty(lp,pid);
     if (l==NULL) return NULL; else return l+1;
}


/*************************************************************************
**
** OleStdSetStringProperty
**
** Purpose:
**    Set a String Propety
**    (takes zero terminated string -- C string)
**
** Parameters:
**    LPSUMINFO FAR *lp             - pointer to open Summary Info struct
**    DWORD pid                        - ID of String Property
**    LPSTR lpsz                       - new value for String Property.
**                                         zero terminated string -- C string.
**                                         May be NULL, in which case the
**                                         propery is cleared.
**
** Return Value:
**    int                              - 1 if successful
**                                     - 0 invalid property id
**
** Comments:
**    The input string is copied.
**
*************************************************************************/

STDAPI_(int) OleStdSetStringProperty(LPSUMINFO lp, DWORD pid, LPSTR lpsz)
{
     LPRSI lprsi=(LPRSI)lp;
     STANDARDSECINMEM FAR* lpSSIM=(STANDARDSECINMEM FAR*)&((LPRSI)lp)-
>section;
     int i;
     if (pid==PID_APPNAME) {
           pid = MAXSTDZ-1;
     } else if (pid<PID_TITLE || pid>PID_REVNUMBER) return 0; else pid -= 2;
     ASSERT(lpSSIM);
     if (lpSSIM->rglpsz[pid]) MemFree(lpSSIM->rglpsz[pid]);
     if ((lpsz==NULL)||(*lpsz==0)) {
           lpSSIM->rglpsz[pid] = NULL;
           return (1);
     }
     i = _fstrlen(lpsz);
     lpSSIM->rglpsz[pid] = (STDZ FAR*)MemAlloc(i+1/*null*/+sizeof(DWORD)*2);
     if (lpSSIM->rglpsz[pid]==NULL) return 0;
     _fstrcpy((LPSTR)&lpSSIM->rglpsz[pid]->rgchars, lpsz);
     lpSSIM->rglpsz[pid]->vtType = VT_LPSTR;
     lpSSIM->rglpsz[pid]->vtByteCount = 0;
     lpSSIM->rglpsz[pid]->VTCB = i;
     return (1);
}


/*************************************************************************
**
** OleStdGetStringZProperty
**
** Purpose:
**    Retrieve a String Propety.
**    (returns zero-terminated with leading byte count string)
**
** Parameters:
**    LPSUMINFO FAR *lp                - pointer to open Summary Info struct
**    DWORD pid                        - ID of Property
**
** Return Value:
**    LPSTZR                           - value of String Property
**                                        (zero-terminated with leading
**                                        byte count)
**
** Comments:
**    String should NOT be freed by caller. Memory for string will be
**    freed when OleStdFreeSummaryInfo is called.
*************************************************************************/

STDAPI_(LPSTZR) OleStdGetStringZProperty(LPSUMINFO lp, DWORD pid)
{
     STANDARDSECINMEM FAR *lpSSIM=(STANDARDSECINMEM FAR*)&((LPRSI)lp)-
>section;
     if (pid==PID_APPNAME) {
           pid = MAXSTDZ-1;
     } else if (pid<PID_TITLE || pid>PID_REVNUMBER) return NULL; else pid -=
2;
     if (lpSSIM->rglpsz[pid]!=NULL) {
           return (LPSTR)&lpSSIM->rglpsz[pid]->VTCB;
     }
     return NULL;
}


/*************************************************************************
**
** OleStdGetDocProperty
**
** Purpose:
**    Retrieve document properties (no. pages, no. words, no. characters)
**
** Parameters:
**    LPSUMINFO FAR *lp             - pointer to open Summary Info struct
**    DWORD FAR *nPage              - (OUT) number of pages in document
**    DWORD FAR *nWords             - (OUT) number of words in document
**    DWORD FAR *nChars             - (OUT) number of charactrs in doc
**
** Return Value:
**    void
**
** Comments:
**
*************************************************************************/

STDAPI_(void) OleStdGetDocProperty(
           LPSUMINFO       lp,
           DWORD FAR*      nPage,
           DWORD FAR*      nWords,
           DWORD FAR*      nChars
)
{
STANDARDSECINMEM FAR *lpSSIM=(STANDARDSECINMEM FAR*)&((LPRSI)lp)->section;
     *nPage=0; *nWords=0; *nChars=0;
     if (lpSSIM->rgInts[0].vtType == VT_I4) *nPage = lpSSIM-
>rgInts[0].value;
```

```
     if (lpSSIM->rgInts[1].vtType == VT_I4) *nWords = lpSSIM-
>rgInts[1].value;
     if (lpSSIM->rgInts[2].vtType == VT_I4) *nChars = lpSSIM-
>rgInts[2].value;
}


/*************************************************************************
**
** OleStdSetDocProperty
**
** Purpose:
**    Set document properties (no. pages, no. words, no. characters)
**
** Parameters:
**    LPSUMINFO FAR *lp             - pointer to open Summary Info struct
**    DWORD nPage                   - number of pages in document
**    DWORD nWords                  - number of words in document
**    DWORD nChars                  - number of charactrs in doc
**
** Return Value:
**    int                           - 1 for success
**                                  - 0 if error occurs
**                                    (there are no errors)
**
** Comments:
**
*************************************************************************/

STDAPI_(int) OleStdSetDocProperty(
           LPSUMINFO       lp,
           DWORD           nPage,
           DWORD           nWords,
           DWORD           nChars
)
{
DWORD vttype=VT_I4;
STANDARDSECINMEM FAR *lpSSIM=(STANDARDSECINMEM FAR*)&((LPRSI)lp)->section;
     if ((nPage|nWords|nChars)==0) {
           vttype = VT_EMPTY;
           nPage=0; nWords=0; nChars=0;
     }
     lpSSIM->rgInts[0].vtType = vttype;
     lpSSIM->rgInts[1].vtType = vttype;
     lpSSIM->rgInts[2].vtType = vttype;
     lpSSIM->rgInts[0].value = nPage;
     lpSSIM->rgInts[1].value = nWords;
     lpSSIM->rgInts[2].value = nChars;
     return 1;
}


/*************************************************************************
**
** OleStdGetThumbNailProperty
```

```
**
** Purpose:
**    Retrieve a Thumbnail Property
**
** Parameters:
**    LPSTREAM lps
**    LPSUMINFO FAR *lp             - pointer to open Summary Info struct
**    DWORD FAR* clipFormatNo       - clipboard format for thumbnail
**                                      (type of value depends on vtcf
**                                      return value.)
**                                      NOTE: ONLY VT_CF_WIN is
**                                      implemented, so clipFormatNo
**                                      will be CF_METAFILEPICT
**    LPSTR FAR* lpszName           - format name if VT_CF_NAME is
**                                      returned
**                                      NOTE: NOT IMPLEMENTED
**    THUMBNAIL FAR* clip           - handle to thumbnail
**                                      for VT_CF_WIN clip will be
**                                      handle to MetafilePict
**                                      NOTE: only VT_CF_WIN IMPLEMENTED
**    DWORD FAR* byteCount          - size of thumbnail stream
**                                      for VT_CF_WIN case this should
**                                      be combined size of both the
**                                      Metafile as well as the
**                                      MetafilePict structure.
**    BOOL transferClip             - transfer ownership of thumbnail
**                                      to caller. (see comment)
**
** Return Value:
**    int vtcfNo                    - OLE thumbnail selector value
**      VT_CF_WIN                   - Windows thumbnail
**                                      (interpret clipFormatNo as
**                                      Windows clipboard format)
**      VT_CF_FMTID                 - (NOT IMPLEMENTED)
**                                      thumbnail format is specified
**                                      by ID. use clipFormatNo.
**                                      (but NOT a Windows format ID)
**
**      VT_CF_NAME                  - (NOT IMPLEMENTED)
**                                      thumbnail format is specified
**                                      by name. use lpszName.
**      VT_CF_EMPTY                 - blank thumbnail
**                                      (clip will be NULL)
**      VT_CF_OOM                   - Memory allocation failure
**
** Comments:
**    NOTE: Currently there is only proper support for VT_CF_WIN.
**    OleStdSetThumbNailProperty does implement VT_CF_FMTID and VT_CF_NAME,
**    however, OleStdGetThumbNailProperty, OleStdReadSummaryInfo and
**    OleStdWriteSummaryInfo only support VT_CF_WIN.
**
**    Note that on input, the thumbnail is read on demand while all the
**    other properties are pre-loaded.  The thumbnail is manipulated as
**    a windows handle to a METAFILEPICT structure, which in turn
**    contains a handle to the METAFILE.  The transferClip argument on
**    GetThumbNail, when set to true, transfers responsibility for
**    storage management of the thumbnail to the caller; that is, after
**    OleStdFreeSummaryInfo has been called, the handle is still valid.
****************************************************************************/

STDAPI_(int) OleStdGetThumbNailProperty(
           LPSTREAM        lps,
           LPSUMINFO       lp,
           DWORD FAR*      clipFormatNo,
           LPSTR FAR*      lpszName,
           THUMBNAIL FAR*  clip,
           DWORD FAR*      byteCount,
           BOOL            transferClip
)
{
     int i;
     LPRSI lprsi=(LPRSI)lp;
     STANDARDSECINMEM FAR *lpSSIM=(STANDARDSECINMEM FAR*)&((LPRSI)lp)-
>section;
     ULONG cbRead, cbToRead;
     LARGE_INTEGER a;
     ULARGE_INTEGER b;
     CHAR FAR *lpst;
     LPMETAFILEPICT lpmfp;
     HANDLE hst, hmfp;
     SCODE sc;
     *byteCount = 0;
     if (lpSSIM->thumb.cBytes==0) return VT_CF_EMPTY;
     if (lpSSIM->thumb.lpByte==NULL) {
           LISet32(a, lprsi->fileOffset);
           sc = GetScode(lps->Seek(a, STREAM_SEEK_SET, &b));
           if (FAILED(sc)) return VT_CF_EMPTY;
           i = (int) lpSSIM->thumb.selector;
           if (i>0||i==VT_CF_FMTID) {
                 if (i>255) return VT_CF_EMPTY;
                 else if (i==VT_CF_FMTID) i = sizeof(FMTID);
                 else lpSSIM->thumb.selector = (DWORD)VT_CF_NAME;
                 cbToRead = ((i+3)>>2)<<2;
                 lpSSIM->thumb.lpstzName=(unsigned char
FAR*)MemAlloc(i+1/*n*/+1);
                 if (lpSSIM->thumb.lpstzName==NULL) return VT_CF_OOM;
                 sc = GetScode(lps->Read(lpSSIM->thumb.lpstzName+1,
cbToRead, &cbRead));
                 if (FAILED(sc)||cbRead!=cbToRead) return VT_CF_EMPTY;
                 *lpSSIM->thumb.lpstzName = i;
                 *(lpSSIM->thumb.lpstzName+i) = 0;
                 lpSSIM->thumb.cBytes -= cbToRead+sizeof(DWORD);
           }
           i = (int) lpSSIM->thumb.selector;
           cbToRead = lpSSIM->thumb.cBytes;
           if (cbToRead>65535) return VT_CF_OOM;
           ASSERT(i!=VT_CF_NAME);
           if (i==VT_CF_WIN) {
                 cbToRead -= sizeof(METAFILEPICT);
```

```
                  hmfp = GlobalAlloc(GMEM_MOVEABLE, sizeof(METAFILEPICT));
                  if (hmfp==NULL) return VT_CF_OOM;
                  hst = GlobalAlloc(GMEM_MOVEABLE, cbToRead);
                  if (hst==NULL) {
                        GlobalFree(hmfp);
                        return VT_CF_OOM;
                  }
                  lpmfp = (LPMETAFILEPICT)GlobalLock(hmfp);
                  sc = GetScode(lps->Read(lpmfp, sizeof(METAFILEPICT),
&cbRead));
                  if (FAILED(sc)||cbRead!=sizeof(METAFILEPICT)) {
                        GlobalUnlock(hmfp);
                        GlobalFree(hmfp);
                        GlobalFree(hst);
                        return VT_CF_EMPTY;
                  }
                  lpst = (CHAR FAR*)GlobalLock(hst);
                  lpmfp->hMF = (HMETAFILE)hst;
                  lpSSIM->thumb.lpByte = (CHAR FAR*)hmfp;
            } else {
                  lpst = (unsigned char FAR*)MemAlloc((int)cbToRead);
                  if (lpst==NULL) return VT_CF_OOM;
                  lpSSIM->thumb.lpByte = lpst;
            }
            sc = GetScode(lps->Read(lpst, cbToRead, &cbRead));
            if (i==VT_CF_WIN) {
                  GlobalUnlock(hst);
                  GlobalUnlock(hmfp);
            }
            if (FAILED(sc)||cbRead!=cbToRead) {
                  if (i==VT_CF_WIN) {
                        GlobalFree(hst);
                        GlobalFree(hmfp);
                  } else MemFree(lpst);
                  lpSSIM->thumb.lpByte = NULL;
                  if ((i==VT_CF_NAME||i==VT_CF_FMTID)&&(lpSSIM-
>thumb.lpstzName!=NULL))
                        MemFree(lpSSIM->thumb.lpstzName);
                  return VT_CF_EMPTY;
            }
      }
      *clipFormatNo = lpSSIM->thumb.clipFormat;
      *byteCount = lpSSIM->thumb.cBytes;
      if(lpszName!=NULL)
            *lpszName = (char FAR*)lpSSIM->thumb.lpstzName+1;
      *clip = (char FAR*)lpSSIM->thumb.lpByte;
      if (transferClip) lpSSIM->thumb.lpByte=NULL;
      return (int)lpSSIM->thumb.selector;
}


/*************************************************************************
**
** OleStdSetThumbNailProperty
```

```
**
** Purpose:
**    Set a Thumbnail Property
**
** Parameters:
**    LPSTREAM lps                  - open SummaryInfo IStream*
**    LPSUMINFO FAR *lp             - pointer to open Summary Info struct
**    int vtcfNo                    - OLE thumbnail selector value
**          VT_CF_WIN                   - Windows thumbnail
**                                        (interpret clipFormatNo as
**                                        Windows clipboard format)
**          VT_CF_FMTID                 - thumbnail format is specified
**                                        by ID. use clipFormatNo.
**                                        (but NOT a Windows format ID)
**
**          VT_CF_NAME                  - thumbnail format is specified
**                                        by name. use lpszName.
**          VT_CF_EMPTY                 - blank thumbnail
**                                        (clip will be NULL)
**
**    DWORD FAR* clipFormatNo       - clipboard format for thumbnail
**                                      used if vtcfNo is VT_CF_WIN or
**                                      VT_CF_FMTID. interpretation of
**                                      value depends on vtcfNo specified.
**                                      (normally vtcfNo==VT_CF_WIN and
**                                      clipFormatNo==CF_METAFILEPICT)
**    LPSTR FAR* lpszName           - format name if vtcfNo is VT_CF_NAME
**    THUMBNAIL clip                - handle to thumbnail
**                                      for VT_CF_WIN clip will be
**                                      handle to MetafilePict
**    DWORD FAR* byteCount          - size of thumbnail stream
**                                      for VT_CF_WIN case this should
**                                      be combined size of both the
**                                      Metafile as well as the
**                                      MetafilePict structure.
**
** Return Value:
**    int                           - 1 for success
**                                  - 0 if error occurs
**
** Comments:
**    NOTE: Currently there is only proper support for VT_CF_WIN.
**    OleStdSetThumbNailProperty does implement VT_CF_FMTID and VT_CF_NAME,
**    however, OleStdGetThumbNailProperty, OleStdReadSummaryInfo and
**    OleStdWriteSummaryInfo only support VT_CF_WIN.
**
**    This function copies lpszName but saves the "clip" handle passed.
**
**    NOTE: overwriting or emptying frees space for clip and name.
**    The thumbnail is manipulated as a windows handle to a
**    METAFILEPICT structure, which in turn contains a handle to the
**    METAFILE.
*************************************************************************/

STDAPI_(int) OleStdSetThumbNailProperty(
           LPSTREAM        lps,
           LPSUMINFO       lp,
           int             vtcfNo,
           DWORD           clipFormatNo,
           LPSTR           lpszName,
           THUMBNAIL       clip,
           DWORD           byteCount
)
{
     int i;
     LPRSI lprsi=(LPRSI)lp;
     STANDARDSECINMEM FAR *lpSSIM=(STANDARDSECINMEM FAR*)&((LPRSI)lp)-
>section;
     LPMETAFILEPICT lpmfp;
     if (lpSSIM==NULL||vtcfNo>0||vtcfNo<VT_CF_EMPTY||
(vtcfNo==VT_CF_NAME&&(lpszName==NULL||*lpszName==0))) {
           return 0;
     }
     if (vtcfNo!=VT_CF_EMPTY&&(clip==0||byteCount==0)) return 0;
     i = (int) lpSSIM->thumb.vtType;
     if (i!=VT_EMPTY) {
           i = (int) lpSSIM->thumb.selector;
           ASSERT(i!=VT_CF_NAME);
           if (i==VT_CF_WIN) {
                 if (lpSSIM->thumb.lpByte!=NULL) {
                       lpmfp = (LPMETAFILEPICT)GlobalLock((HANDLE)
(DWORD)lpSSIM->thumb.lpByte);
                       GlobalFree(lpmfp->hMF);
                       GlobalUnlock((HANDLE)(DWORD)lpSSIM->thumb.lpByte);
                       GlobalFree((HANDLE)(DWORD)lpSSIM->thumb.lpByte);
                 }
           } else {
                 MemFree(lpSSIM->thumb.lpByte);
           }
           if ((i==VT_CF_NAME||i==VT_CF_FMTID)&&(lpSSIM->thumb.lpstzName!
=NULL))
                 MemFree(lpSSIM->thumb.lpstzName);
           lpSSIM->thumb.lpstzName = NULL;
           lpSSIM->thumb.lpByte = NULL;
     }
     if (vtcfNo==VT_CF_EMPTY) {
           lpSSIM->thumb.vtType = VT_EMPTY;
           lpSSIM->thumb.cBytes = 0;
     } else {
           lpSSIM->thumb.vtType = VT_CF;
           lpSSIM->thumb.selector = vtcfNo;
           lpSSIM->thumb.cBytes = byteCount;
           lpSSIM->thumb.clipFormat = clipFormatNo;
           lpSSIM->thumb.lpByte = (unsigned char FAR*)clip;    //just save
the hnadle
           if (vtcfNo==VT_CF_NAME||vtcfNo==VT_CF_FMTID) {
                 i = _fstrlen(lpszName);
                 if (vtcfNo==VT_CF_FMTID) ASSERT(i==sizeof(FMTID));
                 lpSSIM->thumb.lpstzName =
                       (unsigned char FAR*)MemAlloc(i+1/*n*/+1/*null*/);
                 if (lpSSIM->thumb.lpstzName==NULL) {
                       lpSSIM->thumb.vtType = VT_EMPTY;
                       return 0;
                 }
                 _fstrcpy((char FAR*)lpSSIM->thumb.lpstzName+1, lpszName);
                 *lpSSIM->thumb.lpstzName = i;
           }
     }
     return 1;
}


/*************************************************************************
**
** OleStdGetDateProperty
**
** Purpose:
**    Retrieve Data Property
**
** Parameters:
**    LPSUMINFO FAR *lp              - pointer to open Summary Info struct
**    DWORD pid                      - ID of Property
**    int FAR *yr                    - (OUT) year
**    int FAR *mo                    - (OUT) month
**    int FAR *dy                    - (OUT) day
**    DWORD FAR *sc                  - (OUT) seconds
**
** Return Value:
**    void
**
** Comments:
**
*************************************************************************/

STDAPI_(void) OleStdGetDateProperty(
           LPSUMINFO       lp,
           DWORD           pid,
           int FAR*        yr,
           int FAR*        mo,
           int FAR*        dy,
           DWORD FAR*      sc
)
{
     STANDARDSECINMEM FAR *lpSSIM=(STANDARDSECINMEM FAR*)&((LPRSI)lp)-
>section;
     SFFS sffs;
     pid -= PID_EDITTIME;
     *yr = 0; *mo = 0; *dy = 0; *sc = 0;
     if (pid<0||pid>=MAXTIME) return;
     if (lpSSIM->rgTime[pid].vtType == VT_FILETIME) {
           if (pid==0) {
                 //convert from 100ns to seconds
                 ulargeDivide((ULARGE_INTEGER FAR*)&lpSSIM->rgTime[0].time,
10000);
```

```
                 ulargeDivide((ULARGE_INTEGER FAR*)&lpSSIM->rgTime[0].time,
1000);
                 pid = lpSSIM->rgTime[0].time.dwLowDateTime;
                 *sc = pid%((DWORD)60*60*24);
                 pid /= (DWORD)60*60*24;
                 *dy = (int)(pid%(DWORD)30);
                 pid /= (DWORD)30;
                 *mo = (int)(pid%(DWORD)12);
                 *yr = (int)(pid/(DWORD)12);
           } else {
                 if (CoFileTimeToDosDateTime(&lpSSIM->rgTime[pid].time,
                                   &sffs.dateVariable, &sffs.timeVariable))
{
                       *yr = sffs.yr+1980;
                       *mo = sffs.mon;
                       *dy = sffs.dom;
                       *sc = (DWORD)sffs.hr*3600+sffs.mint*60+sffs.sec*2;
                 }
           }
     }
     return;
}



/*************************************************************************
**
** OleStdSetDateProperty
**
** Purpose:
**    Set Data Property
**
** Parameters:
**    LPSUMINFO FAR *lp              - pointer to open Summary Info struct
**    DWORD pid                      - ID of Property
**    int yr                         - year
**    int mo                         - month
**    int dy                         - day
**    DWORD sc                       - seconds
**
** Return Value:
**    int                            - 1 for success
**                                   - 0 if error occurs
**
** Comments:
**    Use all zeros to clear.
**    The following is an example of valid input:
**          yr=1993 mo=1(Jan) dy=1(1st) hr=12(noon) mn=30 sc=23
**    for PID_EDITTIME property, the values are a zero-origin duration
**    of time.
**
*************************************************************************/

STDAPI_(int) OleStdSetDateProperty(
            LPSUMINFO       lp,
            DWORD           pid,
            int             yr,
            int             mo,
            int             dy,
            int             hr,
            int             mn,
            int             sc
)
{
     STANDARDSECINMEM FAR *lpSSIM=(STANDARDSECINMEM FAR*)&((LPRSI)lp)-
>section;
     SFFS sffs;
     pid -= PID_EDITTIME;
     if (pid<0||pid>=MAXTIME) return 0;
     if ((yr|mo|dy|hr|mn|sc)==0) {   //all must be zero
           lpSSIM->rgTime[pid].vtType = VT_EMPTY;
           return 1;
     }
     lpSSIM->rgTime[pid].vtType = VT_FILETIME;
     if (pid==0) {
           lpSSIM->rgTime[0].time.dwLowDateTime =
                       (((((DWORD)yr*365+mo*30)+dy)*24+hr)*60+mn)*60+sc;
           lpSSIM->rgTime[0].time.dwHighDateTime = 0;
           //10^7 nanoseconds/second
           ulargeMultiply((ULARGE_INTEGER FAR*)&lpSSIM->rgTime[0].time,
10000);
           //convert to units of 100 ns
           ulargeMultiply((ULARGE_INTEGER FAR*)&lpSSIM->rgTime[0].time,
1000);
     } else {
           sffs.yr = max(yr-1980,0);
           sffs.mon = mo;
           sffs.dom = dy;
           sffs.hr = hr;
           sffs.mint= mn;
           sffs.sec = sc/2;  //dos is 2 second intervals
           if (!CoDosDateTimeToFileTime(sffs.date, sffs.time,
                       &lpSSIM->rgTime[pid].time)) {
                 lpSSIM->rgTime[pid].vtType = VT_EMPTY;
                 return 0;
           }
     }
     return 1;
}

} //END C
```

TARGTDEV.C (WRAPUI Sample)

```
/*************************************************************************
**
**    OLE 2 Standard Utilities
**
**    olestd.c
**
**    This file contains utilities that are useful for dealing with
**    target devices.
**
**    (c) Copyright Microsoft Corp. 1992 All Rights Reserved
**
*************************************************************************/

#define STRICT  1
#include "ole2ui.h"
#ifndef WIN32
#include <print.h>
#endif

/*
 * OleStdCreateDC()
 *
 * Purpose:
 *
 * Parameters:
 *
 * Return Value:
 *    SCODE  -  S_OK if successful
 */
STDAPI_(HDC) OleStdCreateDC(DVTARGETDEVICE FAR* ptd)
{
     HDC hdc=NULL;
     LPDEVNAMES lpDevNames;
     LPDEVMODE lpDevMode;
     LPSTR lpszDriverName;
     LPSTR lpszDeviceName;
     LPSTR lpszPortName;

     if (ptd == NULL) {
           hdc = CreateDC("DISPLAY", NULL, NULL, NULL);
           goto errReturn;
     }

     lpDevNames = (LPDEVNAMES) ptd; // offset for size field

     if (ptd->tdExtDevmodeOffset == 0) {
           lpDevMode = NULL;
     }else{
           lpDevMode  = (LPDEVMODE) ((LPSTR)ptd + ptd->tdExtDevmodeOffset);
     }

     lpszDriverName = (LPSTR) lpDevNames + ptd->tdDriverNameOffset;
     lpszDeviceName = (LPSTR) lpDevNames + ptd->tdDeviceNameOffset;
     lpszPortName   = (LPSTR) lpDevNames + ptd->tdPortNameOffset;

     hdc = CreateDC(lpszDriverName, lpszDeviceName, lpszPortName,
lpDevMode);

errReturn:
     return hdc;
}


/*
 * OleStdCreateIC()
 *
 * Purpose: Same as OleStdCreateDC, except that information context is
 *          created, rather than a whole device context.  (CreateIC is
 *          used rather than CreateDC).
 *          OleStdDeleteDC is still used to delete the information context.
 *
 * Parameters:
 *
 * Return Value:
 *    SCODE  -  S_OK if successful
 */
STDAPI_(HDC) OleStdCreateIC(DVTARGETDEVICE FAR* ptd)
{
     HDC hdcIC=NULL;
     LPDEVNAMES lpDevNames;
     LPDEVMODE lpDevMode;
     LPSTR lpszDriverName;
     LPSTR lpszDeviceName;
     LPSTR lpszPortName;

     if (ptd == NULL) {
           hdcIC = CreateIC("DISPLAY", NULL, NULL, NULL);
           goto errReturn;
     }

     lpDevNames = (LPDEVNAMES) ptd; // offset for size field

     lpDevMode  = (LPDEVMODE) ((LPSTR)ptd + ptd->tdExtDevmodeOffset);

     lpszDriverName = (LPSTR) lpDevNames + ptd->tdDriverNameOffset;
     lpszDeviceName = (LPSTR) lpDevNames + ptd->tdDeviceNameOffset;
     lpszPortName   = (LPSTR) lpDevNames + ptd->tdPortNameOffset;

     hdcIC = CreateIC(lpszDriverName, lpszDeviceName, lpszPortName,
lpDevMode);

errReturn:
     return hdcIC;
}


/*
```

```
 * OleStdCreateTargetDevice()
 *
 * Purpose:
 *
 * Parameters:
 *
 * Return Value:
 *    SCODE  -  S_OK if successful
 */
STDAPI_(DVTARGETDEVICE FAR*) OleStdCreateTargetDevice(LPPRINTDLG lpPrintDlg)
{
     DVTARGETDEVICE FAR* ptd=NULL;
     LPDEVNAMES lpDevNames, pDN;
     LPDEVMODE lpDevMode, pDM;
     UINT nMaxOffset;
     LPSTR pszName;
     DWORD dwDevNamesSize, dwDevModeSize, dwPtdSize;

     if ((pDN = (LPDEVNAMES)GlobalLock(lpPrintDlg->hDevNames)) == NULL) {
           goto errReturn;
     }

     if ((pDM = (LPDEVMODE)GlobalLock(lpPrintDlg->hDevMode)) == NULL) {
           goto errReturn;
     }

     nMaxOffset =  (pDN->wDriverOffset > pDN->wDeviceOffset) ?
           pDN->wDriverOffset : pDN->wDeviceOffset ;

     nMaxOffset =  (pDN->wOutputOffset > nMaxOffset) ?
           pDN->wOutputOffset : nMaxOffset ;

     pszName = (LPSTR)pDN + nMaxOffset;

     dwDevNamesSize = (DWORD)(nMaxOffset+lstrlen(pszName) + 1/* NULL term
*/);
     dwDevModeSize = (DWORD) (pDM->dmSize + pDM->dmDriverExtra);

     dwPtdSize = sizeof(DWORD) + dwDevNamesSize + dwDevModeSize;

     if ((ptd = (DVTARGETDEVICE FAR*)OleStdMalloc(dwPtdSize)) != NULL) {

           // copy in the info
           ptd->tdSize = (UINT)dwPtdSize;

           lpDevNames = (LPDEVNAMES) &ptd->tdDriverNameOffset;
           _fmemcpy(lpDevNames, pDN, (size_t)dwDevNamesSize);

           lpDevMode=(LPDEVMODE)((LPSTR)&ptd-
>tdDriverNameOffset+dwDevNamesSize);
           _fmemcpy(lpDevMode, pDM, (size_t)dwDevModeSize);

           ptd->tdDriverNameOffset += 4 ;
           ptd->tdDeviceNameOffset += 4 ;
           ptd->tdPortNameOffset   += 4 ;
           ptd->tdExtDevmodeOffset = (UINT)dwDevNamesSize + 4 ;
     }

errReturn:
     GlobalUnlock(lpPrintDlg->hDevNames);
     GlobalUnlock(lpPrintDlg->hDevMode);

     return ptd;
}



/*
 * OleStdDeleteTargetDevice()
 *
 * Purpose:
 *
 * Parameters:
 *
 * Return Value:
 *    SCODE  -  S_OK if successful
 */
STDAPI_(BOOL) OleStdDeleteTargetDevice(DVTARGETDEVICE FAR* ptd)
{
     BOOL res=TRUE;

     if (ptd != NULL) {
           OleStdFree(ptd);
     }

     return res;
}



/*
 * OleStdCopyTargetDevice()
 *
 * Purpose:
 *  duplicate a TARGETDEVICE struct. this function allocates memory for
 *  the copy. the caller MUST free the allocated copy when done with it
 *  using the standard allocator returned from CoGetMalloc.
 *  (OleStdFree can be used to free the copy).
 *
 * Parameters:
 *  ptdSrc      pointer to source TARGETDEVICE
 *
 * Return Value:
 *    pointer to allocated copy of ptdSrc
 *    if ptdSrc==NULL then retuns NULL is returned.
 *    if ptdSrc!=NULL and memory allocation fails, then NULL is returned
 */
STDAPI_(DVTARGETDEVICE FAR*) OleStdCopyTargetDevice(DVTARGETDEVICE FAR*
ptdSrc)
```

```
{
  DVTARGETDEVICE FAR* ptdDest = NULL;

  if (ptdSrc == NULL) {
     return NULL;
  }

  if ((ptdDest = (DVTARGETDEVICE FAR*)OleStdMalloc(ptdSrc->tdSize)) != NULL)
{
     _fmemcpy(ptdDest, ptdSrc, (size_t)ptdSrc->tdSize);
  }

  return ptdDest;
}


/*
 * OleStdCopyFormatEtc()
 *
 * Purpose:
 *  Copies the contents of a FORMATETC structure. this function takes
 *  special care to copy correctly copying the pointer to the TARGETDEVICE
 *  contained within the source FORMATETC structure.
 *  if the source FORMATETC has a non-NULL TARGETDEVICE, then a copy
 *  of the TARGETDEVICE will be allocated for the destination of the
 *  FORMATETC (petcDest).
 *
 *  OLE2NOTE: the caller MUST free the allocated copy of the TARGETDEVICE
 *  within the destination FORMATETC when done with it
 *  using the standard allocator returned from CoGetMalloc.
 *  (OleStdFree can be used to free the copy).
 *
 * Parameters:
 *  petcDest      pointer to destination FORMATETC
 *  petcSrc       pointer to source FORMATETC
 *
 * Return Value:
 *    returns TRUE is copy is successful; retuns FALSE if not successful
 */
STDAPI_(BOOL) OleStdCopyFormatEtc(LPFORMATETC petcDest, LPFORMATETC petcSrc)
{
  if ((petcDest == NULL) || (petcSrc == NULL)) {
     return FALSE;
  }

  petcDest->cfFormat = petcSrc->cfFormat;

  petcDest->ptd      = OleStdCopyTargetDevice(petcSrc->ptd);
  petcDest->dwAspect = petcSrc->dwAspect;
  petcDest->lindex   = petcSrc->lindex;
  petcDest->tymed    = petcSrc->tymed;

  return TRUE;

}
```

```
// returns 0 for exact match, 1 for no match, -1 for partial match (which is
// defined to mean the left is a subset of the right: fewer aspects, null
target
// device, fewer medium).

STDAPI_(int) OleStdCompareFormatEtc(FORMATETC FAR* pFetcLeft, FORMATETC FAR*
pFetcRight)
{
     BOOL bExact = TRUE;

     if (pFetcLeft->cfFormat != pFetcRight->cfFormat)
           return 1;
     else if (!OleStdCompareTargetDevice (pFetcLeft->ptd, pFetcRight->ptd))
           return 1;
     if (pFetcLeft->dwAspect == pFetcRight->dwAspect)
           // same aspects; equal
           ;
     else if ((pFetcLeft->dwAspect & ~pFetcRight->dwAspect) != 0)
           // left not subset of aspects of right; not equal
           return 1;
     else
           // left subset of right
           bExact = FALSE;

     if (pFetcLeft->tymed == pFetcRight->tymed)
           // same medium flags; equal
           ;
     else if ((pFetcLeft->tymed & ~pFetcRight->tymed) != 0)
           // left not subset of medium flags of right; not equal
           return 1;
     else
           // left subset of right
           bExact = FALSE;

     return bExact ? 0 : -1;
}



STDAPI_(BOOL) OleStdCompareTargetDevice
     (DVTARGETDEVICE FAR* ptdLeft, DVTARGETDEVICE FAR* ptdRight)
{
     if (ptdLeft == ptdRight)
           // same address of td; must be same (handles NULL case)
           return TRUE;
     else if ((ptdRight == NULL) || (ptdLeft == NULL))
           return FALSE;
     else if (ptdLeft->tdSize != ptdRight->tdSize)
           // different sizes, not equal
           return FALSE;
#ifdef WIN32
     else if (memcmp(ptdLeft, ptdRight, ptdLeft->tdSize) != 0)
#else
```

```
     else if (_fmemcmp(ptdLeft, ptdRight, (int)ptdLeft->tdSize) != 0)
#endif
            // not same target device, not equal
            return FALSE;

     return TRUE;
}
```

TEMPLATE.H   (WRAPUI Sample)

```
/*
 * TEMPLATE.H
 *
 * CUSTOMIZATION INSTRUCTIONS:
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 *
 *
 *  1.  Replace <FILE> with the uppercased filename for this file.
 *      Lowercase the <FILE>.h entry
 *
 *  2.  Replace <NAME> with the mixed case dialog name in one word,
 *      such as InsertObject
 *
 *  3.  Replace <FULLNAME> with the mixed case dialog name in multiple
 *      words, such as Insert Object
 *
 *  4.  Replace <ABBREV> with the suffix for pointer variables, such
 *      as the IO in InsertObject's pIO or the CI in ChangeIcon's pCI.
 *      Check the alignment of the first variable declaration in the
 *      Dialog Proc after this.  I will probably be misaligned with the
 *      rest of the variables.
 *
 *  5.  Replace <STRUCT> with the uppercase structure name for this
 *      dialog sans OLEUI, such as INSERTOBJECT.  Changes OLEUI<STRUCT>
 *      in most cases, but we also use this for IDD_<STRUCT> as the
 *      standard template resource ID.
 *
 *  6.  Find <UFILL> fields and fill them out with whatever is appropriate.
 *
 *  7.  Delete this header up to the start of the next comment.
 *
 */


/*
 * <FILE>.H
 *
 * Internal definitions, structures, and function prototypes for the
 * OLE 2.0 UI <FULLNAME> dialog.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */


#ifndef <UFILL>
#define <UFILL>

//UFILL>  Move from here to INTERNAL to to OLE2UI.H


typedef struct tagOLEUI<STRUCT>
```

```
     {
     //These IN fields are standard across all OLEUI dialog functions.
     DWORD           cbStruct;       //Structure Size
     DWORD           dwFlags;        //IN-OUT:  Flags
     HWND            hWndOwner;      //Owning window
     LPCSTR          lpszCaption;    //Dialog caption bar contents
     LPFNOLEUIHOOK   lpfnHook;       //Hook callback
     LPARAM          lCustData;      //Custom data to pass to hook
     HINSTANCE       hInstance;      //Instance for customized template name
     LPCSTR          lpszTemplate;   //Customized template name
     HRSRC           hResource;      //Customized template handle

     //Specifics for OLEUI<STRUCT>.  All are IN-OUT unless otherwise spec.
     } OLEUI<STRUCT>, *POLEUI<STRUCT>, FAR *LPOLEUI<STRUCT>;


//API Prototype
UINT FAR PASCAL OleUI<NAME>(LPOLEUI<STRUCT>);


//<FULLNAME> flags
#define <ABBREV>F_SHOWHELP                0x00000001L
<UFILL>


//<FULLNAME> specific error codes
//DEFINE AS OLEUI_<ABBREV>ERR_<ERROR>     (OLEUI_ERR_STANDARDMAX+n)
<UFILL>


//<FULLNAME> Dialog identifiers
//FILL IN DIALOG IDs HERE
<UFILL>





//INTERNAL INFORMATION STARTS HERE

//Internally used structure
typedef struct tag<STRUCT>
     {
     //Keep this item first as the Standard* functions depend on it here.
     LPOLEUI<STRUCT>     lpO<ABBREV>;       //Original structure passed.

     /*
      * What we store extra in this structure besides the original caller's
      * pointer are those fields that we need to modify during the life of
      * the dialog but that we don't want to change in the original
structure
      * until the user presses OK.
      */

     <UFILL>
     } <STRUCT>, *P<STRUCT>;



//Internal function prototypes
//<FILE>.C
BOOL FAR PASCAL <NAME>DialogProc(HWND, UINT, WPARAM, LPARAM);
BOOL            F<NAME>Init(HWND hDlg, WPARAM, LPARAM);
<UFILL>



#endif //<UFILL>
```

TEMPLATE.C   (WRAPUI Sample)

```
/*
 * TEMPLATE.C
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 *
 *
 * CUSTOMIZATION INSTRUCTIONS:
 *
 *  1.  Replace <FILE> with the uppercased filename for this file.
 *      Lowercase the <FILE>.h entry
 *
 *  2.  Replace <NAME> with the mixed case dialog name in one word,
 *      such as InsertObject
 *
 *  3.  Replace <FULLNAME> with the mixed case dialog name in multiple
 *      words, such as Insert Object
 *
 *  4.  Replace <ABBREV> with the suffix for pointer variables, such
 *      as the IO in InsertObject's pIO or the CI in ChangeIcon's pCI.
 *      Check the alignment of the first variable declaration in the
 *      Dialog Proc after this.  I will probably be misaligned with the
 *      rest of the variables.
 *
 *  5.  Replace <STRUCT> with the uppercase structure name for this
 *      dialog sans OLEUI, such as INSERTOBJECT.  Changes OLEUI<STRUCT>
 *      in most cases, but we also use this for IDD_<STRUCT> as the
 *      standard template resource ID.
 *
 *  6.  Find <UFILL> fields and fill them out with whatever is appropriate.
 *
 *  7.  Delete this header up to the start of the next comment.
 */


/*
 * <FILE>.C
 *
 * Implements the OleUI<NAME> function which invokes the complete
 * <FULLNAME> dialog.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */

#define STRICT  1
#include "ole2ui.h"
#include "common.h"
#include "<FILE>.h"




/*
```

```
 * OleUI<NAME>
 *
 * Purpose:
 *  Invokes the standard OLE <FULLNAME> dialog box allowing the user
 *  to <UFILL>
 *
 * Parameters:
 *  lp<ABBREV>            LPOLEUI<NAME> pointing to the in-out structure
 *                  for this dialog.
 *
 * Return Value:
 *  UINT            One of the following codes, indicating success or error:
 *                      OLEUI_SUCCESS           Success
 *                      OLEUI_ERR_STRUCTSIZE    The dwStructSize value is
wrong
 */

STDAPI_(UINT) OleUI<NAME>(LPOLEUI<STRUCT> lp<ABBREV>)
     {
     UINT        uRet;
     HGLOBAL     hMemDlg=NULL;

     uRet=UStandardValidation((LPOLEUISTANDARD)lp<ABBREV>,
sizeof(OLEUI<STRUCT>)
                                          , &hMemDlg);

     if (OLEUI_SUCCESS!=uRet)
           return uRet;

     /*
      * PERFORM ANY STRUCTURE-SPECIFIC VALIDATION HERE!
      * ON FAILURE:
      *  {
      *  if (NULL!=hMemDlg)
      *      FreeResource(hMemDlg)
      *
      *  return OLEUI_<ABBREV>ERR_<ERROR>
      *  }
      */

     //Now that we've validated everything, we can invoke the dialog.
     uRet=UStandardInvocation(<NAME>DialogProc, (LPOLEUISTANDARD)lp<ABBREV>
                                          , hMemDlg,
MAKEINTRESOURCE(IDD_<STRUCT>));

     /*
      * IF YOU ARE CREATING ANYTHING BASED ON THE RESULTS, DO IT HERE.
      */
     <UFILL>

     return uRet;
     }
```

```
/*
 * <NAME>DialogProc
 *
 * Purpose:
 *  Implements the OLE <FULLNAME> dialog as invoked through the
 *  OleUI<NAME> function.
 *
 * Parameters:
 *  Standard
 *
 * Return Value:
 *  Standard
 */

BOOL CALLBACK EXPORT <NAME>DialogProc(HWND hDlg, UINT iMsg, WPARAM wParam,
LPARAM lParam)
     {
     P<STRUCT>               p<ABBREV>;
     BOOL                    fHook=FALSE;


     //Declare Win16/Win32 compatible WM_COMMAND parameters.
     COMMANDPARAMS(wID, wCode, hWndMsg);

     //This will fail under WM_INITDIALOG, where we allocate it.
     p<ABBREV>=(<STRUCT>)PvStandardEntry(hDlg, iMsg, wParam, lParam,
&uHook);

     //If the hook processed the message, we're done.
     if (0!=uHook)
           return (BOOL)uHook;

     //Process the temination message
     if (iMsg==uMsgEndDialog)
           {
           //Free any specific allocations before calling StandardCleanup
           StandardCleanup((PVOID)p<ABBREV>, hDlg);
           EndDialog(hDlg, wParam);
           return TRUE;
           }

     switch (iMsg)
           {
           case WM_INITDIALOG:
                 F<NAME>Init(hDlg, wParam, lParam);
                 return TRUE;


           case WM_COMMAND:
                 switch (wID)
                       {
                       case IDOK:
                             /*
```

```
                              * PERFORM WHATEVER FUNCTIONS ARE DEFAULT HERE.
                              */
                             SendMessage(hDlg, uMsgEndDialog, OLEUI_OK, 0L);
                             break;

                       case IDCANCEL:
                             /*
                              * PERFORM ANY UNDOs HERE, BUT NOT CLEANUP THAT
WILL
                              * ALWAYS HAPPEN WHICH SHOULD BE IN
uMsgEndDialog.
                              */
                             SendMessage(hDlg, uMsgEndDialog, OLEUI_CANCEL,
0L);
                             break;

                       case ID_OLEUIHELP:
                             PostMessage(p<ABBREV>->lpO<ABBREV>->hWndOwner,
uMsgHelp
                                               , (WPARAM)hDlg,
MAKELPARAM(IDD_<STRUCT>, 0));
                             break;
                       }
                 break;
           }
     return FALSE;
     }




/*
 * F<NAME>Init
 *
 * Purpose:
 *  WM_INITIDIALOG handler for the <FULLNAME> dialog box.
 *
 * Parameters:
 *  hDlg            HWND of the dialog
 *  wParam          WPARAM of the message
 *  lParam          LPARAM of the message
 *
 * Return Value:
 *  BOOL            Value to return for WM_INITDIALOG.
 */

BOOL F<NAME>Init(HWND hDlg, WPARAM wParam, LPARAM lParam)
     {
     P<STRUCT>               p<ABBREV>;
     LPOLEUI<STRUCT>         lpO<ABBREV>;
     HFONT                   hFont;

     //1.  Copy the structure at lParam into our instance memory.
     p<ABBREV>=(PSTRUCT)PvStandardInit(hDlg, sizeof(<STRUCT>), TRUE,
&hFont);
```

```
     //PvStandardInit send a termination to us already.
     if (NULL==p<ABBREV>)
           return FALSE;

     lpO<ABBREV>=(LPOLEUI<STRUCT>)lParam);

     p<ABBREV>->lpO<ABBREV>=lpO<ABBREV>;

     //Copy other information from lpO<ABBREV> that we might modify.
     <UFILL>

     //2.  If we got a font, send it to the necessary controls.
     if (NULL!=hFont)
           {
           //Do this for as many controls as you need it for.
           SendDlgItemMessage(hDlg, ID_<UFILL>, WM_SETFONT, (WPARAM)hFont,
0L);
           }


     //3.  Show or hide the help button
     if (!(p<ABBREV>->lpO<ABBREV>->dwFlags & <ABBREV>F_SHOWHELP))
           StandardShowDlgItem(hDlg, ID_OLEUIHELP, SW_HIDE);

     /*
      * PERFORM OTHER INITIALIZATION HERE.  ON ANY LoadString
      * FAILURE POST OLEUI_MSG_ENDDIALOG WITH OLEUI_ERR_LOADSTRING.
      */

     //n.  Call the hook with lCustData in lParam
     UStandardHook((PVOID)p<ABBREV>, hDlg, WM_INITDIALOG, wParam,
lpO<ABBREV>->lCustData);
     return TRUE;
     }
```

UTILITY.H (WRAPUI Sample)

```
/*
 * UTILITY.H
 *
 * Miscellaneous prototypes and definitions for OLE UI dialogs.
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */


#ifndef _UTILITY_H_
#define _UTILITY_H_

//Function prototypes
//UTILITY.C
HCURSOR  WINAPI HourGlassOn(void);
void     WINAPI HourGlassOff(HCURSOR);

BOOL     WINAPI Browse(HWND, LPSTR, LPSTR, UINT, UINT, UINT, DWORD);
int      WINAPI ReplaceCharWithNull(LPSTR, int);
int      WINAPI ErrorWithFile(HWND, HINSTANCE, UINT, LPSTR, UINT);
HFILE WINAPI DoesFileExist(LPSTR lpszFile, OFSTRUCT FAR* lpOpenBuf);


HICON FAR PASCAL    HIconAndSourceFromClass(REFCLSID, LPSTR, UINT FAR *);
BOOL FAR PASCAL FIconFileFromClass(REFCLSID, LPSTR, UINT, UINT FAR *);
LPSTR FAR PASCAL PointerToNthField(LPSTR, int, char);
BOOL FAR PASCAL GetAssociatedExecutable(LPSTR, LPSTR);
HICON    WINAPI HIconFromClass(LPSTR);
BOOL     WINAPI FServerFromClass(LPSTR, LPSTR, UINT);
UINT     WINAPI UClassFromDescription(LPSTR, LPSTR, UINT);
UINT     WINAPI UDescriptionFromClass(LPSTR, LPSTR, UINT);
BOOL     WINAPI FVerbGet(LPSTR, UINT, LPSTR);
LPSTR    WINAPI ChopText(HWND hwndStatic, int nWidth, LPSTR lpch);
void     WINAPI OpenFileError(HWND hDlg, UINT nErrCode, LPSTR lpszFile);


#endif //_UTILITY_H_
```

UTILITY.C   (WRAPUI Sample)

```
/*
 * UTILITY.C
 *
 * Utility routines for functions inside OLE2UI.DLL
 *
 *  General:
 *  ----------------------
 *  HourGlassOn             Displays the hourglass
 *  HourGlassOff            Hides the hourglass
 *
 *  Misc Tools:
 *  ----------------------
 *  Browse                  Displays the "File..." or "Browse..." dialog.
 *  ReplaceCharWithNull     Used to form filter strings for Browse.
 *  ErrorWithFile           Creates an error message with embedded filename
 *  OpenFileError           Give error message for OpenFile error return
 *  ChopText                Chop a file path to fit within a specified width
 *  DoesFileExist           Checks if file is valid
 *
 *  Registration Database:
 *  ----------------------
 *  HIconFromClass          Extracts the first icon in a class's server path
 *  FServerFromClass        Retrieves the server path for a class name
(fast)
 *  UClassFromDescription   Finds the classname given a description (slow)
 *  UDescriptionFromClass   Retrieves the description for a class name
(fast)
 *  FGetVerb                Retrieves a specific verb for a class (fast)
 *
 *
 * Copyright (c)1992 Microsoft Corporation, All Right Reserved
 */

#define STRICT  1
#include "ole2ui.h"
#include <stdlib.h>
#include <commdlg.h>
#include <memory.h>
#include <cderr.h>
#include "common.h"
#include "utility.h"
#include "geticon.h"

#ifdef WIN32
/* 1/23/94 - define LINK_COMMDLG avoid LoadLibrary ("commdlg.dll")*/
#define LINK_COMMDLG
#endif

OLEDBGDATA

/*
 * HourGlassOn
 *
 * Purpose:
 *  Shows the hourglass cursor returning the last cursor in use.
 *
 * Parameters:
 *  None
 *
 * Return Value:
 *  HCURSOR         Cursor in use prior to showing the hourglass.
 */

HCURSOR WINAPI HourGlassOn(void)
     {
     HCURSOR     hCur;

     hCur=SetCursor(LoadCursor(NULL, IDC_WAIT));
     ShowCursor(TRUE);

     return hCur;
     }



/*
 * HourGlassOff
 *
 * Purpose:
 *  Turns off the hourglass restoring it to a previous cursor.
 *
 * Parameters:
 *  hCur            HCURSOR as returned from HourGlassOn
 *
 * Return Value:
 *  None
 */

void WINAPI HourGlassOff(HCURSOR hCur)
     {
     ShowCursor(FALSE);
     SetCursor(hCur);
     return;
     }




/*
 * Browse
 *
 * Purpose:
 *  Displays the standard GetOpenFileName dialog with the title of
 *  "Browse."  The types listed in this dialog are controlled through
 *  iFilterString.  If it's zero, then the types are filled with "*.*"
 *  Otherwise that string is loaded from resources and used.
 *
 * Parameters:
 *  hWndOwner       HWND owning the dialog
 *  lpszFile        LPSTR specifying the initial file and the buffer in
 *                  which to return the selected file.  If there is no
 *                  initial file the first character of this string should
 *                  be NULL.
 *  lpszInitialDir  LPSTR specifying the initial directory.  If none is to
 *                  set (ie, the cwd should be used), then this parameter
 *                  should be NULL.
 *  cchFile         UINT length of pszFile
 *  iFilterString   UINT index into the stringtable for the filter string.
 *  iTitleString    UINT index into the stringtable for the title string.
 *  dwOfnFlags      DWORD flags to OR with OFN_HIDEREADONLY
 *
 * Return Value:
 *  BOOL            TRUE if the user selected a file and pressed OK.
 *                  FALSE otherwise, such as on pressing Cancel.
 */

BOOL WINAPI Browse(HWND hWndOwner, LPSTR lpszFile, LPSTR lpszInitialDir,
UINT cchFile, UINT iFilterString, UINT iTitleString, DWORD dwOfnFlags)
     {
        UINT           cch;
        char           szTitle[256];
        char           szFilters[256];
        OPENFILENAME   ofn;
        BOOL           fStatus;
        DWORD          dwError;
#if !defined( LINK_COMMDLG )
        HINSTANCE hinstCommDlg = NULL;
        typedef BOOL (WINAPI* LPFNGETOPENFILENAME) (OPENFILENAME FAR*);
        LPFNGETOPENFILENAME lpfnGetOpenFileName;
        typedef DWORD (WINAPI* LPFNCOMMDLGEXTENDEDERROR) (void);
        LPFNCOMMDLGEXTENDEDERROR lpfnCommDlgExtendedError;
#endif

     if (NULL==lpszFile || 0==cchFile)
           return FALSE;

     /*
      * REVIEW:  Exact contents of the filter combobox is TBD.  One idea
      * is to take all the extensions in the RegDB and place them in here
      * with the descriptive class name associate with them.  This has the
      * extra step of finding all extensions of the same class handler and
      * building one extension string for all of them.  Can get messy quick.
      * UI demo has only *.* which we do for now.
      */

     if (0!=iFilterString)
           cch=LoadString(ghInst, iFilterString, (LPSTR)szFilters,
sizeof(szFilters));
     else
           {
           szFilters[0]=0;
           cch=1;
           }

     if (0==cch)
           return FALSE;

     ReplaceCharWithNull(szFilters, szFilters[cch-1]);

     if (0!=iTitleString)
           cch=LoadString(ghInst, iTitleString, (LPSTR)szTitle,
sizeof(szTitle));
     else
           {
           szTitle[0]=0;
           cch=1;
           }

     if (0==cch)
           return FALSE;

     //Prior string must also be initialized, if there is one.
     _fmemset((LPOPENFILENAME)&ofn, 0, sizeof(ofn));
     ofn.lStructSize =sizeof(ofn);
     ofn.hwndOwner   =hWndOwner;
     ofn.lpstrFile   =lpszFile;
     ofn.nMaxFile    =cchFile;
     ofn.lpstrFilter =(LPSTR)szFilters;
     ofn.nFilterIndex=1;
     ofn.lpstrTitle  =(LPSTR)szTitle;
     ofn.hInstance   = ghInst;
     ofn.lpTemplateName = MAKEINTRESOURCE(IDD_FILEOPEN);
     if (NULL != lpszInitialDir)
       ofn.lpstrInitialDir = lpszInitialDir;

     ofn.Flags= OFN_HIDEREADONLY | OFN_ENABLETEMPLATE | (dwOfnFlags) ;

#if !defined( LINK_COMMDLG )
     // Load the COMMDLG.DLL library module
     hinstCommDlg = LoadLibrary("COMMDLG.DLL");
     if ((DWORD)hinstCommDlg <= HINSTANCE_ERROR)   /* load failed */
           return FALSE;

     // Retrieve the address of required functions
     lpfnGetOpenFileName = (LPFNGETOPENFILENAME)
                 GetProcAddress(hinstCommDlg, "GetOpenFileName");
     lpfnCommDlgExtendedError = (LPFNCOMMDLGEXTENDEDERROR)
                 GetProcAddress(hinstCommDlg, "CommDlgExtendedError");

     if (lpfnGetOpenFileName == NULL || lpfnCommDlgExtendedError == NULL)
           {
           FreeLibrary(hinstCommDlg);
           return FALSE;
           }

     //On success, copy the chosen filename to the static display
     fStatus = (*lpfnGetOpenFileName) ((LPOPENFILENAME)&ofn);
     dwError = (*lpfnCommDlgExtendedError)();

     FreeLibrary(hinstCommDlg);
     return fStatus;
#else
     //On success, copy the chosen filename to the static display
     fStatus = GetOpenFileName((LPOPENFILENAME)&ofn);
     dwError = CommDlgExtendedError();
     return fStatus;
#endif
     }
```

```
/*
 * ReplaceCharWithNull
 *
 * Purpose:
 *  Walks a null-terminated string and replaces a given character
 *  with a zero.  Used to turn a single string for file open/save
 *  filters into the appropriate filter string as required by the
 *  common dialog API.
 *
 * Parameters:
 *  psz             LPSTR to the string to process.
 *  ch              int character to replace.
 *
 * Return Value:
 *  int             Number of characters replaced.  -1 if psz is NULL.
 */

int WINAPI ReplaceCharWithNull(LPSTR psz, int ch)
     {
     int             cChanged=-1;

     if (NULL!=psz)
           {
           while (0!=*psz)
                 {
                 if (ch==*psz)
                       {
                       *psz=0;
                       cChanged++;
                       }
                 psz++;
                 }
           }
     return cChanged;
     }
```

```
/*
 * ErrorWithFile
 *
 * Purpose:
 *  Displays a message box built from a stringtable string containing
 *  one %s as a placeholder for a filename and from a string of the
 *  filename to place there.
 *
 * Parameters:
 *  hWnd            HWND owning the message box.  The caption of this
 *                  window is the caption of the message box.
 *  hInst           HINSTANCE from which to draw the idsErr string.
 *  idsErr          UINT identifier of a stringtable string containing
 *                  the error message with a %s.
 *  lpszFile        LPSTR to the filename to include in the message.
 *  uFlags          UINT flags to pass to MessageBox, like MB_OK.
 *
 * Return Value:
 *  int             Return value from MessageBox.
 */

int WINAPI ErrorWithFile(HWND hWnd, HINSTANCE hInst, UINT idsErr
                         , LPSTR pszFile, UINT uFlags)
     {
     int             iRet=0;
     HANDLE          hMem;
     const UINT      cb=(2*OLEUI_CCHPATHMAX);
     LPSTR           psz1, psz2, psz3;

     if (NULL==hInst || NULL==pszFile)
           return iRet;

     //Allocate three 2*OLEUI_CCHPATHMAX byte work buffers
     hMem=GlobalAlloc(GHND, (DWORD)(3*cb));

     if (NULL==hMem)
           return iRet;

     psz1=GlobalLock(hMem);
     psz2=psz1+cb;
     psz3=psz2+cb;

     if (0!=LoadString(hInst, idsErr, psz1, cb))
           {
           wsprintf(psz2, psz1, pszFile);

           //Steal the caption of the dialog
           GetWindowText(hWnd, psz3, cb);
           iRet=MessageBox(hWnd, psz2, psz3, uFlags);
           }
     GlobalUnlock(hMem);
     GlobalFree(hMem);
     return iRet;
     }
```

```
/*
 * HIconFromClass
 *
 * Purpose:
 *  Given an object class name, finds an associated executable in the
 *  registration database and extracts the first icon from that
 *  executable.  If none is available or the class has no associated
 *  executable, this function returns NULL.
 *
 * Parameters:
 *  pszClass        LPSTR giving the object class to look up.
 *
 * Return Value:
 *  HICON           Handle to the extracted icon if there is a module
 *                  associated to pszClass.  NULL on failure to either
 *                  find the executable or extract and icon.
 */

HICON WINAPI HIconFromClass(LPSTR pszClass)
     {
     HICON           hIcon;
     char            szEXE[OLEUI_CCHPATHMAX];
     UINT            Index;
     CLSID           clsid;

     if (NULL==pszClass)
          return NULL;

     CLSIDFromString(pszClass, &clsid);

     if (!FIconFileFromClass((REFCLSID)&clsid, szEXE, sizeof(szEXE),
&Index))
          return NULL;

     hIcon=ExtractIcon(ghInst, szEXE, Index);

     if ((HICON)32 > hIcon)
          hIcon=NULL;

     return hIcon;
     }
```

```
/*
 * FServerFromClass
 *
 * Purpose:
 *  Looks up the classname in the registration database and retrieves
 *  the name under CLSID\[Local|InProc]Server.
 *
 * Parameters:
 *  pszClass        LPSTR to the classname to look up.
 *  pszEXE          LPSTR at which to store the server name
 *  cch             UINT size of pszEXE
 *
 * Return Value:
 *  BOOL            TRUE if one or more characters were loaded into pszEXE.
 *                  FALSE otherwise.
 */

BOOL WINAPI FServerFromClass(LPSTR pszClass, LPSTR pszEXE, UINT cch)
{

     DWORD       dw;
     LONG        lRet;
     HKEY        hKey;

     if (NULL==pszClass || NULL==pszEXE || 0==cch)
           return FALSE;

     /*
      * We have to go walking in the registration database under the
      * classname, so we first open the classname key and then check
      * under "\\LocalServer" to get the .EXE.
      */

     //Open up the class key
     lRet=RegOpenKey(HKEY_CLASSES_ROOT, pszClass, &hKey);

     if ((LONG)ERROR_SUCCESS!=lRet)
           return FALSE;

     //Get the executable path.
     dw=(DWORD)cch;
#ifdef WIN32
   lRet=RegQueryValue(hKey, "LocalServer32", pszEXE, &dw);
#else
   lRet=RegQueryValue(hKey, "LocalServer", pszEXE, &dw);
#endif

    if ((LONG)ERROR_SUCCESS!=lRet)
        {
        //Try InprocServer
```

```
#ifdef WIN32
        lRet=RegQueryValue(hKey, "InProcServer32", pszEXE, &dw);
#else
        lRet=RegQueryValue(hKey, "InProcServer32", pszEXE, &dw);
#endif
        }

     RegCloseKey(hKey);

     return ((ERROR_SUCCESS == lRet) && (dw > 0));
}



/*
 * UClassFromDescription
 *
 * Purpose:
 *  Looks up the actual OLE class name in the registration database
 *  for the given descriptive name chosen from a listbox.
 *
 * Parameters:
 *  psz             LPSTR to the descriptive name.
 *  pszClass        LPSTR in which to store the class name.
 *  cb              UINT maximum length of pszClass.
 *
 * Return Value:
 *  UINT            Number of characters copied to pszClass.  0 on failure.
 */

UINT WINAPI UClassFromDescription(LPSTR psz, LPSTR pszClass, UINT cb)
     {
     DWORD           dw;
     HKEY            hKey;
     char            szClass[OLEUI_CCHKEYMAX];
     LONG            lRet;
     UINT            i;

     //Open up the root key.
     lRet=RegOpenKey(HKEY_CLASSES_ROOT, NULL, &hKey);

     if ((LONG)ERROR_SUCCESS!=lRet)
           return 0;

     i=0;
     lRet=RegEnumKey(hKey, i++, szClass, OLEUI_CCHKEYMAX);

     //Walk the available keys
     while ((LONG)ERROR_SUCCESS==lRet)
           {
           dw=(DWORD)cb;
           lRet=RegQueryValue(hKey, szClass, pszClass, &dw);

           //Check if the description matches the one just enumerated
           if ((LONG)ERROR_SUCCESS==lRet)
```

```
                 {
                 if (!lstrcmp(pszClass, psz))
                       break;
                 }

           //Continue with the next key.
           lRet=RegEnumKey(hKey, i++, szClass, OLEUI_CCHKEYMAX);
           }

     //If we found it, copy to the return buffer
     if ((LONG)ERROR_SUCCESS==lRet)
           lstrcpy(pszClass, szClass);
     else
           dw=0L;

     RegCloseKey(hKey);
     return (UINT)dw;
     }
```

```
/*
 * UDescriptionFromClass
 *
 * Purpose:
 *  Looks up the actual OLE descriptive name name in the registration
 *  database for the given class name.
 *
 * Parameters:
 *  pszClass        LPSTR to the class name.
 *  psz             LPSTR in which to store the descriptive name.
 *  cb              UINT maximum length of psz.
 *
 * Return Value:
 *  UINT            Number of characters copied to pszClass.  0 on failure.
 */

UINT WINAPI UDescriptionFromClass(LPSTR pszClass, LPSTR psz, UINT cb)
     {
     DWORD           dw;
     HKEY            hKey;
     LONG            lRet;

     if (NULL==pszClass || NULL==psz)
           return 0;

     //Open up the root key.
     lRet=RegOpenKey(HKEY_CLASSES_ROOT, NULL, &hKey);
     if ((LONG)ERROR_SUCCESS!=lRet)
           return 0;

     //Get the descriptive name using the class name.
     dw=(DWORD)cb;
     lRet=RegQueryValue(hKey, pszClass, psz, &dw);

     RegCloseKey(hKey);

     psz+=lstrlen(psz)+1;
     *psz=0;

     if ((LONG)ERROR_SUCCESS!=lRet)
           return 0;

     return (UINT)dw;
     }



// returns width of line of text. this is a support routine for ChopText
static LONG GetTextWSize(HDC hDC, LPSTR lpsz)
{
     SIZE size;

     if (GetTextExtentPoint(hDC, lpsz, lstrlen(lpsz), (LPSIZE)&size))
           return size.cx;
     else {
           return 0;
     }
}


/*
 * ChopText
 *
 * Purpose:
 *  Parse a string (pathname) and convert it to be within a specified
 *  length by chopping the least significant part
 *
 * Parameters:
 *  hWnd            window handle in which the string resides
 *  nWidth          max width of string in pixels
 *                  use width of hWnd if zero
 *  lpch            pointer to beginning of the string
 *
 * Return Value:
 *  pointer to the modified string
 */
LPSTR WINAPI ChopText(HWND hWnd, int nWidth, LPSTR lpch)
{
#define PREFIX_SIZE    7 + 1
#define PREFIX_FORMAT "%c%c%c...\\"

     char    szPrefix[PREFIX_SIZE];
     BOOL    fDone = FALSE;
     int     i;
     RECT    rc;
     HDC     hdc;
     HFONT   hfont;
     HFONT   hfontOld = NULL;
     SIZE  size;

     if (!hWnd || !lpch)
           return NULL;

     /* Get length of static field. */
     if (!nWidth)
           {
           GetClientRect(hWnd, (LPRECT)&rc);
           nWidth = rc.right - rc.left;
           }

     /* Set up DC appropriately for the static control */
     hdc = GetDC(hWnd);
     hfont = (HFONT)SendMessage(hWnd, WM_GETFONT, 0, 0L);

     if (NULL != hfont)   // WM_GETFONT returns NULL if window uses system
font
           hfontOld = SelectObject(hdc, hfont);

     /* check horizontal extent of string */
     GetTextExtentPoint(hdc, lpch, lstrlen(lpch), &size);
     if (size.cx > nWidth)
           {

           /* string is too long to fit in static control; chop it */
           /* set up new prefix & determine remaining space in control */
           wsprintf((LPSTR) szPrefix, PREFIX_FORMAT, lpch[0], lpch[1],
lpch[2]);
           GetTextExtentPoint(hdc, (LPSTR)szPrefix, lstrlen(szPrefix),
&size);
           nWidth -= size.cx;

           /*
           ** advance a directory at a time until the remainder of the
           ** string fits into the static control after the "x:\...\" prefix
           */
           while (!fDone)
                 {

#ifdef DBCS
                 while (*lpch && (*lpch != '\\'))
                       lpch = AnsiNext(lpch);
                 if (*lpch)
                       lpch = AnsiNext(lpch);
#else
                 while (*lpch && (*lpch++ != '\\'));
#endif
```

```
                 if (*lpch)
                       GetTextExtentPoint(hdc, lpch, lstrlen(lpch), &size);

                 if (!*lpch || size.cx <= nWidth)
                       {
                       if (!*lpch)
                             /*
                             ** Nothing could fit after the prefix; remove
the
                             ** final "\" from the prefix
                             */
                             szPrefix[lstrlen((LPSTR) szPrefix) - 1] = 0;

                             /* rest or string fits -- stick prefix on front
*/
                             for (i = lstrlen((LPSTR) szPrefix) - 1; i >= 0;
--i)
                                   *--lpch = szPrefix[i];
                             fDone = TRUE;
                       }
                 }
           }

     if (NULL != hfont)
           SelectObject(hdc, hfontOld);
     ReleaseDC(hWnd, hdc);

     return(lpch);

#undef PREFIX_SIZE
#undef PREFIX_FORMAT
}


/*
 * OpenFileError
 *
 * Purpose:
 *  display message for error returned from OpenFile
 *
 * Parameters:
 *  hDlg            HWND of the dialog.
 *  nErrCode        UINT error code returned in OFSTRUCT passed to OpenFile
 *  lpszFile        LPSTR file name passed to OpenFile
 *
 * Return Value:
 *  None
 */
void WINAPI OpenFileError(HWND hDlg, UINT nErrCode, LPSTR lpszFile)
{
     switch (nErrCode) {
           case 0x0005:    // Access denied
                 ErrorWithFile(hDlg, ghInst, (UINT)IDS_CIFILEACCESS,
lpszFile, MB_ICONEXCLAMATION | MB_OK);
```

```
                break;

           case 0x0020:    // Sharing violation
                ErrorWithFile(hDlg, ghInst, (UINT)IDS_CIFILESHARE,
lpszFile, MB_ICONEXCLAMATION | MB_OK);
                break;

           case 0x0002:    // File not found
           case 0x0003:    // Path not found
                ErrorWithFile(hDlg, ghInst, (UINT)IDS_CIINVALIDFILE,
lpszFile, MB_ICONEXCLAMATION | MB_OK);
                break;

           default:
                ErrorWithFile(hDlg, ghInst, (UINT)IDS_CIFILEOPENFAIL,
lpszFile, MB_ICONEXCLAMATION | MB_OK);
                break;
     }
}

#define chSpace        ' '
#define chPeriod       '.'
#define PARSE_EMPTYSTRING   -1
#define PARSE_INVALIDDRIVE  -2
#define PARSE_INVALIDPERIOD -3
#define PARSE_INVALIDDIRCHAR    -4
#define PARSE_INVALIDCHAR   -5
#define PARSE_WILDCARDINDIR -6
#define PARSE_INVALIDNETPATH    -7
#define PARSE_INVALIDSPACE  -8
#define PARSE_EXTENTIONTOOLONG  -9
#define PARSE_DIRECTORYNAME -10
#define PARSE_FILETOOLONG   -11

/
*----------------------------------------------------------------------------
 * ParseFile
 * Purpose:  Determine if the filename is a legal DOS name
 * Input:    Long pointer to a SINGLE file name
 *           Circumstance checked:
 *           1) Valid as directory name, but not as file name
 *           2) Empty String
 *           3) Illegal Drive label
 *           4) Period in invalid location (in extention, 1st in file name)
 *           5) Missing directory character
 *           6) Illegal character
 *           7) Wildcard in directory name
 *           8) Double slash beyond 1st 2 characters
 *           9) Space character in the middle of the name (trailing spaces
OK)
 *          10) Filename greater than 8 characters
 *          11) Extention greater than 3 characters
 * Notes:
 *   Filename length is NOT checked.
 *   Valid filenames will have leading spaces, trailing spaces and
 *     terminating period stripped in place.
 *
 * Returns:  If valid, LOWORD is byte offset to filename
 *                     HIWORD is byte offset to extention
 *                           if string ends with period, 0
 *                           if no extention is given, string length
 *           If invalid, LOWORD is error code suggesting problem (< 0)
 *                       HIWORD is approximate offset where problem found
 *                       Note that this may be beyond the offending
character
 *----------------------------------------------------------------------------
*/

static long ParseFile(LPSTR lpstrFileName)
{
  short nFile, nExt, nFileOffset, nExtOffset;
  BOOL bExt;
  BOOL bWildcard;
  short nNetwork = 0;
  BOOL  bUNCPath = FALSE;
  LPSTR lpstr = lpstrFileName;

/* Strip off initial white space.  Note that TAB is not checked */
/* because it cannot be received out of a standard edit control */
/* 30 January 1991  clarkc                                      */
  while (*lpstr == chSpace)
       lpstr++;


  if (!*lpstr)
     {
       nFileOffset = PARSE_EMPTYSTRING;
       goto FAILURE;
     }

  if (lpstr != lpstrFileName)
     {
       lstrcpy(lpstrFileName, lpstr);
       lpstr = lpstrFileName;
     }

  if (*AnsiNext(lpstr) == ':')
     {
       char cDrive = (*lpstr | (BYTE) 0x20);  /* make lowercase */

/* This does not test if the drive exists, only if it's legal */
       if ((cDrive < 'a') || (cDrive > 'z'))
           {
             nFileOffset = PARSE_INVALIDDRIVE;
             goto FAILURE;
           }
       lpstr = AnsiNext(AnsiNext(lpstr));
     }

  if ((*lpstr == '\\') || (*lpstr == '/'))
```

```
     {
       if (*++lpstr == chPeriod)                /* cannot have c:\. */
           {
             if ((*++lpstr != '\\') && (*lpstr != '/'))   /* unless it's
stupid */
                 {
                   if (!*lpstr)        /* it's the root directory */
                         goto MustBeDir;

                   nFileOffset = PARSE_INVALIDPERIOD;
                   goto FAILURE;
                 }
             else
                   ++lpstr;   /* it's saying top directory (again), thus
allowed */
           }
       else if ((*lpstr == '\\') && (*(lpstr-1) == '\\'))
           {
/* It seems that for a full network path, whether a drive is declared or
 * not is insignificant, though if a drive is given, it must be valid
 * (hence the code above should remain there).
 * 13 February 1991           clarkc
 */
             ++lpstr;            /* ...since it's the first slash, 2 are
allowed */
             nNetwork = -1;      /* Must receive server and share to be real
*/
             bUNCPath = TRUE;    /* No wildcards allowed if UNC name
*/
           }
       else if (*lpstr == '/')
           {
             nFileOffset = PARSE_INVALIDDIRCHAR;
             goto FAILURE;
           }
     }
  else if (*lpstr == chPeriod)
     {
       if (*++lpstr == chPeriod)  /* Is this up one directory? */
             ++lpstr;
       if (!*lpstr)
             goto MustBeDir;
       if ((*lpstr != '\\') && (*lpstr != '/'))
           {
             nFileOffset = PARSE_INVALIDPERIOD;
             goto FAILURE;
           }
       else
             ++lpstr;   /* it's saying directory, thus allowed */
     }

  if (!*lpstr)
     {
       goto MustBeDir;
     }
```

```
/* Should point to first char in 8.3 filename by now */
  nFileOffset = nExtOffset = nFile = nExt = 0;
  bWildcard = bExt = FALSE;
  while (*lpstr)
     {
/*
 *  The next comparison MUST be unsigned to allow for extended characters!
 *  21 Feb 1991   clarkc
 */
       if (*lpstr < chSpace)
           {
             nFileOffset = PARSE_INVALIDCHAR;
             goto FAILURE;
           }
       switch (*lpstr)
           {
             case '"':             /* All invalid */
             case '+':
             case ',':
             case ':':
             case ';':
             case '<':
             case '=':
             case '>':
             case '[':
             case ']':
             case '|':
                 {
                   nFileOffset = PARSE_INVALIDCHAR;
                   goto FAILURE;
                 }

             case '\\':      /* Subdirectory indicators */
             case '/':
                 nNetwork++;
                 if (bWildcard)
                   {
                       nFileOffset = PARSE_WILDCARDINDIR;
                       goto FAILURE;
                   }

                 else if (nFile == 0)        /* can't have 2 in a row */
                   {
                       nFileOffset = PARSE_INVALIDDIRCHAR;
                       goto FAILURE;
                   }
                 else
                   {                         /* reset flags */
                       ++lpstr;
                       if (!nNetwork && !*lpstr)
                         {
                             nFileOffset = PARSE_INVALIDNETPATH;
                             goto FAILURE;
                         }
                       nFile = nExt = 0;
                       bExt = FALSE;
                   }

                 break;

             case chSpace:
                 {
                   LPSTR lpSpace = lpstr;

                   *lpSpace = '\0';
                   while (*++lpSpace)
                       {
                         if (*lpSpace != chSpace)
                             {
                               *lpstr = chSpace;        /* Reset string,
abandon ship */
                               nFileOffset = PARSE_INVALIDSPACE;
                               goto FAILURE;
                             }
                       }
                 }
                 break;

             case chPeriod:
                 if (nFile == 0)
                   {
                       if (*++lpstr == chPeriod)
                             ++lpstr;
                       if (!*lpstr)
                             goto MustBeDir;

                       if ((*lpstr != '\\') && (*lpstr != '/'))
                         {
                             nFileOffset = PARSE_INVALIDPERIOD;
                             goto FAILURE;
                         }

                       ++lpstr;              /* Flags are already set */
                   }
                 else if (bExt)
                   {
                       nFileOffset = PARSE_INVALIDPERIOD;  /* can't have one
in ext */
                       goto FAILURE;
                   }
                 else
                   {
                       nExtOffset = 0;
                       ++lpstr;
                       bExt = TRUE;
                   }
                 break;

             case '*':
```

```
             case '?':
                 if (bUNCPath)
                   {
                       nFileOffset = PARSE_INVALIDNETPATH;
                       goto FAILURE;
                   }
                 bWildcard = TRUE;
/* Fall through to normal character processing */

             default:
                 if (bExt)
                   {
                       if (++nExt == 1)
                             nExtOffset = lpstr - lpstrFileName;
                       else if (nExt > 3)
                         {
                             nFileOffset = PARSE_EXTENTIONTOOLONG;
                             goto FAILURE;
                         }
                       if ((nNetwork == -1) && (nFile + nExt > 11))
                         {
                             nFileOffset = PARSE_INVALIDNETPATH;
                             goto FAILURE;
                         }
                   }
                 else if (++nFile == 1)
                       nFileOffset = lpstr - lpstrFileName;
                 else if (nFile > 8)
                   {
                       /* If it's a server name, it can have 11 characters
*/
                       if (nNetwork != -1)
                         {
                             nFileOffset = PARSE_FILETOOLONG;
                             goto FAILURE;
                         }
                       else if (nFile > 11)
                         {
                             nFileOffset = PARSE_INVALIDNETPATH;
                             goto FAILURE;
                         }
                   }

                 lpstr = AnsiNext(lpstr);
                 break;
           }
     }

/* Did we start with a double backslash but not have any more slashes? */
  if (nNetwork == -1)
     {
       nFileOffset = PARSE_INVALIDNETPATH;
       goto FAILURE;
     }
  if (!nFile)
      {
MustBeDir:
       nFileOffset = PARSE_DIRECTORYNAME;
       goto FAILURE;
      }

  if ((*(lpstr - 1) == chPeriod) &&          /* if true, no extention wanted
*/
                   (*AnsiNext(lpstr-2) == chPeriod))
       *(lpstr - 1) = '\0';               /* Remove terminating period   */
  else if (!nExt)
FAILURE:
       nExtOffset = lpstr - lpstrFileName;

  return(MAKELONG(nFileOffset, nExtOffset));
}


/*
 * DoesFileExist
 *
 * Purpose:
 *  Determines if a file path exists
 *
 * Parameters:
 *  lpszFile        LPSTR - file name
 *  lpOpenBuf       OFSTRUCT FAR* - points to the OFSTRUCT structure that
 *                      will receive information about the file when the
 *                      file is first opened. this field is filled by the
 *                      Windows OpenFile API.
 *
 * Return Value:
 *  HFILE   HFILE_ERROR - file does NOT exist
 *          file handle (as returned from OpenFile) - file exists
 */
HFILE WINAPI DoesFileExist(LPSTR lpszFile, OFSTRUCT FAR* lpOpenBuf)
{
     long        nRet;
     int         i;
     static char *arrIllegalNames[] = {
           "LPT1",
           "LPT2",
           "LPT3",
           "COM1",
           "COM2",
           "COM3",
           "COM4",
           "CON",
           "AUX",
           "PRN"
     };

     // Check if file name is syntactically correct.
     //   (OpenFile sometimes crashes if path is not syntactically correct)
     nRet = ParseFile(lpszFile);
     if (LOWORD(nRet) < 0)
           goto error;

     // Check is the name is an illegal name (eg. the name of a device)
     for (i=0; i < (sizeof(arrIllegalNames)/sizeof(arrIllegalNames[0])); i+
+) {
           if (lstrcmpi(lpszFile, arrIllegalNames[i])==0)
                 goto error; // illegal name FOUND
     }
     return OpenFile(lpszFile, lpOpenBuf, OF_EXIST);

error:
     _fmemset(lpOpenBuf, 0, sizeof(OFSTRUCT));
     lpOpenBuf->nErrCode = 0x0002;   // File not found
     return HFILE_ERROR;
}
```

WN_DOS.H (WRAPUI Sample)

```
/*************************************************************************
**
**    OLE 2.0 Property Set Utilities
**
**    wn_dos.h
**
**    This file contains file contains data structure defintions,
**    function prototypes, constants, etc. for Windows 3.x form of
**    DOS calls. This is used by the SUMINFO OLE 2.0 Property Set
**    utilities used to manage the Summary Info property set.
**
**    (c) Copyright Microsoft Corp. 1990 - 1992 All Rights Reserved
**
*************************************************************************/

#ifndef WN_DOS_H
#define WN_DOS_H

#include <dos.h>

#define WIN 1

#define cbMaxFile 146 //from inc\path.h
#define SEEK_FROM_BEGINNING 0
#define SEEK_FROM_END 2
#define chDOSPath ('\\')        // FUTURE: not used all places it could be
#define chDOSWildAll    '*'     /* DOS File name wild card. */
#define chDOSWildSingle '?'



// Close, seek, delete, rename, flush, get attributes, read, write
/* RPC TEMP
int  FCloseOsfnWin(WORD);
#define FCloseOsfn(osfn)    FCloseOsfnWin(osfn)
long DwSeekDwWin(WORD,LONG,WORD);
#define DwSeekDw(osfn, dwSeek, bSeekFrom)   DwSeekDwWin(osfn, dwSeek,
bSeekFrom)
EC  EcDeleteSzFfnameWin(char *);
#define EcDeleteSzFfname(szFile) EcDeleteSzFfnameWin(szFile)
EC  EcRenameSzFfnameWin(char *,char *);
#define EcRenameSzFfname(szFileCur,szFileNew)
EcRenameSzFfnameWin(szFileCur,szFileNew)
int FFlushOsfnWin(int);
#define FFlushOsfn(osfn)    FFlushOsfnWin(osfn)
WORD DaGetFileModeSzWin(char *);
#define DaGetFileModeSz(szFile) DaGetFileModeSzWin(szFile)
int  CbReadOsfnWin(int, void far *, UINT);
int  CbWriteOsfnWin(int, void far *, UINT);
#define CbWriteOsfn(osfn,lpch,cbWrite)  CbWriteOsfnWin(osfn,lpch,cbWrite)
*/
#define WinOpenFile(sz,ofs,n)   OpenFile(sz,ofs,n)
```

```
#define SeekHfile(f,off,kind) _llseek(f,off,kind)
#define CbReadOsfn(osfn,lpch,cbRead)    CbReadOsfnWin(osfn,lpch,cbRead)
#define CbReadHfile(f,buf,n) _lread(f,buf,n)
#define CbReadOsfnWin(f,buf,n) CbReadHfile(f,buf,n)
#define EcFindFirst4dm(a,b,c) _dos_findfirst((const char *)(b),c,(struct
find_t*)a)
#define EcFindNext4dm(a) _dos_findnext((struct find_t*)a)
#define FHfileToSffsDate(handle,date,time) _dos_getftime(handle, (unsigned
*)(date), (unsigned *)(time))
#define SeekHfile(f, off, kind) _llseek(f,off,kind)

/* buffer structure to be used with EcFindFirst() and EcFindNext() */
typedef struct _SFFS
     { /* Search Find File Structure */
     uchar buff[21]; // dos search info
     uchar wAttr;
     union
           {
           unsigned short timeVariable;    /*RPC47*/
           BF time:16;
           struct
                 {
                 BF sec : 5;
                 BF mint: 6;
                 BF hr  : 5;
                 };
           };
     union
           {
           unsigned short dateVariable;
           BF date:16;
           struct
                 {
                 BF dom : 5;
                 BF mon : 4;
                 BF yr  : 7;
                 };
           };
     ulong cbFile;
     uchar szFileName[13];
     } SFFS;

// find first file/find next file
#define PszFromPsffs(psffs)     ((psffs)->szFileName)
#define CopySzFilePsffs(psffs,sz)   OemToAnsi((char HUGE *)&((psffs)-
>szFileName[0]),(char HUGE *)(sz))
#define CbSzFilePsffs(psffs)    CbSz((psffs)->szFileName)
#define CbFileSizePsffs(psffs)  (psffs)->cbFile
#define AttribPsffs(psffs)      (psffs)->wAttr
#define EcFindFirstCore(psffs, sz, wAttr) EcFindFirst(psffs, sz, wAttr)
/*RPC22*/
#define FDotPsffs(psffs) ((psffs)->szFileName[0]=='.')   /*RPC23*/
#define AppendSzWild(sz) {int i=_fstrlen((char FAR *)(sz)); sz[i]='*';
sz[i+1]='.'; sz[i+2]='*'; sz[i+3]='\0';}
// disk free space
```

```
unsigned long LcbDiskFreeSpaceWin(int);
#define LcbDiskFreeSpace(chDrive) LcbDiskFreeSpaceWin(chDrive)

// date and time    /*RPC39*/
/*
typedef struct _TIM {                    // Time structure returned by
OsTime
     CHAR minutes, hour, hsec, sec;
     } TIM;

typedef struct _DAT {                    // Date structure returned by
OsDate
     int  year;
     CHAR month, day, dayOfWeek;
     } DAT;
*/
#define TIM dostime_t    /*RPC39*/
#define DAT dosdate_t
#define OsTimeWin(TIM) _dos_gettime(TIM)
#define OsDateWin(DAT) _dos_getdate(DAT)



/* DOS File Attributes */
#define DA_NORMAL       0x00
#define DA_READONLY     0x01
#define DA_HIDDEN       0x02
#define DA_SYSTEM       0x04
#define DA_VOLUME       0x08
#define DA_SUBDIR       0x10
#define DA_ARCHIVE      0x20
#define DA_NIL          0xFFFF  /* Error DA */
#define dosxSharing     32      /* Extended error code for sharing viol. */
#define nErrNoAcc       5       /* OpenFile error code for Access Denied */
#define nErrFnf         2       /* OpenFile error code for File Not Found */

/* Components of the Open mode for OpenSzFfname (DOS FUNC 3DH) */
#define MASK_fINH       0x80
#define MASK_bSHARE     0x70
#define MASK_bACCESS    0x07

#define bSHARE_DENYRDWR 0x10
#define bSHARE_DENYWR   0x20
#define bSHARE_DENYNONE 0x40

/* Seek-from type codes passed to DOS function 42H */

#define SF_BEGINNING    0       /* Seek from beginning of file */
#define SF_CURRENT      1       /* Seek from current file pointer */
#define SF_END          2       /* Seek from end of file */


typedef struct _DOSDTTM /* DOS DaTe TiMe */
           {
```

```
           union
                 {
                 long lDOSDttm;
                 struct
                       {
                       BF day: 5;
                       BF month:   4;
                       BF year:    7;
                       BF sec: 5;
                       BF mint:    6;
                       BF hours:   5;
                       } S1;
                 } U1;
           } DOSDTTM;

int  FOsfnIsFile(int);

void DateStamp(int, LONG *,  int);
int  DosxError(void);
int  ShellExec(int, int);

#endif //WN_DOS_H
```

ADVF

The **ADVF** enumeration values are flags used by a container object to specify the requested behavior when setting up an advise sink or a caching connection with an object. These values have different meanings depending on the type of connection in which they are used, and each interface uses its own subset of the flags.

```
typedef enum tagADVF
{
     ADVF_NODATA              = 1,
     ADVF_PRIMEFIRST          = 4,
     ADVF_ONLYONCE            = 2,
     ADVF_DATAONSTOP          = 64,
     ADVFCACHE_NOHANDLER      = 8,
     ADVFCACHE_FORCEBUILTIN = 16,
     ADVFCACHE_ONSAVE         = 32
} ADVF;
```

Elements

ADVF_NODATA

For data advisory connections (**IDataObject::DAdvise** or **IDataAdviseHolder::Advise**), this flag requests the data object not to send data when it calls **IAdviseSink::OnDataChange**. The recipient of the change notification can later request the data by calling **IDataObject::GetData**. The data object can honor the request by passing TYMED_NULL in the **STGMEDIUM** parameter, or it can provide the data anyway. For example, the data object might have multiple advisory connections, not all of which specified ADVF_NODATA, in which case the object might send the same notification to all connections. Regardless of the container's request, its **IAdviseSink** implementation must check the **STGMEDIUM** parameter because it is responsible for releasing the medium if it is not TYMED_NULL.

For cache connections (**IOleCache::Cache**), this flag requests that the cache not be updated by changes made to the running object. Instead, the container will update the cache by explicitly calling **IOleCache::SetData**. This situation typically occurs when the iconic aspect of an object is being cached.

ADVF_NODATA is not a valid flag for view advisory connections (**IViewObject::SetAdvise**) and it returns E_INVALIDARG.

ADVF_PRIMEFIRST

Requests that the object not wait for the data or view to change before making an initial call to **IAdviseSink::OnDataChange** (for data or view advisory connections) or updating the cache (for cache connections). Used with ADVF_ONLYONCE, this parameter provides an asynchronous **GetData** call.

ADVF_ONLYONCE

Requests that the object make only one change notification or cache update before deleting the connection.

ADVF_ONLYONCE automatically deletes the advisory connection after sending one data or view notification. The advisory sink receives only one **IAdviseSink** call. A nonzero connection ID is returned if the connection is established so the caller can use it to delete the connection prior to the first change notification.

For data change notifications, the combination of ADVF_ONLYONCE and ADVF_PRIMEFIRST provides, in effect, an asynchronous **IDataObject::GetData** call.

When used with caching, ADVF_ONLYONCE updates the cache one time only, on receipt of the first OnDataChange notification. After the update is complete, the advisory connection between the object and the cache is disconnected. The source object for the advisory connection calls the

IAdviseSink::Release method.

ADVF_DATAONSTOP

For data advisory connections, assures accessibility to data. This flag indicates that when the data object is closing, it should call **IAdviseSink::OnDataChange** providing data with the call. Typically, this value is used in combination with ADVF_NODATA. Without this value, by the time an **OnDataChange** call without data reaches the sink, the source might have completed its shutdown and the data might not be accessible. Sinks that specify this value should accept data provided in **OnDataChange** if it is being passed, because they may not get another chance to retrieve it.

For cache connections, this flag indicates that the object should update the cache as part of object closure.

ADVF_DATAONSTOP is not a valid flag for view advisory connections.

ADVFCACHE_NOHANDLER

Synonym for ADVFCACHE_FORCEBUILTIN, which is used more often.

ADVFCACHE_FORCEBUILTIN

This value is used by DLL object applications and object handlers that perform the drawing of their objects. ADVFCACHE_FORCEBUILTIN instructs OLE to cache presentation data to ensure that there is a presentation in the cache. This value is not a valid flag for data or view advisory connections. For cache connections, this flag caches data that requires only code shipped with OLE (or the underlying operating system) to be present in order to produce it with **IDataObject::GetData** or **IViewObject::Draw**. By specifying this value, the container can ensure that the data can be retrieved even when the object or handler code is not available.

ADVFCACHE_ONSAVE

For cache connections, this flag updates the cached representation only when the object containing the cache is saved. The cache is also updated when the OLE object transitions from the running state back to the loaded state (because a subsequent save operation would require rerunning the object). This value is not a valid flag for data or view advisory connections.

Comments

For a data or view advisory connection, the container uses the **ADVF** constants when setting up a connection between an **IAdviseSink** instance and and either an **IDataObject** or **IViewObject** instance. These connections are set up using the **IDataObject::DAdvise**, **IDataAdviseHolder::Advise**, or **IViewObject::SetAdvise** methods.

For a caching connection, the constants are specified in the **IOleCache::Cache** method to indicate the container's requests on how the object should update its cache.

These constants are also used in the *advf* member of the **STATDATA** structure. This structure is used by **IEnumSTATDATA** to describe the enumerated connections, and the *advf* member indicates the flags that were specified when the advisory or cache connection was established. When **STATDATA** is used for an **IOleObject::EnumAdvise** enumerator, the *advf* member is indeterminate.

See Also

IDataAdviseHolder, **IDataObject**, **IEnumSTATDATA**, **IOleCache**, **IViewObject**

BIND_FLAGS

The **BIND_FLAGS** enumeration values are used to control aspects of moniker binding operations. The values are used in the **BIND_OPTS** structure . Callers of **IMoniker** methods can specify values from this enumeration, and implementors of **IMoniker** methods can use these values in determining what they should do.

The **BIND_FLAGS** enumeration is defined in OBJIDL.IDL.

```
typedef enum tagBIND_FLAGS
{
     BIND_MAYBOTHERUSER     = 1,
     BIND_JUSTTESTEXISTENCE = 2,
} BIND_FLAGS;
```

Elements

BIND_MAYBOTHERUSER

If this flag is specified, the moniker implementation can interact with the end user. If not present, the moniker implementation should not interact with the user in any way, such as by asking for a password for a network volume that needs mounting. If prohibited from interacting with the user when it otherwise would, a moniker implementation can use a different algorithm that does not require user interaction, or it can fail with the error MK_MUSTBOTHERUSER.

BIND_JUSTTESTEXISTENCE

If this flag is specified, the caller is not interested in having the operation carried out, but only in learning whether the operation could have been carried out had this flag not been specified. For example, this flag lets the caller indicate only an interest in finding out whether an object actually exists by using this flag in a **IMoniker::BindToObject** call. Moniker implementations can, however, ignore this possible optimization and carry out the operation in full. Callers must be able to deal with both cases.

See Also

BIND_OPTS, **IBindCtx**

BINDSPEED

The **BINDSPEED** enumeration values indicate approximately how long the caller will wait to bind to an object. Callers of the **IOleItemContainer::GetObject** method specify values from this enumeration, and implementors of that method use these values as a guideline for how quickly they must complete their operation.

The **BINDSPEED** enumeration is defined in OLEIDL.IDL.

```
typedef enum tagBINDSPEED
{
     BINDSPEED_INDEFINITE   = 1,
     BINDSPEED_MODERATE     = 2,
     BINDSPEED_IMMEDIATE    = 3
} BINDSPEED;
```

Elements

BINDSPEED_INDEFINITE
Indicates that there is no time limit on the binding operation.

BINDSPEED_MODERATE
Indicates that the **IOleItemContainer::GetObject** operation must be completed in a moderate amount of time. If this flag is specified, the implementation of **IOleItemContainer::GetObject** should return MK_E_EXCEEEDEDDEADLINE unless the object is one of the following:

- Already in the running state
- A pseudo-object (i.e., an object internal to the item container, such as a cell-range in a spreadsheet or a character-range in a word processor).
- An object supported by an in-process server (so it is always in the running state when it is loaded). In this case, **IOleItemContainer::GetObject** should load the designated object, and, if the **OleIsRunning** API function indicates that the object is running, return successfully.

BINDSPEED_IMMEDIATE
Indicates that the caller will wait only a short time. In this case, **IOleItemContainer::GetObject** should return MK_E_EXCEEEDEDDEADLINE unless the object is already in the running state or is a pseudo-object.

Comments

The system-supplied item moniker implementation is the primary caller of **IOleItemContainer::GetObject**. The **BINDSPEED** value that it specifies depends on the deadline specified by the caller of the moniker operation.

The deadline is stored in the *dwTickCountDeadline* field of the **BIND_OPTS** structure in the bind context passed to the moniker operation. This value is based on the return value of the **GetTickCount** API function. If *dwTickCountDeadline* is zero, indicating no deadline, the item moniker implementation specifies BINDSPEED_INDEFINITE. (This is the default *dwTickCountDeadline* value for a bind context returned by the **CreateBindCtx** API function.) If the difference between *dwTickCountDeadline* and the value returned by the **GetTickCount** API function is greater than 2500, the item moniker implementation specifies BINDSPEED_MODERATE. If the difference is less than 2500, the item moniker implementation specifies BINDSPEED_IMMEDIATE.

Implementations of **IOleItemContainer::GetObject** can use the **BINDSPEED** value as a shortcut approximation of the binding deadline, or they can use the **IBindCtx** instance parameter to determine the exact deadline.

See Also

BIND_OPTS, **IBindCtx::GetBindOptions**, **IOleItemContainer::GetObject**

CALLTYPE

The **CALLTYPE** enumeration constants are the call type used by **IMessageFilter::HandleInComingCall**.

Defined in the **IMessageFilter** interface (*msgflt.idl*).

```
typedef enum tagCALLTYPE
{
     CALLTYPE_TOPLEVEL             = 1,
     CALLTYPE_NESTED               = 2,
     CALLTYPE_ASYNC                = 3,
     CALLTYPE_TOPLEVEL_CALLPENDING = 4,
     CALLTYPE_ASYNC_CALLPENDING    = 5
} CALLTYPE;
```

Elements

CALLTYPE_TOPLEVEL
toplevel call - no outgoing call

CALLTYPE_NESTED
callback on behalf of previous outgoing call - should always handle

CALLTYPE_ASYNC
aysnchronous call - can NOT be rejected

CALLTYPE_TOPLEVEL_CALLPENDING
new toplevel call with new LID

CALLTYPE_ASYNC_CALLPENDING
async call - can NOT be rejected

See Also

IMessageFilter::HandleInComingCall, **IMessageFilter**

CLSCTX

The **CLSCTX** enumeration constants specify the execution context for a class object. Different pieces of code can be associated with one CLSID for use in different execution contexts.

```
typedef enum tagCLSCTX
{
     CLSCTX_INPROC_SERVER   = 1,
     CLSCTX_INPROC_HANDLER  = 2,
     CLSCTX_LOCAL_SERVER    = 4
} CLSCTX;
```

Elements

CLSCTX_INPROC_SERVER
The code that creates and manages objects of this class runs in the same process as the caller of the function specifying the class context.

CLSCTX_INPROC_HANDLER
The code that manages objects of this class is an in-process handler. This is a DLL that runs in the client process and implements client-side structures of this class when instances of the class are accessed remotely.

CLSCTX_LOCAL_SERVER
The EXE code that creates and manages objects of this class is loaded in a separate process space (runs on same machine but in a different process).

To indicate that multiple contexts are acceptable, you can string multiple values together with boolean ORs. The contexts are tried in the order in which they are listed.

The following table shows the use and reasons for CLSCTX values in calls to **CoGetClassObject**, which is used in the implementation of other OLE functions:

Function Called	**Context Flag Used**
OleLoad	CLSCTX_INPROC_HANDLER | CLSCTX_INPROC_SERVER Putting an OLE object into the loaded state requires in-process access; but, it doesn't matter if all of the object's function is presently available.
IRunnableObject::Run	CLSCTX_INPROC_SERVER | CLSCTX_LOCAL_SERVER Running an OLE object requires connecting to the full code of the object wherever it is located.
CoUnMarshalInterface	CLSCTX_INPROC_HANDLER Unmarshalling needs the form of the class designed for remote access.

See Also

CoCreateInstance, **CoGetClassObject**, **CoRegisterClassObject**

DATADIR

The **DATADIR** enumeration values specify the formats for which the **IDataObject::EnumFormatEtc** method supplies an enumerator. The values are used in the *dwDirection* parameter used in a call to the **IDataObject::EnumFormatEtc** method.

```
typedef enum tagDATADIR
{
     DATADIR_GET = 1,
     DATADIR_SET = 2
} DATADIR;
```

Elements

DATADIR_GET
: Requests that **IDataObject::EnumFormatEtc** supply an enumerator for the formats that can be specified in **IDataObject::GetData**.

DATADIR_SET
: Requests that **IDataObject::EnumFormatEtc** supply an enumerator for the formats that can be specified in **IDataObject::SetData**.

See Also

IDataObject

DISCARDCACHE

The **DISCARDCACHE** enumeration values are used in the **IOleCache2::DiscardCache** method to specify what to do with caches that are to be discarded from memory if their dirty bit has been set. The *dwDiscardOptions* parameter specifies whether or not to save these caches.

Defined in the

```
typedef enum tagDISCARDCACHE
{
     DISCARDCACHE_SAVEIFDIRTY    = 0,
     DISCARDCACHE_NOSAVE         = 1
} DISCARDCACHE;
```

Elements

DISCARDCACHE_SAVEIFDIRTY
  Indicates that the cache is to be saved to disk.

DISCARDCACHE_NOSAVE
  Indicates that the cache can be discarded without saving it.

See Also

IOleCache2::DiscardCache

DROPEFFECT

The **DoDragDrop** function and many of the methods in the **IDropSource** and **IDropTarget** interfaces pass information about the effects of a drag-and-drop operation in a **DROPEFFECT** enumeration.

Valid drop effect values are the result of applying the OR operation to the values contained in the **DROPEFFECT** enumeration:

```
typedef enum tagDROPEFFECT
{
     DROPEFFECT_NONE   = 0,
     DROPEFFECT_COPY   = 1,
     DROPEFFECT_MOVE   = 2,
     DROPEFFECT_LINK   = 4,
     DROPEFFECT_SCROLL = 0x80000000,
     }DROPEFFECT;
```

These values have the following meaning:

DROPEFFECT name	**Value**	**Description**
DROPEFFECT_NONE	0	Drop target cannot accept the data.
DROPEFFECT_COPY	1	Drop results in a copy. The original data is untouched by the drag source.
DROPEFFECT_MOVE	2	Drag source should remove the data.
DROPEFFECT_LINK	4	Drag source should create a link to the original data.
DROPEFFECT_SCROLL	0x80000000	Scrolling is about to start or is currently occurring in the target. This value is used in addition to the other values.

Note Your application should always mask values from the **DROPEFFECT** enumeration to ensure compatibility with future implementations. Presently, only four of the 32 bit positions in a **DROPEFFECT** have meaning. In the future, more interpretations for the bits will be added. Drag sources and drop targets should carefully mask these values appropriately before comparing. They should never compare a **DROPEFFECT** against, say, DROPEFFECT_COPY by

```
if (dwDropEffect == DROPEFFECT_COPY)...
```

Instead, the application should always mask for the value or values being sought:

```
if (dwDropEffect & DROPEFFECT_COPY) == DROPEFFECT_COPY)...
```

or

```
if (dwDropEffect & DROPEFFECT_COPY)...
```

This allows for the definition of new drop effects, while preserving backwards compatibility with existing code.

See Also

DoDragDrop, **IDropSource**, **IDropTarget**

DVASPECT

The **DVASPECT** enumeration values specify the desired data or view aspect of the object when drawing or getting data.

```
typedef enum tagDVASPECT
{
     DVASPECT_CONTENT    = 1,
     DVASPECT_THUMBNAIL  = 2,
     DVASPECT_ICON       = 4,
     DVASPECT_DOCPRINT   = 8
} DVASPECT;
```

Elements

DVASPECT_CONTENT
: Provide a representation of an object so it can be displayed as an embedded object inside of a container. This value is typically specified for compound document objects. The presentation can be provided for the screen or printer.

DVASPECT_THUMBNAIL
: Provide a thumbnail representation of an object so it can be displayed in a browsing tool. The thumbnail is approximately a 120 by 120 pixel, 16-color (recommended) device-independent bitmap potentially wrapped in a metafile.

DVASPECT_ICON
: Provide an iconic representation of an object.

DVASPECT_DOCPRINT
: Provide a representation of the object on the screen as though it were printed to a printer using the Print command from the File menu. The described data may represent a sequence of pages.

Comments

This enumeration is used in the *dwAspect* field of the **FORMATETC** structure. Only one **DVASPECT** value can be used to specify a single presentation aspect in a **FORMATETC** structure. The **FORMATETC** structure is used in **IDataObject**, **IAdviseSink**, and **IOleCache** methods. The **DVASPECT** enumeration values are also used in **IViewObject** methods.

See Also

IAdviseSink, **IDataObject**, **IOleObject**, **IViewObject**, **IViewObject2**, **OleDraw**, **FORMATETC**, **OBJECTDESCRIPTOR**

EXTCONN

The **EXTCONN** enumeration specifies the type of external connection existing on an embedded object. Currently, the only supported type is EXTCONN_STRONG, meaning that the external connection is a link. This E**XTCONN** constant is used in the **IExternalConnection::AddConnection** and **IExternalConnection::ReleaseConnection** methods.

```
typedef enum tagEXTCONN
{
     EXTCONN_STRONG    = 0X0001,
     EXTCONN_WEAK            = 0X0002,
     EXTCONN_CALLABLE  = 0X0004
} EXTCONN;
```

Elements

EXTCONN_STRONG
If this value is specified, the external connection must keep the object alive until all strong external connections are cleared through **IExternalConnection::ReleaseConnection**.

EXTCONN_WEAK
This value is currently not used.

EXTCONN_CALLABLE
This value is currently not used.

LOCKTYPE

The **LOCKTYPE** enumeration values indicate the type of locking requested for the specified range of bytes. The values are used in the **ILockBytes::LockRegion** and **IStream::LockRegion** methods.

Defined in the **IStream** interface (*stream.idl*).

```
typedef enum tagLOCKTYPE
{
     LOCK_WRITE       = 1,
     LOCK_EXCLUSIVE   = 2,
     LOCK_ONLYONCE    = 4
} LOCKTYPE;
```

Elements

LOCK_WRITE
: If this lock is granted, the specified range of bytes can be opened and read any number of times, but writing to the locked range is prohibited except for the owner that was granted this lock.

LOCK_EXCLUSIVE
: If this lock is granted, writing to the specified range of bytes is prohibited except for the owner that was granted this lock.

LOCK_ONLYONCE
: If this lock is granted, no other LOCK_ONLYONCE lock can be obtained on the range. Usually this lock type is an alias for some other lock type. Thus, specific implementations can have additional behavior associated with this lock type.

MKRREDUCE

The **MKRREDUCE** enumeration constants are used to specify how far the moniker should be reduced. They are used in the **IMoniker::Reduce** method. **MKRREDUCE** is defined in OBJIDL.IDL.

```
typedef enum tagMKRREDUCE
{
     MKRREDUCE_ONE           = 3<<16,
     MKRREDUCE_TOUSER        = 2<<16,
     MKRREDUCE_THROUGHUSER   = 1<<16,
     MKRREDUCE_ALL           = 0
} MKRREDUCE;
```

Elements

MKRREDUCE_ONE
Performs only one step of reducing the moniker. In general, the caller must have specific knowledge about the particular kind of moniker to take advantage of this option.

MKRREDUCE_TOUSER
Reduce the moniker to a form that the user identifies as a persistent object. If no such point exists, then this option should be treated as MKRREDUCE_ALL.

MKRREDUCE_THROUGHUSER
Reduce the moniker to where any further reduction would reduce it to a form that the user does not identify as a persistent object. Often, this is the same stage as MKRREDUCE_TOUSER.

MKRREDUCE_ALL
Reduce the moniker until it is in simplest form, that is, reduce it to itself.

See Also

IMoniker::Reduce

MKSYS

The **MKSYS** enumeration constants indicate the moniker's class. They are returned from the **IMoniker::IsSystemMoniker** method. **MKSYS** is defined in OBJIDL.IDL.

```
typedef enum tagMKSYS
{
     MKSYS_NONE                  = 0,
     MKSYS_GENERICCOMPOSITE        = 1,
     MKSYS_FILEMONIKER           = 2,
     MKSYS_ANTIMONIKER           = 3,
     MKSYS_ITEMMONIKER           = 4,
     MKSYS_POINTERMONIKER        = 5
} MKSYS;
```

Elements

MKSYS_NONE
  Indicates a custom moniker implementation.

MKSYS_GENERICCOMPOSITE
  Indicates the system's generic composite moniker class.

MKSYS_FILEMONIKER
  Indicates the system's file moniker class.

MKSYS_ANTIMONIKER
  Indicates the system's anti-moniker class.

MKSYS_ITEMMONIKER
  Indicates the system's item moniker class.

MKSYS_POINTERMONIKER
  Indicates the system's pointer moniker class.

See Also

IMoniker::IsSystemMoniker

MSHCTX

The **MSHCTX** enumeration constants specify the destination context, that is, the process in which the unmarshaling is to be done. These flags are used in the **IMarshal** and **IStdMarshalInfo** interfaces and in the **CoMarshalInterface** and **CoGetStandardMarshal** API functions.

MSHCTX is defined in WTYPES.IDL.

```
typedef enum tagMSHCTX
{
     MSHCTX_LOCAL                = 0,
     MSHCTX_NOSHAREDMEM          = 1,
     MSHCTX_DIFFERENTMACHINE     = 2,
     MSHCTX_INPROC               = 3
} MSHCTX;
```

Elements

MSHCTX_LOCAL
The unmarshaling process is local and has shared memory access with the marshaling process.

MSHCTX_NOSHAREDMEM
The unmarshaling process does not have shared memory access with the marshaling process.

MSHCTX_DIFFERENTMACHINE
The unmarshaling process is on a different machine. The marshaling code cannot assume that a particular piece of application code is installed on that machine.

MSHCTX_INPROC
The unmarshaling will be done in another apartment in the same process. If your object supports multiple threads, your custom marshaler can pass a direct pointer instead of creating a proxy object.

See Also

CoGetStandardMarshal, **CoMarshalInterface**, **IMarshal**, **IStdMarshalInfo**

MSHLFLAGS

The **MSHLFLAGS** enumeration constants determine why the marshaling is to be done. These flags are used in the **IMarshal** interface and the **CoMarshalInterface** and **CoGetStandardMarshal** API functions.

MSHFLAGS is defined in WTYPES.IDL.

```
typedef enum tagMSHLFLAGS
{
     MSHLFLAGS_NORMAL           = 0,
     MSHLFLAGS_TABLESTRONG      = 1,
     MSHLFLAGS_TABLEWEAK        = 2
} MSHLFLAGS;
```

Elements

MSHLFLAGS_NORMAL
: The marshaling is occurring because an interface pointer is being passed from one process to another. This is the normal case. The data packet produced by the marshaling process will be unmarshaled in the destination process. The marshaled data packet can be unmarshaled just once, or not at all. If the receiver unmarshals the data packet successfully, the **CoReleaseMarshalData** API function is automatically called on the data packet as part of the unmarshaling process. If the receiver does not or cannot unmarshal the data packet, the sender must call the **CoReleaseMarshalData** API function on the data packet.

MSHLFLAGS_TABLESTRONG
: The marshaling is occurring because the data packet is to be stored in a globally-accessible table from which it can be unmarshaled one or more times, or not at all. The presence of the data packet in the table counts as a strong reference to the interface being marshaled, meaning that it is sufficient to keep the object alive. When the data packet is removed from the table, the table implementor must call the **CoReleaseMarshalData** API function on the data packet.

MSHLFLAGS_TABLESTRONG is used by the **RegisterDragDrop** API function when registering a window as a drop target. This keeps the window registered as a drop target no matter how many times the end-user drags across the window. The **RevokeDragDrop** API function calls **CoReleaseMarshalData**.

MSHLFLAGS_TABLEWEAK
: The marshaling is occurring because the data packet is to be stored in a globally-accessible table from which it can be unmarshaled one or more times, or not at all. However, the presence of the data packet in the table acts as a weak reference to the interface being marshaled, meaning that it is not sufficient to keep the object alive. When the data packet is removed from the table, the table implementor must call the **CoReleaseMarshalData** API function on the data packet.

MSHLFLAGS_TABLEWEAK is typically used when registering an object in the Running Object Table (ROT). This prevents the object's entry in the ROT from keeping the object alive in the absence of any other connections. See **IRunningObjectTable::Register** for more information.

See Also

CoGetStandardMarshal, **CoMarshalInterface**, **IMarshal**

OLECLOSE

The **OLECLOSE** enumeration constants are used in the **IOleObject::Close** method to determine whether the object should be saved before closing. These are defined *ole2.h* and in the **IOleObject** interface (*oleobj.idl*).

```
typedef enum tagOLECLOSE
{
     OLECLOSE_SAVEIFDIRTY  = 0,
     OLECLOSE_NOSAVE       = 1,
     OLECLOSE_PROMPTSAVE   = 2
} OLECLOSE;
```

Elements

OLECLOSE_SAVEIFDIRTY
: Indicates that the object should be saved if it is dirty.

OLECLOSE_NOSAVE
: Indicates that the object should not be saved, even if it is dirty. This flag is typically used when an object is being deleted.

OLECLOSE_PROMPTSAVE
: Indicates that if the object is dirty, the **IOleObject::Close** implementation should display a dialog to let the end user determine whether to save the object. However, if the object is in the running state but its user interface is invisible, the the end user should not be prompted, and the close should be handled as if OLECLOSE_SAVEIFDIRTY had been specified.

See Also

IOleObject::Close

OLECONTF

The **OLECONTF** enumeration indicate the kind of objects to be enumerated by the returned **IEnumUnknown** interface. **OLECONTF** contains a set of bitwise constants used in the **IOleContainer::EnumObjects** method.

```
typedef enum tagOLECONTF
{
     OLECONTF_EMBEDDINGS    = 1;
     OLECONTF_LINKS         = 2;
     OLECONTF_OTHERS        = 4;
     OLECONTF_ONLYUSER      = 8;
     OLECONTF_ONLYIFRUNNING = 16
} OLECONTF;
```

Elements

OLECONTF_EMBEDDINGS
Enumerate the embedded objects in the container.

OLECONTF_LINKS
Enumerate the linked objects in the container.

OLECONTF_OTHERS
Enumerate all objects in the container that are not OLE compound document objects (i.e.; objects other than linked or embedded objects). Use this flag to enumerate the container's pseudo-objects.

OLECONTF_ONLYUSER
Enumerate only those objects the user is aware of. For example; hidden named-ranges in Microsoft Excel would not be enumerated using this value.

OLECONTF_ONLYIFRUNNING
Enumerate only those linked or embedded objects that are currently in the running state for this container.

See Also

IEnumUnknown, **IOleContainer::EnumObjects**

OLEGETMONIKER

The **OLEGETMONIKER** enumeration constants indicate the requested behavior of the **IOleObject::GetMoniker** and **IOleClientSite::GetMoniker** methods. These constants are defined in *ole2.h* and in the **IOleObject** interface (*oleobj.idl*).

```
typedef enum tagOLEGETMONIKER
{
     OLEGETMONIKER_ONLYIFTHERE = 1,
     OLEGETMONIKER_FORCEASSIGN = 2,
     OLEGETMONIKER_UNASSIGN    = 3,
     OLEGETMONIKER_TEMPFORUSER = 4
} OLEGETMONIKER;
```

Elements

OLEGETMONIKER_ONLYIFTHERE
: Indicates that if a moniker for the object or container does not exist, **GetMoniker** should return E_FAIL and not assign a moniker.

OLEGETMONIKER_FORCEASSIGN
: Indicates that if a moniker for the object or container does not exist, **GetMoniker** should create one.

OLEGETMONIKER_UNASSIGN
: Indicates that **IOleClientSite::GetMoniker** can release the object's moniker (although it is not required to do so). This constant is not valid in **IOleObject::GetMoniker**.

OLEGETMONIKER_TEMPFORUSER
: Indicates that if a moniker for the object does not exist, **IOleObject::GetMoniker** can create a temporary moniker that can be used for display purposes (**IMoniker::GetDisplayName**) but not for binding. This is enables the object server to return a descriptive name for the object without incurring the overhead of creating and maintaining a moniker until a link is actually created.

Comments

If the OLEGETMONIKER_FORCEASSIGN flag causes a container to create a moniker for the object, the container should notify the object by calling the **IOleObject::SetMoniker** method.

See Also

IMoniker, **IOleClientSite::GetMoniker**, **IOleObject::GetMoniker**

OLELINKBIND

The **OLELINKBIND** enumeration constants control binding operations to a link source. They are used in the **IOleLink::BindToSource** method, and **OLELINKBIND** is defined in OLEIDL.IDL.

```
typedef enum tagOLELINKBIND
{
    OLELINKBIND_EVENIFCLASSDIFF = 1,
} OLELINKBIND;
```

Element

OLELINKBIND_EVENIFCLASSDIFF
Indicates that the binding operation should proceed even if the current class of the link source is different from the last time the link was bound. For example, the link source could be a Lotus spreadsheet that was converted to an Excel spreadsheet.

See Also

IOleLink::BindToSource

OLEMISC

The **OLEMISC** enumeration is a set of bitwise constants that can be combined to describe miscellaneous characteristics of an object or class of objects. A container can call the **IOleObject::GetMiscStatus** method to determine the **OLEMISC** bits set for an object. The values specified in an object server's CLSID\MiscStatus entry in the registration database are based on the **OLEMISC** enumeration. These constants are also used in the *dwStatus* member of the **OBJECTDESCRIPTOR** structure. **OLEMISC** is defined in *ole2.h* and in the **IOleObject** interface (*oleobj.idl*).

```
typedef enum tagOLEMISC // bitwise
{
     OLEMISC_RECOMPOSEONRESIZE            = 1,
     OLEMISC_ONLYICONIC                   = 2,
     OLEMISC_INSERTNOTREPLACE             = 4,
     OLEMISC_STATIC                       = 8,
     OLEMISC_CANTLINKINSIDE               = 16,
     OLEMISC_CANLINKBYOLE1                = 32,
     OLEMISC_ISLINKOBJECT                 = 64,
     OLEMISC_INSIDEOUT                    = 128,
     OLEMISC_ACTIVATEWHENVISIBLE          = 256
     OLEMISC_RENDERINGISDEVICEINDEPENDENT = 512
} OLEMISC;
```

Elements

OLEMISC_RECOMPOSEONRESIZE

Indicates that when the container resizes the space allocated to displaying one of the object's presentations, the object wants to recompose the presentation. This means that on resize, the object wants to do more than scale its picture. If this bit is set, the container should force the object to the running state and call **IOleObject::SetExtent** with the new size.

OLEMISC_ONLYICONIC

Indicates that the object has no useful content view other than its icon. From the user's perspective, the Display As Icon checkbox (in the Paste Special dialog box) for this object should always be checked, and should not be uncheckable. Note that such an object should still have a drawable content aspect; it will look the same as its icon view.

OLEMISC_INSERTNOTREPLACE

Indicates that the object has initialized itself from the data in the container's current selection. Containers should examine this bit after calling **IOleObject::InitFromData** to initialize an object from the current selection. If set, the container should insert the object beside the current selection rather than replacing the current selection. If this bit is not set, the object being inserted replaces the current selection.

OLEMISC_STATIC

Indicates that this object is a static object, which is an object that contains only a presentation; it contains no native data. See **OleCreateStaticFromData**.

OLEMISC_CANTLINKINSIDE

Indicates that this object cannot be the link source that when bound to activates (runs) the object. If the object is selected and copied to the clipboard, the object's container can offer a link in a clipboard data transfer that, when bound, must connect to the outside of the object. The user would see the object selected in its container, not open for editing. Rather than doing this, the container can simply refuse to offer a link source when transferring objects with this bit set. Examples of objects that have this bit set include OLE1 objects, static objects, and links.

OLEMISC_CANLINKBYOLE1
: Indicates that this object can be linked to by OLE 1 containers. This bit is used in the *dwStatus* member of the **OBJECTDESCRIPTOR** structure transferred with the Object and Link Source Descriptor formats. An object can be linked to by OLE 1 containers if it is an untitled document, a file, or a selection of data within a file. Embedded objects or pseudo-objects that are contained within an embedded object cannot be linked to by OLE 1 containers (i.e., OLE 1 containers cannot link to link sources that, when bound, require more than one object server to be run.

OLEMISC_ISLINKOBJECT
: This object is a link object. This bit is significant to OLE 1 and is set by the OLE 2 link object; object applications have no need to set this bit.

OLEMISC_INSIDEOUT
: This object is capable of activating in-place, without requiring installation of menus and toolbars to run. Several such objects can be active concurrently. Some containers, such as forms, may choose to activate such objects automatically.

OLEMISC_ACTIVATEWHENVISIBLE
: This bit is set only when OLEMISC_INSIDEOUT is set, and indicates that this object prefers to be activated whenever it is visible. Some containers may always ignore this hint.

OLEMISC_RENDERINGISDEVICEINDEPENDENT
: This object does not pay any attention to target devices. Its presention data will be the same in all cases.

See Also

IOleObject::GetMiscStatus, **OBJECTDESCRIPTOR**

OLERENDER

The **OLERENDER** enumeration constants are used in the various object creation API functions to indicate the type of caching requested for the newly created object.

```
typedef enum tagOLERENDER
{
     OLERENDER_NONE   = 0;
     OLERENDER_DRAW   = 1;
     OLERENDER_FORMAT = 2;
     OLERENDER_ASIS   = 3
} OLERENDER;
```

Elements

OLERENDER_NONE
The client is not requesting any locally cached drawing or data retrieval capabilities in the object. *pFormatEtc* is ignored for this option.

OLERENDER_DRAW
The client will draw the content of the object on the screen (a NULL target device) using **IViewObject:Draw**. The object itself determines the data formats that need to be cached. With this render option, only the *ptd* and *dwAspect* members of *pFormatEtc* are significant, since the object may cache things differently depending on the parameter values. However, *pFormatEtc* can legally be NULL here, in which case the object is to assume the display target device and the DVASPECT_CONTENT aspect.

OLERENDER_FORMAT
The client will pull one format from the object using **IDataObject::GetData()**. The format of the data to be cached is passed in *pFormatEtc*, which may not in this case be NULL.

OLERENDER_ASIS
The client is not requesting any locally cached drawing or data retrieval capabilities in the object. *pFormatEtc* is ignored for this option. The difference between this and the OLERENDER_FORMAT value is important in such functions as **OleCreateFromData()** and **OleCreateLinkFromData()**.

See Also

OleCreate, **OleCreateFromData**, **OleCreateFromFile**, **OleCreateLink**, **OleCreateLinkFromData**, **OleCreateLinkToFile**, **OleCreateStaticFromData**

OLEUIPASTEFLAG

This enumeration is used to indicate the user options that are available to the user when pasting this format, and within which group or list of choices (Paste, Paste Link, etc) this entry is to be available. OLEUIPASTEFLAG is used by the **OLEUIPASTEENTRY** structure. It

```
typedef enum tagOLEUIPASTEFLAG
{
     OLEUIPASTE_ENABLEICON    = 2048,
     OLEUIPASTE_PASTEONLY     = 0,
     OLEUIPASTE_PASTE         = 512,
     OLEUIPASTE_LINKANYTYPE   = 1024,
     OLEUIPASTE_LINKTYPE1     = 1,
     OLEUIPASTE_LINKTYPE2     = 2,
     OLEUIPASTE_LINKTYPE3     = 4,
     OLEUIPASTE_LINKTYPE4     = 8,
     OLEUIPASTE_LINKTYPE5     = 16,
     OLEUIPASTE_LINKTYPE6     = 32,
     OLEUIPASTE_LINKTYPE7     = 64,
     OLEUIPASTE_LINKTYPE8     = 128
} OLEUIPASTEFLAG;
```

OLEUIPASTE_ENABLEICON

If the container does not specify this flag for the entry in the **OLEUIPASTEENTRY**array passed as input to **OleUIPasteSpecial**, the DisplayAsIcon button will be unchecked and disabled when the user selects the format that corresponds to the entry.

OLEUIPASTE_PASTEONLY

Indicates that the entry in the OLEUIPASTEENTRY array is valid for pasting only.

OLEUIPASTE_PASTE

Indicates that the entry in the **OLEUIPASTEENTRY**array is valid for pasting. It may also be valid for linking if any of the following linking flags are specified. If it is valid for linking, then the following flags indicate which link types are acceptable by OR'ing together the appropriate OLEUIPASTE_LINKTYPE<#> values. These values correspond as follows to the array of link types passed to **OleUIPasteSpecial**:

OLEUIPASTE_LINKTYPE1=arrLinkTypes[0]
OLEUIPASTE_LINKTYPE2=arrLinkTypes[1]
OLEUIPASTE_LINKTYPE3=arrLinkTypes[2]
OLEUIPASTE_LINKTYPE4=arrLinkTypes[3]
OLEUIPASTE_LINKTYPE5=arrLinkTypes[4]
OLEUIPASTE_LINKTYPE6=arrLinkTypes[5]
OLEUIPASTE_LINKTYPE7=arrLinkTypes[6]
OLEUIPASTE_LINKTYPE8=arrLinkTypes[7]

where,

UINT arrLinkTypes[8] is an array of registered clipboard formats for linking. A maximum of 8 link types is allowed.

OLEUPDATE

The **OLEUPDATE** enumeration constants are used to indicate whether the linked object updates the cached data for the linked object automatically or only when the container calls either the **IOleObject::Update** or **IOleLink::Update** methods. The constants are used in the **IOleLink** interface.

The **OLEUPDATE** enumeration is defined in OLEIDL.IDL..

```
typedef enum tagOLEUPDATE
{
     OLEUPDATE_ALWAYS = 1;
     OLEUPDATE_ONCALL = 3
} OLEUPDATE;
typedef OLEUPDATE *LPOLEUPDATE;
```

Elements

OLEUPDATE_ALWAYS
Update the link object whenever possible, this option corresponds to the Automatic update option in the Links dialog box.

OLEUPDATE_ONCALL
Update the link object only when **IOleObject::Update** or **IOleLink::Update** is called, this option corresponds to the Manual update option in the Links dialog box.

See Also

IOleLink::SetUpdateOptions, **IOleLink::GetUpdateOptions**

OLEVERBATTRIB

The **OLEVERBATTRIB** enumeration constants are used in the **OLEVERB** structure to describe the attributes of a specified verb for an object. Defined in the **IEnumOLEVERB** interface (*eoverb.idl*).

```
typedef enum tagOLEVERBATTRIB
{
     OLEVERBATTRIB_NEVERDIRTIES    = 1,
     OLEVERBATTRIB_ONCONTAINERMENU = 2
} OLEVERBATTRIB;
```

Elements

OLEVERBATTRIB_NEVERDIRTIES
OLEVERBATTRIB_ONCONTAINERMENU
Indicates a verb that should appear in the container's menu of verbs for this object.

See Also

IEnumOLEVERB, **OLEVERB**

OLEWHICHMK

The **OLEWHICHMK** enumeration constants indicate which part of an object's moniker is being set or retrieved. These constants are used in the **IOleObject** and **IOleClientSite** interfaces, and are defined in *ole2.h* and in the **IOleObject** interface (*oleobj.idl*).

```
typedef enum tagOLEWHICHMK
{
     OLEWHICHMK_CONTAINER = 1,
     OLEWHICHMK_OBJREL    = 2,
     OLEWHICHMK_OBJFULL   = 3
} OLEWHICHMK;
```

Elements

OLEWHICHMK_CONTAINER
: Indicates the moniker of the object's container, Typically this is a file moniker. This moniker is not persistently stored inside the object, since the container can be renamed even while the object is not loaded.

OLEWHICHMK_OBJREL
: Indicates the moniker of the object relative to its container. Typically this is an item moniker, and it is part of the persistent state of the object. If this moniker is composed on to the end of the container's moniker, the resulting moniker is the full moniker of the object.

OLEWHICHMK_OBJFULL
: Indicates the full moniker of the object. Binding to this moniker results in a connection to the object. This moniker is not persistently stored inside the object, since the container can be renamed even while the object is not loaded.

See Also

IOleClientSite::GetMoniker, **IOleObject::GetMoniker**, **IOleObject::SetMoniker**

PENDINGMSG

The **PENDINGMSG** enumeration constants are return values of **IMessageFilter::MessagePending**.

Defined in the **IMessageFilter** interface (*msgflt.idl*).

```
typedef enum tagPENDINGMSG
{
     PENDINGMSG_CANCELCALL      = 0,
     PENDINGMSG_WAITNOPROCESS   = 1,
     PENDINGMSG_WAITDEFPROCESS  = 2
} PENDINGMSG;
```

Elements

PENDINGMSG_CANCELCALL
  cancel the outgoing call

PENDINGMSG_WAITNOPROCESS
  wait for the return and don't dispatch the message

PENDINGMSG_WAITDEFPROCESS
  wait and dispatch the message

See Also

IMessageFilter::MessagePending

PENDINGTYPE

The **PENDINGTYPE** enumeration constants indicate the level of nesting in the **IMessageFilter::MessagePending** method.

Defined in the **IMessageFilter** interface (*msgflt.idl*).

```
typedef enum tagPENDINGTYPE
{
     PENDINGTYPE_TOPLEVEL = 1,
     PENDINGTYPE_NESTED   = 2
} PENDINGTYPE;
```

Elements

PENDINGTYPE_TOPLEVEL
toplevel call

PENDINGTYPE_NESTED
nested call

See Also

IMessageFilter::MessagePending

REGCLS

The **REGCLS** enumeration defines flags used in **CoRegisterClassObject** to control the type of connections to the class object. It is defined as follows:

```
typedef enum tagREGCLS
{
     REGCLS_SINGLEUSE        = 0,
     REGCLS_MULTIPLEUSE      = 1,
     REGCLS_MULTI_SEPARATE   = 2,
     } REGCLS;
```

Elements

REGCLS_SINGLEUSE
: Specifies that once an application has connected to the class object with **CoGetClassObject**, the class object is removed from public view so that no other applications can connect to it. This flag is commonly used for single document interface (SDI) applications. Specifying this flag does not affect the responsibility of the object application to call **CoRevokeClassObject**; it must always call **CoRevokeClassObject** when it is finished with an object class.

REGCLS_MULTIPLEUSE
: Specifies that multiple applications can connect to the class object through calls to **CoGetClassObject**.

REGCLS_MULTI_SEPARATE
: Similar to REGCLS_MULTIPLEUSE, except that REGCLS_MULTI_SEPARATE does not automatically register the class object as CLSCTX_INPROC_SERVER for a local server. Instead, it provides separate control over each context. When a class is registered this way, if that server tries to bind to an object with its own class id, it will start another copy of the server.

Comments

In **CoRegisterClassObject**, members of both the **REGCLS** and the **CLSCTX** enumerations, taken together, determine how the class object is registered.

The following table summarizes the allowable flag combinations and the object registrations affected by the combinations:

	REGCLS_ SINGLEUSE	**REGCLS_ MULTIPLEUSE**	**REGCLS_ MULTI_ SEPARATE**	**Other**
CLSCTX_ INPROC_ SERVER	Error	Inproc	Inproc	Error
CLSCTX_ LOCAL_ SERVER	Local	Inproc/local	Local	Error
Both of the above	Error	Inproc/local	Inproc/local	Error
Other	Error	Error	Error	Error

REGCLS_MULTIPLEUSE in combination with CLSCTX_LOCAL_SERVER automatically registers the class object as an in-process server (CLSCTX_INPROC_SERVER). In contrast, registering a class object as a local server and specifying REGCLS_MULTIPLE_SEPARATE does not register the class object as an in-process server (registers the object with the CLSCTX_LOCAL_SERVER flag, but does not automatically add the CLSCTX_INPROC_SERVER flag as is the case when you specify the REGCLS_MULTIPLEUSE flag. This distinction is important in applications that are both OLE containers and OLE embeddings, allowing a container/server to be inserted into itself.

In general, the following two registrations have the same effect -- they register class objects as both multiple-use and as in-process servers:

```
CLSCTX_LOCAL_SERVER, REGCLS_MULTIPLEUSE
```

```
(CLSCTX_INPROC_SERVER|CLSCTX_LOCAL_SERVER), REGCLS_MULTI_SEPARATE
```

The following registers the class object only as a multiple-use local server:

```
CLSCTX_LOCAL_SERVER, REGCLS_MULTI_SEPARATE
```

See Also

CoGetClassObject, **CoRegisterClassObject**, **CoRevokeClassObject**, **DllGetClassObject**

SERVERCALL

The **SERVERCALL** enumeration constants indicate the status of server call - returned by **IMessageFilter::HandleInComingCall** and passed to **IMessageFilter::RetryRejectedCall**.

Defined in the **IMessageFilter** interface (*msgflt.idl*).

```
typedef enum tagSERVERCALL
{
     SERVERCALL_ISHANDLED  = 0,
     SERVERCALL_REJECTED   = 1,
     SERVERCALL_RETRYLATER = 2
} SERVERCALL;
```

Elements

SERVERCALL_ISHANDLED
SERVERCALL_REJECTED
SERVERCALL_RETRYLATER

See Also

IMessageFilter::HandleInComingCall, **IMessageFilter::RetryRejectedCall**

STATFLAG

The **STATFLAG** enumeration values indicate whether the method should try to return a name in the *pwcsName* member of the **STATSTG** structure. The values are used in the **ILockBytes::Stat**, **IStorage::Stat**, and **IStream::Stat** methods to save memory when the *pwcsName* member is not needed.

Defined in the **IOLETypes** pseudo-interface (*oletyp.idl*).

```
typedef enum tagSTATFLAG
{
     STATFLAG_DEFAULT = 0,
     STATFLAG_NONAME  = 1
} STATFLAG;
```

Elements

STATFLAG_DEFAULT
Requests that the statistics include the *pwcsName* member of the **STATSTG** structure.

STATFLAG_NONAME
Requests that the statistics not include the *pwcsName* member of the **STATSTG** structure. If the name is omitted, there is no need for the **Stat** methods to allocate and free memory for the string value for the name and the method can save an **Alloc** and **Free** operation.

See Also

ILockBytes::Stat, **IStorage::Stat**, **IStream::Stat**

STATSTATE

The **STATSTATE** enumeration values indicate state information about the storage object and are used as a mask. The values are used in the **IStorage::SetStateBits** method.

Defined in the ().

```
typedef enum tagSTATSTATE
{
     STATSTATE_DOC = 1,
     STATSTATE_CONVERT  = 2,
     STATSTATE_FILESTGSAME  = 4
} STATSTATE;
```

Elements

STATSTATE_DOC
.

Indicates that the storage object is a document file. This bit is set on the root storage object as part of a normal File/Save sequence. With nested storage objects, the application manages the storage objects and sets or clears this bit as appropriate. If the nested object is an embedded object, this bit can be ignored. It is cleared in a newly created storage object. However, some applications might use this bit to enable editing an embedded storage object without first copying the object to the file system. For example, a mail application might set this bit for attachments so the attachments can be edited without copying them first to a file.

STATSTATE_CONVERT

Indicates that a convert operation was done on this storage object while it was in a passive state.

STATSTATE_FILESTGSAME

Indicates that the embedded object and document representations for the storage object are the same. Thus, the storage object can be saved in a document file simply by copying the storage object bits.

See Also

IStorage::SetStateBits.

STGC

The **STGC** enumeration constants specify the conditions for performing the commit operation in the **IStorage::Commit** and **IStream::Commit** methods.

Defined in the **IOLETypes** pseudo-interface (*oletyp.idl*).

```
typedef enum tagSTGC
{
     STGC_DEFAULT                           = 0,
     STGC_OVERWRITE                         = 1,
     STGC_ONLYIFCURRENT                     = 2,
     STGC_DANGEROUSLYCOMMITMERELYTODISKCACHE = 4
} STGC;
```

Elements

STGC_DEFAULT

Indicates that none of the other values apply. You can specify this condition or some combination of the other three. You would use this value mainly to make your code more readable.

STGC_OVERWRITE

Indicates that the commit operation can overwrite existing data to reduce overall space requirements. This value is not recommended for typical usage because it is not as robust as the default case. In this case, it is possible for the commit to fail after the old data is overwritten but before the new data is completely committed. Then, neither the old version nor the new version of the storage object will be intact.

You can use this value in cases where:

- the user has indicated a willingness to risk losing the data
- the low memory save sequence will be used to safely save the storage object to a smaller file
- a previous commit returned STG_E_MEDIUMFULL but overwriting the existing data would provide enough space to commit changes to the storage object

Note that the commit operation checks for adequate space before any overwriting occurs. Thus, even with this value specified, if the commit operation fails due to space requirements, the old data will remain safe. The case where data loss can occur is when the commit operation fails due to some reason other than lack of space and the STGC_OVERWRITE value was specified.

STGC_ONLYIFCURRENT

Prevents multiple users of a storage object from overwriting one another's changes. The commit operation occurs only if there have been no changes to the saved storage object since the user most recently opened the storage object. Thus, the saved version of the storage object is the same version that the user has been editing. If other users have changed the storage object, the commit operation fails and returns the STG_E_NOTCURRENT value. You can override this behavior by calling the **Commit** method again using the STGC_DEFAULT value.

STGC_DANGEROUSLYCOMMITMERELYTODISKCACHE

Commits the changes to a write-behind disk cache, but does not save the cache to the disk. In a write-behind disk cache, the operation that writes to disk actually writes to a disk cache thus increasing performance. The cache is eventually written to the disk but usually not until after the write operation has already returned. The performance increase comes at the expense of an increased risk of losing data if a problem occurs before the cache is saved and the data in the cache is lost.

If you do not specify this value, then committing changes to root level storage objects is robust even if a disk cache is used. The two-phase commit process ensures that data is stored on the disk and not just to the disk cache.

Comments

You can specify STGC_DEFAULT or some combination of the other three values. Typically, you would use STGC_ONLYIFCURRENT to protect the storage object in cases where more than one user can edit the object simultaneously.

See Also

IStorage, **IStream**

STGFMT

The **STGFMT** enumeration values indicate the format of a storage object and are used in the **STATSTG** structure and in the **StgCreateDocFile** and **StgIsStorageFile** API functions.

Defined in the **IOLETypes** pseudo-interface (*oletyp.idl*).

```
typedef enum tagSTGFMT
{
     STGFMT_DOCUMENT  = 0,
     STGFMT_DIRECTORY = 1,
     STGFMT_CATALOG   = 2,
     STGFMT_FILE      = 3
} STGFMT;
```

Elements

STGFMT_DOCUMENT
Indicates a document format.

STGFMT_DIRECTORY
Indicates a directory format.

STGFMT_CATALOG
Indicates a catalog format.

STGFMT_FILE
Indicates a file format.

See Also

STATSTG, **StgCreateDocFile**, **StgIsStorageFile**

STGM

The **STGM** enumeration values are used in the storage and stream interfaces to indicate the conditions for creating and deleting the object and access modes for the object.

Defined in the ().

```
typedef enum tagSTGM
{
     STGM_READ = OF_READ,            // 0x0000
     STGM_WRITE = OF_WRITE,          // 0x0001
     STGM_READWRITE = OF_READWRITE,  // 0x0002

     STGM_SHARE_DENY_NONE = OF_SHARE_DENY_NONE,    // 0x0040
     STGM_SHARE_DENY_READ = OF_SHARE_DENY_READ,    // 0x0030
     STGM_SHARE_DENY_WRITE = OF_SHARE_DENY_WRITE,  // 0x0020
     STGM_SHARE_EXCLUSIVE = OF_SHARE_EXCLUSIVE,    // 0x0010

     STGM_DIRECT = 0x00000000,
     STGM_TRANSACTED = 0x00010000,

     STGM_CREATE = OF_CREATE       // 0x1000,
     STGM_CONVERT = 0x00020000,
     STGM_FAILIFTHERE = 0x00000000,

     STGM_PRIORITY = 0x00040000,

     STGM_DELETEONRELEASE = 0x04000000,

} STGM;
```

Elements

STGM_READ, STGM_WRITE, STGM_READWRITE

For stream objects, STGM_READ allows you to call the **IStream::Read** method. For storage objects, you can enumerate the storage elements and open them for reading.

STGM_WRITE lets you save changes to the object.

STGM_READWRITE is the combination of STGM_READ and STGM_WRITE.

STGM_SHARE_DENY_NONE, STGM_SHARE_DENY_READ, STGM_SHARE_DENY_WRITE, STGM_SHARE_EXCLUSIVE

STGM_SHARE_DENY_NONE specifies that subsequent openings of the object are not denied read or write access.

STGM_SHARE_DENY_READ prevents others from subsequently opening the object in STGM_READ mode. It is typically used on a root storage object.

STGM_SHARE_DENY_WRITE prevents others from subsequently opening the object in STGM_WRITE mode. This value is typically used to prevent unnecessary copies made of an object opened by multiple users. If this value is not specified, a snapshot copy must be made for each subsequent opening. Thus, you can improve performance by specifying this value.

STGM_SHARE_EXCLUSIVE is the combination of STGM_SHARE_DENY_READ and STGM_SHARE_DENY_WRITE.

STGM_DIRECT, STGM_TRANSACTED

In direct mode, each change to a storage element is written as it occurs. This is the default.

In transacted mode, changes are buffered and are written only if an explicit commit operation is called. The changes can be ignored by calling the **Revert** method in the **IStream** or **IStorage** interfaces. Transacted mode is not supported in the OLE-provided implementation of compound files.

STGM_CREATE, STGM_CONVERT, STGM_FAILIFTHERE

STGM_CREATE indicates that an existing storage object or stream should be removed before the new one replaces it. A new object is created when this flag is specified only if the existing object has been successfully removed.

STGM_CONVERT flag is used in three situations:

- when you are trying to create a storage object on disk but a file of that name already exists
- when you are trying to create a stream inside a storage object but a stream with the specified name already exists
- when you are creating a byte array object but one with the specified name already exists

STGM_CONVERT creates the new object while preserving existing data in a stream named **CONTENTS**. In the case of a storage object or a byte array, the old data is flattened to a stream regardless of whether the existing file or byte array currently contains a layered storage object.

STGM_FAILIFTHERE causes the create operation to fail if an existing object with the specified name exists. In this case, STG_E_FILEALREADYEXISTS is returned. STGM_FAILIFTHERE applies to both storage objects and streams.

STGM_PRIORITY

Opens the storage object with exclusive access to the most recently committed version. Thus, other users cannot commit changes to the object while you have it open in priority mode. You gain performance benefits for copy operations but you prevent others from committing changes. So you should limit the time you keep objects open in priority mode. You must specify STGM_DIRECT and STGM_READ with priority mode.

STGM_DELETEONRELEASE

Indicates that the underlying file is to be automatically destroyed when the root storage object is released. This capability is most useful for creating temporary files.

Comments

You can combine these flags but you can only choose one flag from each group of related flags.

See Also

IStream::Read.

STGMOVE

The **STGMOVE** enumeration values indicate whether a storage element is to be moved or copied. They are used in the **IStorage::MoveElementTo** method.

Defined in the **IOLETypes** pseudo-interface (*oletyp.idl*).

```
typedef enum tagSTGMOVE
{
     STGMOVE_MOVE = 0,
     STGMOVE_COPY = 1
} STGMOVE;
```

Elements

STGMOVE_MOVE
Indicates the method should move the data from the source to the destination.

STGMOVE_COPY
Indicates the method should copy the data from the source to the destination. A copy is the same as a move except the source element is not removed after copying the element to the destination. Copying an element on top of itself is undefined.

See Also

IStorage::MoveElementTo

STGTY

The **STGTY** enumeration values are used in the *type* member of the **STATSTG** structure to indicate the type of the storage element. A storage element is a storage object, a stream object, or a byte array object (LOCKBYTES).

Defined in the **IStream** interface (*stream.idl*).

```
typedef enum tagSTGTY
{
     STGTY_STORAGE    = 1,
     STGTY_STREAM     = 2,
     STGTY_LOCKBYTES  = 3,
     STGTY_PROPERTY   = 4
} STGTY;
```

Elements

STGTY_STORAGE
  Indicates that the storage element is a storage object.

STGTY_STREAM
  Indicates that the storage element is a stream object.

STGTY_LOCKBYTES
  Indicates that the storage element is a byte array object.

STGTY_PROPERTY
  Indicates that the storage element is a property storage object.

See Also

IStream, **STATSTG**

STREAM_SEEK

The **STREAM_SEEK** enumeration values specify the origin from which to calculate the new seek pointer location. They are used for the *dworigin* parameter in the **IStream::Seek** method. The new seek position is calculated using this value and the *dlibMove* parameter.

Defined in the **IStream** interface (*stream.idl*).

```
typedef enum tagSTREAM_SEEK
{
     STREAM_SEEK_SET = 0,
     STREAM_SEEK_CUR = 1,
     STREAM_SEEK_END = 2
} STREAM_SEEK;
```

Elements

STREAM_SEEK_SET
Indicates that the new seek pointer is an offset relative to the beginning of the stream. In this case, the *dlibMove* parameter is the new seek position relative to the beginning of the stream.

STREAM_SEEK_CUR
Indicates that the new seek pointer is an offset relative to the current seek pointer location. In this case, the *dlibMove* parameter is the signed displacement from the current seek position.

STREAM_SEEK_END
Indicates that the new seek pointer is an offset relative to the end of the stream. In this case, the *dlibMove* parameter is the new seek position relative to the end of the stream.

See Also

IStream::Seek

TYMED

The **TYMED** enumeration values indicate the type of storage medium being used in a data transfer. They are used in the **STGMEDIUM** or **FORMATETC** structures.

```
typedef [transmit_as(long)] enum tagTYMED
{
     TYMED_HGLOBAL   = 1;
     TYMED_FILE      = 2;
     TYMED_ISTREAM   = 4;
     TYMED_ISTORAGE  = 8;
     TYMED_GDI       = 16;
     TYMED_MFPICT    = 32;
     TYMED_ENHMF     = 64;
     TYMED_NULL      = 0
} TYMED;
```

Elements

TYMED_HGLOBAL
: The storage medium is a global memory handle (HGLOBAL). Allocate the global handle with the GMEM_SHARE flag. If the **STGMEDIUM** *punkForRelease* member is NULL, the destination process should use **GlobalFree** to release the memory.

TYMED_FILE
: The storage medium is a disk file identified by a path. If the **STGMEDIUM** *punkForRelease* member is NULL, the destination process should use **OpenFile** to delete the file.

TYMED_ISTREAM
: The storage medium is a stream object identified by an **IStream** pointer. Use **IStream::Read** to read the data. If the **STGMEDIUM** *punkForRelease* member is NULL, the destination process should use **IStream::Release** to release the stream component.

TYMED_ISTORAGE
: The storage medium is a storage component identified by an **IStorage** pointer. The data is in the streams and storages contained by this **IStorage** instance. If the **STGMEDIUM** *punkForRelease* member is NULL, the destination process should use **IStorage::Release** to release the storage component.

TYMED_GDI
: The storage medium is a GDI component (HBITMAP). If the **STGMEDIUM** *punkForRelease* member is NULL, the destination process should use **DeleteObject** to delete the bitmap.

TYMED_MFPICT
: The storage medium is a metafile (HMETAFILE). Use the Windows or WIN32 API functions to access the metafile's data. If the **STGMEDIUM** *punkForRelease* member is NULL, the destination process should use **DeleteMetaFile** to delete the bitmap.

TYMED_ENHMF
: The storage medium is an enhanced metafile. If the **STGMEDIUM** *punkForRelease* member is NULL, the destination process should use **DeleteEnhMetaFile** to delete the bitmap.

TYMED_NULL
: Indicates that no data is being passed.

Comments

During data transfer operations, a storage medium is specified. This medium must be released after the data transfer operation. The provider of the medium indicates its choice of ownership scenarios in the value it provides in the **STGMEDIUM** structure. A NULL value for the **IUNKNOWN** field indicates that the receiving body of code owns and can free the medium. A non-NULL pointer specifies that

ReleaseStgMedium can always be called to free the medium.

See Also

FORMATETC, **IAdviseSink**, **IDataObject**, **IOleCache**, **ReleaseStgMedium**, **STGMEDIUM**

USERCLASSTYPE

The **USERCLASSTYPE** enumeration constants indicate the different variants of the display name associated with a class of objects. They are used in the **IOleObject::GetUserType** method and the **OleRegGetUserType** function. These constants are defined in *ole2.h* and in the **IOleObject** interface (*oleobj.idl*).

```
typedef enum tagUSERCLASSTYPE
{
     USERCLASSTYPE_FULL    = 1,
     USERCLASSTYPE_SHORT   = 2,
     USERCLASSTYPE_APPNAME = 3,
} USERCLASSTYPE;
```

Elements

USERCLASSTYPE_FULL
Indicates the full type name of the class.

USERCLASSTYPE_SHORT
Indicates a short name (maximum of 15 characters) that is used for popup menus and the Links dialog box.

USERCLASSTYPE_APPNAME
Indicates the name of the application servicing the class and is used in the Result text in dialogs.

See Also

IOleObject::GetUserType, **OleRegGetUserType**

BindMoniker QuickInfo

Binds the specified moniker, that is, locates the object identified by the moniker, which activates the object if it isn't active already, and retrieves the specified interface pointer to that object.

```
HRESULT BindMoniker(
  LPMONIKER pmk,             //Pointer to the moniker to be bound
  DWORD grfOpt,              //Reserved
  REFIID iidResult,          //IID of interface pointer desired
  LPVOID FAR *ppvResult      //Receives interface pointer
 );
```

Parameters

pmk
Points to the moniker that is to be bound.

grfOpt
Reserved for future use; must be zero.

iidResult
Specifies the IID of the interface pointer desired.

ppvResult
Receives a pointer to the requested interface. If an error occurs, *ppvResult* is NULL; otherwise, the caller is responsible for calling **IUnknown::Release**.

Return Values

S_OK
Indicates that the moniker was successfully bound.

MK_E_NOOBJECT
Indicates that the object identifed by *pmk* could not be found. If *pmk* is a composite moniker, the object indicated is some intermediate object identified in the composite.

E_OUTOFMEMORY
Indicates insufficient memory.

E_INVALIDARG
Indicates one or more invalid arguments.

The return value can also be any of the error values returned by **IMoniker::BindToObject**.

Comments

The **BindMoniker** helper function packages the following functionality:

```
IBindCtx *pbc;
CreateBindCtx(0, &pbc);
pmk->BindToObject(pbc, NULL, riid, ppvObj);
pbc->Release();
```

BindMoniker is a helper function that is convenient for a moniker client to call to bind one moniker. A moniker client uses a moniker to acquire an interface pointer to the object that the moniker identifies. If you have several monikers to bind in quick succession, and if you know that those monikers will activate the same objects, it may be more efficient to call the **IMoniker::BindToObject** method directly so you can use the same bind context for all the monikers. See the **IBindCtx** interface for more information.

The most common examples of moniker clients are applications that act as link containers; that is, container applications that allow their documents to contain linked objects. However, link containers are a special case in that they generally do not need to make direct calls to **IMoniker** methods. Instead, they manipulate linked objects through the **IOleLink** interface; the default handler implements this interface and calls the appropriate **IMoniker** methods as needed.

See Also

CreateBindCtx, **IMoniker::BindToObject**

CLSIDFromProgID QuickInfo

Looks up a CLSID from a ProgID in the registry.

```
HRESULT CLSIDFromProgID(
    LPCOLESTR lpszProgID,        //Points to the ProgID whose CLSID is requested
    LPCLSID pclsid               //Receives a pointer to the created CLSID
  );
```

Parameters

lpszProgID
Points to the ProgID whose CLSID is requested.

pclsid
Receives a pointer to the location of the created CLSID on return.

Return Values

S_OK
Indicates the CLSID was created successfully.

CO_E_CLASSSTRING
Indicates the registered CLSID for the ProgID is invalid.

REGDB_E_WRITEREGDB
Indicates an error occurred writing to the registry.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Indicates one or more arguments are invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

Comments

Given a ProgID, **CLSIDFromProgID** looks up its associated CLSID from a ProgID in the registry. If the ProgID cannot be found in the registry, **CLSIDFromProgID** creates an OLE 1 CLSID for it. Because of the restrictions placed on OLE 1 CLSID values, **CLSIDFromProgID** and **CLSIDFromString** are the *only* two functions that can be used to generate a CLSID for an OLE 1 object.

See Also

ProgIDFromCLSID

CLSIDFromString QuickInfo

Converts a string generated by the **StringFromCLSID** function back into the original CLSID.

```
HRESULT CLSIDFromString(
    LPOLESTR lpsz,        //Points to the string representation of the CLSID
    LPCLSID pclsid        //Receives a pointer to the CLSID on return
  );
```

Parameters

lpsz
Points to the string representation of the CLSID.

pclsid
Receives a pointer to the CLSID on return.

Return Values

S_OK
Indicates the CLSID was returned successfully.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Indicates one or more arguments are invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

Comments

Because of the restrictions placed on OLE 1 CLSID values, **CLSIDFromProgID** and **CLSIDFromString** are the *only* two functions that can be used to generate a CLSID for an OLE 1 object.

See Also

CLSIDFromProgID, **StringFromCLSID**

CoBuildVersion QuickInfo

This function is obsolete.

CoCreateFreeThreadedMarshaler

Creates an object that implements special marshalling behavior. This is a helpler function that makes it easier to define objects with VTBLs that different apartment model threads can access directly.

```
HRESULT CoCreateFreeThreadedMarshaler(
   LPUNKNOWN punkOuter,          //Points to the stream from which the object is to be marshaled
   LPUNKNOWN                     //The interface requested from the unmarshaled object
punkMarshaler
  );
```

Parameters

punkOuter

Specifies a pointer to the controlling IUnknown of the object that is going to have the special marshaling behavior specified by the object this function creates.

punkMarshaler

[out] On return, contains a pointer to the private IUnknown of an object that implements this special marshaling behavior.

Return Values

S_OK

Indicates the marshaler was created successfully.

E_OUTOFMEMORY

Indicates that the marshaler object could not be created.

Comments

CoCreateFreeThreadedMarshaler is a helper API that makes it easier to define an object that allows direct access to its VTBL from multiple apartment-model threads of execution, and uses standard marshaling if a reference to the object is sent to another process.

When there is a request for a pointer to an object, the marshal context of the request is determined by the specified member of the MSHCTX enumeration. When the marshal context is MSHCTX_INPROC, meaning that the pointer to the object need only be marshaled across different threads in the same process, the marshal code only has to copy the pointer to the interface being marshaled into the marshaling stream. For any other marshaling context, use standard marshaling to complete the operation.

CoCreateFreeThreadedMarshaler implements code that defines this context-dependent behavior, and, on return, writes to its *punkMarshaler* parameter a pointer to the private unknown for a object that does this standard type of marshaling. The function then aggregates this marshaler object with the object that should have this marshaling behavior.

See Also

CoMarshalInterThreadInterfaceInStream, **CoGetInterfaceAndReleaseStream**

CoCreateGuid [QuickInfo]

Creates a GUID, a unique 128-bit integer used for CLSIDs and interface identifiers.

```
HRESULT CoCreateGuid(
   GUID   *pguid        //Receives a pointer to the GUID on return
  );
```

Parameter

pguid
Receives a pointer to the requested GUID on return.

Return Value

S_OK
Indicates the GUID was successfully created.

Win32 errors are returned by **UuidCreate** but wrapped as an HRESULT.

Comments

The **CoCreateGuid** function calls **UuidCreate** (documented in the *RPC Programmer's Guide and Reference*), which creates a GUID, a globally unique 128-bit integer used for CLSIDs and interface IDs. To a very high degree of certainty, this function returns a unique value - no other invocation, on the same or any other system (networked or not) should return the same value.

CoCreateInstance QuickInfo

Creates a single uninitialized object of the kind associated with the specified CLSID and returns any requested interface pointer. Call **CoCreateInstance** when you want to create a single object based on a CLSID. If you want to create multiple objects from a single class object, refer to the CoGetClassObject function.

STDAPI CoCreateInstance(
  REFCLSID *rclsid*,   //Class identifier
  LPUNKNOWN *pUnkOuter*,   //Object is or isn't part of an aggregate
  DWORD *dwClsContext*,   //Context for running executable code
  REFIID *riid*,   //Interface identifier
  LPVOID * *ppv*   //Pointer to storage of interface pointer
 );

Parameters

rclsid
Specifies the CLSID that will associate the correct data and code.

pUnkOuter
If **NULL**, indicates that the object is not being created as part of an aggregate. If non-**NULL**, points to the controlling unknown of the aggregate that will use the newly created object.

dwClsContext
Specifies the context in which the executable is to be run. The values are taken from the enumeration CLSCTX.

riid
Specifies the interface to be used to communicate with the object.

ppv
Points to where to return the pointer to the requested interface.

Return Values

S_OK
Indicates that an instance of the specified object class was successfully created.

REGDB_E_CLASSNOTREG
Indicates that a specified class is not registered in the registration database.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Indicates one or more arguments are invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

CLASS_E_NOAGGREGATION
Indicates this class cannot be created as part of an aggregate.

Comments

The **CoCreateInstance** helper function connects to the class object associated with the specified CLSID, creates an uninitialized instance, and releases the class object. It encapsulates the following functionality:

```
CoGetClassObject(rclsid, dwClsContext, NULL, IID_IClassFactory,
      &pCF);
hresult = pCF->CreateInstance(pUnkOuter, riid, ppvObj)
pCF->Release();
```

It is convenient to use this function when you need to create only a single instance through a class object, and do not need to keep the class object around to create other instances. If you do, see the **CoGetClassObject** function.

See Also

CoGetClassObject, **IClassFactory::CreateInstance**

CoCreateStandardMalloc

This function is obsolete. The system-provided implementation is the only one available.

CoDisconnectObject QuickInfo

Disconnects all remote process connections being maintained on behalf of all the interface pointers on a specified object. Only the process that actually manages the object should call **CoDisconnectObject**.

```
STDAPI CoDisconnectObject(
    IUnknown * pUnk,          //Points to the object
    DWORD dwReserved          //Reserved for future use
  );
```

Parameters

pUnk
Points to the object to be disconnected. This does not necessarily have to be an **IUnknown** pointer. It can be a pointer to any interface supported by the object, as long as the interface is derived from **IUnknown**.

dwReserved
Reserved for future use; must be zero.

Return Values

S_OK
Indicates all connections to remote processes were successfully deleted.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Indicates one or more arguments are invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

E_FAIL
Indicates an unspecified error occurred.

Comments

The **CoDisconnectObject** function is not used in the typical course of processing (clients of interfaces should use **IUnknown::Release**). The primary purpose of **CoDisconnectObject** is to give an application control over connections from other processes that were made to objects managed by the application. For example, suppose an OLE container loads an embedded object and then links are established between the embedded object and outside containers. If the container application wants to exit, it is not appropriate for these remote connections to keep the object active beyond the lifetime of its container. When the object's container closes, such links should be disconnected. To forcefully close any existing connection(s), the object's container can call **CoDisconnectObject**.

Before calling **CoDisconnectObject**, the container should first call **IOleObject::Close** for all OLE objects. The objects will send **OnClose** notifications to inform all clients of the impending closure.

Calling this function is a privileged operation, to be invoked only by the process in which the object is actually managed.

See Also

IOleObject::Close

CoDosDateTimeToFileTime QuickInfo

Converts the MS-DOS representation of the time and date to a **FILETIME** structure, which Win32 uses to determine the date and time.

```
BOOL CoDosDateTimeToFileTime(
    WORD nDosDate,              //The 16-bit MS-DOS date
    WORD nDosTime,              //The 16-bit MS-DOS time
    FILETIME * lpFileTime       //Pointer to the FILETIME structure
  );
```

Parameters

nDosDate
Specifies the 16-bit MS-DOS date.

nDosTime
Specifies the 16-bit MS-DOS time.

lpFileTime
Points to the **FILETIME** structure.

Return Values

TRUE
Indicates the **FILETIME** structure was created successfully.

FALSE
Indicates the **FILETIME** structure was not created successfully, probably because of invalid arguments.

Comments

The **FILETIME** structure and the **CoDosDateTimeToFileTime** and **CoFileTimeToDosDateTime** functions are part of the Win32 API definition. They are provided for compatibility in all OLE implementations, but are redundant on Win32 platforms.

MS-DOS records file dates and times as packed 16-bit values. An MS-DOS date has the following format:

Bits	**Contents**
0-4	Days of the month (1-31).
5-8	Months (1 = January, 2 = February, and so forth).
9-15	Year offset from 1980 (add 1980 to get actual year).

An MS-DOS time has the following format:

Bits	**Contents**
0-4	Seconds divided by 2.
5-10	Minutes (0-59).
11-15	Hours (0-23 on a 24-hour clock).

See Also

CoFileTimeToDosDateTime, **CoFileTimeNow**

CoFileTimeNow QuickInfo

Returns the current time as a **FILETIME** structure.

```
HRESULT CoFileTimeNow(
   FILETIME   * lpFileTime        //Points to where to return the FILETIME structure
  );
```

Parameter

lpFileTime
  Points to where to return the **FILETIME** structure.

Return Values

S_OK
  Indicates the current time was converted to a **FILETIME** structure.

E_OUTOFMEMORY
  Out of memory.

E_INVALIDARG
  Indicates *lpFileTime* is invalid.

E_UNEXPECTED
  Indicates an unexpected error occurred.

E_FAIL
  Indicates the current time was not converted to a **FILETIME** structure.

See Also

CoDosDateTimeToFileTime, **CoFileTimeToDosDateTime**

CoFileTimeToDosDateTime

Converts a **FILETIME** into MS-DOS date and time values.

```
BOOL CoFileTimeToDosDateTime(
   FILETIME   * lpFileTime,       //Points to the FILETIME structure to be converted
   LPWORD lpDosDate,              //Points to the 16-bit MS-DOS date
   LPWORD lpDosTime               //Points to the 16-bit MS-DOS time
  );
```

Parameters

lpFileTime
Points to the **FILETIME** structure to be converted.

lpDosDate
Points to the 16-bit MS-DOS date.

lpDosTime
Points to the 16-bit MS-DOS time.

Return Values

TRUE
Indicates the **FILETIME** structure was converted successfully.

FALSE
Indicates the **FILETIME** structure was not converted successfully.

Comments

This is the inverse of the operation provided by the **CoDosDateTimeToFileTime** function.

See Also

CoDosDateTimeToFileTime, **CoFileTimeNow**

CoFreeAllLibraries QuickInfo

Frees all the DLLs that have been loaded with the **CoLoadLibrary** function, regardless of whether they are currently in use.

void CoFreeAllLibraries();

Comments

The OLE Component Object Model(COM) library maintains a list of loaded DLLs for each process that **CoFreeAllLibraries** uses to unload the libraries. The **CoUninitialize** function calls **CoFreeAllLibraries** internally, so OLE applications usually have no need to call this function directly.

See Also

CoLoadLibrary, **CoFreeLibrary**, **CoFreeUnusedLibraries**, **CoUninitialize**

CoFreeLibrary QuickInfo

Frees a library that was previously loaded with the *bAutoFree* parameter of the **CoLoadLibrary** function set to FALSE.

```
void CoFreeLibrary(
    HINSTANCE  hInst      //The handle of the library module to be freed
  );
```

Parameter

hInst

Specifies the handle to the library module to be freed, as returned by **CoLoadLibrary**.

Comments

The **CoFreeLibrary** function should be called to free a library that is to be freed explicitly. This is established when the library is loaded with the *bAutoFree* parameter of **CoLoadLibrary** set to FALSE. It is illegal to free a library explicitly when the corresponding **CoLoadLibrary** call specifies that it be freed automatically (the *bAutoFree* parameter is set to TRUE).

See Also

CoFreeAllLibraries, **CoFreeUnusedLibraries**, **CoLoadLibrary**

CoFreeUnusedLibraries QuickInfo

Unloads any DLLs that have been loaded with the *bAutoFree* parameter of the **CoLoadLibrary** function set to TRUE, but are no longer in use.

void CoFreeUnusedLibraries();

Comments

Applications can call **CoFreeUnusedLibraries** periodically to free resources. It is most efficient to call it either at the top of a message loop or in some idle-time task. **CoFreeUnusedLibraries** internally calls **DllCanUnloadNow** for DLLs that implement and export that function.

See Also

CoFreeLibrary, **CoFreeUnusedLibraries**, **CoLoadLibrary**, **DLLCanUnloadNow**

CoGetClassObject QuickInfo

Provides the caller with a pointer to an interface on a class object associated with a specified CLSID. The **CoGetClassObject** function transparently locates, and if necessary, dynamically loads the executable code required to do this.

Call **CoGetClassObject** when you want to create multiple objects through a class object for which there is a CLSID in the system registry. You would then call **IClassFactory::CreateInstance** to create an uninitialized object. It is not always necessary to go through this process. To create a single object, call instead the **CoCreateInstance** function, which encapsulates connecting to the class object, creating the instance, and releasing the class object. OLE also provides many other ways to create an object in the form of numerous helper functions and interface member functions whose function is to create objects of a single type and provide a pointer to an interface on that object.

```
STDAPI CoGetClassObject(
  REFCLSID rclsid,              //CLSID associated with the class object
  DWORD dwClsContext,           //Context for running executable code
  LPVOID pvReserved,            //Must be NULL
  REFIID riid,                  //Interface identifier
  LPVOID * ppv                  //On return, pointer to location of interface pointer
 );
```

Parameters

rclsid
Specifies the CLSID that will associate the correct data and code.

dwClsContext
Specifies the context in which the executable code is to be run. For information on the context values and their use, see the **CLSCTX** enumeration.

pvReserved
Reserved for future use. Must be NULL.

riid
Specifies the interface to be used to communicate with the class object. This interface is almost always **IClassFactory** (represented by the *riid* **IID_IClassFactory**).

ppv
Points to where the resulting interface pointer is to be stored.

Return Values

S_OK
Location and connection to the specified class object was successful.

REGDB_E_CLASSNOTREG
CLSID is not properly registered.

E_NOINTERFACE
The object pointed to by *ppv* does not support the interface identified by *riid*.

REGDB_E_READREGDB
Error reading the registration database.

CO_E_DLLNOTFOUND
In process DLL or handler DLL not found (depends on context).

CO_E_APPNOTFOUND
EXE not found (CLSCTX_LOCAL_SERVER only).

E_ACCESSDENIED
General access failure (returned from **LoadLib/CreateProcess**).

CO_E_ERRORINDLL

EXE has error in image.

CO_E_APPDIDNTREG
EXE was launched, but it didn't register class object (may or may not have shutdown).

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

E_NOINTERFACE
The **QueryInterface** on the class object returned E_NOINTERFACE.

Comments

A class object in OLE is an intermediate object that supports an interface that permits operations common to a group of objects. The objects in this group are instances derived from the same object definition represented by a single CLSID. Usually, the interface on a class object is **IClassFactory**, through which you can create object instances of a given definition (class).

A call to **CoGetClassObject** creates, initializes, and gives the caller access (through an interface specified with the *riid* parameter) to the class object. The class object is the one associated with the CLSID that you specify in the *rclsid* parameter. Details of how code and data are located and associated are transparent to the caller, as is the dynamic loading of associated code not already loaded. There are two places to find a CLSID for a given object definition:

- The registry holds an association between CLSIDs and file suffixes, and between CLSIDs and file signatures for determining the class of an object.
- When an object is saved to persistent storage, its CLSID is stored with its data.

To create and initialize embedded or linked OLE document objects, it is not necessary to call **CoGetClassObject** directly. Instead, call one of the **OleCreate** or **OleCreate***Xxx* helper functions. These functions encapsulate the entire object instantiation and initialization process, and call, among other functions, **CoGetClassObject**.

The *riid* parameter specifies the interface the client will use to communicate with the class object. In most cases, this interface is **IClassFactory**. This provides access to the **IClassFactory::CreateInstance** member function, through which the caller can then create an uninitialized object of the kind specified in its implementation. All object definitions registered in the system with a CLSID must implement **IClassFactory**.

In rare cases, however, you may want to specify some other interface that defines operations common to a set of objects. For example, in the way OLE implements monikers, the interface on the class object is **IParseDisplayName**, used to transform the display name of an object into a moniker.

The *dwClsContext* parameter specifies the execution context, allowing one CLSID to be associated with different pieces of code in different execution contexts. The **CLSCTX** enumeration , defined in COMPOBJ.H, specifies the available context flags. **CoGetClassObject** consults (as appropriate for the context indicated) both the registration database and the class objects that are currently registered by calling the **CoRegisterClassObject** function.

See Also

CoRegisterClassObject, **CoRevokeClassObject**, **OleLoad**, **CLSCTX**

CoGetCurrentProcess QuickInfo

Returns a value that is unique to the current thread. It can be used to avoid PROCESSID reuse problems.

DWORD CoGetCurrentProcess();

Return Value

DWORD value
    Indicates a unique value for the current thread that can be used to avoid PROCESSID reuse problems.

Comments

The **CoGetCurrentProcess** function returns a value that is effectively unique, because it is not used again until 2 (32) more threads have been created on the current workstation or until the workstation is rebooted.

Using the value returned from a call to **CoGetCurrentProcess** can help you maintain tables that are keyed by threads or in uniquely identifying a thread to other threads or processes.

Using the value returned by **CoGetCurrentProcess** is more robust than using the HTASK task handle value returned by the Win32 function **GetCurrentTask**, because Windows task handles can be reused relatively quickly when a window's task dies.

CoGetInterfaceAndReleaseStream

Unmarshals a buffer containing an interface pointer and releases the stream when an interface pointer has been marshalled from another thread to the calling thread.

```
HRESULT CoGetInterfaceAndReleaseStream(
   LPSTREAM * ppStm,              //Points to the stream from which the object is to be marshalled
   REFIID riid,                   //The interface requested from the unmarshalled object
   LPVOID * ppv                   //On return, location of the interface pointer
  );
```

Parameters

pstm
   Points to the stream containing the interface to be unmarshaled.

riid
   Specifies the interface requested from the unmarshaled object.

ppv
   [Out] On return, points to the location of the unmarshaled interface pointer.

Return Values

S_OK
   Indicates the output interface was unmarshaled and the stream was released.

E_INVALIDARG
   Indicates that input arguments are invalid.

Other results from **CoUnmarshalInterface**

Comments

CoGetInterfaceAndReleaseStream is a companion helper API to **CoMarshalInterThreadInterfaceInStream**, reversing the result of the latter.   It unmarshals the buffer and releases the stream that contained the interface. If the unmarshaling fails, the stream is still released because there is no effective way to recover from a failure of this kind.

See Also

CoMarshalInterThreadInterfaceInStream

CoGetMalloc QuickInfo

Retrieves the default OLE task memory allocator (a pointer to the system implementation of the **IMalloc** interface) so applications can use it to manage memory.

```
HRESULT CoGetMalloc(
   DWORD dwMemContext,        //Indicates if memory is private or shared
   LPMALLOC * ppMalloc        //Receives pointer to memory allocator on return
  );
```

Parameters

dwMemContext
Indicates this is reserved; the value must be 1.

ppMalloc
Receives a pointer to the memory allocator on return.

Return Values

S_OK
Indicates the allocator was retrieved successfully.

E_INVALIDARG
Indicates either *dwMemContext* is not equal to 1, or *ppMalloc* is an invalid out pointer.

E_OUTOFMEMORY
Inidcates pointer wasn't returned because the system is out of memory.

Comments

The pointer to the **IMalloc** interface pointer received through the *ppMalloc* parameter cannot be called from a remote process—each process must have its own allocator.

See Also

IMalloc, **CoTaskMemAlloc**

CoGetMarshalSizeMax QuickInfo

Retrieves an upper bound on the number of bytes needed to marshal the specified interface pointer on the specified object.

```
STDAPI CoGetMarshalSizeMax(
    ULONG *pulSize,              //Receives maximum size
    REFIID riid,                 //Identifies interface to be marshalled
    IUnknown * pUnk,             //Interface pointer to be marshaled
    DWORD dwDestContext,         //Specifies destination process
    LPVOID pvDestContext,        //Reserved for future use
    DWORD mshlflags              //Specifies reason for marshalling
  );
```

Parameters

pulSize
Receives the maximum marshal size; a value of zero means that the size of the data is unknown.

riid
Specifies the IID of the interface pointer to be marshalled. This interface must be derived from the **IUnknown** interface.

pUnk
Indicates the interface pointer to be marshalled; can be NULL.

dwDestContext
Specifies the destination context, that is, the process in which the unmarshaling will be done. The legal values for *dwDestContext* are taken from the enumeration **MSHCTX**.

pvDestContext
Indicates reserved for use with future **MSHCTX** values. This must be set to NULL.

mshlflags
Specifies why marshalling is taking place. The legal values for *mshlflags* are taken from the enumeration **MSHLFLAGS**.

Return Values

S_OK
Indicates the upper bound was returned successfully.

CO_E_NOTINITIALIZED
Indicates the **CoInitialize** or **OleInitialize** function was not called on the current thread before this function was called.

E_UNEXPECTED
Indicates an unexpected error occurred.

Comments

You typically call this function before calling the **CoMarshalInterface** function. See the description of that function for information on calling these functions.

See Also

CoMarshalInterface, **IMarshal::GetMarshalSizeMax**

CoGetStandardMarshal QuickInfo

Returns an **IMarshal** pointer to the system's default marshalling implementation. Use this implementation to create a proxy for the specified interface pointer in another process.

```
STDAPI CoGetStandardMarshal(
    REFIID riid,                  //Identifies interface being marshalled
    IUnknown * pUnk,              //Interface pointer being marshalled
    DWORD dwDestContext,          //Specifies destination process
    LPVOID pvDestContext,         //Reserved for future use
    DWORD mshlflags,              //Specifies reason for marshalling
    LPMARSHAL * ppMarshal         //Receives pointer to standard marshaller
  );
```

Parameters

riid
: Specifies the IID of the interface pointer to be marshalled. This interface must be derived from the **IUnknown** interface.

pUnk
: Indicates interface pointer to be marshalled. This interface pointer does not have to have the same IID specified by the *riid* parameter; it can be a pointer to any **IUnknown**-derived interface on the object. The standard marshaller will call the **QueryInterface** function on this pointer requesting the interface specified by *riid*.

dwDestContext
: Specifies the destination context, that is, the process in which the unmarshalling will be done. The legal values for *dwDestContext* are taken from the enumeration **MSHCTX**.

pvDestContext
: Reserved for use with future **MSHCTX** values.

mshlflags
: Specifies why marshalling is taking place. The legal values for *mshlflags* are taken from the enumeration **MSHLFLAGS**.

ppMarshal
: Receives the standard marshaller.

Return Values

S_OK
: Indicates that the **IMarshal** instance was returned successfully.

CO_E_NOTINITIALIZED
: Indicates that the **CoInitialize** or **OleInitialize** function was not called on the current thread before this function was called.

E_OUTOFMEMORY
: Out of memory.

E_INVALIDARG
: Indicates one or more arguments are invalid.

E_UNEXPECTED
: Indicates an unexpected error occurred.

E_FAIL
: Indicates an unspecified error.

Comments

You typically do not need to call this function. If you are performing custom marshalling (that is, writing your own implementation of **IMarshal**), you can delegate certain destination contexts to the standard marshaller. You should delegate any destination contexts that you don't fully understand or don't want to handle. The standard marshaller returned by this function is the one used by OLE when an object does not support the **IMarshal** interface.

See Also

IMarshal

CoGetTreatAsClass [QuickInfo]

Returns the CLSID of an object that can emulate the specified object.

```
HRESULT CoGetTreatAsClass(
    REFCLSID clsidOld,        //CLSID of object that is being emulated
    LPCLSID pclsidNew         //Points to location of CLSID for object that can emulate clsidOld
  );
```

Parameters

clsidOld
Specifies the CLSID of the object that can be emulated or treated as an object with a different CLSID.

pclsidNew
Points to the location where the CLSID that can emulate *clsidOld* objects is retrieved. This parameter cannot be NULL. If there is no emulation information for *clsidOld* objects, then the *clsidOld* parameter is returned.

Return Values

S_OK
Indicates that a new CLSID was successfully returned.

S_FALSE
Indicates that there is no emulation information for the *clsidOld* parameter and that the **pclsidNew* parameter is set to *clsidOld*.

REGDB_E_READREGDB
Indicates an error reading the registration database.

See the **CLSIDFromString** function for other possible errors.

Comments

This function returns the **TreatAs** entry in the registration database for the specified object. If there is no **TreatAs** entry for the specfied object, this function returns the CLSID of the original object (*clsidOld*).

Use this function to obtain the CLSID of an object application that can emulate the specified object. The end-user can request that a specified object be treated as an object of a different class, or a setup program can register one class of objects to be treated as objects of a different class. For example, a spreadsheet application may be able to read and write spreadsheets from a different application. The first application can then emulate the objects of the second by treating them as its own.

In the first case, the end-user makes the request through the Convert To dialog box. For example, the end-user might wish to edit a spreadsheet created by one application using a different application that can read and write the spreadsheet format of the original application. In the second case, an application's setup program can specify to activate objects of one class as objects of a different class. For example, when the application is updated, the objects created with the earlier version can be activated and treated as objects of the new version.

In either case, the object application or the setup program calls the **CoTreatAsClass** function to set the **TreatAs** entry in the registration database. Subsequently, all objects of the original CLSID are activated and treated as objects of the second CLSID.

See Also

CoTreatAsClass

CoInitialize QuickInfo

The **CoInitialize** function initializes the Component Object Model(COM) library. You must initialize the library before you can call its functions. Applications must call **CoInitialize** before they make any other COM library calls with two exceptions: the **CoGetMalloc** function, and memory allocation calls.

```
HRESULT CoInitialize(
  LPVOID pvReserved        //Reserved, must be NULL
 );
```

Parameter

pvReserved
In 32-bit OLE, this parameter must be NULL. The 32-bit version of OLE does not support applications replacing OLE's allocator and if the parameter is not NULL, **CoInitialize** returns E_INVALIDARG.

Return Values

S_OK
Indicates the library was initialized successfully.

S_FALSE
Indicates that the library is already initialized or that it could not release default allocator.

E_OUTOFMEMORY
Indicates that it was unable to initialize because the system is out of memory.

E_INVALIDARG
Indicates the argument is invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

Comments

You need to call this before you call any of the OLE library functions unless you call the **OleInitialize** function, which calls **CoInitialize** internally.

Typically, **CoInitialize** is called only once in the process that uses the OLE library. There can be multiple calls, but subsequent calls return S_FALSE. To close the library gracefully, each successful call to **CoInitialize**, including those that return S_FALSE, *must* be balanced by a corresponding call to the **CoUninitialize** function.

See Also

CoInitialize, **OleInitialize**

CoIsHandlerConnected QuickInfo

Determines if a remote object is connected to its corresponding object.

```
BOOL CoIsHandlerConnected(
    LPUNKNOWN pUnk          //Points to the remoted object
  );
```

Parameter

pUnk
Specifies the object in question.

Return Values

TRUE
Indicates the object either is not remote or is remote and is still valid (it is still connected to its remote handler).

FALSE
Indicates the object is remote and is invalid (no longer connected to its remote handler).

Comments

The **CoIsHandlerConnected** function determines the validity of a remote object specified by a pointer to its controlling unknown interface (the *pUnk* parameter). If the specified remote object is remote and is still connected to its remote handler, or is not a remote object, the function returns TRUE. If the specified object is still remote but is not connected to its remote handler, the function returns FALSE, and the caller should respond by releasing the object.

CoIsOle1Class QuickInfo

Determines if a given CLSID represents an OLE 1 object. This is one of several OLE compatibility functions. You can use these functions to determine if an object class is from OLE 1 and to convert objects between OLE 1 and OLE 2 storage formats.

```
BOOL CoIsOle1Class(
   REFCLSID rclsid,          //CLSID to check
  );
```

Parameter

rclsid
Indicates the reference to the CLSID to check.

Return Values

S_TRUE
Indicates that the CLSID refers to an OLE 1 object.

S_FALSE
Indicates that the CLSID does not refer to an OLE 1 object.

Comments

The **CoIsOle1Class** function is useful for preventing linking to embedded OLE 1 objects within a container. Once a container has determined that copied data represents an embedded object, **CoIsOle1Class** can be called to determine whether the embedded object is an OLE 1 object. If **CoIsOle1Class** returns S_TRUE, the container does not offer Link Source.

See Also

OleConvertIStorageToOLESTREAM, **OleConvertIStorageToOLESTREAMEx**, **OleConvertOLESTREAMToIStorage**, **OleConvertOLESTREAMToIStorageEx**

CoLoadLibrary QuickInfo

Loads a specific DLL into the caller's process. The **CoGetClassObject** function calls **CoLoadLibrary** internally; applications should not call it directly.

```
HINSTANCE CoLoadLibrary(
   LPOLESTR lpszLibName,          //Points to the name of the library to be loaded
   BOOL bAutoFree                 //Indicates whether library is automatically freed
  );
```

Parameters

lpszLibName
Points to the name of the library to be loaded. The use of this name is the same as in the Win32 function **LoadLibrary**.

bAutoFree
If TRUE, indicates that this library is freed when it is no longer needed, through a call to either the **CoFreeUnusedLibraries** or **CoUninitialize** functions. If FALSE, the library should be explicitly freed with the **CoFreeLibrary** function.

Return Values

Module Handle
Indicates handle of the loaded library.

NULL
Indicates library could not be loaded.

Comments

The **CoLoadLibrary** function is called internally by the **CoGetClassObject** function when the class context (CLSCTX) indicates a DLL. **CoLoadLibrary** loads a DLL specified by the *lpszLibName* parameter into the process that called **CoGetClassObject**. Containers should not call **CoLoadLibrary** directly.

Internally, a reference count is kept on the loaded DLL, by using **CoLoadLibrary** to increment the count and the **CoFreeLibrary** function to decrement it.

See Also

CoFreeAllLibraries, **CoFreeLibrary**, **CoFreeUnusedLibraries**, **CoGetClassObject**

CoLockObjectExternal QuickInfo

Called either to lock an object to ensure that it stays in memory, or to release such a lock. Call **CoLockObjectExternal** to place a strong lock on an object to ensure that it stays in memory.

```
STDAPI CoLockObjectExternal(
  IUnknown * pUnk,              //Pointer to object to be locked or unlocked
  BOOL fLock,                   //TRUE = lock, FALSE = unlock
  BOOL fLastUnlockReleases      //TRUE = release all pointers to object
 );
```

Parameters

pUnk
Points to the object (through its **IUnknown** interface) to be locked or unlocked.

fLock
Specifies whether the object is to be locked or released. Specifying TRUE holds a reference to the object (keeping it in memory), locking it independently of external or internal **AddRef/Release** operations, registrations, or revocations. If *fLock* is TRUE, *fLastLockReleases* is ignored. FALSE releases a lock previously set with a call to this function.

fLastUnlockReleases
Specifies whether a given lock is the last reference that is supposed to keep an object alive. If it is, TRUE releases all pointers to the object (there may be other references that are not supposed to keep it alive).

Return Values

S_OK
Indicates the object was locked successfully.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Indicates one or more arguments are invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

Comments

The **CoLockObjectExternal** function prevents the reference count of an object from going to zero, thereby "locking" it into existence until the lock is released. The same function (with different parameters) releases the lock. The lock is implemented by having the system call **IUnknown::AddRef** on the object. The system then waits to call **IUnknown::Release** on the object until a later call to **CoLockObjectExternal** with *fLock* set to FALSE. This function can be used to maintain a reference count on the object on behalf of the end-user, because it acts outside of the object, as does the user.

The **CoLockObjectExternal** function *must* be called in the process in which the object actually resides (the EXE process, not the process in which handlers may be loaded).

Calling **CoLockObjectExternal** sets a strong lock on an object. A strong lock keeps an object in memory, while a weak lock does not. Strong locks are required, for example, during a silent update to an OLE embedding. The embedded object's container must remain in memory until the update process is complete. There must also be a strong lock on an application object to ensure that the application stays alive until it has finished providing services to its clients. All external references place a strong reference lock on an object.

The **CoLockObjectExternal** function is typically called in the following situations:

- Object applications should call **CoLockObjectExternal** with both *fLock* and *fLastLockReleases* set to TRUE when they become visible. This call creates a strong lock on behalf of the user. When the application is closing, free the lock with a call to **CoLockObjectExternal**, setting *fLock* to FALSE and *fLastLockReleases* to TRUE.

- A call to **CoLockObjectExternal** on the application object can also be used in the implementation of **IOleContainer::LockContainer**.

There are several things to be aware of when you use **CoLockObjectExternal** in the implementation of **IOleContainer::LockContainer**. An embedded object would call **IOleContainer::LockContainer** on its container to keep it running (to lock it) in the absence of other reasons to keep it running. When the embedded object becomes visible, the container must weaken its connection to the embedded object with a call to the **OleSetContainedObject** function, so other connections can affect the object.

Unless an application manages all aspects of its application and document shutdown completely with calls to **CoLockObjectExternal**, the container must keep a private lock count in **IOleContainer::LockContainer** so that it exits when the lock count reaches zero and the container is invisible. Maintaining all aspects of shutdown, and thereby avoiding keeping a private lock count, means that **CoLockObjectExternal** should be called whenever one of the following conditions occur:

- A document is created and destroyed or made visible or invisible.
- An application is started and shut down by the user.
- A pseudo-object is created and destroyed.

For debugging purposes, it may be useful to keep a count of the number of external locks (and unlocks) set on the application.

Note The end-user has explicit control over the lifetime of an application, even if there are external locks on it. That is, if a user decides to close the application (File, Exit), it *must* shut down. In the presence of external locks (such as the lock set by **CoLockObjectExternal**), the application can call the **CoDisconnectObject** function to force these connections to close prior to shutdown.

See Also

IOleContainer::LockContainer, **OleSetContainedObject**

CoMarshalHresult QuickInfo

Marshals an HRESULT to the specified stream so it can be unmarshaled using the **CoUnmarshalHresult** function.

```
STDAPI CoMarshalHresult(
    IStream * pStm,           //Stream to be used for marshalling
    HRESULT hresult           //HRESULT to be marshalled
  );
```

Parameters

pStm
Points to the stream used for marshalling.

hresult
Specifies the HRESULT in the originating process.

Return Values

S_OK
Indicates the HRESULT was marshalled successfully.

STG_E_INVALIDPOINTER
Indicates bad pointer passed in for *pStm*.

STG_E_MEDIUMFULL
Indicates the medium is full.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Indicates one or more arguments are invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

Comments

You typically do not need to call this function. If you are performing custom marshalling (that is, writing your own implementation of **IMarshal**) and you need to marshal an HRESULT from one process to another, either as a parameter or a return code, you must call this function. An HRESULT is process-specific, so an HRESULT valid in one process might not be valid in another.

Note that the HRESULT returned by **CoMarshalHresult** indicates the success or failure of the marshalling process and is unrelated to the HRESULT parameter.

See Also

CoUnmarshalHresult

CoMarshalInterface QuickInfo

Marshals an interface pointer into the specified stream. Marshalling consists of writing the appropriate data into the stream so the **CoUnmarshalInterface** function can use that data to initialize a proxy object in another process. **CoMarshalInterface** can marshal only interfaces derived from **IUnknown**.

```
STDAPI CoMarshalInterface(
    IStream *pStm,              //Stream used for marshalling
    REFIID riid,                //Identifies interface being marshalled
    IUnknown *pUnk,             //Interface pointer being marshalled
    DWORD dwDestContext,        //Specifies destination process
    void *pvDestContext,        //Reserved for future use
    DWORD mshlflags             //Specifies reason for marshalling
  );
```

Parameters

pStm
Points to the stream to be used during marshalling.

riid
Specifies the IID of the interface pointer being marshaled. This interface must be derived from the **IUnknown** interface.

pUnk
Points to the interface pointer to be marshalled.

dwDestContext
Specifies the destination context, which is the process in which the unmarshalling will be done. The legal values for *dwDestContext* are taken from the enumeration **MSHCTX**.

pvDestContext
Indicates reserved for future use; must be set to zero by the caller.

mshlflags
Specifies why marshalling is taking place. The legal values for *mshlflags* are taken from the **MSHLFLAGS** enumeration.

Return Values

S_OK
Indicates the interface pointer was marshalled successfully.

CO_E_NOTINITIALIZED
Indicates the **CoInitialize** or **OleInitialize** function was not called on the current thread before this function was called.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Indicates one or more arguments are invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

E_FAIL
Indicates an unspecified error.

IStream errors
See the **IStream** interface for information on possible stream access errors.

Comments

The main steps that **CoMarshalInterface** performs are the following:

1. The function checks whether custom marshaling is supported by calling the **QueryInterface** function on *pUnk* and asking for an **IMarshal** interface pointer. If the object does not support custom marshalling, the function gets an **IMarshal** interface pointer to the system's standard marshalling implementation.
2. Using whichever **IMarshal** interface pointer it has acquired, the function calls **IMarshal::GetUnmarshalClass** to get the CLSID of the unmarshaller (the proxy to be created). Then it writes the CLSID to the stream.
3. The function calls **IMarshal::MarshalInterface** to marshal the interface pointer.

You typically do not need to call this function. OLE calls it from within interface proxies or stubs that marshal an interface pointer.

The only situation in which you might need to call this function is if you are performing custom marshalling (that is, writing your own implementation of **IMarshal**). Your implementation of **IMarshal::MarshalInterface** writes to a stream the data needed to initialize a custom proxy in the destination context. If your custom proxy will need access to a private object, you can include an interface pointer to that object as part of the data you write to the stream. In such situations, if you want to use standard marshalling when passing the interface pointer, you can call **CoMarshalInterface** to do so; this is an example of using standard marshaling within your implementation of custom marshaling.

See Also

CoUnmarshalInterface, **IMarshal::MarshalInterface**

CoMarshalInterThreadInterfaceInStream

Passes streams containing interfaces between different threads, and is a helper that allows you to pass interface pointers easily and correctly between threads.

```
HRESULT CoMarshalInterThreadInterfaceInStream(
   REFIID riid,                    //Interface exposed by the object to be marshaled
   LPUNKNOWN pUnk,                 //Points to the object to be marshaled
   LPSTREAM * ppStm                //Output pointer to the stream interface
  );
```

Parameters

riid
Specifies the interface that is to be marshalled.

pUnk
Points to the **IUnknown** of the object to be marshalled, thus specifying the object.

ppStm
[Out] On return, points to the stream containing the interface .

Return Values

S_OK
Output stream contains marshalled interface.

E_OUTOFMEMORY
There was not enough memory to complete the call.

E_INVALIDARG
One or more arguments are invalid.

Comments

CoMarshalInterThreadInterfaceInStream is a helper function that facilitates passing streams containing interfaces between threads, although you could implement it yourself from existing APIs if desired. It creates a stream and passes it to **CoMarshalInterface**, then passes the pointer to that stream back to the caller. The stream it returns is guaranteed to behave correctly when different threads access it. The receiving thread can then call the corresponding helper function **CoGetInterfaceAndReleaseStream** to get the interface pointer and release the stream object.

See Also

CoGetInterfaceAndReleaseStream

CoRegisterClassObject QuickInfo

Registers an EXE class object with OLE so that other applications can connect to it. EXE object applications should call **CoRegisterClassObject** on startup. It can also be used to register internal objects for use by the same EXE or other code (such as DLLs) that the EXE uses.

```
STDAPI CoRegisterClassObject(
    REFCLSID rclsid,              //Class identifier to be registered
    IUnknown * pUnk,              //Pointer to the class object
    DWORD dwClsContext,           //Context for running executable code
    DWORD flags,                  //How to connect to the class object
    LPDWORD * lpdwRegister        //Pointer to the value returned
  );
```

Parameters

rclsid
Specifies the CLSID registered.

pUnk
Points to the class object whose availability is being published.

dwClsContext
Specifies the context in which the executable code is to be run. For information on these context values, see the **CLSCTX** enumeration.

flags
Determines how connections are made to the class object. For information on these flags, see the **REGCLS** enumeration.

lpdwRegister
Points to the value returned by **CoRegisterClassObject** that identifies the class object; later used by the **CoRevokeClassObject** function to revoke the registration.

Return Values

S_OK
Indicates the class object was registered successfully.

CO_E_OBJISREG
Indicates already registered in the class object table.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Indicates one or more arguments are invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

Comments

Only EXE object applications call **CoRegisterClassObject**. Servers that can create and support more than one kind of object (such as multiple types of embeddable objects) must also register each of class of object it supports. Object handlers or DLL object applications do not call this function; instead, they must implement and export the **DllGetClassObject** function.

When a multiple-use EXE object application is started, it must create a class object with the **IClassFactory** interface on it and register it with a call to **CoRegisterClassObject**. Object applications that support several different object definitions must allocate a different class object for each.

Multiple registrations of the same class object are independent and do not produce an error. Each subsequent registration yields a unique key in *lpdwRegister*.

Multiple document interface (MDI) applications must register their class objects. Single document interface (SDI) applications must register their class objects only if they can be started by means of the **/Embedding** switch.

The server for a class object should call **CoRevokeClassObject** to revoke the class object when all of the following are true:

- There are no existing instances of the object definition
- There are no locks on the class object
- The application providing services to the class object is not under user control (not visible to the user on the display).

After the class object is revoked, when its reference count reaches zero, the class object can be destroyed, allowing the application to exit.

For information on the *flags* parameter, refer to the **REGCLS** enumeration.

See Also

CoGetClassObject, **CoRevokeClassObject**, **DllGetClassObject**, **REGCLS**, **CLSCTX**

CoRegisterMallocSpy

Registers an implementation of the **IMallocSpy** interface in OLE, thereafter requiring OLE to call its wrapper methods around every call to the corresponding **IMalloc** method. **IMallocSpy** is defined in OLE to allow debugging of memory allocations.

```
HRESULT CoRegisterMallocSpy(
  LPMALLOCSPY pMallocSpy        //Pointer to an instance of IMallocSpy
 );
```

Parameter

pMallocSpy
Points to an instance of the **IMallocSpy** implementation.

Return Values

S_OK
Indicates the **IMallocSpy** object is successfully registered.

CO_E_OBJISREG
Indicates there is already a registered spy.

E_INVALIDARG
Indicates the call to the **QueryInterface** function in *pMallocSpy* for **IID_IMallocSpy** is unsuccessful, or *pMallocSpy* is NULL.

Comments

The **CoRegisterMallocSpy** function registers the **IMallocSpy** object, which is used to debug calls to **IMalloc** methods. The function calls **QueryInterface** on the pointer **pMallocSpy** for the interface **IID_IMallocSpy**. This is to ensure that *pMallocSpy* really supports an implementation of *IMallocSpy*. By the rules of OLE, it is expected that a successful call to **QueryInterface** has added a reference (through the **AddRef** function) to the **IMallocSpy** object. That is, **CoRegisterMallocSpy** does not directly call **AddRef** on *pMallocSpy*, but fully expects that the **QueryInterface** call will.

When the **IMallocSpy** object is registered, whenever there is a call to one of the **IMalloc** methods, OLE first calls the corresponding **IMallocSpy** pre-method. Then, after executing the **IMalloc** method, OLE calls the corresponding **IMallocSpy** post-method. For example, whenever there is a call to **IMalloc::Alloc**, from whatever source, OLE calls **IMallocSpy::PreAlloc**, calls **IMalloc::Alloc**, and after that allocation is completed, calls **IMallocSpy::PostAlloc**.

See Also

IMallocSpy, **CoRevokeMallocSpy**, **CoGetMalloc**

CoRegisterMessageFilter QuickInfo

Registers with OLE the instance of an EXE application's **IMessageFilter** interface, which is to be used for handling concurrency issues. DLL object applications cannot register a message filter.

```
HRESULT CoRegisterMessageFilter(
   LPMESSAGEFILTER lpMessageFilter,          //IMessageFilter interface supplied by app
   LPMESSAGEFILTER * lplpMessageFilter       //Prior IMessageFilter instance if non-NULL
  );
```

Parameters

lpMessageFilter
Points to an **IMessageFilter** interface supplied by the application. Can be NULL, indicating that the current **IMessageFilter** registration should be revoked.

lplpMessageFilter
If this parameter is non-NULL, returns a pointer to the previously registered **IMessageFilter** instance. If NULL, indicates no previous **IMessageFilter** instance was registered.

Return Values

S_OK
Indicates the **IMessageFilter** instance registered or revoked successfully.

S_FALSE
Indicates an error registering or revoking **IMessageFilter** instance.

CoReleaseMarshalData QuickInfo

Destroys a previously marshalled data packet.

```
STDAPI CoReleaseMarshalData(
    IStream * pStm          //Stream containing data packet
  );
```

Parameter

pStm
Points to the stream that contains the data packet which is to be destroyed.

Return Values

S_OK
Indicates the data packet was successfully destroyed.

STG_E_INVALIDPOINTER
Indicates an **IStream** error dealing with the *pStm* parameter.

CO_E_NOTINITIALIZED
Indicates that the **CoInitialize** or **OleInitialize** function was not called on the current thread before this function was called.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Indicates *pStm* is invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

E_FAIL
Indicates an unspecified error.

Comments

The main steps that **CoReleaseMarshalData** performs are the following:

1. The function reads a CLSID from the stream.
2. If standard marshalling is being used, the function gets an **IMarshal** pointer to an instance of the standard unmarshaller. If custom marshalling is being used, the function creates a proxy by calling the **CoCreateInstance** function, passing the CLSID it read from the stream, and requests an **IMarshal** interface pointer to the newly created proxy.
3. Using whichever **IMarshal** interface pointer it has acquired, the function calls **IMarshal::ReleaseMarshalData**.

You typically do not call this function. The only situation in which you might need to call this function is if you use custom marshalling (write and use your own implementation of **IMarshal**). Examples of when **CoReleaseMarshalData** should be called include the following situations:

- An attempt was made to unmarshal the data packet, but it failed.
- A marshalled data packet was removed from a global table.

As an analogy, the data packet can be thought of as a reference to the original object, just as if it were another interface pointer being held on the object. Like a real interface pointer, that data packet must be released at some point. The use of **IMarshal::ReleaseMarshalData** to release data packets is analogous to the use of **IUnknown::Release** to release interface pointers.

Note that you do not need to call **CoReleaseMarshalData** after a successful call of the **CoUnmarshalInterface** function; that function releases the marshal data as part of the processing that it does.

See Also

IMarshal::ReleaseMarshalData

CoRevokeClassObject QuickInfo

Informs OLE that a class object, previously registered with the **CoRegisterClassObject** function, is no longer available for use.

```
HRESULT CoRevokeClassObject(
  DWORD dwRegister        //Token on class object returned from call to CoRegisterClassObject
 );
```

Parameter

dwRegister
Specifies the token previously returned from the **CoRegisterClassObject** function.

Return Values

S_OK
Indicates the class object was successfully revoked.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Indicates the *dwRegister* parameter does not map to a registered class object.

E_UNEXPECTED
Indicates an unexpected error occurred.

Comments

If a call to **CoRevokeClassObject** is successful, it means that the class object has been removed from the global class object table. However, if other clients still have pointers to the class object and have caused the reference count to be incremented by calls to **IUnknown::AddRef**, the reference count will not be zero. When this occurs, applications may benefit if subsequent calls (with the obvious exceptions of **IUnknown::AddRef** and **IUnknown::Release**) to the class object fail.

The object application *must* call **CoRevokeClassObject** to revoke registered class objects before exiting the program. Class object implementors should call **CoRevokeClassObject** as part of the release sequence. You must specifically revoke the class object even when you have specified the *flags* value REGCLS_SINGLEUSE in a call to **CoRegisterClassObject**, indicating that only one application can connect to the class object.

See Also

CoGetClassObject, **CoRegisterClassObject**

CoRevokeMallocSpy

Revokes a registered **IMallocSpy** object.

HRESULT CoRevokeMallocSpy();

Return Values

S_OK
: Indicates the **IMallocSpy** object is successfully revoked.

CO_E_OBJNOTREG
: Indicates no spy is currently registered.

E_ACCESSDENIED
: Indicates a spy is registered but there are outstanding allocations (not yet freed) made while this spy was active.

Comments

The **IMallocSpy** object is released when it is revoked. This release corresponds to the call to **IUnknown::AddRef** in the implementation of the **QueryInterface** function by the **CoRegisterMallocSpy** function. The implementation of the **IMallocSpy** interface should then do any appropriate clean-up.

If the return code is E_ACCESSDENIED, there are still outstanding allocations that were done while the spy was active. The registered spy cannot be revoked at this time because it may have attached arbitrary headers and/or trailers to these allocations which only the spy knows about. Only the spy's **PreFree** (or **PreRealloc**) method knows how to account for these headers and trailers. Before returning E_ACCESSDENIED, **CoRevokeMallocSpy** notes internally that a revoke is pending. When the outstanding allocations have been freed, the revoke proceeds automatically, releasing the **IMallocSpy** object. This means that it is necessary to call **CoRevokeMallocSpy** only once for each call to **CoRegisterMallocSpy**, even if E_ACCESSDENIED is returned.

See Also

IMallocSpy, **CoRegisterMallocSpy**, **CoGetMalloc**

CoTaskMemAlloc [QuickInfo]

Allocates a block of task memory in the same way that **IMalloc::Alloc** does.

```
LPVOID CoTaskMemAlloc(
  ULONG cb        //Size in bytes of memory block to be allocated
 );
```

Parameter

cb
Specifies the size (in bytes) of the memory block to be allocated.

Return Values

Allocated memory block
Indicates memory block allocated successfully.

NULL
Indicates insufficient memory available.

Comments

The **CoTaskMemAlloc** function uses the default allocator to allocate a memory block in the same way that **IMalloc::Alloc** does. It is not necessary to call the **CoGetMalloc** function before calling **CoTaskMemAlloc**.

The initial contents of the returned memory block are undefined - there is no guarantee that the block has been initialized. The allocated block may be larger than *cb* bytes because of the space required for alignment and for maintenance information.

If *cb* is zero, **CoTaskMemAlloc** allocates a zero-length item and returns a valid pointer to that item. If there is insufficient memory available, **CoTaskMemAlloc** returns NULL.

Note Applications should always check the return value from this method, even when requesting small amounts of memory, because there is no guarantee the memory will be allocated.

See Also

IMalloc::Alloc, **CoGetMalloc**, **CoTaskMemFree**, **CoTaskMemRealloc**

CoTaskMemFree   QuickInfo

Frees a block of task memory previously allocated through a call to the CoTaskMemAlloc or CoTaskMemRealloc function.

```
void CoTaskMemFree(
    void pv          //Points to memory block to be freed
  );
```

Parameter

pv
Points to the memory block to be freed.

Comments

The **CoTaskMemFree** function, using the default OLE allocator, frees a block of memory previously allocated through a call to the **CoTaskMemAlloc** or **CoTaskMemRealloc** function.

The number of bytes freed equals the number of bytes that were originally allocated or reallocated. After the call, the memory block pointed to by *pv* is invalid and can no longer be used.

Note The *pv* parameter can be NULL, in which case this method has no effect.

See Also

CoTaskMemAlloc, **CoTaskMemRealloc**, **CoGetMalloc**, **IMalloc::Free**

CoTaskMemRealloc QuickInfo

Changes the size of a previously allocated block of task memory.

```
LPVOID CoTaskMemRealloc(
   LPVOID pv,          //Points to memory block to be reallocated
   ULONG cb            //Size in bytes of block to be reallocated
  );
```

Parameters

pv
Points to the memory block to be reallocated. It can be a NULL pointer, as discussed in the Comments.

cb
Specifies the size in bytes of the memory block to be reallocated. It can be zero, as discussed in the following remarks.

Return Values

Reallocated memory block
Indicates memory block successfully reallocated.

NULL
Indicates insufficient memory or *cb* is zero and *pv* is not NULL.

Comments

The **CoTaskMemRealloc** function changes the size of a previously allocated memory block in the same way that **IMalloc::Realloc** does. It is not necessary to call the **CoGetMalloc** function to get a pointer to the OLE allocator before calling **CoTaskMemRealloc**.

The *pv* argument points to the beginning of the memory block. If *pv* is NULL, **CoTaskMemRealloc** allocates a new memory block in the same way as the **CoTaskMemAlloc** function. If *pv* is not NULL, it should be a pointer returned by a prior call to **CoTaskMemAlloc**.

The *cb* argument specifies the size (in bytes) of the new block. The contents of the block are unchanged up to the shorter of the new and old sizes, although the new block can be in a different location. Because the new block can be in a different memory location, the pointer returned by **CoTaskMemRealloc** is not guaranteed to be the pointer passed through the *pv* argument. If *pv* is not NULL and *cb* is zero, then the memory pointed to by *pv* is freed.

CoTaskMemRealloc returns a void pointer to the reallocated (and possibly moved) memory block. The return value is NULL if the size is zero and the buffer argument is not NULL, or if there is not enough memory available to expand the block to the given size. In the first case, the original block is freed; in the second, the original block is unchanged.

The storage space pointed to by the return value is guaranteed to be suitably aligned for storage of any type of object. To get a pointer to a type other than **void**, use a type cast on the return value.

See Also

CoTaskMemAlloc, **CoTaskMemFree**, **CoGetMalloc**, **IMalloc::Realloc**

CoTreatAsClass QuickInfo

Establishes or removes an emulation so objects of one class are treated as objects of a different class.

```
STDAPI CoTreatAsClass(
   REFCLSID clsidOld,        //CLSID for the original object to be emulated
   REFCLSID clsidNew         //CLSID for the new object that emulates the original
  );
```

Parameters

clsidOld
Specifies the CLSID of the object to be emulated.

clsidNew
Specifies the CLSID of the object that should emulate the original object. This replaces any existing emulation for *clsidOld*. Can be CLSID_NULL, in which case any existing emulation for *clsidOld* is removed.

Return Values

S_OK
Indicates the emulation was successfully established or removed.

REGDB_E_CLASSNOTREG
Indicates the *clsidOld* parameter is not properly registered in the registration database.

REGDB_E_READREGDB
Indicates error reading from registration database.

REGDB_E_WRITEREGDB
Indicates error writing to registration database.

E_INVALIDARG
Indicates the *clsidOld* parameter specifies an OLE 1 class.

Comments

This function sets the **TreatAs** entry in the registration database for the specified object. After this entry is set, the **CoGetClassObject** function and object handlers (including the default handler) are transparently forwarded to the new CLSID when they consult the registration database for the old CLSID. For example, launching the object application for *clsidOld* would actually launch the object application for *clsidNew* instead. The **IOleObject::EnumVerbs** method in the default handler enumerates the verbs from *clsidNew* instead of *clsidOld*.

During installation, setup programs should call **CoTreatAsClass**, setting the *clsidNew* parameter to CLSID_NULL to remove any existing emulation for the classes they install.

If you set a class to emulate itself, that is, the values for *clsidNew* and *clsidOld* are the same, this function has two possible results depending on the **AutoTreatAs** key in the registration database. If there is a CLSID assigned to the **AutoTreatAs** key, then the **AutoTreatAs** CLSID is assigned to the **TreatAs** key. If there is no value assigned to the **AutoTreatAs** key, then the **TreatAs** entry is removed.

The **CoTreatAsClass** function does not validate whether an appropriate registration database entry for *clsidNew* currently exists.

See Also

CoGetTreatAsClass

CoUninitialize QuickInfo

Closes the OLE Component Object Model(COM) library, freeing any resources that it maintains and forcing all RPC connections to close.

void CoUninitialize();

Comments

The **CoInitialize** and **CoUninitialize** calls must be balanced - if there are multiple calls to the **CoInitialize** function, there must be the same number of calls to **CoUninitialize**: Only the **CoUninitialize** call corresponding to the **CoInitialize** call that initialized the library can close it.

The **OleUninitialize** function calls **CoUninitialize** internally, so applications that call **OleUninitialize** do not also need to call **CoUninitialize**.

The **CoUninitialize** function should be called on application shutdown, as the last call made to the COM library after the application hides its main windows and falls through its main message loop. If there are open conversations remaining, **CoUninitialize** starts a modal message loop and dispatches any pending messages from the containers or server for this OLE application. By dispatching the messages, **CoUninitialize** ensures that the application does not quit before receiving all of its pending messages. Non-OLE messages are discarded.

See Also

CoInitialize, **OleUninitialize**

CoUnmarshalHresult QuickInfo

Unmarshals an **HRESULT** type from the specified stream.

```
STDAPI CoUnmarshalHresult(
    LPSTREAM pStm,          //Stream used for unmarshalling
    HRESULT * phresult      //Receives the HRESULT
  );
```

Parameters

pStm
Points to the stream from which the **HRESULT** is to be unmarshalled.

phresult
Receives the unmarshalled **HRESULT**.

Return Values

S_OK
Indicates the **HRESULT** was unmarshalled successfully.

STG_E_INVALIDPOINTER
Indicates *pStm* is an invalid pointer.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Indicates one or more arguments are invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

Comments

You typically do not need to call this function. The only situation in which you would need to call this function is if you are performing custom marshalling (that is, writing your own implementation of **IMarshal**). You must use **CoUnmarshalHresult** to unmarshal **HRESULT**s previously marshalled by a call to the **CoMarshalHresult** function.

See Also

CoMarshalHresult

CoUnmarshalInterface QuickInfo

Initializes a proxy object using data written into the stream by a previous call to the **CoMarshalInterface** function, and returns an interface pointer to that proxy object.

```
STDAPI CoUnmarshalInterface(
    IStream * pStm,         //Stream used for unmarshalling
    REFIID riid,            //IID of interface pointer desired
    void ** ppv             //Receives the interface pointer
  );
```

Parameters

pStm
Points to the stream from which the interface pointer is to be unmarshalled.

riid
Specifies the IID of the interface pointer desired.

ppv
Receives the interface pointer.

Return Values

S_OK
Indicates the interface was unmarshalled successfully.

STG_E_INVALIDPOINTER
Indicates *pStm* is an invalid pointer.

CO_E_NOTINITIALIZED
Indicates the **CoInitialize** or **OleInitialize** function was not called on the current thread before this function was called.

CO_E_OBJNOTCONNECTED
Indicates the object application has been disconnected from the remoting system (for example, as a result of a call to the **CoDisconnectObject** function).

REGDB_E_CLASSNOTREG
Indicates an error occurred reading the registration database.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Indicates one or more arguments are invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

E_NOINTERFACE
Indicates the final **QueryInterface** of this function for the requested interface returned E_NOINTERFACE.

E_FAIL
Indicates an unspecified error.

CoCreateInstance errors
Indicates an error occurred when creating the handler (standard or custom marshaling).

Comments

The main steps that **CoUnmarshalInterface** performs are the following:

1. The function reads a CLSID from the stream.
2. If standard marshalling is being used, the function gets an **IMarshal** pointer to an instance of the standard unmarshaller. If custom marshalling is being used, the function creates a proxy by calling

the **CoCreateInstance** function, passing the CLSID it read from the stream, and requests an **IMarshal** interface pointer to the newly created proxy.

3. Using whichever **IMarshal** interface pointer it has acquired, the function then calls **IMarshal::UnmarshalInterface** and, if appropriate, **IMarshal::ReleaseMarshalData**.

You typically do not need to call this function. The primary caller of this function is OLE, from within interface proxies or stubs that unmarshal an interface pointer.

The only place where you might need to call this function is from your implementation of **IMarshal::UnmarshalInterface**, if you are performing custom marshaling. If your implementation of **IMarshal::MarshalInterface** called the **CoMarshalInterface** function to marshal an interface pointer to the stream, you must call **CoUnmarshalInterface** to retrieve that interface pointer. The retrieved interface pointer is typically a pointer to a private object in the originating process.

See Also

CoMarshalInterface, **IMarshal::UnmarshalInterface**

CreateAntiMoniker **QuickInfo**

Creates and returns a new anti-moniker.

WINOLEAPI CreateAntiMoniker(
**LPMONIKER FAR ****ppmk* //Receives the anti-moniker
);

Parameter

ppmk
Receives an **IMoniker** interface pointer to the new anti-moniker. The returned pointer is NULL if an error occurs; if non-NULL, the function has called **IUnknown::AddRef** on the parameter and the caller is responsible for calling **IUnknown::Release**.

Return Values

S_OK
Indicates the anti-moniker was created successfully.

E_OUTOFMEMORY
Indicates insufficient memory.

Comments

You would call this function only if you are writing your own moniker class (implementing the **IMoniker** interface). If you want an anti-moniker to act as the inverse for your moniker class, you can call **CreateAntiMoniker** from your implementation of **IMoniker::Inverse**.

The inverse of a moniker is analogous to the ".." directory in MS-DOS file systems; the ".." directory acts as the inverse to any other directory name, because appending ".." to a directory name results in an empty path. In the same way, the inverse of a moniker typically is also the inverse of all monikers in the same class. However, it is not necessarily the inverse of a moniker of a different class.

An anti-moniker can be used as the inverse for certain classes of monikers. For example, the system-provided implementations of file monikers, item monikers, and pointer monikers all use anti-monikers as their inverse; consequently, an anti-moniker composed to the right of a file, item, or pointer moniker composes to nothing.

If you're a moniker client, you typically do not know the class of the moniker you're using. If you need to get the inverse of a moniker, you should always call **IMoniker::Inverse** rather that the **CreateAntiMoniker** function, because you cannot be certain that the moniker you're using considers an anti-moniker to be its inverse. For example, if you need to remove the last piece of a composite moniker, call **IMoniker::Enum** on the composite, specifying FALSE as the first parameter; this call creates an enumerator that returns the component monikers in reverse order. Use the enumerator to retrieve the last piece of the composite, and call **IMoniker::Inverse** on that moniker. The moniker returned by **IMoniker::Inverse** will remove the last piece of the composite.

If you are writing a new moniker class and you have no special requirements for its inverse, you can use **CreateAntiMoniker** in your implementation of the **IMoniker::Inverse** method, and then check for an anti-moniker in your implementation of **IMoniker::ComposeWith**.

See Also

IMoniker::Inverse, **IMoniker::ComposeWith**, **IMoniker - Anti-Moniker Implementation**

CreateBindCtx QuickInfo

Creates a bind context. A bind context is an object that stores information about a particular moniker-binding operation. A bind context is required as a parameter in many methods of the **IMoniker** interface and in certain functions related to monikers.

```
STDAPI CreateBindCtx(
    DWORD reserved,         //Reserved for future use
    LPBC FAR* ppbc          //Receives the pointer to the bind context
  );
```

Parameters

reserved
Indicates reserved for future use; must be zero.

ppbc
Receives an **IBindCtx** interface pointer to the new bind context. The returned pointer is NULL if an error occurs; if non-NULL, the caller is responsible for calling **IUnknown::Release** on the parameter.

Return Values

S_OK
Indicates the bind context was allocated and initialized successfully.

E_OUTOFMEMORY
Indicates insufficient memory.

Comments

The most common use of **CreateBindCtx** is in binding a moniker (getting a pointer to an interface by ideintifying it through a moniker). This X typically takes the following steps:

1. Create a bind context by calling the **CreateBindCtx** function.
2. Call the **IMoniker::BindToObject** method on the moniker, retrieving an interface pointer.
3. Release the bind context.
4. Use the interface pointer.
5. Release the interface pointer.

Here's a code fragment that illustrates these steps:

```
// pMnk is an IMoniker * that points to a previously-acquired moniker
IFoo *pFoo;
IBindCtx *pbc;

CreateBindCtx( 0, &pbc );
pMnk->BindToObject( pbc, NULL, IID_IFoo, &pFoo );
pbc->Release();
// pFoo now points to the object; safe to use pFoo
pFoo->Release();
```

Bind contexts are also used in other methods of the **IMoniker** interface besides **IMoniker::BindToObject** and in the **MkParseDisplayName** function.

A bind context retains references to the objects that are bound during the binding operation. This causes the bound objects to remain active until the bind context is released. This can improve performance in subsequent binding operations if they are trying to bind to the same objects. However, this potential performance enhancement should be balanced against the costs of keeping the objects activated unnecessarily (i.e., keeping the object's server running).

In general, you can benefit from reusing a bind context for a few sequential bind operations if those operations are likely to require the same objects, but you should release the bind context rather than holding onto it for a long period. If you don't release the bind context, objects registered with the bind context will remain active unnecessarily.

A bind context contains a **BIND_OPTS** structure, which contains parameters that apply to all steps in a binding operations. When you create a bind context using **CreateBindCtx**, the fields of the **BIND_OPTS** structure are initialized to the following values:

```
cbStruct = sizeof(BIND_OPTS)
grfFlags = 0
grfMode = STGM_READWRITE
dwTickCountDeadline = 0.
```

You can call the **IBindCtx::SetBindOptions** method to modify these values, if you want values other from the defaults.

See Also

BIND_OPTS, **IBindCtx**, **IMoniker**, **MkParseDisplayName**

CreateDataAdviseHolder QuickInfo

Returns a new instance of an OLE-provided implementation of the **IDataAdviseHolder** interface.

STDAPI CreateDataAdviseHolder(
  IDataAdviseHolder ***ppDAHolder* //Pointer to the new instance of IDataAdviseHolder
 );

Parameter

ppDAHolder
 Points to the location where the pointer to the new advise holder object should be returned.

Return Values

S_OK
 The **IDataAdviseHolder** interface was instantiated and the pointer returned.

E_OUTOFMEMORY
 Indicates that OLE ran out of memory and **IDataAdviseHolder** could not be instantiated.

Comments

The **IDataAdviseHolder** interface is called by your implementation of the **IDataObject**::DAdvise interface to keep track of connections to advise sinks and to send notifications when necessary.

See Also

IDataAdviseHolder

CreateDataCache QuickInfo

Returns a new instance of an OLE-provided implementation of a cache.

```
HRESULT CreateDataCache(
   LPUNKNOWN pUnkOuter,        //Pointer to controlling unknown of aggregate
   REFCLSID rclsid,            //CLSID used to generate icon labels
   REFIID riid,                //Interface used to communicate with cache
   LPVOID FAR *ppvObj          //Pointer to returned cache object
  );
```

Parameters

pUnkOuter
: Points to the controlling unknown if the cache is to be created as part of an aggregate. This parameter may be NULL, indicating that there is no aggregate.

rclsid
: Specifies the CLSID used to generate icon labels. This value is typically CLSID_NULL.

riid
: Specifies the interface that will be used by the caller to communicate with the cache. This value is typically **IID_IOleCache.**

ppvObj
: Points to the location of the returned cache object.

Return Values

S_OK
: Indicates the OLE-provided cache was instantiated and the pointer returned.

E_INVALIDARG
: Indicates one or more arguments are invalid.

E_OUTOFMEMORY
: Indicates the interface could not be instantiated due to a lack of memory.

E_NOINTERFACE
: Indicates the interface represented by *riid* is not supported by the object. The parameter *ppvObj* is set to NULL.

Comments

The cache created by **CreateDataCache** supports the **IOleCache**, **IOleCache2**, and **IOleCacheControl** interfaces for controlling the cache. It also supports the **IPersistStorage**, **IDataObject** (without advise sinks), **IViewObject**, and **IViewObject2** interfaces.

See Also

IOleCache, **IOleCache2**, **IOleCacheControl**

CreateFileMoniker QuickInfo

Creates a file moniker based on the specified path.

```
WINOLEAPI CreateFileMoniker(
   LPCOLESTR lpszPathName,       //Pathname to be used
   LPMONIKER FAR *ppmk           //Receives the file moniker
  );
```

Parameters

lpszPathName

Points to a zero-terminated string containing the path on which this moniker is based. For Win32 applications, the **LPCOLESTR** type indicates a wide character string (two bytes per character); otherwise, the string has one byte per character.

ppmk

Receives an **IMoniker** interface pointer to the new file moniker. The returned pointer is NULL if an error occurs; if non-NULL, the function has called **IUnknown::AddRef** on the parameter and the caller is responsible for calling **IUnknown::Release**.

Return Values

S_OK

Indicates the moniker was created successfully.

MK_E_SYNTAX

Indicates an error in the syntax of a path was encountered while creating a moniker.

E_OUTOFMEMORY

Indicates insufficient memory.

Comments

CreateFileMoniker should be called by moniker providers, to create a file moniker when that is suitable for identifying its objects. Moniker providers hand out monikers to identify their objects so they are accessible to other parties. You must use file monikers if the objects being identified are stored in files. If each object resides in its own file, file monikers are the only type you need. For objects smaller than a file, you need to use another type of moniker (for example, item monikers) in addition to file monikers. Your objects must also implement the **IPersistFile** interface for them to be loaded when a file moniker is bound.

The most common example of moniker providers are OLE server applications that support linking. For OLE server applications that support linking only to file-based documents in their entirety, file monikers are the only type of moniker needed. OLE server applications that support linking to objects smaller than a document (such as sections of a document or embedded objects), must use item monikers as well as file monikers.

The *lpszPathName* can be a relative path, a UNC path (e.g., \\*server**share**path*), or a drive letter based path (e.g., c:\). If based on a relative path, the resulting moniker will need to be composed onto another file moniker before it can be bound.

A file moniker cannot be composed to the right of any other kind of moniker. It is possible to compose two file monikers together if the first moniker is based on an absolute path and the other is a relative path, resulting in a single file moniker based on the combination of the two paths. To identify objects stored within a file, it is necessary to compose other types of monikers (usually item monikers) to the right of a file moniker.

See Also

IMoniker - File Moniker Implementation

CreateGenericComposite QuickInfo

Performs a generic composition of *pmkFirst* and *pmkRest* and returns the resulting moniker.

```
WINOLEAPI CreateGenericComposite(
   LPMONIKER pmkFirst,                  //First moniker
   LPMONIKER pmkRest,                   //Second moniker
   LPMONIKER FAR *ppmkComposite         //Receives the composite
  );
```

Parameters

pmkFirst
Points to the moniker to be composed to the left of the *pmkRest* moniker. Can point to any kind of moniker, including a generic composite.

pmkRest
Points to the moniker to be composed to the right of the *pmkFirst* moniker. Can point to any kind of moniker, including a generic composite.

ppmkComposite
Receives an **IMoniker** pointer to the moniker that is the result of composing *pmkFirst* and *pmkRest*. If either *pmkFirst* or *pmkRest* are NULL, the returned pointer is the one that is non-NULL. If both *pmkFirst* and *pmkRest* are NULL, or if an error occurs, the returned pointer is NULL. If the returned pointer is non-NULL, the function has called **IUnknown::AddRef** on the parameter and the caller is responsible for calling **IUnknown::Release**.

Return Values

S_OK
Indicates the two input monikers were successfully composed.

MK_E_SYNTAX
Indicates the two monikers could not be composed due to an error in the syntax of a path (for example, if both *pmkFirst* and *pmkRest* are file monikers based on absolute paths).

E_OUTOFMEMORY
Indicates insufficient memory.

Comments

You would call this function only if you are writing your own moniker class (implementing the **IMoniker** interface). If you want to perform a generic composition in your implementation of the **IMoniker::ComposeWith** method, call this function from there.

Composition is the process of joining two monikers together. Sometimes two monikers of specific classes can be combined; for example, a file moniker representing an incomplete path and another file moniker representing a relative path can be combined to form a single file moniker representing the complete path. This is an example of "non-generic" composition. "Generic" composition, on the other hand, can connect any two monikers, no matter what their classes. Because a non-generic composition depends on the class of the monikers involved, it can be done only in the implementation of the **IMoniker::ComposeWith** method for a particular class. You can define new types of non-generic compositions if you write a new moniker class. In contrast, it is only necessary to call the CreateGenericComposite for generic compositions.

If you are a moniker client and you need to compose two monikers together, you should call the **IMoniker::ComposeWith** method. This gives the first moniker a chance to perform a non-generic composition. The only time you should call **CreateGenericComposite** is if you are writing a new moniker class. In the implementation of **IMoniker::ComposeWith**, you should call **CreateGenericComposite** once you have determined that your implementation cannot do a non-generic composition.

During the process of composing the two monikers, **CreateGenericComposite** makes all possible simplifications. Consider the example where *pmkFirst* is the generic composite moniker, *A* (°) *B* (°) *C*, and *pmkRest* is the generic composite moniker, $C_{(-1)}$ (°) $B_{(-1)}$ (°) *Z* (where $C_{(-1)}$ is the inverse of *C*). The function first composes *C* to $C_{(-1)}$, which composes to nothing. Then it composes *B* and $B_{(-1)}$ to nothing. Finally, it composes *A* to *Z*, and returns a pointer to the generic composite moniker, *A* (°) *Z*.

See Also

IMoniker::ComposeWith, **IMoniker - Generic Composite Moniker Implementation**

CreateILockBytesOnHGlobal [QuickInfo]

Returns an OLE-provided implementation of the **ILockBytes** interface which allows you to build a storage object in memory.

```
HRESULT CreateILockBytesOnHGlobal(
    HGLOBAL hGlobal,             //Memory handle for the byte array object
    BOOL fDeleteOnRelease,       //Indicates whether to free memory when the object is released
    ILockBytes ** ppLkbyt        //Points to location for pointer to the new byte array object
  );
```

Parameters

hGlobal

Contains the memory handle allocated by the **GlobalAlloc** function. The handle must be allocated as moveable and nondiscardable. If the handle is to be shared between processes, it must also be allocated as shared. New handles should be allocated with a size of zero. If *hGlobal* is NULL, the **CreateILockBytesOnHGlobal** function internally allocates a new shared memory block of size zero.

fDeleteOnRelease

Indicates whether the underlying handle for this byte array object should be automatically freed when the object is released.

ppLkbyt

Points to the location where this method places the pointer for the new byte array object, that is, the new instance of the **ILockBytes** interface.

Return Values

S_OK

Indicates the **ILockBytes** instance was created successfully.

E_INVALIDARG

Indicates an invalid value was specified for the *hGlobal* parameter.

E_OUTOFMEMORY

Indicates not enough memory to create an **ILockBytes** instance.

Comments

This function creates a byte array object in memory using the OLE-provided implementation of the **ILockBytes** interface. The returned byte array object can be the basis for a compound file. You can call the **StgCreateDocfileOnILockBytes** function to build a compound file on top of this byte array object. The **ILockBytes** instance calls the **GlobalReAlloc** function to grow the memory block as needed.

The current contents of the memory block are undisturbed by the creation of the new byte array object. After creating the **ILockBytes** instance, you can use the **StgOpenStorageOnILockBytes** function to reopen a previously existing storage object already contained in the memory block.

Note If you free the *hGlobal* memory handle, the byte array object is no longer valid. You must call the **ILockBytes::Release** method before freeing the memory handle.

The value of the *hGlobal* parameter can be changed by a subsequent call to the **GlobalReAlloc** function; thus, you cannot rely on this value after the byte array object is created.

See Also

StgOpenStorageOnILockBytes

CreateItemMoniker [QuickInfo]

Creates an item moniker based on a string that identifies an object within a container, typically a compound document.

WINOLEAPI CreateItemMoniker(
   LPCOLESTR *lpszDelim*,   //Delimiter string
   LPCOLESTR *lpszItem*,   //Item name
   LPMONIKER FAR **ppmk*   //Receives the item moniker
 );

Parameters

lpszDelim
  Points to a zero-terminated string containing the delimiter (typically "!") used to separate this item's display name from the display name of its container. For Win32 applications, the **LPOLESTR** type indicates a wide character string (two bytes per character); otherwise, the string has one byte per character.

lpszItem
  Points to a zero-terminated string containing the container's name for the object being identified. This name can later be used to retrieve a pointer to the object in a call to **IOleItemContainer::GetObject**.

ppmk
  Receives an **IMoniker** pointer to the new item moniker. The returned pointer is NULL if an error occurs; if non-NULL, the function has called **IUnknown::AddRef** on the parameter and the caller is responsible for calling **IUnknown::Release**.

Return Values

S_OK
  Indicates the moniker was created successfully.

E_OUTOFMEMORY
  Indicates insufficient memory.

Comments

A moniker providerthat can identify its objects with an item moniker would call CreateItemMoniker. A moniker provider hands out monikers to identify its objects so they are accessible to other parties. Item monikers identify objects that are contained within another object and can be individually identified using a string. The container of the object must also implement the **IOleContainer** interface so the objects can be loaded when a item moniker is bound.

The most common example of moniker providers are OLE applications that support linking. If your OLE application supports OLE linking to objects smaller than a file-based documents, you need to use item monikers. A server application that allows linking to a selection within a document, uses item monikers to identify those objects. Container applications that allow linking to embedded objects use item monikers to identify the embedded objects.

The *lpszItem* parameter is the name used by the document to uniquely identify the object. For example, if the object being identified is a cell range in a spreadsheet, an appropriate name might be something like "A1:E7." If the object being identified is an embedded object, an appropriate name might be something like "embedobj1." The document containing the object must provide an implementation of the **IOleItemContainer** interface that can interpret the name and locate the corresponding object. This allows the item moniker to be bound to the object it identifies.

Item monikers are typically not used in isolation; they must be composed with a moniker that identifies the document that contains the object. For example, if the object being identified is a cell range contained in a file-based document, the item moniker identifying that object must be composed with the file moniker identifying that document; this results in a composite moniker like "C:\work\sales.xls!A1:E7." It's also possible to have nested containers, for example, if an object is contained within an embedded object inside another document. The complete moniker of such an object would look like "C:\work\report.doc!embedobj1!A1:E7." In such a situation, each container object must call **CreateItemMoniker** and provide its own implementation of the **IOleItemContainer** interface.

See Also

IMoniker::ComposeWith, **IOleItemContainer**, **IMoniker - Item Moniker Implementation**

CreateOleAdviseHolder QuickInfo

Creates an advise holder for managing compound document notifications. It returns an instance of the OLE implementation of the **IOleAdviseHolder** interface.

HRESULT CreateOleAdviseHolder(
  ppOAHolder   //Pointer to a pointer
 );

Parameter

ppOAHolder
  Points to where to return the new **IOleAdviseHolder** instance.

Return Values

S_OK
  Indicates that the new OLE advise holder returned successfully.

E_OUTOFMEMORY
  Indicates insufficient memory to create new advise holder.

Comments

The helper function **CreateOleAdviseHolder** is an OLE implementation of the **IOleAdviseHolder** interface. Anyone who wants to write their own implementation of this interface may do so; but the advise holder returned by **CreateOleAdviseHolder** will suffice for the great majority of applications. If your application has a need to implement **IOleAdviseHolder::EnumAdvise**; you will need to implement your own advise holder because the one provided by OLE does not support this method.

See Also

IOleAdviseHolder

CreatePointerMoniker QuickInfo

Creates a pointer moniker based on a specified interface pointer.

```
WINOLEAPI CreatePointerMoniker(
   LPUNKNOWN punk,              //Interface pointer to be used
   LPMONIKER FAR *ppmk          //Receives the pointer moniker
  );
```

Parameters

punk
Points to an interface pointer to the object to be identified by the resulting moniker.

ppmk
Receives an **IMoniker** interface pointer to the new pointer moniker. The returned pointer is NULL if an error occurs; if non-NULL, the function has called **IUnknown::AddRef** on the parameter and the caller is responsible for calling **IUnknown::Release**.

Return Values

S_OK
Indicates the pointer moniker was created successfully.

E_UNEXPECTED
Indicates an unexpected error occurred.

E_OUTOFMEMORY
Indicates insufficient memory.

Comments

You rarely need to call this function.

A pointer moniker is essentially a wrapping of an interface pointer that allows it to be passed to those interfaces that require monikers. Pointer monikers allow an object that has no persistent representation to participate in a moniker binding operation. Binding a pointer moniker is done by calling the pointer's **IUnknown::QueryInterface** method.

See Also

IMoniker - Pointer Moniker Implementation

CreateStreamOnHGlobal QuickInfo

Returns an OLE-provided implementation of the **IStream** interface with the stream object stored in global memory.

```
HRESULT CreateStreamOnHGlobal(
    HGLOBAL hGlobal,              //Memory handle for the stream object
    BOOL fDeleteOnRelease,        //Indicates whether to free memory when the object is released
    LPSTREAM * ppstm              //Points to location for pointer to the new stream object
  );
```

Parameters

hGlobal
Contains the memory handle allocated by the **GlobalAlloc** function. The handle must be allocated as moveable and nondiscardable. If the handle is to be shared between processes, it must also be allocated as shared. New handles should be allocated with a size of zero. If *hGlobal* is NULL, the **CreateStreamOnHGlobal** function internally allocates a new shared memory block of size zero.

fDeleteOnRelease
Indicates whether the underlying handle for this stream object should be automatically freed when the stream object is released.

ppstm
Points to the location where this method places a pointer to the new stream object, that is, the new **IStream** instance. It cannot be NULL.

Return Values

S_OK
Indicates the stream object was created successfully.

E_OUTOFMEMORY
Indicates the stream could not be created due to lack of memory.

E_INVALIDARG
Indicates invalid value for one of the parameters.

Comments

This function creates a stream object in memory using the OLE-provided implementation of the **IStream** interface. The returned stream object supports both reading and writing, is not transacted, and does not support locking.

The initial contents of the stream are the current contents of the memory block provided in the *hGlobal* parameter. If the *hGlobal* paramter is NULL, this function internally allocates memory.

The current contents of the memory block are undisturbed by the creation of the new stream object. Thus, you can use this function to open an existing stream in memory.

The initial size of the stream is the size of the memory handle returned by the Win32 **GlobalSize** function. Because of rounding, this is not necessarily the same size that was originally allocated for the handle. If the logical size of the stream is important, you should follow the call to this function with a call to the **IStream::SetSize** method.

See Also

CreateStreamOnHGlobal, **IStream::SetSize**

GlobalSize in Win32

DllCanUnloadNow

Determines whether the DLL that implements it is in use. It is called by OLE, not by the application. If the DLL is not in use, the calling function can safely unload it from memory.

Note OLE does not provide this function. DLLs that support the OLE Component Object Model(COM) should implement and export **DllCanUnloadNow**.

HRESULT DllCanUnloadNow();

Return Values

S_OK
 Indicates that the DLL can be unloaded.

S_FALSE
 Indicates that the DLL cannot be unloaded now.

Comments

A call to **DllCanUnloadNow** determines whether the DLL from which it is exported is still in use. A DLL is no longer in use when it is not managing any existing objects (the reference count on all of its objects is 0).

Notes to Callers

You should not have to call **DllCanUnloadNow** directly. OLE calls it only through a call to the **CoFreeUnusedLibraries** function. When it returns S_OK, **CoFreeUnusedLibraries** safely frees the DLL.

Notes to Implementors

You need to implement **DllCanUnloadNow** in, and export it from, DLLs that are to be dynamically loaded through a call to the **CoGetClassObject** function. (You also need to implement and export the **DllGetClassObject** function in the same DLL).

If a DLL loaded by **CoGetClassObject** fails to export **DllCanUnloadNow**, the DLL will not be unloaded until the application calls the **CoUninitialize** function to release the OLE libraries.

If the DLL links to another DLL, returning S_OK from **DllCanUnloadNow** will also cause the second, dependent DLL to be unloaded. To eliminate the possibility of a crash, the DLL should call the **CoLoadLibrary** function(*path_to_second DLL*, TRUE), which forces the COM library to reload the second DLL and set it up for a call to **CoFreeUnusedLibraries** to free it separately when appropriate.

DllCanUnloadNow should return S_FALSE if there are any existing references to objects that the DLL manages.

See Also

DllGetClassObject

DllGetClassObject QuickInfo

Retrieves the class object from a DLL object handler or object application. It is called from within the **CoGetClassObject** function when the class context is a DLL.

Note OLE does not provide this function. DLLs that support the OLE Component Object Model must implement **DllGetClassObject** in OLE object handlers or DLL applications.

```
HRESULT DllGetClassObject(
  REFCLSID rclsid,     //CLSID for the class object
  REFIID riid,         //Identifier that identifies the interface that communicates with the
                       class object
  LPVOID  * ppv        //Points to the pointer of the communicating interface
   );
```

Parameters

rclsid
Specifies the CLSID that will associate the correct data and code.

riid
Specifies the interface that the caller is to use to communicate with the class object. Usually, this is **IID_IClassFactory**.

ppv
Points either to the pointer of the requested interface or, if an error occurs, to NULL.

Return Values

S_OK
Indicates that the object was retrieved successfully.

CLASS_E_CLASSNOTAVAILABLE
Indicates that the DLL does not support the class (object definition).

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Indicates that one or more arguments are invalid.

E_UNEXPECTED
Indicates that an unexpected error occurred.

Comments

If a call to the **CoGetClassObject** function finds the class object that is to be loaded in a DLL, **CoGetClassObject** uses the DLL's exported **DllGetClassObject** function.

Notes to Callers

You should not call **DllGetClassObject** directly. When an object is defined in a DLL, **CoGetClassObject** calls the **CoLoadLibrary** function to load the DLL, which, in turn, calls **DllGetClassObject**.

Notes to Implementors

You need to implement **DllGetClassObject** in (and export it from) DLLs that support the OLE Component Object Model.

Example

Following is an example (in C++) of an implementation of **DllGetClassObject**. In this example, **DllGetClassObject** creates a class factory object and calls its **QueryInterface** method to retrieve a pointer to the interface requested in *riid*. The implementation safely releases the reference it holds to the **IClassFactory** interface because it returns a reference-counted pointer to **IClassFactory** to the caller.

```
HRESULT_export  PASCAL DllGetClassObject
        (REFCLSID rclsid, REFIID riid, LPVOID * ppvObj)
{
     HRESULT hres = E_OUTOFMEMORY;
     *ppvObj = NULL;

     CClassFactory *pClassFactory = new CClassFactory(rclsid);
     if (pClassFactory != NULL)   {
       hRes = pClassFactory->QueryInterface(riid, ppvObj);
       pClassFactory->Release();
     }
     return hRes;
}
```

See Also

CoGetClassObject, **DllCanUnloadNow**

DoDragDrop QuickInfo

Carries out an OLE drag and drop operation.

```
HRESULT DoDragDrop(
  IDataObject * pDataObject,          //Data object being dragged
  IDropSource * pDropSource,          //Instance of IDropSource for source application
  DWORD dwOKEffect,                   //Specifies the effects allowed by the source
  DWORD * pdwEffect                   //Points to effects of the OLE drag and drop on the source
 );
```

Parameters

pDataObject
Points to an instance of the **IDataObject** interface, which provides the data being dragged.

pDropSource
Points to an instance of the **IDropSource** interface, which is used to communicate with the source during the drag operation.

dwOKEffect
Specifies the effects the source allows in the OLE drag and drop operation. Most significant is whether it permits a move. The *dwOKEffect* and *pdwEffect* parameters obtain values from the **DROPEFFECT** enumeration. For a list of values, see **DROPEFFECT**.

pdwEffect
Points to a value that indicates how the OLE drag and drop operation affected the source data. The *pdwEffect* parameter is set only if the operation is not canceled.

Return Values

DRAGDROP_S_DROP
Indicates the OLE drag and drop operation was successful.

DRAGDROP_S_CANCEL
Indicates the OLE drag and drop operation was canceled.

E_OUTOFMEMORY
Out of memory.

E_UNSPEC
Indicates an unexpected error occurred.

Comments

If you are developing an application that can act as a data source for an OLE drag and drop operation, you must call **DoDragDrop** when you detect that the user has started an OLE drag and drop operation.

The **DoDragDrop** function enters a loop in which it calls various methods in the **IDropSource** and **IDropTarget** interfaces.

1. The **DoDragDrop** function determines the window under the current cursor location. It then checks to see if this window is a valid drop target.
2. If the window is a valid drop target, **DoDragDrop** calls **IDropTarget::DragEnter**. This method returns an effect code indicating what would happen if the drop actually occurred. For a list of valid drop effects, see the **DROPEFFECT** enumeration.
3. **DoDragDrop** calls **IDropSource::GiveFeedback** with the effect code so that the drop source interface can provide appropriate visual feedback to the user. The *pDropSource* pointer passed into **DoDragDrop** specifies the appropriate **IDropSource** interface.
4. **DoDragDrop** tracks mouse cursor movements and changes in the keyboard or mouse button state.
    - If the user moves out of a window, **DoDragDrop** calls **IDropTarget::DragLeave**.
    - If the mouse enters another window, **DoDragDrop** determines if that window is a valid drop target

and then calls **IDropTarget::DragEnter** for that window.

- If the mouse moves but stays within the same window, **DoDragDrop** calls **IDropTarget::DragOver**.

5. If there is a change in the keyboard or mouse button state, **DoDragDrop** calls **IDropSource::QueryContinueDrag** and determines whether to continue the drag, to drop the data, or to cancel the operation based on the return value.
    - If the return value is S_OK, **DoDragDrop** first calls **IDropTarget::DragOver** to continue the operation. This method returns a new effect value and **DoDragDrop** then calls **IDropSource::GiveFeedback** with the new effect so appropriate visual feedback can be set. For a list of valid drop effects, see the **DROPEFFECT** enumeration. **IDropTarget::DragOver** and **IDropSource::GiveFeedback** are paired so that as the mouse moves across the drop target, the user is given the most up-to-date feedback on the mouse's position.
    - If the return value is DRAGDROP_S_DROP, **DoDragDrop** calls **IDropTarget::Drop**. The **DoDragDrop** function returns the last effect code to the source, so the source application can perform the appropriate operation on the source data, for example, cut the data if the operation was a move.
    - If the return value is DRAGDROP_S_CANCEL, the **DoDragDrop** function calls **IDropTarget::DragLeave**.

See Also

IDropSource::GiveFeedback

FACILITY_NT_BIT

```
#define FACILITY_NT_BIT 0x10000000
```

Defines bits so macros are guaranteed to work.

FAILED

```
#define FAILED(Status) ((HRESULT)(Status)<0)
```

Provides a generic test for failure on any status value. Negative numbers indicate failure.

GetClassFile QuickInfo

Returns the CLSID associated with the given filename.

HRESULT GetClassFile(
  LPCWSTR *szFileName*, //Pointer to filename for which you are requesting a CLSID
  CLSID * *pclsid* //Pointer to location for returning the CLSID
 );

Parameters

szFileName
Points to the filename for which you are requesting the associated CLSID.

pclsid
Points to the location where the the associated CLSID is returned.

Return Values

S_OK
Indicates the CLSID was successfully returned.

MK_E_CANTOPENFILE
Indicates unable to open the specified filename.

MK_E_INVALIDEXTENSION
Indicates the specified extension in the registry is invalid.

Note This function can also return any file system errors.

Comments

The CLSID returned by this function is used, for example, by File Monikers to locate the object application that can open the file. For example, when a link to a file-based document is activated, the file moniker uses the returned CLSID to locate the application for the link source.

The following strategies are used to determine an appropriate CLSID:

1. If the file contains a storage object, as determined by a call to the **StgIsStorageFile** function, the **GetClassFile** function returns the CLSID that was written with the **IStorage::SetClass** method.
2. If the file is not a storage object, the **GetClassFile** function attempts to match various bits in the file against a pattern in the registry. A pattern in the registry can contain a series of entries of the form:

   ```
   regdb key = offset, cb, mask, value
   ```

   The value of the *offset* item is an offset from the beginning or end of the file and the *cb* item is a length in bytes. These two values represent a particular byte range in the file. (A negative value for the *offset* item is interpreted from the end of the file). The *mask* value is a bit mask that is used to perform a logical AND operation with the byte range specifed by *offset* and *cb*. The result of the logical AND operation is compared with the *value* item. If the *mask* is omitted, it is assumed to be all ones.

   Each pattern in the registry is compared to the file in the order of the patterns in the database. The first pattern where each of the *value* items matches the result of the AND operation determines the CLSID of the file. For example, the pattern contained in the following entries of the registry requires that the first four bytes be AB CD 12 34 and that the last four bytes be FE FE FE FE:

   ```
   HKEY_CLASSES_ROOT
      FileType
        {12345678-0000-0001-C000-000000000095}
          0 = 0, 4, FFFFFFFF, ABCD1234
          1 = -4, 4, , FEFEFEFE
   ```

If a file contains such a pattern, the CLSID {12345678-0000-0001-C000-000000000095} will be associated with this file.

3. If the above strategies fail, the **GetClassFile** function searches for the File Extension key in the registry that corresponds to the *.ext* portion of the filename. If the database entry contains a valid CLSID, this function returns that CLSID.
4. Otherwise, MK_E_INVALIDEXTENSION is returned.

See Also

WriteClassStg

GetConvertStg QuickInfo

Returns the current value of the convert bit for the specifed storage object.

```
HRESULT GetConvertStg(
   IStorage * pStg          //Points to the storage object
  );
```

Parameter

pStg
Points to the storage object from which to retrieve the convert bit.

Return Values

S_OK
Indicates the convert bit is set to TRUE.

S_FALSE
Indicates the convert bit is cleared (FALSE).

STG_E_ACCESSDENIED
Indicates cannot access the storage object.

IStorage::OpenStream, **IStorage::OpenStorage**, and **IStream::Read** storage and stream access errors.

Comments

This function is used by object applications that support converting the object from one format to another. The object application must be able to read the storage object using the format of its previous CLSID and write the object using the format of its new CLSID to support converting the object. For example, a spreadsheet created by one application can be converted to the format used by a different application.

The convert bit is set by a call to the **SetConvertStg** function. A container application can call this function on the request of an end user, or a setup program can call it when installing a new version of an application. An end user requests converting an object through the Convert To dialog box. When an object is converted, the new CLSID is permanently assigned to the object, so the object is subsequently associated with the new CLSID.

Then, when the object is activated, the object application calls the **GetConvertStg** function to retrieve the value of the convert bit from the storage object. If the bit is set, the object's CLSID has been changed, and the object application must read the old format and write the new format for the storage object.

After retrieving the bit value, the object application should clear the convert bit by calling the **SetConvertStg** function with its *fConvert* parameter set to FALSE.

See Also

SetConvertStg

GetHGlobalFromILockBytes QuickInfo

Returns a global memory handle to a byte array object created using the **CreateILockBytesOnHGlobal** function.

```
HRESULT GetHGlobalFromILockBytes(
   ILockBytes * pLkbyt,          //Points to the byte array object
   HGLOBAL * phglobal            //Points to the current memory handle for the specified byte array
  );
```

Parameters

pLkbyt
Points to the byte array object previously created by a call to the **CreateILockBytesOnHGlobal** function.

phglobal
Points to the current memory handle used by the specified byte array object.

Return Values

S_OK
Indicates the handle was returned successfully.

E_INVALIDARG
Indicates invalid value specified for the *pLkbyt* parameter. It can also indicate that the byte array object passed in is not one created by the **CreateILockBytesOnHGlobal** function.

Comments

The handle this function returns might be different from the original handle due to intervening calls to the **GlobalRealloc** function.

The contents of the returned memory handle can be written to a clean disk file, and then opened as a storage object using the **StgOpenStorage** function.

This function only works within the same process from which the byte array was created.

See Also

StgOpenStorage

GetHGlobalFromStream QuickInfo

Returns the global memory handle to a stream that was created using the **CreateStreamOnHGlobal** function.

```
HRESULT GetHGlobalFromStream(
    IStream * pstm,           //Points to the stream object
    HGLOBAL * phglobal        //Points to the current memory handle for the specified stream
  );
```

Parameters

pstm
  Points to the stream object previously created by a call to the **CreateStreamOnHGlobal** function.

phglobal
  Points to the current memory handle used by the specified stream object.

Return Values

S_OK
  Indicates the handle was successfully returned.

E_INVALIDARG
  Indicates invalid value specified for the *pstm* parameter. It can also indicate that the stream object passed in is not one created by a call to the **CreateStreamOnHGlobal** function.

Comments

The handle this function returns may be different from the original handle due to intervening **GlobalRealloc** calls.

This function only works within the same process from which the byte array was created.

See Also

CreateStreamOnHGlobal

GlobalRealloc in Win32

GetRunningObjectTable QuickInfo

Returns an **IRunningObjectTable** pointer to the local Running Object Table (ROT).

```
HRESULT GetRunningObjectTable(
   DWORD reserved,                    //Reserved
   LPRUNNINGOBJECTTABLE *pprot        //Receives an IRunningObjectTable pointer
  );
```

Parameters

reserved
Indicates reserved for future use.

pprot
Receives an **IRunningObjectTable** interface pointer to the local ROT. If an error occurs, **pprot* is undefined; otherwise, the caller is responsible for calling **IUnknown::Release** on the parameter.

Return Values

S_OK
Indicates that an **IRunningObjectTable** pointer was successfully returned.

E_UNEXPECTED
Indicates an unexpected error.

Comments

Each workstation has a local ROT that maintains a table of the objects that have been registered as running on that machine. This function returns an **IRunningObjectTable** interface pointer, which provides access to that table.

Moniker providers, which hand out monikers identifying objects to make them accessible to others, should call **GetRunningObjectTable**. Use the interface pointer returned by this function to register your objects when they begin running, to record the times that those objects are modified, and to revoke their registrations when they stop running. See the **IRunningObjectTable** interface for more information.

The most common types of moniker provider are compound-document link sources. This includes server applications that support linking to their documents (or portions of a document) and container applications that support linking to embeddings within their documents. Server applications that do not support linking can also use the ROT to cooperate with container applications that support linking to embeddings.

If you're writing your own moniker class (implementing the **IMoniker** interface) and you need an interface pointer to the ROT, use the **IBindCtx::GetRunningObjectTable** method rather than the **GetRunningObjectTable** function. This allows future implementations of the **IBindCtx** interface to modify binding behavior.

See Also

IBindCtx::GetRunningObjectTable, **IMoniker**, **IRunningObjectTable**

GetScode

```
#define GetScode(hr) ((SCODE) (hr))
```

Extracts the **SCODE** from an **HRESULT**

This macro is obsolete and should not be used.

HRESULT_CODE

```
#define HRESULT_CODE(hr) ((hr) & 0xFFFF)
```

Returns the error code part of the **HRESULT**.

HRESULT_FACILITY

```
#define HRESULT_FACILITY(hr) (((hr) >> 16) & 0x1fff)
```

Returns the facility from the **HRESULT**.

HRESULT_FROM_NT

```
#define HRESULT_FROM_NT(x) ((HRESULT) ((x) | FACILITY_NT_BIT))
```

Maps an NT status value into an **HRESULT**.

HRESULT_FROM_WIN32

```
#define HRESULT_FROM_WIN32(x) (x ? ((HRESULT) (((x) & 0x0000FFFF) | (FACILITY_WIN32 <<
    16) | 0x80000000)) : 0 )
```

Maps a WIN32 error value into an **HRESULT**. Note that this assumes WIN32 errors fall in the range of -32k to 32k.

HRESULT_SEVERITY

```
#define HRESULT_SEVERITY(hr) (((hr) >> 31) & 0x1)
```

Returns the severity bit from the **HRESULT**.

IIDFromString [QuickInfo]

Converts a string generated by the **StringFromIID** function back into the original interface identifier.

```
HRESULT IIDFromString(
    LPOLESTR lpsz,          //Points to the string representation of the IID
    LPIID lpiid             //Receives a pointer to the requested IID on return
  );
```

Parameters

lpsz
Points to the string representation of the IID.

lpiid
Receives a pointer to the requested IID on return.

Return Values

S_OK
Indicates that the string was successfully converted.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Indicates that one or more arguments are invalid.

Comments

The function converts the interface ID in a way that guarantees different interface IDs will always be converted to different strings.

See Also

StringFromIID

IsAccelerator QuickInfo

Determines whether the keystroke maps to an accelerator in the given accelerator table.

```
BOOL IsAccelerator(
    HACCEL hAccel,           //Handle to accelerator table
    INT cAccelEntries,       //Number of entries in the accelerator table
    LPMSG lpMsg,             //Points to the keystroke message to be translated
    WORD * lpwCmd            //Where to return the corresponding command ID
  );
```

Parameters

hAccel
Specifies the handle to the accelerator table.

cAccelEntries
Specifies the number of entries in the accelerator table.

lpMsg
Points to the keystroke message to be translated.

lpwCmd
Points to where to return the corresponding command ID if there is an accelerator for the keystroke. It may be NULL.

Return Values

TRUE
Indicates the message is for the object application.

FALSE
Indicates the message is not for the object and should be forwarded to the container.

Comments

While active in-place, the object *always* has first chance to translate the keystrokes into accelerators. If the keystroke corresponds to one of its accelerators, the object must *not* call the OleTranslateAccelerator function – even if its call to the Windows **TranslateAccelerator** function fails. Failure to process keystrokes in this manner can lead to inconsistent behavior.

If the keystroke is not one of the object's accelerators, then the object must call **OleTranslateAccelerator** to let the container try its accelerator translation.

The object application can call **IsAccelerator** to determine if the accelerator message belongs to it. Some object applications do accelerator translation on their own and do not call **TranslateAccelerator**. Those applications will not call **IsAccelerator**, because they already have the information.

See Also

OleTranslateAccelerator

TranslateAccelerator in Win32

IS_ERROR

```
#define IS_ERROR(Status) ((unsigned long)(Status) >> 31 == SEVERITY_ERROR)
```

Provides a generic test for errors on any status value.

IsEqualGUID   QuickInfo

Determines whether two GUIDs are equal.

```
BOOL IsEqualGUID(
   REFGUID rguid1,        //The GUID to compare to rguid2
   REFGUID rguid2         //The GUID to compare to rguid1
  );
```

Parameters

rguid1
Specifies the GUID to compare to *rguid2*.

rguid2
Specifies the GUID to compare to *rguid1*.

Return Values

TRUE
Indicates the GUIDs are equal.

FALSE
Indicates the GUIDs are not equal.

Comments

IsEqualGUID is used by the **IsEqualCLSID** and **IsEqualIID** functions.

See Also

IsEqualCLSID, **IsEqualIID**

IsEqualCLSID

Determines whether two CLSIDs are equal.

```
BOOL IsEqualCLSID(
    REFCLSID rclsid1,          //The CLSID to compare to rclsid2
    REFCLSID rclsid2           //The CLSID to compare to rclsid1
  );
```

Parameters

rclsid1
    Specifies the CLSID to compare to *rclsid2*.

rclsid2
    Specifies the CLSID to compare to *rclsid1*.

Return Values

TRUE
    Indicates that the CLSIDs are equal.

FALSE
    Indicates that the CLSIDs are not equal.

See Also

IsEqualGUID, **IsEqualIID**

IsEqualIID

Determines whether two interface IDs are equal.

```
BOOL IsEqualIID(
    REFGUID riid1,          //The interface ID to compare to riid2
    REFGUID riid2           //The interface ID to compare to riid1
  );
```

Parameters

riid1
Specifies the interface ID to compare with *riid2*.

riid2
Specifies the interface IID to compare with *riid1*.

Return Values

TRUE
Indicates that the interface IDs are equal.

FALSE
Indicates that the interface IDs are not equal.

See Also

IsEqualGUID, **IsEqualCLSID**

IsValidIid

This function is obsolete.

IsValidInterface

This function is obsolete.

IsValidPtrIn

This function is obsolete.

IsValidPtrOut

This function is obsolete.

MAKE_HRESULT

```
#define MAKE_HRESULT(sev,fac,code) \HRESULT) (((unsigned long)(sev)<<31) | ((unsigned long)(fac)<<16) | ((unsigned long)(code))) )
```

Creates an **HRESULT** value from component pieces of the 32-bit value.

MAKE_SCODE

```
#define MAKE_SCODE(sev,fac,code) \((SCODE) (((unsigned long)(sev)<<31) | ((unsigned long)
    (fac)<<16) | ((unsigned long)(code))) )
```

Returns an **SCODE** given an **HRESULT**.

MkParseDisplayName QuickInfo

Converts a string into a moniker that identifies the object named by the string. This is the inverse of the **IMoniker::GetDisplayName** operation, which retrieves the display name associated with a moniker.

```
WINOLEAPI MkParseDisplayName(
    LPBC pbc,                   //Bind context to be used
    LPCOLESTR szUserName,       //Display name
    ULONG FAR *pchEaten,        //Receives number of characters consumed
    LPMONIKER FAR *ppmk         //Receives moniker built from display name
  );
```

Parameters

pbc
Points to the binding context to be used in this binding operation.

szUserName
Points to a zero-terminated string containing the display name to be parsed. For Win32 applications, the **LPOLESTR** type indicates a wide character string (two bytes per character); otherwise, the string has one byte per character.

pchEaten
Receives the number of characters of *szUserName* that were consumed. If the function is successful, **pchEaten* is the length of *szUserName*; otherwise, it is the number of characters successfully parsed.

ppmk
Receives a pointer to the moniker that was built from *szUserName*. The returned pointer is NULL if an error occurs; if non-NULL, the function has called **IUnknown::AddRef** on the parameter and the caller is responsible for calling **IUnknown::Release**.

Return Values

S_OK
Indicates the parse operation was successful and the moniker was created.

MK_E_SYNTAX
Indicates an error in the syntax of a filename or an error in the syntax of the resulting composite moniker.

E_OUTOFMEMORY
Indicates insufficient memory.

In addition, the return value can be any of the error return values from **IMoniker::BindToObject**, **IOleItemContainer::GetObject**, or **IParseDisplayName::ParseDisplayName**.

Comments

The **MkParseDisplayName** function parses a human-readable name into a moniker that can be used to identify a link source. The resulting moniker can be a simple moniker (such as a file moniker), or it can be a generic composite made up of the component moniker pieces. For example, the following display name:

```
"c:\mydir\somefile!item 1"
```

could be parsed into the following generic composite moniker:

```
(FileMoniker based on "c:\mydir\somefile")  (°)  (ItemMoniker based on "item 1")
```

The most common situation in which you need to call **MkParseDisplayName** is in the implementation of the standard Links dialog, which allows an end-user to specify the source of a linked object by typing in a string. You may also need to call **MkParseDisplayName** if your application supports a macro language that permits remote references (reference to elements outside of the document).

Note that parsing a display name often requires activating the same objects that would be activated during a binding operation. Therefore, parsing a display name can be as expensive as binding to the object that it refers. Objects that are bound during the parsing operation are cached in the passed-in bind context. If you plan to bind the moniker returned by **MkParseDisplayName**, the best time to do so is immediately after the function returns, using the same bind context; this removes the need to activate objects a second time.

The **MkParseDisplayName** function parses as much of the display name as it understands into a moniker. The function then calls **IMoniker::ParseDisplayName** on the newly created moniker, passing the remainder of the display name. The moniker returned by **IMoniker::ParseDisplayName** is composed onto the end of the existing moniker and, if any of the display name remains unparsed, **IMoniker::ParseDisplayName** is called on the result of the composition. This process is repeated until the entire display name has been parsed.

The **MkParseDisplayName** function attempts the following strategies to parse the beginning of the display name, using the first one that succeeds:

1. The function looks in the Running Object Table for file monikers corresponding to all prefixes of *szDisplayName* that consist solely of valid filename characters. This strategy can identify documents that are as yet unsaved.
2. The function checks the maximal prefix of *szDisplayName*, which consists solely of valid filename characters, to see if an OLE 1 document is registered by that name (this may require some DDE broadcasts). In this case, the returned moniker is an internal moniker provided by the OLE 1 compatibility layer of OLE 2.
3. The function consults the file system to check whether a prefix of *szDisplayName* matches an existing file. The filename can be drive absolute, drive relative, working-directory relative, or begin with an explicit network share name. This is the common case.
4. If the initial character of *szDisplayName* is an '@', the function finds the longest string immediately following it that conforms to the legal ProgID syntax. The function converts this string to a CLSID using the **CLSIDFromProgID** function. If the CLSID represents an OLE 2 class, the function loads the corresponding class object and asks for an **IParseDisplayName** interface pointer. The resulting **IParseDisplayName** interface is then given the whole string to parse, starting with the '@'. If the CLSID represents an OLE 1 class, then the function treats the string following the ProgID as an OLE1/DDE link designator having <*filename*>!<*item*> syntax.

See Also

IMoniker::ParseDisplayName, **IMoniker::GetDisplayName**, **IParseDisplayName**

MonikerCommonPrefixWith QuickInfo

Returns a moniker based on the common prefix that the two monikers share. This function is intended for use only in **IMoniker::CommonPrefixWith** implementations.

```
WINOLEAPI MonikerCommonPrefixWith(
    LPMONIKER pmkThis,                //First moniker being compared
    LPMONIKER pmkOther,               //Second moniker being compared
    LPMONIKER FAR *ppmkCommon         //Receives common prefix
  );
```

Parameters

pmkThis
Points to one of the monikers to compare.

pmkOther
Points to the other moniker to compare.

ppmkCommon
Receives a pointer to a moniker based on the common prefix of *pmkThis* and *pmkOther*. The returned pointer is NULL if an error occurs; if non-NULL, the function has called **IUnknown::AddRef** on the parameter and the caller is responsible for calling **IUnknown::Release**.

Return Values

S_OK
Indicates a common prefix exists that is neither *pmkThis* nor *pmkOther*.

MK_S_HIM
Indicates the entire *pmkOther* moniker is a prefix of the *pmkThis* moniker.

MK_S_ME
Indicates the entire *pmkThis* moniker is a prefix of the *pmkOther* moniker.

MK_S_US
Indicates the *pmkThis* and *pmkOther* monikers are equal.

MK_E_NOPREFIX
Indicates the monikers have no common prefix.

MK_E_NOTBINDABLE
Indicates this function was called on a relative moniker. It is not meaningful to take the common prefix of relative monikers.

E_UNEXPECTED
Indicates an unexpected error occurred.

E_OUTOFMEMORY
Indicates insufficient memory.

Comments

Call **MonikerCommonPrefixWith** only if you are writing your own moniker class (implementing the **IMoniker** interface). Call this function from your implementation of the **IMoniker::CommonPrefixWith** method.

Your implementation of **IMoniker::CommonPrefixWith** should first check whether the other moniker is of a type you recognize and handle in a special way. If not, you should call **MonikerCommonPrefixWith**, passing itself as *pmkThis* and the other moniker as *pmkOther*. **MonikerCommonPrefixWith** correctly handles the cases where either moniker is a generic composite.

You should call this function only if *pmkSrc* and *pmkDest* are both absolute monikers (where an absolute moniker is either a file moniker or a generic composite whose left-most component is a file moniker, and where the file moniker represents an absolute path). Do not call this function on relative monikers.

See Also

IMoniker::CommonPrefixWith

MonikerRelativePathTo QuickInfo

Returns a moniker that, when composed onto the end of the first specified moniker (or one with a similar structure), yields the second specified moniker. This function is intended for use only by **IMoniker::RelativePathTo** implementations.

```
WINOLEAPI MonikerRelativePathTo(
   LPMONIKER pmkSrc,                 //Moniker identifying source
   LPMONIKER pmkDest,                //Moniker identifying destination
   LPMONIKER FAR * ppmkRelPath,      //Receives relative moniker
   BOOL dwReserved                   //Reserved; must be non-zero
  );
```

Parameters

pmkSrc
Points to the moniker that, when composed with the relative moniker to be created, produces *pmkDest*. This moniker identifies the "source" of the relative moniker to be created.

pmkDest
Points to the moniker to be expressed relative to *pmkSrc*. This moniker identifies the "destination" of the relative moniker to be created.

ppmkRelPath
Receives a pointer to the relative moniker. The returned pointer is NULL if an error occurs; if non-NULL, the function has called **IUnknown::AddRef** on the parameter and the caller is responsible for calling **IUnknown::Release**.

dwReserved
Reserved; must be non-zero.

Return Values

S_OK
Indicates a meaningful relative path has been returned.

MK_S_HIM
Indicates the only form of the relative path is the other moniker.

MK_E_NOTBINDABLE
Indicates that *pmkSrc* is a relative moniker, such as an item moniker. *pmkSrc* must be composed with the moniker of its container before a relative path can be determined.

E_INVALIDARG
Specifies the parameter *dwReserved* was set to zero.

E_UNEXPECTED
Indicates an unexpected error occurred.

E_OUTOFMEMORY
Indicates insufficient memory.

Comments

Call **MonikerRelativePathTo** only if you are writing your own moniker class (implementing the **IMoniker** interface). Call this function from your implementation of the **IMoniker::RelativePathTo** method.

Your implementation of **IMoniker::RelativePathTo** should first check whether the other moniker is of a type you recognize and handle in a special way. If not, you should call **MonikerRelativePathTo**, passing itself as *pmkThis* and the other moniker as *pmkOther*. **MonikerRelativePathTo** correctly handles the cases where either moniker is a generic composite.

You should call this function only if *pmkSrc* and *pmkDest* are both absolute monikers, where an absolute moniker is either a file moniker or a generic composite whose left-most component is a file moniker, and where the file moniker represents an absolute path. Do not call this function on relative monikers.

See Also

IMoniker::RelativePathTo

OleBuildVersion QuickInfo

This function is obsolete.

OleConvertIStorageToOLESTREAM QuickInfo

Converts the specified storage object from OLE 2 structured storage to the OLE 1 storage model but does not include the presentation data. This is one of several compatibility functions. You can use these functions to determine if an object class is from OLE 1 and to convert objects between OLE 1 and OLE 2 storage formats.

```
HRESULT OleConvertIStorageToOLESTREAM(
  IStorage * pStg,            //Points to the OLE 2 storage object to be converted
  LPOLESTREAM lpolestm        //Points to the stream where the OLE1 storage is written
 );
```

Parameters

pStg
Points to the storage object to be converted to an OLE 1 storage.

lpolestm
Points to a stream where the persistent representation of the object is saved using the OLE 1 storage model.

Return Values

S_OK
Indicates the storage object was successfully converted and the **OLESTREAM** structure contains the persistent representation of an OLE 1 object.

CONVERT10_E_STG_NO_STD_STREAM
Indicates object cannot be converted because its storage is missing a stream.

CONVERT10_S_NO_PRESENTATION
Indicates the specified storage object contains a Paintbrush object in DIB format and there is no presentation data in the **OLESTREAM**.

E_INVALIDARG
Indicates invalid value for the *pstg* or *lpolestm* parameters.

Comments

This function converts an OLE 2 storage object to OLE 1 format. The **OLESTREAM** code implemented for OLE 1 must be available.

On entry, the stream pointed to by *lpolestm* should be created and positioned just as it would be for an **OleSaveToStream** call. On exit, the stream contains the persistent representation of the object using OLE 1 storage.

Note Paintbrush objects are dealt with differently from other objects because their native data is in DIB format. When Paintbrush objects are converted using **OleConvertIStorageToOLESTREAM**, no presentation data is added to the **OLESTREAM**. To include presentation data, use the **OleConvertIStorageToOLESTREAMEx** function instead.

See Also

CoIsOle1Class, **OleConvertIStorageToOLESTREAMEx**, **OleConvertOLESTREAMToIStorage**, **OleConvertOLESTREAMToIStorageEx**

OleConvertIStorageToOLESTREAMEx   QuickInfo

Converts the specified storage object from OLE 2 structured storage to the OLE 1 storage model, including the presentation data. This is one of several compatibility functions. You can use these functions to determine if an object class is from OLE 1 and to convert objects between OLE 1 and OLE 2 storage formats.

```
HRESULT OleConvertIStorageToOLESTREAMEx(
    IStorage   * pStg,           //Points to the OLE 2 storage object to be converted
    CLIPFORMAT  cfFormat,        //Presentation data format
    LONG   lWidth,               //Width in HIMETRIC
    LONG   lHeight,              //Height in HIMETRIC
    DWORD   dwSize,              //Size of data in bytes
    STGMEDIUM   pmedium,         //Points to STGMEDIUM of data
    LPOLESTREAM   lpolestm       //Points to the stream where the OLE1 storage is written
  );
```

Parameters

pStg
Points to the storage object to be converted to an OLE 1 storage.

cfFormat
Specifies the format of the presentation data. May be NULL, in which case the *lWidth, lHeight, dwSize,* and *pmedium* parameters are ignored. *lWidth*
Specifies the width of the object presentation data in HIMETRIC units.

lHeight
Specifies the height of the object presentation data in HIMETRIC units.

dwSize
Specifies the size of the data to be converted, in bytes.

pmedium
Points to the **STGMEDIUM** structure for the serialized data to be converted. See **STGMEDIUM** for more information.

lpolestm
Points to a stream where the persistent representation of the object is saved using the OLE 1 storage model.

Return Values

S_OK
Indicates the conversion was completed successfully.

DV_E_STGMEDIUM
Specifies the *hGlobal* member of **STGMEDIUM** is NULL.

E_INVALIDARG
Specifies the *dwSize* parameter is NULL; or the *pstg* parameter is invalid; or the *lpolestm* parameter is invalid.

DV_E_TYMED
Specifies the *tymed* member of the **STGMEDIUM** structure is not TYMED_HGOLBAL or TYMED_ISTREAM.

Comments

This function converts an OLE 2 storage object to OLE 1 format. It differs from the **OleConvertIStorageToOLESTREAM** function in that the presentation data to be written to the OLE 1 storage is passed in.

Because **OleConvertIStorageToOLESTREAMEx** can specify which presentation data to convert, it can be used by applications that do not use OLE default caching resources but do use OLE's conversion resources.

The value of the tymed member of **STGMEDIUM** must be either TYMED_HGLOBAL or TYMED_ISTREAM. The medium is not released by **OleConvertIStorageToOLESTREAMEx**.

See Also

CoIsOle1Class, **OleConvertIStorageToOLESTREAM**, **OleConvertOLESTREAMToIStorage**, **OleConvertOLESTREAMToIStorageEx**

OleConvertOLESTREAMToIStorage QuickInfo

Converts the specified object from the OLE 1 storage model to an OLE 2 structured storage object without specifying presentation data. This is one of several compatibility functions. You can use these functions to determine if an object class is from OLE 1 and to convert objects between OLE 1 and OLE 2 storage formats.

```
HRESULT OleConvertOLESTREAMToIStorage(
  LPOLESTREAM lpolestm,             //Points to the stream where the OLE 1 storage is written
  IStorage * pstg,                  //Points to OLE 2 storage object
  const DVTARGETDEVICE * ptd        //Points to target device
 );
```

Parameters

lpolestm
Points to a stream that contains the persistent representation of the object in the OLE 1 storage format.

pstg
Points to the OLE 2 structured storage object.

ptd
Points to the target device for which the OLE 1 object is rendered.

Return Values

S_OK
Indicates the object was successfully converted.

CONVERT10_S_NO_PRESENTATION
Indicates object either has no presentation data or uses native data for its presentation.

DV_E_DVTARGETDEVICE or DV_E_DVTARGETDEVICE_SIZE
Indicates invalid value for *ptd*.

E_INVALIDARG
Indicates invalid value for *lpolestm*.

Comments

This function converts an OLE 1 object to an OLE 2 structured storage object. You can use this function to update OLE 1 objects to OLE 2 objects when a new version of the object application supports OLE 2.

On entry, the *lpolestm* parameter should be created and positioned just as it would be for an **OleLoadFromStream** function call. On exit, the *lpolestm* parameter is positioned just as it would be on exit from an **OleLoadFromStream** function, and the *pstg* parameter contains the uncommitted persistent representation of the OLE 2 storage object.

For OLE 1 objects that use native data for their presentation, the **OleConvertOLESTREAMToIStorage** function returns CONVERT10_S_NO_PRESENTATION. On receiving this return value, callers should call **IOleObject::Update** to get the presentation data so that it can be written to storage.

The following steps describe the conversion process:

1. Create a root **IStorage** object by calling the **StgCreateDocfile** function(..., &*pstg*).
2. Open the OLE 1 file (using **OpenFile** or another OLE 1 technique).
3. Using the OLE 1 procedure for reading files, read from the file until an OLE object is encountered.
4. Allocate an **IStorage** object from the root **IStorage** created in step 1:

```
pstg->lpVtbl->CreateStorage(...&pStgChild);
hRes = OleConvertIStorageToOLESTREAM(polestm, pStgChild);
```

```
  hRes = OleLoad(pStgChild, &IID_IOleObject, pClientSite, ppvObj);
```

5. Repeat step 3 until the file is completely read.

See Also

CoIsOle1Class, **OleConvertIStorageToOLESTREAM**, **OleConvertIStorageToOLESTREAMEx**, **OleConvertOLESTREAMToIStorageEx**

OleConvertOLESTREAMToIStorageEx QuickInfo

Converts the specified object from the OLE 1 storage model to an OLE 2 structured storage object including presentation data. This is one of several compatibility functions. You can use these functions to determine if an object class is from OLE 1 and to convert objects between OLE 1 and OLE 2 storage formats.

```
HRESULT OleConvertOLESTREAMToIStorageEx(
    LPOLESTREAM lpolestm,        //Points to the a stream where the OLE1 storage is written
    IStorage * pstg,             //Points to OLE 2 storage object
    CLIPFORMAT * pcfFormat,      //Points to location where presentation data is returned
    LONG * plWidth,              //Points to location where width value is returned
    LONG * plHeight,             //Points to location where height value is returned
    DWORD * pdwSize,             //Points to location where size is returned
    STGMEDIUM pmedium            //Points to location where the STGMEDIUM structure is returned
   );
```

Parameters

lpolestm
Points to a stream that contains the persistent representation of the object in the OLE 1 storage format.

pstg
Points to the OLE 2 structured storage object.

pcfFormat
Points to the location where the format of the presentation data is returned. May be NULL, indicating the absence of presentation data.

plWidth
Points to the location where the width value (in HIMETRIC) of the presentation data is returned.

plHeight
Points to the location where the height value (in HIMETRIC) of the presentation data is returned.

pdwSize
Points to the location where the size in bytes of the converted data is returned.

pmedium
Points to the location where the **STGMEDIUM** structure for the converted serialized data is returned.

Return Values

S_OK
Indicates the conversion was completed successfully.

DV_E_TYMED|
Specifies the value of the *tymed* member of **STGMEDIUM** is not TYMED_ISTREAM or TYMED_NULL.

Comments

This function converts an OLE 1 object to an OLE 2 structured storage object. You can use this function to update OLE 1 objects to OLE 2 objects when a new version of the object application supports OLE 2.

This function differs from the **OleConvertOLESTREAMToIStorage** function in that the presentation data read from the **OLESTREAM** structure is passed out and the newly created OLE 2 storage object does not contain a presentation stream.

Since this function can specify which presentation data to convert, it can be used by applications that do not use OLE's default caching resources but do use the conversion resources.

The *tymed* member of **STGMEDIUM** can only be TYMED_NULL or TYMED_ISTREAM. If TYMED_NULL, then the data will be returned in a global handle through the *hGlobal* member of **STGMEDIUM**, otherwise data will be written into the *pstm* member of this structure.

See Also

CoIsOle1Class, **OleConvertIStorageToOLESTREAM**, **OleConvertIStorageToOLESTREAMEx**, **OleConvertOLESTREAMToIStorage**

OleCreate QuickInfo

Creates an embedded object identified by a CLSID. You use it typically to implement the menu item that allows the end user to insert a new object.

```
STDAPI OleCreate(
  REFCLSID rclsid,              //CLSID of embedded object to be created
  REFIID riid,                  //Interface used to communicate with new object
  DWORD renderopt,              //RENDEROPT value indicating cached capabilities
  FORMATETC * pFormatEtc,       //FORMATETC value, depending upon RENDEROPT value
  IOleClientSite * pClientSite, //Client site pointer to request services from the container
  IStorage * pStg,              //IStorage instance to be used as storage for the object
  void ** ppvObject             //On return, pointer to new object
 );
```

Parameters

rclsid
Specifies the CLSID of the embedded object that is to be created.

riid
Identifies the interface, usually **IID_IOleObject**, through which the caller will communicate with the new object.

renderopt
A value from the enumeration **OLERENDER**, indicating the locally cached drawing capabilities the newly created object is to have. The **OLERENDER** value chosen affects the possible values for the *pFormatEtc* parameter.

pFormatEtc
Depending on which of the **OLERENDER** flags is used as the value of *renderopt*, may be one of the **FORMATETC** enumeration values. Refer to the **OLERENDER** enumeration for restrictions. This parameter, along with the *renderopt* parameter, specifies what the new object can cache initially.

pClientSite
Points to an instance of **IOleClientSite**, the primary interface through which the object will request services from its container. Value may be NULL.

pStg
Points to an instance of the **IStorage** interface to be used as the storage for the object. This parameter may not be NULL.

ppvObject
Upon successful return, receives a pointer to the interface requested in *riid* on the newly created object.

Return Values

S_OK
Indicates embedded object created successfully.

E_OUTOFMEMORY
Indicates embedded object cannot be created due to lack of memory.

Comments

The **OleCreate** function creates a new embedded object, and is typically called to implement the menu item Insert New Object**.** When **OleCreate** returns, the object it has created is blank (contains no data), unless *renderopt* is OLERENDER_DRAW or OLERENDER_FORMAT, and is loaded. Containers typically then call the **OleRun** function or **IOleObject::DoVerb** to show the object for initial editing.

The *rclsid* parameter specifies the CLSID of the requested object. CLSIDs of registered objects are stored in the registry. When an application user selects Insert Object, a selection box allows the user to select the type of object desired from those in the system registry. When **OleCreate** is used to implement the Insert Object menu item, the CLSID associated with the selected item is assigned to the *rclsid* parameter of **OleCreate**.

The *riid* parameter specifies the interface the client will use to communicate with the new object. Upon successful return, the *ppvObject* parameter holds a pointer to the requested interface.

The created object's cache contains information that allows a presentation of a contained object when the container is opened. Information about what should be cached is passed in the *renderopt* and *pFormatetc* values. When **OleCreate** returns, the created object's cache is not necessarily filled. Instead, the cache is filled the first time the object enters the running state. The caller can add additional cache control with a call to **IOleCache::Cache** after the return of **OleCreate** and before the object is run. If *renderopt* is OLERENDER_DRAW or OLERENDER_FORMAT, **OleCreate** requires that the object support the **IOleCache** interface. There is no such requirement for any other value of *renderopt*.

If *pClientSite* is non-NULL, **OleCreate** calls **IOleObject::SetClientSite** with the *pClientSite* parameter. **IOleClientSite** is the primary interface by which an object requests services from its container.

If *pClientSite* is NULL, you must make a specific call to **IOleObject::SetClientSite** before attempting any operations.

See Also

OLERENDER, **FORMATETC**, **IOleClientSite**, **IOleObject**

OleCreateDefaultHandler QuickInfo

Creates a new instance of the default embedding handler. This instance is initialized so it creates a local server when the embedded object enters the running state.

```
HRESULT OleCreateDefaultHandler(
    REFCLSID clsid,               //Identifies OLE server to be loaded
    LPUNKNOWN pUnkOuter,          //Controlling unknown if aggregated; else NULL
    REFIID riid,                  //Interface for communicating with handler
    LPVOID FAR *lplpvObj          //Receives pointer to interface on handler
  );
```

Parameters

clsid
  Specifies CLSID identifying the OLE server to be loaded when the embedded object enters the running state.

pUnkOuter
  Points to the controlling **IUnknown** interface if the handler is to be aggregated; NULL if it is not to be aggregated.

riid
  Specifies the interface, usually **IID_IOleObject**, through which the caller will communicate with the handler.

lplpObj
  On return, receives a pointer to the interface requested in *riid* on the newly created handler.

Return Values

E_OUTOFMEMORY
  Indicates a handler cannot be created because the system is out of memory

NOERROR
  Indicates the creation operation was successful.

Comments

OleCreateDefaultHandler creates a new instance of the default embedding handler, initialized so it creates a local server identified by the *clsid* parameter when the embedded object enters the running state.

If the given object definition does not have a special handler, a call to **OleCreateDefaultHandler** produces the same results as a call to the **CoCreateInstance** function with the class context parameter assigned the value CLSCTX_INPROC_HANDLER.

When to Use

If you are writing a handler and want to use the services of the default handler, call **OleCreateDefaultHandler**. OLE also calls it internally when the CLSID specified in an object creation call is not registered.

See Also

CoCreateInstance, **CLSCTX**

OleCreateEmbeddingHelper QuickInfo

Creates an OLE embedding helper object using application-supplied code aggregated with pieces of the OLE default object handler. This helper object can be created and used in a specific context and role, as determined by the caller.

```
HRESULT OleCreateEmbeddingHelper(
    REFCLSID clsid,                 //The ID of the class to be helped
    LPUNKNOWN pUnkOuter,            //Points to controlling IUnknown
    DWORD flags,                    //Specifies the purpose for the helper
    LPCLASSFACTORY pCF,             //Points to IClassFactory instance
    REFIID riid,                    //The ID of the interface sought by caller
    LPVOID * lplpvObj               //Points to return point for helper
  );
```

Parameters

clsid
Specifies the CLSID of the class to be helped.

pUnkOuter
Points to the controlling **IUnknown** interface if the handler is to be aggregated; NULL if it is not to be aggregated.

flags
Specifies the role and creation context for the embedding helper; see the following comments for legal values.

pCF
Points to the application's **IClassFactory** instance; can be NULL.

riid
Specifies the interface ID desired by the caller.

lplpvObj
Points to where to return the newly created embedding helper.

Return Values

S_OK
Indicates that the OLE embedding helper was created successfully.

E_NOINTERFACE
Indicates that the interface is not supported by the object.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Indicates that one or more arguments are invalid.

E_UNEXPECTED
Indicates that an unexpected error occurred.

Comments

The **OleCreateEmbeddingHelper** function is used to create a new instance of the OLE default handler, which can be used to support objects in various roles. The caller passes a pointer to its **IClassFactory** implementation to **OleCreateEmbeddingHelper**. This object and the default handler are then aggregated to create the new embedding helper object.

The **OleCreateEmbeddingHelper** function is usually used to support one of the following implementations:

- An object application that is being used as both a container and a server, and which supports inserting objects into itself. In this case, the application registers its CLSID for different contexts by calling the **CoRegisterClassObject** function and specifying CLSCTX_LOCAL_SERVER | CLSCTX_INPROC_SERVER as the *dwClsctx* parameter, and REGCLS_MULTI_SEPARATE as the *flags* parameter. The local class is used to create the object and the in-process class creates the embedding helper, passing in the pointer to the first object's class factory in *pCF*.

- An in-process object handler, in which case, the DLL creates the embedding helper by passing in a pointer to a private implementation of **IClassFactory** in *pCF.*

The *flags* parameter indicates how the embedding helper is to be used and how and when the embedding helper is initialized. The values for *flags* are obtained by OR-ing together values from the following table:

Values for *flags* Parameter	**Purpose**
EMBDHLP_INPROC_HANDLER	Creates an embedding helper that can be used with DLL object applications; specifically, the helper exposes the caching features of the default object handler.
EMBDHLP_INPROC_SERVER	Creates an embedding helper that can be used with in-process servers.
EMBDHLP_CREATENOW	EMBDHLP_CREATENOW creates the application-supplied piece of the object immediately.
EMBDHLP_DELAYCREATE	EMBDHLP_DELAYCREATE delays creating the application-supplied piece of the object until the object is put into its running state.

Calling

```
OleCreateEmbeddingHelper
     (clsid, pUnkOuter, EMBDHLP_INPROC_HANDLER | EMBDHLP_CREATENOW,
     NULL, riid, ppvObj)
```

is the same as calling

```
OleCreateDefaultHandler(clsid, pUnkOuter, riid, ppvObj)
```

See Also

OleCreateDefaultHandler

OleCreateFromData QuickInfo

Creates an embedded object from a data transfer object retrieved either from the clipboard or as part of an OLE drag and drop operation. It is intended to be used to implement a paste from an OLE drag and drop operation.

```
HRESULT OleCreateFromData(
  LPDATAOBJECT pSrcDataObj,      //Data transfer object holding data used to create new object
  REFIID riid,                   //Interface to be used to communicate with the new object
  DWORD renderopt,               //Value from OLERENDER
  LPFORMATETC pFormatEtc,        //Depending on renderopt, pointer to value from FORMATETC
  LPOLECLIENTSITE pClientSite,   //Points to instance of IOleClientSite
  LPSTORAGE pStg,                //Points to instance of IStorage used to store object
  LPVOID FAR* ppvObj             //Receives a pointer to the interface requested in riid
 );
```

Parameters

pSrcDataObj
Points to the data transfer object that holds the data from which the object is created.

riid
Identifies the interface the caller later uses to communicate with the new object (usually **IID_IOleObject)**.

renderopt
A value from the enumeration **OLERENDER** that indicates the locally cached drawing or data retrieval capabilities the newly created object is to have. Additional considerations are described in the Comments section below.

pFormatEtc
A value from the enumeration **OLERENDER** that indicates the locally cached drawing or data retrieval capabilities the newly created object is to have. The **OLERENDER** value chosen affects the possible values for the *pFormatEtc* parameter.

pClientSite
Points to an instance of **IOleClientSite**, the primary interface through which the object will request services from its container. May be NULL.

pStg
Points to an instance of the **IStorage** interface to be used as the storage for the object. This parameter may not be NULL.

ppvObj
Receives a pointer to the interface requested in *riid* on the newly created object on return.

Return Values

S_OK
Indicates that the embedded object was created successfully.

OLE_E_STATIC
Indicates OLE can create only a static object.

DV_E_FORMATETC
Indicates no acceptable formats are available for object creation.

Comments

The **OleCreateFromData** function creates an embedded object from an instance of the **IDataObject** interface. The data object in this case is either the type retrieved from the clipboard with a call to the **OleGetClipboard** function or is part of an OLE drag and drop operation (the data object is passed to a call to **IDropTarget::Drop**).

If either the FileName or FileNameW clipboard format (CF_FILENAME) is present in the data transfer object, and CF_EMBEDDEDOBJECT or CF_EMBEDSOURCE do not exist, **OleCreateFromData** creates a package containing the indicated file. (Generally, it takes the first available format.) The Microsoft Windows NT File Manager places these formats on the clipboard when the user selects the File / Copy To Clipboard menu command.

If **OleCreateFromData** cannot create a package, it tries to create an object using the CF_EMBEDDEDOBJECT format. If that format is not available, **OleCreateFromData** tries to create it with the CF_EMBEDSOURCE format. If neither of these formats is available and the data transfer object supports the **IPersistStorage** interface, **OleCreateFromData** calls **IPersistStorage::Save** to have the object save itself.

If an existing linked object is selected, then copied, it appears on the clipboard as just another embeddable object. Consequently, a paste operation that invokes **OleCreateFromData** may create a linked object. After the paste operation, the container should call the **QueryInterface** function, requesting IID_IOleLink, to determine if a linked object was created.

Use the *renderopt* and *pFormatetc* parameters to control the caching capability of the newly created object. For information on how to determine what is to be cached, refer to the **OLERENDER** enumeration for a description of the interaction between *renderopt* and *pFormatetc*. However, values of *renderopt* also specifically affect the way **OleCreateFromData** initializes the cache.

When **OleCreateFromData** uses either the CF_EMBEDDEDOBJECT or the CF_EMBEDSOURCE clipboard format to create the embedded object, the main difference between the two is where the cache-initialization data is stored:

- CF_EMBEDDEDOBJECT indicates that the source is an existing embedded object. It already has in its cache the appropriate data, and OLE uses this data to initialize the cache of the new object.
- CF_EMBEDSOURCE indicates that the source data object contains the cache initialization information in formats other than CF_EMBEDSOURCE. **OleCreateFromData** uses these to initialize the cache of the newly embedded object.

The *renderopt* values affect cache initialization as follows:

Value	Description
OLERENDER_DRAW & OLERENDER_FORMAT	If the presentation information to be cached is currently present in the appropriate cache initialization pool, it is used. (Appropriate locations are in the source data object cache for CF_EMBEDDEDOBJECT, and in the other formats in the source data object for CF_EMBEDSOURCE.) If the information is not present, the cache is initially empty, but will be filled the first time the object is run. No other formats are cached in the newly created object.
OLERENDER_NONE	Nothing is to be cached in the newly created object. If the source has the CF_EMBEDDEDOBJECT format, any existing cached data that has been copied is removed.
OLERENDER_ASIS	If the source has the CF_EMBEDDEDOBJECT format, the cache of the new object is to contain the same cache data as the source

object. For CF_EMBEDSOURCE, nothing is to be cached in the newly created object.

This option should be used by more sophisticated containers. After this call, such containers would call **IOleCache::Cache** and **IOleCache::Uncache** to set up exactly what is to be cached. For CF_EMBEDSOURCE, they would then also call **IOleCache::InitCache**.

See Also

OleCreate

OleCreateFromFile QuickInfo

Creates an embedded object from the contents of a named file.

```
OleCreateFromFile(
    REFCLSID rclsid,                    //Reserved. Must be CLSID_NULL
    LPCOLESTR lpszFileName,             //Points to full path name of file used ot create object
    REFIID riid,                        //Interface to be used to communicate with new object
    DWORD renderopt,                    //Value from OLERENDER
    LPFORMATETC pFormatEtc,             //FORMATETC value depending on OLERENDER value
    LPOLECLIENTSITE pClientSite,        //Pointer to an instance of IOleClientSite
    LPSTORAGE pStg,                     //Pointer to IStorage instance to be used as object storage
    LPVOID FAR* ppvObj                  //Receives a pointer to the interface requested in riid
  );
```

Parameters

rclsid
Reserved. Must be CLSID_NULL.

lpszFileName
Points to the full path name specifying the file from which the object should be initialized.

riid
Identifies the interface the caller later uses to communicate with the new object (usually **IID_IOleObject**).

renderopt
A value from the enumeration **OLERENDER** that indicates the locally cached drawing or data-retrieval capabilities the newly created object is to have. The **OLERENDER** value chosen affects the possible values for the *pFormatEtc* parameter.

pFormatEtc
Depending on which of the **OLERENDER** flags is used as the value of *renderopt*, can be one of the **FORMATETC** enumeration values. Refer also to the **OLERENDER** enumeration for restrictions.

pClientSite
Points to an instance of **IOleClientSite**, the primary interface through which the object will request services from its container. May be NULL.

pStg
Points to an instance of the **IStorage** interface to be used as the storage for the object. This parameter may not be NULL.

ppvObj
Receives a pointer to the interface requested in *riid* on the newly created object on return.

Return Values

S_OK
Indicates embedded object successfully created.

STG_E_FILENOTFOUND
Indicates file not bound.

OLE_E_CANT_BINDTOSOURCE
Indicates not able to bind to source.

STG_E_MEDIUMFULL
Indicates the medium is full.

DV_E_TYMED
Indicates invalid TYMED.

DV_E_LINDEX
Indicates invalid LINDEX.

DV_E_FORMATETC
Indicates invalid **FORMATETC** structure.

E_OUTOFMEMORY
Indicates OLE libraries have not been properly initialized.

Comments

The **OleCreateFromFile** function creates a new embedded object from the contents of a named file. If the ProgID in the registration database contains the PackageOnFileDrop key, it creates a package. If not, the function calls the **GetClassFile** function to get the CLSID associated with the *lpszFileName* parameter, and then creates an OLE 2-embedded object associated with that CLSID. The *rclsid* parameter of **OleCreateFromFile** will always be ignored, and should be set to CLSID_NULL.

As for other **OleCreate***Xxx* functions, the newly created object is not shown to the user for editing, which requires a **DoVerb** operation. It is used in to implement insert file operations, such as the Create from File command in Word for Windows.

See Also

GetClassFile

OleCreateLink [QuickInfo]

Creates an OLE compound-document linked object.

```
HRESULT OleCreateLink(
   LPMONIKER pmkLinkSrc,          //Points to IMoniker instance indicating source of linked object
   REFIID riid,                   //Interface to be used to communicate with the new object
   DWORD renderopt,               //Value from OLERENDER
   LPFORMATETC pFormatEtc,        //FORMATETC value depending on renderopt value
   LPOLECLIENTSITE pClientSite,   //Pointer to an instance of IOleClientSite
   LPSTORAGE pStg,                //Pointer to an IStorage instance used as the object storage
   LPVOID FAR* ppvObj             //Receives a pointer to the interface requested in riid
  );
```

Parameters

pmkLinkSrc
Points to an **IMoniker** interface instance that indicates the source of the linked object.

riid
Identifies the interface the caller later uses to communicate with the new object (usually **IID_IOleObject**).

renderopt
Specifies a value from the enumeration **OLERENDER** that indicates the locally cached drawing or data-retrieval capabilities the newly created object is to have. Additional considerations are described in the Comments section below.

lpFormatEtc
Specifies a value from the enumeration **OLERENDER** that indicates the locally cached drawing or data retrieval capabilities the newly created object is to have. The **OLERENDER** value chosen affects the possible values for the *pFormatEtc* parameter.

pClientSite
Points to an instance of **IOleClientSite**, the primary interface through which the object will request services from its container. May be NULL.

pStg
Points to an instance of the **IStorage** interface to be used as the storage for the object. This parameter may not be NULL.

ppvObj
Receives a pointer to the interface requested in *riid* on the newly created object on return.

Return Values

S_OK
Indicates the compound-document linked object was created successfully.

OLE_E_CANT_BINDTOSOURCE
Indicates not able to bind to source. Binding is necessary to get the cache's initialization data.

Comments

Call **OleCreateLink** whenever a container creates a linked object within itself.

See Also

IOleObject::SetMoniker, **IOleClientSite::GetMoniker**

OleCreateLinkFromData **QuickInfo**

Creates a linked object from a data transfer object retrieved either from the clipboard or as part of an OLE drag and drop operation.

```
HRESULT OleCreateLinkFromData(
    LPDATAOBJECT pSrcDataObj,        //Points to data transfer object used to create linked object
    REFIID riid,                     //Interface to be used to communicate with the new object
    DWORD renderopt,                 //OLERENDER value
    LPFORMATETC pFormatEtc,          //FORMATETC value, depends on OLERENDER value
    LPOLECLIENTSITE pClientSite,     //Points to IOleClientSite instance
    LPSTORAGE pStg,                  //Pointer to an IStorage instance used as the object storage
    LPVOID FAR* ppvObj               //On return, points to requested object
  );
```

Parameters

pSrcDataObj
Points to the data transfer object from which the linked object is to be created.

riid
Identifies the interface the caller later uses to communicate with the new object (usually **IID_IOleObject**).

renderopt
Specifies a value from the enumeration **OLERENDER** that indicates the locally cached drawing or data-retrieval capabilities the newly created object is to have. Additional considerations are described in the Comments section below.

pFormatEtc
Specifies a value from the enumeration **OLERENDER** that indicates the locally cached drawing or data-retrieval capabilities the newly created object is to have. The **OLERENDER** value chosen affects the possible values for the *pFormatEtc* parameter.

pClientSite
Points to an instance of **IOleClientSite**, the primary interface through which the object will request services from its container. May be NULL.

pStg
Points to an instance of the **IStorage** interface to be used as the storage for the object. This parameter may not be NULL.

ppvObj
Receives a pointer to the interface requested in *riid* on the newly created object on return.

Return Values

S_OK
Indicates the linked object was created successfully.

CLIPBRD_E_CANT_OPEN
Indicates not able to open the clipboard.

OLE_E_CANT_GETMONIKER
Indicates not able to extract the object's moniker.

OLE_E_CANT_BINDTOSOURCE
Indicates not able to bind to source. Binding is necessary to get the cache's initialization data.

Comments

The **OleCreateLinkFromData** function is used to implement either a paste-link or a drag-link operation.

Its operation is similar to that of the **OleCreateFromData** function, except that it creates a link, and looks for different data formats. If the CF_LINKSOURCE format is not present, and either the FileName or FileNameW clipboard format is present in the data transfer object, **OleCreateLinkFromData** creates a package containing the link to the indicated file.

You use the *renderopt* and *pFormatetc* parameters to control the caching capability of the newly created object. For information on how to determine what is to be cached, refer to the **OLERENDER** enumeration for a description of the interaction between *renderopt* and *pFormatetc*. However, values of *renderopt* also specifically affect the way **OleCreateLinkFromData** initializes the cache as follows:

Value	**Description**
OLERENDER_DRAW, OLERENDER_FORMAT	If the presentation information is in the other formats in the source data object,this information is used. If the information is not present, the cache is initially empty, but will be filled the first time the object is run. No other formats are cached in the newly created object.
OLERENDER_NONE, OLERENDER_ASIS	Nothing is to be cached in the newly created object.

See Also

OleCreateLink

OleCreateLinkToFile [QuickInfo]

Creates an object that is linked to a file.

```
STDAPI OleCreateLinkToFile(
    LPWSTR lpszFileName,            //Points to source of linked object
    REFIID riid,                    //Interface to be used to communicate with the new object
    DWORD renderopt,                //Value from OLERENDER
    LPFORMATETC pFormatEtc,         //FORMATETC value depending on renderopt value
    IOleClientSite * pClientSite,   //Pointer to an instance of IOleClientSite
    IStorage * pStg,                //Pointer to an IStorage instance used as the object storage
    void ** ppvObj                  //Receives a pointer to the interface requested in riid
  );
```

Parameters

lpszFileName
Points to the source file to be linked to.

riid
Identifies the interface the caller later uses to communicate with the new object (usually **IID_IOleObject**).

renderopt
Specifies a value from the enumeration **OLERENDER** that indicates the locally cached drawing or data-retrieval capabilities the newly created object is to have. Additional considerations are described in the Comments section below.

pFormatEtc
Specifies a value from the enumeration **OLERENDER** the indicates the locally cached drawing or data retrieval capabilities the newly created object is to have. The **OLERENDER** value chosen affects the possible values for the *pFormatEtc* parameter.

pClientSite
Points to an instance of **IOleClientSite**, the primary interface through which the object will request services from its container. May be NULL.

pStg
Points to an instance of the **IStorage** interface to be used as the storage for the object. This parameter may not be NULL.

ppvObj
Receives a pointer to the interface requested in *riid* on the newly created object on return.

Return Values

S_OK
Indicates the object was created successfully.

STG_E_FILENOTFOUND
Indicates file name is invalid.

OLE_E_CANT_BINDTOSOURCE
Indicates not able to bind to source.

Comments

The **OleCreateLinkToFile** function differs from the **OleCreateLink** function because it can create links to non-OLE aware files (as well as files that are OLE-aware) using the Windows Packager.

See Also

OleCreateLink

OleCreateMenuDescriptor QuickInfo

Creates and returns an OLE menu descriptor (that is, an OLE-provided data structure that describes the menus) for OLE to use when dispatching menu messages and commands.

```
HOLEMENU OleCreateMenuDescriptor(
    HMENU hmenuCombined,                    //Specifies a handle to the combined menu
    LPOLEMENUGROUPWIDTHS                    //Number of menus in each group
lpMenuWidths
   );
```

Parameters

hmenuCombined
Specifies a handle to the combined menu created by the object.

lpMenuWidths
Points to an array of six LONG values giving the number of menus in each group.

Return Value

Returns the handle to the descriptor, or NULL if insufficient memory is available.

Comments

The **OleCreateMenuDescriptor** function can be called by the object to create a descriptor for the composite menu. OLE then uses this descriptor to dispatch menu messages and commands.

See Also

OleDestroyMenuDescriptor

OleCreateStaticFromData QuickInfo

The **OleCreateStaticFromData** function creates a static object (containing only a representation, with no native data) from a data transfer object.

```
HRESULT OleCreateStaticFromData(
    LPDATAOBJECT pSrcDataObj,         //Data transfer object holding data used to create new object
    REFIID riid,                      //Interface to be used to communicate with the new object
    DWORD renderopt,                  //Value from OLERENDER
    LPFORMATETC pFormatEtc,           //Depending on renderopt, pointer to value from FORMATETC
    LPOLECLIENTSITE pClientSite,      //Points to instance of IOleClientSite
    LPSTORAGE pStg,                   //Points to instance of IStorage used to store object
    LPVOID FAR* ppvObj                //Receives a pointer to the interface requested in riid
  );
```

Parameters

pSrcDataObj
Points to the data transfer object that holds the data from which the object will be created.

riid
Identifies the interface with which the caller is to communicate with the new object (usually **IID_IOleObject**).

renderopt
A value from the enumeration **OLERENDER** indicating the locally cached drawing or data-retrieval capabilities that the container wants in the newly created component. It is an error to pass the render options OLERENDER_NONE or OLERENDER_ASIS to this function.

pFormatEtc
Depending on which of the **OLERENDER** flags is used as the value of *renderopt*, may be one of the **FORMATETC** enumeration values. Refer to the **OLERENDER** enumeration for restrictions.

pClientSite
Points to an instance of **IOleClientSite**, the primary interface through which the object will request services from its container. May be NULL.

pStg
Points to an instance of the **IStorage** interface to be used as the storage for the object. This parameter may not be NULL.

ppvObj
When the function returns successfully, contains a pointer to the interface requested in *riid* on the newly created object.

Return Value

S_OK
Indicates that the object was successfully created.

Comments

The **OleCreateStaticFromData** function can convert any object that provides an **IDataObject** interface to a static object. It is useful in implementing the Convert To Picture option for OLE linking or embedding.

Static objects can be created only if the source supports one of the OLE-rendered clipboard formats: CF_METAFILEPICT, CF_DIB, or CF_ BITMAP, and CF_ENHMF.

Static objects can be pasted from the clipboard using **OleCreateStaticFromData**. The **OleQueryCreateFromData** function will return OLE_S_STATIC if one of CF_METAFILEPICT, CF_DIB, or CF_BITMAP is present and an OLE format is not present. But, in that case, the **OleCreateFromData** function does not automatically create the static object – if the container will be pasting a static object it should call **OleCreateStaticFromData**.

The new static object is of class CLSID_StaticMetafile (in the case of CF_METAFILEPICT) and CLSID_StaticDib (in the case of CF_DIB or CF_BITMAP). The static object sets the OLEMISC_STATIC and OLE_CANTLINKINSIDE bits returned from **IOleObject::GetMiscStatus**. The static object will have the aspect DVASPECT_CONTENT and a LINDEX of -1.

The *pDataObject* is still valid after **OleCreateStaticFromData** returns. It is the caller's responsibility to free *pDataObject* – OLE does not release it.

There cannot be more than one presentation stream in a static object.

Note The **OLESTREAM**<->**IStorage** conversion functions also convert static objects.

See Also

OleCreateFromData

OleDestroyMenuDescriptor QuickInfo

Called by the container to free the shared menu descriptor allocated by the **OleCreateMenuDescriptor** function.

```
void OleDestroyMenuDescriptor(
    HOLEMENU holemenu        //Handle to the shared menu descriptor
  );
```

Parameter

holemenu

Specifies a handle to the shared menu descriptor that was returned by the **OleCreateMenuDescriptor** function.

Return Value

None. (This function does not indicate failure.)

See Also

OleCreateMenuDescriptor

OleDoAutoConvert QuickInfo

Automatically converts an object to a new class if conversion is needed.

```
HRESULT OleDoAutoConvert(
    IStorage * pStg,            //Pointer to storage object to be converted
    LPCLSID pClsidNew           //Points to new CLSID of converted object
  );
```

Parameters

pStg
Points to the persistent storage of the object to be converted.

pClsidNew
Points to the new CLSID for the object being converted. If there was no automatic conversion, this may be the same as the original class.

Return Values

S_OK
Indicates no conversion is needed or a conversion was successfully completed.

REGDB_E_KEYMISSING
Indicates cannot read a key from the registration database.

E_OUTOFMEMORY
Indicates the conversion was not successful due to a lack of memory.

E_INVALIDARG
Indicates one or more arguments are invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

See also the **OleGetAutoConvert** function for other error return values.

See the **IStorage::OpenStorage** and **IStorage::OpenStream** methods for possible errors when accessing storage and stream objects.

See also the **IStream** interface for other error return values when it is not possible to determine the existing CLSID or when it is not possible to update the storage object with new information.

Comments

This function automatically converts an object if automatic conversion has previously been specified in the registry by the **OleSetAutoConvert** function. Object conversion means that the object is permanently associated with a new CLSID. Automatic conversion is typically specified by the setup program for a new version of an object application so that objects created by its older versions can be automatically updated.

A container application that supports object conversion should call the **OleDoAutoConvert** function each time it loads an object. If the container uses the **OleLoad** helper function, it need not call **OleDoAutoConvert** explicitly because **OleLoad** calls it internally.

The **OleDoAutoConvert** function first determines whether any conversion is required by calling the **OleGetAutoConvert** function. If no conversion is required, the function returns S_OK. Otherwise, the storage object is modified and converted by activating the new object application. The new object application reads the existing data format but when the object is saved, the new native format for the object application is saved.

If the object to be automatically converted is an OLE 1 object, the ItemName string is stored in a stream called "\1Ole10ItemName." If this stream does not exist, the object's item name is NULL.

The storage object must be in the unloaded state when **OleDoAutoConvert** is called.

See Also

OleSetAutoConvert

OleDraw QuickInfo

The **OleDraw** helper function can be used to draw objects more easily. You can use it instead of calling **IViewObject::Draw** directly.

```
STDAPI OleDraw(
  IUnknown * pUnk,          //Points to the view object to be drawn
  DWORD dwAspect,           //Specifies how the object is to be represented
  HDC hdcDraw,              //Identifies the device context on which to draw
  LPCRECT lprcBounds        //Specifies the rectangle in which the object is drawn
 );
```

Parameters

pUnk
Points to the view object that is to be drawn.

dwAspect
Specifies how the object is to be represented. Representations include content, an icon, a thumbnail, or a printed document. Valid values are taken from the enumeration **DVASPECT**. See **DVASPECT** for more information.

hdcDraw
Specifies the device context on which to draw. Cannot be a metafile device context.

lprcBounds
Points to a **RECT** structure specifying the rectangle in which the object should be drawn. This parameter is converted to a **RECTL** structure and passed to **IViewObject::Draw**.

Return Values

S_OK
Indicates object was successfully drawn.

E_INVALIDARG
Indicates one or more arguments are invalid.

E_OUTOFMEMORY
Indicates ran out of memory.

OLE_E_BLANK
Indicates there is no data to draw from.

E_ABORT
Indicates the draw operation was aborted.

VIEW_E_DRAW
Indicates an error occurred in drawing.

OLE_E_INVALIDRECT
Indicates the rectangle is invalid.

DV_E_NOIVIEWOBJECT
Indicates the object doesn't support the **IViewObject** interface.

Comments

The **OleDraw** helper function calls the **QueryInterface** method for the object specified (*pUnk*), asking for an **IViewObject** interface on that object. Then, **OleDraw** converts the **RECT** structure to a **RECTL** structure, and calls **IViewObject::Draw** as follows:

```
lpViewObj->Draw(dwAspect,-1,0,0,0,hdcDraw,&rectl,0,0,0);
```

Do not use **OleDraw** to draw into a metafile because it does not specify the *lprcWBounds* parameter required for drawing into metafiles.

See Also

IViewObject::Draw

OleDuplicateData QuickInfo

Duplicates the data found in the specified handle and returns a handle to the duplicated data. The source data is in a clipboard format. You use this function when implementing some of the data transfer interfaces such as **IDataObject**.

```
HANDLE OleDuplicateData(
    HANDLE hSrc,              //Handle of the source data
    CLIPFORMAT cfFormat,      //Clipboard format of the source data
    UINT uiFlags              //Flags used in global memory allocation
  );
```

Parameters

hSrc
Specifies the handle of the source data.

cfFormat
Specifies the clipboard format of the source data.

uiFlags
Specifies the flags to be used in allocate global memory for the copied data. These flags are passed to **GlobalAlloc**. If the value of *uiFlags* is NULL, GMEM_MOVEABLE is used as a default flag.

Return Values

HANDLE
Indicates data was successfully duplicated and the handle to the new data is the return value of the function.

NULL
Indicates error duplicating data.

Comments

The CF_METAFILEPICT, CF_PALETTE, or CF_BITMAP formats receive special handling. They are GDI handles and a new GDI object must be created instead of just copying the bytes. All other formats are duplicated byte-wise. For the formats that are duplicated byte-wise, *hSrc* must be a global memory handle.

OleFlushClipboard QuickInfo

Carries out the clipboard shutdown sequence. It also releases the instance of the **IDataObject** interface that was placed on the clipboard by the **OleSetClipboard** function.

HRESULT OleFlushClipboard();

Return Values

S_OK
  Indicates the clipboard has been flushed.

CLIPBRD_E_CANT_OPEN
  Indicates the Windows **OpenClipboard** function used within **OleFlushClipboard** failed.

CLIPBRD_E_CANT_CLOSE
  Indicates the Windows **CloseClipboard** function used within **OleFlushClipboard** failed.

Comments

If you are writing an application that can act as the source of a clipboard operation, you can call **OleFlushClipboard** to flush the clipboard and release the instance of **IDataObject** when your application is closed (e.g., if the user exits from your application). A call to the **OleSetClipboard** function using NULL will empty the clipboard.

The difference is that **OleFlushClipboard** leaves all formats offered by the data transfer object, including the OLE 1 compatibility formats, on the clipboard so they are available after application shutdown. In addition to OLE 1 compatibility formats, these include all formats offered on a global handle medium (all except for TYMED_FILE) and formatted with a NULL target device.

When your application closes, you should call **OleSetClipboard** using NULL if there is no need to leave data on the clipboard after shutdown or if data will be placed on the clipboard using the standard Windows clipboard functions. Applications should call **OleFlushClipboard** to enable pasting and paste-linking of OLE objects after shutdown.

Before calling **OleFlushClipboard**, it must be convenient to check the clipboard. You can call the **OleIsCurrentClipboard** function to determine if your data is still on the clipboard.

See Also

OleSetClipboard, **OleIsCurrentClipboard**, **IDataObject**,

OleGetAutoConvert QuickInfo

Returns the new CLSID to which the specified object can be automatically converted. This conversion CLSID is stored in the registration database.

```
HRESULT OleGetAutoConvert(
   REFCLSID clsidOld,         //CLSID of the object to be converted
   LPCLSID pClsidNew          //Pointer to new CLSID for object being converted
  );
```

Parameters

clsidOld
Specifies the CLSID for an object that can be automatically converted.

pClsidNew
Points to location where the new CLSID, if any, is returned. The original object, *clsidOld*, is automatically converted to this new CLSID when the object is opened. If there is no auto-conversion CLSID stored in the registration database, the value *clsidOld* is returned; the *pClsidNew* parameter cannot be NULL.

Return Values

S_OK
Indicates that a value was successfully returned through the *pclsidNew* parameter.

REGDB_E_CLASSNOTREG
Indicates the *clsidOld* CLSID is not properly registered in the registration database.

REGDB_E_READREGDB
Indicates error reading the registration database.

REGDB_E_KEYMISSING
Indicates auto convert is not active or there was no registration entry for the *clsidOld* parameter.

E_OUTOFMEMORY
Indicates the value was not returned due to a lack of memory.

E_INVALIDARG
Indicates one or more arguments are invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

Comments

This function returns the AutoConvertTo entry in the registration database for the specified object, which specifies whether objects of a given CLSID are to be automatically converted to a new CLSID. This is usually set up to convert files created by older versions of an application to the current version. If there is no AutoConvertTo entry, this function returns the value of *clsidOld*. This function is typically called by the **OleDoAutoConvert** function to determine if the object specified needs to be converted. If the object has an AutoConvertTo entry, then it will be converted.

Automatic conversion is typically set up during the installation of an object application. The setup program can take advantage of the AutoConvertTo subkey to tag a class of objects for automatic conversion to a different class of objects.

For example, a new version of a spreadsheet application may need to convert spreadsheets that were created with earlier versions of the application. These earlier version spreadsheet objects have a different CLSID than the new version. The setup program for the new version of the spreadsheet application tags objects in the registration database that have the old CLSID so they are automatically updated when they are loaded.

To tag the old spreadsheet objects, the setup program calls the **OleSetAutoConvert** function and passes it to the CLSIDs of the source and destination object classes. The **OleSetAutoConvert** function creates the required entry in the registration database. These entries appear in the registration database as subkeys of the CLSID key:

```
CLSID\{clsid}=MainUserTypeName\AutoConvertTo = clsid
```

See Also

OleSetAutoConvert, **OleDoAutoConvert**

OleGetClipboard QuickInfo

Retrieves a data object that you can use to access the clipboard contents.

```
HRESULT OleGetClipboard(
  IDataObject ** ppDataObj        //Points to location where a pointer to the data object is placed
 );
```

Parameter

ppDataObj
Points to the location where a pointer to the retrieved data object can be placed.

Return Values

S_OK
Indicates the data object was successfully retrieved.

E_OUTOFMEMORY
Indicates the data object was not returned due to a lack of memory.

CLIPBRD_E_CANT_CLOSE
Indicates the Windows **CloseClipboard** function used within **OleGetClipboard** failed.

CLIPBRD_E_CANT_OPEN
Indicates the Windows **OpenClipboard** function used with **OleGetClipboard** failed.

Comments

If you are writing an application that can accept data from the clipboard, you call **OleGetClipboard** to get a pointer to a data object that you can use to retrieve the contents of the clipboard.

There are three cases that the **OleGetClipboard** function handles:

1. The application that placed data on the clipboard with the **OleSetClipboard** function is still running.
2. The application that placed data on the clipboard with the **OleSetClipboard** function has subsequently called the **OleFlushClipboard** function.
3. There is data from a non-OLE application on the clipboard.

In the first case, the clipboard data object returned by the **OleGetClipboard** function may forward calls as necessary to the original data object placed on the clipboard and, thus, can potentially make RPC calls.

In the second case, OLE creates a default data object and returns it to the user. However, because the data on the Clipboard originated from an **OleSetClipboard** call, the OLE-provided data object contains more accurate information about the type of data on the Clipboard. In particular, the original medium (TYMED) on which the data was offered is known. Thus, if a data-source application offers a particular clipboard format, for example cfFOO, on an **IStorage** object and calls the **OleFlushClipboard** function, the storage object is copied into memory and the *hglobal* memory handle is put on the Clipboard. Then, when the **OleGetClipboard** function creates its default data object, the *hglobal* from the clipboard again becomes an **IStorage** object. Furthermore, the **FORMATETC** enumerator and the **IDataObject::QueryGetData** method would all correctly indicate that the original clipboard format (cfFOO) is again available on a TYMED_ISTORAGE.

In the third case, OLE still creates a default data object, but there is no special information about the data in the Clipboard formats (particularly for application-defined Clipboard formats). Thus, if an hGlobal-based storage medium were put on the Clipboard directly by a call to the **SetClipboardData** function, the **FORMATETC** enumerator and the **IDataObject::QueryGetData** method would not indicate that the data was available on a storage medium. A call to the **IDataObject::GetData** method for TYMED_ISTORAGE would succeed, however. Because of these limitations, it is strongly recommended that OLE-aware applications interact with the Clipboard using the OLE Clipboard functions.

The clipboard data object created by the **OleGetClipboard** function has a fairly extensive **IDataObject** implementation. The OLE-provided data object can convert OLE 1 clipboard format data into the representation expected by an OLE 2 caller. Also, any structured data is available on any structured or flat medium, and any flat data is available on any flat medium. However, GDI objects (such as metafiles and bitmaps) are only available on their respective mediums.

Note that the *tymed* member of the **FORMATETC** structure used in the **FORMATETC** enumerator contains the union of supported mediums. Applications looking for specific information (such as whether CF_TEXT is available on TYMED_HGLOBAL) should do the appropriate bit masking when checking this value.

If you call the **OleGetClipboard** function, you should only hold on to the returned **IDataObject** for a very short time. It consumes resources in the application that offered it.

See Also

OleSetClipboard

OleGetIconOfClass QuickInfo

Returns a handle to a metafile containing an icon and a string label for the specified CLSID.

```
HGLOBAL OleGetIconOfClass(
    REFCLSID rclsid,              //CLSID for which information is requested
    LPOLESTR lpszLabel,           //Points to string to use as label for icon
    BOOL fUseTypeAsLabel          //Indicates whether to use CLSID's user type name as icon label
  );
```

Parameters

rclsid
Specifies the CLSID for which the icon and string are requested.

lpszLabel
Points to a string to use as a label for the icon.

fUseTypeAsLabel
Indicates whether or not to use the user type string in the CLSID as the icon label.

Return Value

HGLOBAL
The *hGlobal* returned is a handle to a metafile that contains an icon and label for the specified CLSID. If the CLSID cannot be found in the registration database, NULL is returned.

See Also

OleGetIconOfFile, **OleMetafilePictFromIconAndLabel**

OleGetIconOfFile QuickInfo

Returns a handle to a metafile containing an icon and string label for the specified filename.

```
HGLOBAL OleGetIconOfFile(
   LPOLESTR lpszPath,          //Points to string that specifies the file for which info is requested
   BOOL fUseFileAsLabel        //Indicates whether to use the filename as the icon label
  );
```

Parameters

lpszPath
Points to a file for which the icon and string are requested.

fUseFileAsLabel
Indicates whether or not to use the filename as the icon label.

Return Value

HGLOBAL
The *hGlobal* returned is a handle to a metafile that contains an icon and label for the specified file. If there is no CLSID in the registration database for the file, then the string "Document" is used. If the value of *lpszPath* is NULL, then NULL is returned.

See Also

OleGetIconOfClass, **OleMetafilePictFromIconAndLabel**

OleInitialize QuickInfo

The **OleInitialize** function initializes the OLE library. You must initialize the library before you can call OLE functions.

```
HRESULT OleInitialize(
  LPVOID pvReserved      //Reserved, must be NULL
 );
```

Parameter

pvReserved
In 32-bit OLE, this parameter must be NULL. The 32-bit version of OLE does not support applications replacing OLE's allocator and if the parameter is not NULL, **OleInitialize** returns E_INVALIDARG.

Return Values

S_OK
Indicates the library was initialized successfully.

S_FALSE
Indicates OLE library is already initialized; a pointer to the IMalloc implementation was not used.

OLE_E_WRONGCOMPOBJ
Indicates COMPOBJ.DLL is the wrong version for OLE2.DLL.

E_OUTOFMEMORY
Indicates the library was not initialized because the system was out of memory.

E_INVALIDARG
Indicates an argument was invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

Comments

Applications *must* call **OleInitialize** before calling any other function in the OLE library.

Typically, **OleInitialize** is called only once in the process that uses the OLE library. There can be multiple calls, but subsequent calls return S_FALSE. To close the library gracefully, each successful call to **OleInitialize**, including those that return S_FALSE, *must* be balanced by a corresponding call to the **OleUninitialize** function.

See Also

OleUninitialize, **CoInitialize**

OleIsCurrentClipboard QuickInfo

Determines whether the **IDataObject** interface previously placed on the clipboard by the **OleSetClipboard** function is still on the clipboard.

```
HRESULT OleIsCurrentClipboard(
  IDataObject * pDataObj       //The data object previously copied or cut
 );
```

Parameter

pDataObject
Points to the instance of **IDataObject** that the caller previously placed on the clipboard.

Return Values

S_OK
Indicates that **IDataObject** is currently on the clipboard and the caller is the owner of the clipboard.

S_FALSE
Indicates that **IDataObject** is not on the clipboard.

Notes to Callers

OleIsCurrentClipboard only works for the data object used in the **OleSetClipboard** function. It cannot be called by the consumer of the data object to determine if the object that was on the clipboard at the previous **OleGetClipboard** call is still on the clipboard.

See Also

OleFlushClipboard, **OleSetClipboard**

OleIsRunning QuickInfo

Determines whether an object is currently in the running state.

```
BOOL OleIsRunning(
   LPOLEOBJECT pObject          //Pointer to IOleObject interface
  );
```

Parameter

pObject
  Points to an the object's **IOleObject** interface.

Return Value

The return value is TRUE if the object is running; otherwise, it is FALSE.

Comments

You can use **OleIsRunning** and **IRunnableObject::IsRunning** interchangeably. **OleIsRunning** queries the object for a pointer to the **IRunnableObject** interface. If successful, the function returns the results of calling the **IRunnableObject::IsRunning** method.

Note The implementation of **OleIsRunning** in earlier versions of OLE differs from that described here.

For more information on using this function, see **IRunnableObject::IsRunning**.

See Also

IRunnableObject::IsRunning

OleLoad [QuickInfo]

Loads into memory an object nested within a specified storage object.

```
HRESULT OleLoad(
    IStorage * pStg,                //Points to the storage object from which to load
    REFIID riid,                    //Interface to use for the object being loaded
    IOleClientSite * pClientSite,   //Points to the client site for the object
    LPVOID * ppvObj                 //Points to the newly loaded object
  );
```

Parameters

pStg
Points to the storage object from which to load the specified object.

riid
Specifies the interface that the caller wants to use when talking to the object once it is loaded.

pClientSite
Points to the client site for the object being loaded.

ppvObj
Points to the newly loaded object.

Return Values

S_OK
Indicates the object was loaded successfully.

E_OUTOFMEMORY
Indicates the object could not be loaded due to lack of memory.

E_NOINTERFACE
Indicates the object does not support the specified interface.

See also the **IPersistStorage::Load** method for other error return values.

Comments

OLE containers load objects into memory by calling this function. When calling the **OleLoad** function, the container application passes in a pointer to the open storage object in which the nested object is stored. Typically, the nested object to be loaded is a child storage object to the container's root storage object. Using the OLE information stored with the object, the object handler (usually, the default handler) attempts to load the object. On completion of the **OleLoad** function, the object is said to be in the loaded state with its object application not running.

Some applications load all of the object's native data. Containers often defer loading the contained objects until required to do so. For example, until an object is scrolled into view and needs to be drawn, it does not need to be loaded.

The **OleLoad** function performs the following steps:

1. If necessary, performs an automatic conversion of the object (see the **OleDoAutoConvert** function).
2. Gets the CLSID from the open storage object by calling the **IStorage::Stat** method.
3. Calls the **CoCreateInstance** function to create an instance of the handler. If the handler code is not available, the default handler is used (see the **OleCreateDefaultHandler** function).
4. Calls the **IOleObject::SetClientSite** method with the *pClientSite* parameter to inform the object of its client site.
5. Calls the **QueryInterface** method for the **IPersistStorage** interface. If successful, the **IPersistStorage::Load** method is invoked for the object.
6. Queries and returns the interface identified by the *riid* parameter.

See Also

ReadClassStg, **IClassFactory::CreateInstance**, **IPersistStorage::Load**

OleLoadFromStream QuickInfo

Loads an object from the stream.

```
HRESULT OleLoadFromStream(
   IStream * pStm,           //Points to stream from which object is to be loaded
   REFIID iidInterface,      //Interface to use for the object being loaded
    void ** ppvObj           //Points to the newly loaded object
  );
```

Parameters

pStm
Points to the stream from which the object is to be loaded.

iidInterface
Specifies the interface the caller wants to use to talk to the object once it is loaded.

ppvObj
Points to the newly loaded object.

Return Values

S_OK
Indicates the object was successfully loaded.

E_OUTOFMEMORY
Indicates the object could not be loaded due to a lack of memory.

E_NOINTERFACE
Indicates the specified interface is not supported.

See also the **ReadClassStm** function for other error return values.

See also the **CoCreateInstance** function for other error return values.

See also the **IPersistStorage::Load** method for other error return values.

Comments

This function can be used to load an object that supports the **IPersistStream** interface. The CLSID of the object must immediately precede the object's data in the stream.

If the CLSID for the stream is CLSID_NULL, the *ppvObj* parameter is set to NULL.

See Also

OleSaveToStream

OleLockRunning QuickInfo

Locks an already running object into its running state or unlocks it from its running state.

```
HRESULT OleLockRunning(
  LPUNKNOWN pUnknown,        //Pointer to IUnknown interface
  BOOL fLock,                //Flag indicating whether object is locked
  BOOL fLastUnlockCloses     //Flag indicating whether to close object
 );
```

Parameters

pUnknown
Points to an object's **IUnknown** interface in order to obtain a pointer to **IRunnableObject::LockRunning**.

fLock
TRUE locks the object into its running state. FALSE unlocks the object from its running state.

fLastUnlockCloses
TRUE specifies that if the connection being released is the last external lock on the object, the object should close. FALSE specifies that the object should remain open until closed by the user or another process.

Return Values

S_OK
Indicates that the object was successfully locked or unlocked.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Indicates one or more arguments are invalid .

E_UNEXPECTED
Indicates an unexpected error occurred.

Comments

The **OleLockRunning** function saves you the trouble of calling the **IRunnableObject::LockRunning** method. You can use **OleLockRunning** and **IRunnableObject::LockRunning** interchangeably. **OleLockRunning** queries for an **IRunnableObject** pointer. If successful, it returns the results of calling **IRunnableObject::LockRunning**.

Note The implementation of **OleLockRunning** in earlier versions of OLE differs from that described here.

For more information on using this function, see **IRunnableObject::LockRunning**.

See Also

CoLockObjectExternal, **IRunnableObject::LockRunning**, **OleNoteObjectVisible**

OleMetafilePictFromIconAndLabel QuickInfo

Creates a **METAFILEPICT** structure that contains a metafile in which the icon and label are drawn.

```
HGLOBAL OleMetafilePictFromIconAndLabel(
    HICON hIcon,                    //Handle to the icon to be drawn into the metafile
    LPOLESTR lpszLabel,             //Points to the string to be used as the icon label
    LPOLESTR lpszSourceFile,        //Points to the string that contains the path to the icon file
    UINT iIconIndex                 //Index of icon in lpszSourceFile
  );
```

Parameters

hIcon
Handle to the icon that is to be drawn into the metafile.

lpszLabel
Points to the string to be used as the icon label.

lpszSourceFile
Points to the string that contains the path and filename of the icon file. This string can be obtained from the user or from the registration database.

iIconIndex
Provides the index to the icon within the *lpszSourceFile* file.

Return Value

HGLOBAL
Indicates the *hGlobal* returned is the handle to a **METAFILEPICT** structure containing the icon and label. The metafile uses the MM_ANISOTROPIC mapping mode.

See Also

OleGetIconOfClass, **OleGetIconOfFile**

OleNoteObjectVisible QuickInfo

Increments or decrements an external reference that keeps the object in the running state.

```
STDAPI OleNoteObjectVisible(
   LPUNKNOWN pUnknown,       //Points to an IUnknown instance
   BOOL fVisible             //Indicates whether object is visible
  );
```

Parameters

pUnknown
Points to an **IUnknown** instance for the object that is to be locked or unlocked.

fVisible
Indicates whether the object is visible. If TRUE, OLE holds the object visible and alive regardless of external or internal **IUnknown::AddRef** and **IUnknown::Release** operations; registrations; or revokes. If FALSE, OLE releases its hold and the object can be closed.

Return Values

S_OK
Indicates the object was successfully locked or unlocked.

E_OUTOFMEMORY
Indicates insufficient memory to perform the operation.

E_INVALIDARG
Indicates an invalid argument.

E_UNEXPECTED
Indicates an unexpected error.

Comments

The **OleNoteObjectVisible** function calls the **CoLockObjectExternal** function. It is provided as a separate function to reinforce the need to lock an object when it becomes visible to the user and to release the object when it becomes invisible. This creates a strong lock on behalf of the user to ensure that the object cannot be closed by its container while it is visible.

See Also

CoLockObjectExternal

OleQueryCreateFromData **QuickInfo**

Checks whether a data object has one of the formats that would allow it to become an embedded object through a call to either the **OleCreateFromData** or **OleCreateStaticFromData** function.

```
STDAPI OleQueryCreateFromData(
   IDataObject * pSrcDataObject        //Points to the data transfer object to be queried
  );
```

Parameter

pSrcDataObject
   Points to the data transfer object to be queried.

Return Values

S_OK
   Indicates formats that support embedded-object creation are present.

S_FALSE
   Indicates no formats are present that support either embedded- or static-object creation.

OLE_S_STATIC
   Indicates formats that support static-object creation are present.

Comments

When an application retrieves a data transfer object through a call to the **OleGetClipboard** function, the application should call **OleQueryCreateFromData** as part of the process of deciding to enable or disable the **Edit/Paste** or **Edit/Paste Special...** commands. It tests for the presence of the following formats in the data object:

CF_EMBEDDEDOBJECT
CF_EMBEDSOURCE
cfFileName
CF_METAFILEPICT
CF_DIB
CF_BITMAP

Determining that the data object has one of these formats does not absolutely guarantee that the object creation will succeed, but is intended to help the process.

If **OleQueryCreateFromData** finds one of the CF_METAFILEPICT, CF_BITMAP, or CF_DIB formats and none of the other formats, it returns OLE_S_STATIC, indicating that you should call the **OleCreateStaticFromData** function to create the embedded object.

If **OleQueryCreateFromData** finds one of the other formats (CF_EMBEDDEDOBJECT, CF_EMBEDSOURCE, or cfFileName), even in combination with the static formats, it returns S_OK, indicating that you should call the **OleCreateFromData** function to create the embedded object.

See Also

OleCreateFromData, **OleCreateStaticFromData**, **OleQueryLinkFromData**

OleQueryLinkFromData   QuickInfo

The **OleQueryLinkFromData** function determines whether an OLE linked object (rather than an OLE embedded object) can be created from a clipboard data object.

```
STDAPI OleQueryLinkFromData(
   IDataObject * pSrcDataObject        //Points to the clipboard data object to be used to create the new object
  );
```

Parameter

pSrcDataObject
Points to the clipboard data object from which the object is to be created.

Return Value

Returns S_OK if the **OleCreateLinkFromData** function can be used to create the linked object; otherwise S_FALSE.

Comments

The **OleQueryLinkFromData** function is similar to the **OleQueryCreateFromData** function, but determines whether an OLE linked object (rather than an OLE embedded object) can be created from the clipboard data object. If the return value is S_OK, the application can then attempt to create the object with a call to **OleCreateLinkFromData**, although this is not guaranteed to create a link successfully.

OleRegGetUserType [QuickInfo]

Returns from the registry the user type of the specified class. Developers of custom DLL object applications use this function to emulate the behavior of the OLE default handler.

```
HRESULT OleRegGetUserType(
    REFCLSID clsid,              //Reference to the class identifier
    DWORD dwFormOfType,          //Specifies form of type name
    LPOLESTR * pszUserType       //Pointer to storage of string pointer
  );
```

Parameters

clsid
Specifies the class whose user type is requested.

dwFormOfType
Specifies a value that describes the form of the user-presentable string from the enumeration **USERCLASSTYPE**.

pszUserType
Points to storage of the returned user type.

Return Values

S_OK
Indicates the user type was returned successfully.

E_OUTOFMEMORY
Indicates there is insufficient memory to complete the operation.

REGDB_E_CLASSNOTREG
Indicates there is no CLSID registered for the class object.

REGDB_E_READREGDB
Indicates there was an error reading the registry.

OLE_E_REGDB_KEY
Indicates the *ProgID = MainUserTypeName* and *CLSID = MainUserTypeName* keys are missing from the registry.

Comments

Object applications can ask OLE to get the user type name of a specified class in one of two ways. One way is to call **OleRegGetUserType**. The other is to return OLE_S_USEREG in response to calls by the default object handler to **IOleObject::GetUserType**. OLE_S_USEREG instructs the default handler to call **OleRegGetUserType**. Because DLL object applications cannot return OLE_S_USEREG, they must call **OleRegGetUserType** rather than delegating the job to the object handler.

The **OleRegGetUserType** function and its sibling functions, **OleRegGetMiscStatus**, **OleRegEnumFormatEtc**, and **OleRegEnumVerbs**, provide a way for developers of custom DLL object applications to emulate the behavior of OLE's default object handler in getting information about objects from the registry. By using these functions, you avoid the considerable work of writing your own, and the pitfalls inherent in working directly in the registry. In addition, you get future enhancements and optimizations of these functions without having to code them yourself.

See Also

IOleObject::GetUserType, **OleRegGetMiscStatus**, **OleRegEnumFormatEtc**, **OleRegEnumVerbs**, **USERCLASSTYPE**

OleRegGetMiscStatus QuickInfo

Returns from the registry miscellaneous information about the presentation and behaviors supported by the specified class. Used by developers of custom DLL object applications to emulate the behavior of the OLE default handler.

```
HRESULT OleRegGetMiscStatus(
    REFCLSID clsid,           //Reference to class identifier
    DWORD dwAspect,           //Value specifying aspect of requested class
    DWORD * pdwStatus         //Pointer to returned status information
  );
```

Parameters

clsid
Specifies the class whose status information is requested.

dwAspect
Specifies the aspect of the class whose information is requested.

pdwStatus
Points to storage of the returned status information.

Return Values

S_OK
Indicates the status information was returned successfully.

E_OUTOFMEMORY
Indicates there is insufficient memory to complete the operation.

REGDB_E_CLASSNOTREG
Indicates no CLSID is registered for the class object.

REGDB_E_READREGDB
Indicates an error occurred reading the registry.

OLE_E_REGDB_KEY
Indicates the GetMiscStatus key is missing from the registry.

Comments

Object applications can ask OLE to get miscellaneous status information in one of two ways. One way is to call **OleRegGetMiscStatus.** The other is to return OLE_S_USEREG in response to calls by the default object handler to **IOleObject::GetMiscStatus**. OLE_S_USEREG instructs the default handler to call **OleRegGetMiscStatus**. Because DLL object applications cannot return OLE_S_USEREG, they must call **OleRegGetMiscStatus** rather than delegating the job to the object handler.

OleRegGetMiscStatus and its sibling functions, **OleRegGetUserType**, **OleRegEnumFormatEtc**, and **OleRegEnumVerbs**, provide a way for developers of custom DLL object applications to emulate the behavior of OLE's default object handler in getting information about objects from the registry. By using these functions, you avoid the considerable work of writing your own, and the pitfalls inherent in working directly in the registry. In addition, you get future enhancements and optimizations of these functions without having to code them yourself.

See Also

DVASPECT, **FORMATETC**, **OLEMISC**, **IOleObject::GetMiscStatus**, **OleRegEnumFormatEtc**, **OleRegEnumVerbs**, **OleRegGetUserType**

OleRegEnumFormatEtc QuickInfo

Enumerates data formats that an OLE object server has registered in the registration database. An object application or object handler calls this function to obtain an enumeration of those formats. Developers of custom DLL object applications use this function to emulate the behavior of the default object handler.

```
HRESULT OleRegEnumFormatEtc(
  REFCLSID clsid,                          //Reference to class identifier
  DWORD dwDirection,                       //Value specifying data formats
  LPENUMFORMATETC * ppenumFormatetc        //Pointer to returned format information
 );
```

Parameters

clsid
Specifies the class whose formats are being requested.

dwDirection
Specifies whether to enumerate formats that can be passed to **IDataObject::GetData** or formats that can be passed to **IDataObject::SetData**. Valid values are taken from the enumeration **DATADIR**.

ppenumFormatetc
Points to where to return the enumeration.

Return Values

S_OK
Indicates the enumerator was returned successfully.

E_OUTOFMEMORY
Indicates there is insufficient memory to complete the operation.

REGDB_E_CLASSNOTREG
Indicates there is no CLSID registered for the class object.

REGDB_E_READREGDB
Indicates there was an error reading the registry.

OLE_E_REGDB_KEY
Indicates the DataFormats/GetSet key is missing from the registry.

Comments

Object applications can ask OLE to enumerate supported data formats in one of two ways. One way is to call **OleRegEnumFormatEtc.** The other is to return OLE_S_USEREG in response to calls by the default object handler to **IDataObject::EnumFormatEtc**. OLE_S_USEREG instructs the default handler to call **OleRegEnumFormatEtc**. Because DLL object applications cannot return OLE_S_USEREG, they must call **OleRegEnumFormatEtc** rather than delegating the job to the object handler.

The **OleRegEnumFormatEtc** function and its sibling functions, **OleRegGetUserType**, **OleRegGetMiscStatus**, and **OleRegEnumVerbs**, provide a way for developers of custom DLL object applications to emulate the behavior of OLE's default object handler in getting information about objects from the registry. By using these functions, you avoid the considerable work of writing your own, and the pitfalls inherent in working directly in the registry. In addition, you get future enhancements and optimizations of these functions without having to code them yourself.

See Also

IDataObject::EnumFormatEtc

OleRegEnumVerbs QuickInfo

Returns an enumeration of the registered verbs for the specified class. Developers of custom DLL object applications use this function to emulate the behavior of the default object handler.

```
HRESULT OleRegEnumVerbs(
   REFCLSID clsid,                          //Reference to class identifier
   LPENUMOLEVERB * ppenumOleVerb            //Pointer to returned enumerator
  );
```

Parameters

clsid
Specifies the class whose verbs are being requested.

ppenumOleVerb
Points to where to return the enumeration.

Return Values

S_OK
Indicates the enumerator was returned successfully.

E_OUTOFMEMORY
Indicates there is insufficient memory to complete the operation.

OLEOBJ_E_NOVERBS
Indicates no verbs are registered for the class.

REGDB_E_CLASSNOTREG
Indicates no CLSID is registered for the class object.

REGDB_E_READREGDB
Indicates an error occurred reading the registry.

OLE_E_REGDB_KEY
Indicates the DataFormats/GetSet key is missing from the registry.

Comments

Object applications can ask OLE to enumerate supported verbs in one of two ways. One way is to call **OleRegEnumVerbs**. The other way is to return OLE_S_USEREG in response to calls by the default object handler to **IOleObject::EnumVerbs**. OLE_S_USEREG instructs the default handler to call **OleRegEnumVerbs**. Because DLL object applications cannot return OLE_S_USEREG, they must call **OleRegEnumVerbs** rather than delegating the job to the object handler.

The **OleRegEnumVerbs** function and its sibling functions, **OleRegGetUserType**, **OleRegGetMiscStatus**, and **OleRegEnumFormatEtc**, provide a way for developers of custom DLL object applications to emulate the behavior of OLE's default object handler in getting information about objects from the registry. By using these functions, you avoid the considerable work of writing your own, and the pitfalls inherent in working directly in the registry. In addition, you get future enhancements and optimizations of these functions without having to code them yourself.

See Also

IOleObject::EnumVerbs

OleRun QuickInfo

Runs an object.

```
HRESULT OleRun(
  LPUNKNOWN pUnknown        //Pointer to IUnknown interface
 );
```

Parameter

pUnknown
Points to an object's **IUnknown** interface in order to obtain a pointer from the **IRunnableObject** interface in order to call **Run**.

Return Values

S_OK
Indicates the object was successfully placed in the running state.

OLE_E_CLASSDIFF
Indicates the source of an OLE link has been converted to a different class.

E_INVALIDARG
Indicates an argument is invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

Comments

The implementation of **OleRun** was changed in OLE 2.01 to coincide with the publication of the **IRunnableObject** interface. You can use **OleRun** and **IRunnableObject::Run** interchangeably. **OleRun** queries the object for a pointer to **IRunnableObject**. If successful, the function returns the results of calling the **IRunnableObject::Run** method.

Note The implementation of **OleRun** in earlier versions of OLE differs from that described here.

For more information on using this function, see **IRunnableObject::Run**.

See Also

IOleLink::BindToSource, **IRunnableObject::Run**

OleSave QuickInfo

Saves an object opened in transacted mode into the specified storage object.

```
HRESULT OleSave(
   IPersistStorage * pPS,        //Points to the object to be saved
   IStorage * pStg,              //Points to the destination storage to which pPS is saved
   BOOL fSameAsLoad              //Indicates whether the object was loaded from pstg or not
  );
```

Parameters

pPS
Points to the object to be saved.

pStg
Points to the destination storage object to which *pPS* is to be saved.

fSameAsLoad
TRUE indicates that *pStg* is the same storage object from which the object was loaded or created; FALSE indicates that *pstg* was loaded or created from a different storage object.

Return Values

S_OK
Indicates the object was successfully saved.

STG_E_MEDIUMFULL
Indicates the object could not be saved due to lack of disk space.

See also the **IPersistStorage::Save** method for other error return values.

Comments

The **OleSave** helper function handles the common scenario in which an object is open in transacted mode and is to be saved into the specified storage object which uses the OLE-provided compound file implementation. Other scenarios can be handled with the **IPersistStorage** and **IStorage** interfaces directly.

The **OleSave** function does the following:

1. Calls the **IPersistStorage::GetClassID** method to get the CLSID.
2. Writes the CLSID to the storage object using the **WriteClassStg** function.
3. Calls the **IPersistStorage::Save** method to save the object.
4. If there were no errors on the save; calls the **IPersistStorage::Commit** method to commit the changes.

Note Static objects are saved into a stream called CONTENTS. Static metafile objects get saved in "placeable metafile format" and static DIB data gets saved in "DIB file format." These formats are defined to be the OLE standards for metafile and DIB. All data transferred using an **IStream** interface or a file (that is; via **IDataObject::GetDataHere**) must be in these formats. Also; all objects whose default file format is a metafile or DIB must write their data into a CONTENTS stream using these standard formats.

See Also

IStorage, **IPersistStorage**

OleSaveToStream QuickInfo

Saves an **IPersistStream** object to the specified stream.

```
HRESULT OleSaveToStream(
   IPersistStream * pPStm,        //Points to the object to be saved
   IStream * pStm                 //Points to the destination stream to which the object is saved
  );
```

Parameters

pPStm
Points to the **IPersistStorage** object to be saved to the stream. Can be NULL, which has the effect of writing CLSID_NULL to the stream.

pStm
Points to the stream in which the object is to be saved.

Return Values

S_OK
Indicates the object was successfully saved.

STG_E_MEDIUMFULL
Indicates there is no space left on device to save the object.

See also the **WriteClassStm** function for other error return values.

See also the **IPersistStream::Save** method for other error return values.

Comments

This function simplifies saving an **IPersistStream** object to a stream. The object is saved preceded by its CLSID. The **OleSaveToStream** function performs the following steps:

1. Calls the **IPersistStream::GetClassID** method to get the object's CLSID.
2. Writes the CLSID to the stream with the **WriteClassStm** function.
3. Calls the **IPersistStream::Save** method with *fClearDirty* set to TRUE. Note that this clears the dirty bit in the object.

See Also

OleLoadFromStream

OleSetAutoConvert [QuickInfo]

Tags an object for automatic conversion to a different class when the object is loaded.

HRESULT OleSetAutoConvert(
    REFCLSID *clsidOld*,   //CLSID to be converted
    REFCLSID *clsidNew*   //New CLSID after conversion
  );

Parameters

clsidOld
    Specifies the CLSID of the object class to be converted.

clsidNew
    Specifies the CLSID of the object class that should replace *clsidOld*. This new CLSID replaces any existing auto-conversion information in the registration database for *clsidOld*. If this value is CLSID_NULL, any existing auto-conversion information for *clsidOld* is removed from the registration database.

Return Values

S_OK
    Indicates the object was tagged successfully.

REGDB_E_CLASSNOTREG
    Indicates the CLSID is not properly registered in the registration database.

REGDB_E_READREGDB
    Indicates error reading from the registration database.

REGDB_E_WRITEREGDB
    Indicates error writing to the registration database.

REGDB_E_KEYMISSING
    Indicates cannot read a key from the registration database.

E_OUTOFMEMORY
    Indicates the auto-conversion cannot be completed due to a lack of memory.

E_INVALIDARG
    Indicates one or more arguments are invalid.

E_UNEXPECTED
    Indicates an unexpected error occurred.

Comments

The **OleSetAutoConvert** function sets the entry for AutoConvertTo entry for *clsidOld* in the registration database. This function does no validation of whether an appropriate registration database entry for *clsidNew* currently exists. When it is called, the **OleSetAutoConvert** function records the new CLSID for the object under the AutoConvertTo subkey in the registration database for *clsidOld*.

Object conversion means that the object is permanently associated with a new CLSID. Automatic conversion is typically specified by the setup program for a new version of an object application, so that objects created by its older versions can be automatically updated. You can write the setup program to tag a class of objects for automatic conversion to a different class of objects.

For example, a new version of a spreadsheet application may need to convert spreadsheets that were created with earlier versions of the application. These earlier version spreadsheet objects have a different CLSID than the new version. The setup program for the new version of the spreadsheet application tags objects in the registration database that have the old CLSID so they are automatically updated when they are loaded.

When installing a new CLSID, setup programs may call **OleSetAutoConvert** for various *clsidOld* values for older versions of their applications. They also should call **OleSetAutoConvert**, specifying the new class as the *clsidOld* parameter, and setting the *clsidNew* parameter to CLSID_NULL to remove any existing conversion for the new class.

The **OleSetAutoConvert** function creates the required entry in the registration database. These entries appear in the registration database as subkeys of the CLSID key:

```
CLSID\{clsid}=MainUserTypeName\AutoConvertTo = clsid
```

See Also

OleDoAutoConvert

OleSetClipboard QuickInfo

Places the specified **IDataObject** instance on the clipboard making it accessible to the **OleGetClipboard** function.

```
HRESULT OleSetClipboard(
    IDataObject * pDataObj        //The data object being copied or cut
  );
```

Parameter

pDataObj
Points to an **IDataObject** instance from which the data to be placed on the clipboard can be obtained. This parameter can be NULL; in which case the clipboard is emptied.

Return Values

S_OK
Indicates the **IDataObject** interface was placed on the clipboard.

CLIPBRD_E_CANT_OPEN
Indicates the Windows **OpenClipboard** function used within **OleSetClipboard** failed.

CLIPBRD_E_CANT_EMPTY
Indicates the Windows **EmptyClipboard** function used within **OleSetClipboard** failed.

CLIPBRD_E_CANT_CLOSE
Indicates the Windows **CloseClipboard** function used within **OleSetClipboard** failed.

Comments

If you are writing an application that can act as the source of a clipboard operation, you must:

1. Create an instance of the **IDataObject** interface for the data being copied or cut to the clipboard. This **IDataObject** should be the same as the **IDataObject** used in OLE drag and drop operations.
2. Call **OleSetClipboard** to place the **IDataObject** on the clipboard and make it accessible to the **OleGetClipboard** function. **OleSetClipboard** also calls the **IUnknown::AddRef** method on your data object.
3. If you wish, release the **IDataObject** once you have placed it on the clipboard to free the **IUnknown::AddRef** counter in your application.
4. If the user is cutting data to the clipboard, remove the data from the document.

All formats are offered on the clipboard using delayed rendering. The formats necessary for OLE 1 compatibility are synthesized from the OLE 2 formats that are present and are also put on the clipboard.

The **OleSetClipboard** function assigns ownership of the clipboard to an internal OLE window handle. The reference count of the data object is increased by 1. This is done so that delayed rendering can be used. The reference count is decreased by a call to the **OleFlushClipboard** function or by a subsequent call to **OleSetClipboard** using NULL.

The **OleSetClipboard** function empties the current clipboard before any new data is made available. When an application opens the clipboard (either directly or indirectly by calling the Windows **OpenClipboard** function), the clipboard cannot be used by any other application until it is closed. If the clipboard is currently open by another application, **OleSetClipboard** fails. The internal OLE window handle satisfies WM_RENDERFORMAT messages by delegating them to the instance of **IDataObject** that is on the clipboard.

Passing NULL for **IDataObject** is legal and it empties the clipboard. If the contents of the clipboard are the result of a previous **OleSetClipboard** call and the clipboard is released; the **IDataObject** that was passed to the previous call is released. The clipboard owner should use this as a signal that the data it previously offered is no longer on the clipboard.

If you need to leave the data on the clipboard after your application is closed, you can call **OleFlushClipboard** instead of **OleSetClipboard** using NULL.

See Also

OleGetClipboard, **OleIsCurrentClipboard**,

OleSetContainedObject QuickInfo

Notifies an object that it is embedded in an OLE container, which ensures that reference counting is done correctly for containers that support links to embedded objects.

```
STDAPI OleSetContainedObject(
    LPUNKNOWN pUnk               //Pointer to embedded object
    BOOL fContained              //Indicates if the object is embedded
  );
```

Parameters

pUnk
Specifies a pointer to the **IUnknown** interface of the object

fContained
Indicates TRUE if the object is an embedded object; FALSE otherwise.

Return Values

S_OK
Indicates object was notified successfully.

E_OUTOFMEMORY
Indicates insufficient memory to perform the operation.

E_INVALIDARG
Indicates an invalid argument.

E_UNEXPECTED
Indicates an unexpected error.

Comments

The implementation of **OleSetContainedObject** was changed in OLE 2.01 to coincide with the publication of the **IRunnableObject** interface. You can use **OleSetContainedObject** and the **IRunnableObject::SetContainedObject** method interchangeably. The **OleSetContainedObject** function queries the object for a pointer to the **IRunnableObject** interface. If successful, the function returns the results of calling **IRunnableObject::SetContainedObject**.

Note The implementation of **OleSetContainedObject** in earlier versions of OLE differs from that described here.

See Also

IRunnableObject::SetContainedObject

OleSetMenuDescriptor QuickInfo

Installs or removes OLE dispatching code from the container's frame window.

```
HRESULT OleCreateMenuDescriptor(
   HOLEMENU holemenu,                          //Handle to the composite menu descriptor
   HWND hwndFrame,                             //Handle to the container's frame window
   HWND hwndActiveObject,                      //Handle to the object's in-place activation window
   LPOLEINPLACEFRAME lpFrame,                  //Container's frame window
   LPOLEINPLACEACTIVEOBJECT lpActiveObj        //Active in-place object
 );
```

Parameters

holemenu
Specifies the handle to the composite menu descriptor returned by the **OleCreateMenuDescriptor** function. If NULL, the dispatching code is unhooked.

hwndFrame
Specifies the handle to the container's frame window where the in-place composite menu is to be installed.

hwndActiveObject
Specifies the handle to the object's in-place activation window. OLE dispatches menu messages and commands to this window.

lpFrame
Points to the container's frame window.

lpActiveObj
Points to the active in-place object.

Return Values

S_OK
Indicates the menu was installed correctly.

E_INVALIDARG
Indicates one or more arguments are invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

E_FAIL
Indicates a Windows function call failed, indicating a resource allocation failure or improper arguments.

Comments

The container should call **OleSetMenuDescriptor** to install the dispatching code on *hwndFrame* when the object calls the **IOleInPlaceFrame::SetMenu** method, or to remove the dispatching code by passing NULL as the value for *holemenu* to **OleSetMenuDescriptor**.

OLE installs the context-sensitive help F1 message filter for the application if both the *lpFrame* and *lpActiveObj* parameters are non-NULL. Otherwise, the application must supply its own message filter.

See Also

OleCreateMenuDescriptor, **IOleInPlaceFrame::SetMenu**

OleTranslateAccelerator   QuickInfo

Called by the object application, allowing the container to translate its accelerators pointed to by *lpFrameInfo*.

```
HRESULT OleTranslateAccelertor(
  LPOLEINPLACEFRAME lpFrame,                //Possible location to send keystrokes
  LPOLEINPLACEFRAMEINFO lpFrameInfo,        //Accelerator table obtained from container
  LPMSG lpmsg                               //Structure containing the keystroke
 );
```

Parameters

lpFrame
Points to the **IOleInPlaceFrame** interface where the keystroke might be sent.

lpFrameInfo
Points to an **OLEINPLACEFRAMEINFO** structure containing the accelerator table obtained from the container.

lpmsg
Points to an **MSG** structure containing the keystroke.

Return Values

S_OK
Indicates the keystroke was processed.

S_FALSE
Indicates the object should continue processing this message.

E_INVALIDARG
Indicates one of the arguments is invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

Comments

While an object is UI active, it *always* has first chance at translating any messages received. If the object does not want to translate the message, it calls **OleTranslateAccelerator** to give the container a chance. If the keyboard input matches an accelerator found in the container-provided accelerator table, **OleTranslateAccelerator** passes the message and its command identifier on to the container through the **IOleInPlaceFrame::TranslateAccelerator** method. This method returns S_OK if the keystroke is consumed; otherwise it returns S_FALSE.

The **OleTranslateAccelerator** function is intended to be called only by local server object applications and not by in-process server objects. In the case of a local server object application, keyboard input goes directly to the object application's message pump. If the object does not translate the key, **OleTranslateAccelerator** gives the container application an opportunity to do it. In the case of an in-process server object, the keyboard input goes directly to the container's message pump.

Note   Accelerator tables for containers should be defined so they will work properly with object applications that do their own accelerator keystroke translations. These tables should take the form:

```
"char", wID, VIRTKEY, CONTROL
```

This is the most common way to describe keyboard accelerators. Failure to do so can result in keystrokes being lost or sent to the wrong object during an in-place session.

Objects can call the **IsAccelerator** function to see whether the accelerator keystroke belongs to the object or the container.

See Also

IsAccelerator, **IOleInPlaceFrame::TranslateAccelerator**

OleUIAddVerbMenu

Adds the Verb menu for the specified object to the given menu.

```
BOOL OleUIAddVerbMenu(
    LPOLEOBJECT *lpOleObj,      //Points to the selected object
    LPCTSTR lpszShortType,      //The short name corresponding to the object
    HMENU hMenu,                //The handle to the menu to modify
    UINT uPos,                  //The position of the menu item.
    UINT uIDVerbMin,            //The value at which to start the verbs
    UINT uIDVerbMax,            //The maximum ID value for object verbs
    BOOL bAddConvert,           //Specifies whether to add convert item
    UINT idConvert,             //The value to use for the convert item
    HMENU FAR * lphMenu         //Points to the cascading verb menu, if created
  );
```

Parameters

lpOleObj
The **LPOLEOBJECT** pointing to the selected object. If this is NULL, then a default disabled menu item is created.

lpszShortType
The **LPTSTR** with short type name (AuxName==2) corresponding to the *lpOleObj*. If the string is NOT known, then NULL may be passed. If NULL is passed, then **IOleObject::GetUserType** will be called to retrieve it. If the caller has easy access to the string, then it is faster to pass it in.

hMenu
The **HMENU** in which to make modifications.

uPos
The position of the menu item

iIDVerbMin
The **UINT** ID value at which to start the verbs.

uIDVerbMax
The **UINT** Maximum ID value to be used for object verbs. If *uIDVerbMax* is 0, then no maximum ID value is used.

bAddConvert
The BOOL specifying whether or not to add a Convert item to the bottom of the menu (preceded by a separator).

idConvert
The **UINT** ID value to use for the Convert menu item, if *bAddConvert* is TRUE.

lphMenu
The **HMENU FAR *** of the cascading verb menu if it's created. If there is only one verb, this will be filled with NULL.

Return Values

TRUE
Indicates *lpOleObj* was valid and at least one verb was added to the menu

FALSE
Indicates *lpOleObj* was NULL and a disabled default menu item was created.

Comments

If the object has one verb, the verb is added directly to the given menu. If the object has multiple verbs, a cascading sub-menu is created.

OleUIBusy

Invokes the standard Busy dialog box, allowing the user to manage concurrency.

```
UINT OleUIBusy(
   LPOLEUIBUSY   lpBZ    //Points to the initialization structure
  );
```

Parameter

lpBZ
Points to an **OLEUIBUSY** structure that contains information used to initialize the dialog box.

Return Values

Standard Success/Error Definitions

OLEUI_FALSE
Unknown failure (unused).

OLEUI_SUCCESS
No error, same as OLEUI_OK

OLEUI_OK
The user pressed theOK button.

OLEUI_CANCEL
Indicates that user has pressed the Cancel button and that the caller should cancel the operation.

OLEUI_BZ_SWITCHTOSELECTED
Indicates that the user has pressed Switch To and **OleUIBusy** was unable to determine how to switch to the blocking application. In this case, the caller should either take measures to attempt to resolve the conflict itself, if possible, or retry the operation. **OleUIBusy** will only return OLEUI_BZ_SWITCHTOSELECTED if the user has pressed the Switch To button, *hTask* is NULL and the BZ_NOTREESPONDING flag is set.

OLEUI_BZ_RETRYSELECTED
Indicates that the user has either pressed the Retry button or attempted to resolve the conflict (probably by switching to the blocking application). In this case, the caller should retry the operation.

OLEUI_BZ_CALLUNBLOCKED
Indicates that the dialog has been informed that the operation is no longer blocked.

Standard Field Validation Errors

OLEUI_ERR_STANDARDMIN
Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

OLEUI_ERR_STRUCTURENULL
The pointer to an OLEUIXXX structure passed into the function was NULL.

OLEUI_ERR_STRUCTUREINVALID
Insufficient permissions for read or write access to an OLEUIXXX structure.

OLEUI_ERR_CBSTRUCTINCORRECT
The *cbstruct* value is incorrect.

OLEUI_ERR_HWNDOWNERINVALID
The *hWndOwner* value is invalid.

OLEUI_ERR_LPSZCAPTIONINVALID
The *lpszCaption* value is invalid.

OLEUI_ERR_LPFNHOOKINVALID
The *lpfnHook* value is invalid.

OLEUI_ERR_HINSTANCEINVALID
The *hInstance* value is invalid.

OLEUI_ERR_LPSZTEMPLATEINVALID
The *lpszTemplate* value is invalid.

OLEUI_ERR_HRESOURCEINVALID
The *hResource* value is invalid.

Initialization Errors

OLEUI_ERR_FINDTEMPLATEFAILURE
Unable to find the dialog template.

OLEUI_ERR_LOADTEMPLATEFAILURE
Unable to load the dialog template.

OLEUI_ERR_DIALOGFAILURE
Dialog initialization failed.

OLEUI_ERR_LOCALMEMALLOC
A call to **LocalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_GLOBALMEMALLOC
A call to **GlobalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_LOADSTRING
Unable to **LoadString** localized resources from the library.

OLEUI_ERR_OLEMEMALLOC
A call to the standard *IMalloc* allocator failed.

Function Specific Errors

OLEUI_ERR_STANDARDMAX
Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

OLEUI_BZERR_HTASKINVALID
The *hTask* specified in the *hTask* member of the **OLEUIBUSY** structure is invalid.

Comments

The standard OLE Server Busy dialog box notifies the user that the server application is not receiving messages. The dialog then asks the user to either cancel the operation, switch to the task which is blocked, or continue waiting.

See Also

OLEUIBUSY

OleUICanConvertOrActivateAs

Determines if there are any OLE object classes in the registry that can be used to convert or activate the specified CLSID from.

```
BOOL OleUICanConvertOrActivateAs(
    REFCLSID rClsid,            //CLSID of the specified class
    BOOL fIsLinkedObject,       //Indicates whether the original object was a linked object
    WORD wFormat                //The format of the original class
  );
```

Parameters

rClsid
Points to the CLSID of the specified class.

fIsLinkedObject
TRUE if the original object is a linked object; FALSE otherwise.

wFormat
Specifies the format of the original class.

Return Values

TRUE
The specified class can be converted to or activated as another class.

FALSE
The specified class cannot be converted to or activated as another class.

Comments

Notes to Implementors

OleUICanConvertOrActivateAs searches the Registry for classes that include *wFormat* in their \Conversion\Readable\Main, \Conversion\ReadWriteable\Main, and \DataFormats\DefaultFile entries.

This function is useful for determining if a Convert... menu item should be disabled. If the CF_DISABLEDISPLAYASICON flag is specified in the call to **OleUIConvert**, then the Convert... menu item should be enabled only if **OleUICanConvertOrActivateAs** returns TRUE.

See Also

OleUIConvert

OleUIChangeIcon

Invokes the standard Change Icon dialog box, which allows the user to select an icon from an icon file, executable, or DLL.

```
UINT OleUIChangeIcon(
   LPOLEUICHANGEICON lpCI    //Points to the in-out structure for this dialog
  );
```

Parameter

lpCI
Points to the in-out structure for this dialog.

Return Values

Standard Success/Error Definitions

OLEUI_FALSE
Unknown failure (unused).

OLEUI_SUCCESS
No error, same as OLEUI_OK

OLEUI_OK
The user pressed the OK button.

OLEUI_CANCEL
The user pressed the Cancel button.

Standard Field Validation Errors

OLEUI_ERR_STANDARDMIN
Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

OLEUI_ERR_STRUCTURENULL
The pointer to an OLEUIXXX structure passed into the function was NULL.

OLEUI_ERR_STRUCTUREINVALID
Insufficient permissions for read or write access to an OLEUIXXX structure.

OLEUI_ERR_CBSTRUCTINCORRECT
The *cbstruct* value is incorrect.

OLEUI_ERR_HWNDOWNERINVALID
The *hWndOwner* value is invalid.

OLEUI_ERR_LPSZCAPTIONINVALID
The *lpszCaption* value is invalid.

OLEUI_ERR_LPFNHOOKINVALID
The *lpfnHook* value is invalid.

OLEUI_ERR_HINSTANCEINVALID
The *hInstance* value is invalid.

OLEUI_ERR_LPSZTEMPLATEINVALID
The *lpszTemplate* value is invalid.

OLEUI_ERR_HRESOURCEINVALID
The *hResource* value is invalid.

Initialization Errors

OLEUI_ERR_FINDTEMPLATEFAILURE
Unable to find the dialog template.

OLEUI_ERR_LOADTEMPLATEFAILURE
Unable to load the dialog template.

OLEUI_ERR_DIALOGFAILURE
Dialog initialization failed.

OLEUI_ERR_LOCALMEMALLOC
A call to **LocalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_GLOBALMEMALLOC
A call to **GlobalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_LOADSTRING
Unable to **LoadString** localized resources from the library.

OLEUI_ERR_OLEMEMALLOC
A call to the standard *IMalloc* allocator failed.

Function Specific Errors

OLEUI_ERR_STANDARDMAX
Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

OLEUI_CIERR_MUSTHAVECLSID
The *clsid* member was not the current CLSID.

OLEUI_CIERR_MUSTHAVECURRENTMETAFILE
The *hMetaPict* member was not the current metafile.

OLEUI_CIERR_SZICONEXEINVALID
The *szIconExe* value was invalid.

Comment

OleUIChangeIcon uses information contained in the **OLEUICHANGEICON** structure.

See Also

OLEUICHANGEICON

OleUIChangeSource

Invokes the Change Source dialog box, allowing the user to change the source of a link.

UINT OleUIChangeSource(
LPOLEUICHANGESOURCE *lpCS* //Points to the in-out structure
);

Parameter

lpCS
Points to the in-out structure for this dialog.

Return Values

Standard Success/Error Definitions

OLEUI_FALSE
Unknown failure (unused).

OLEUI_SUCCESS
No error, same as OLEUI_OK.

OLEUI_OK
The user pressed the OK button.

OLEUI_CANCEL
The user pressed the Cancel button.

Standard Field Validation Errors

OLEUI_ERR_STANDARDMIN
Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

OLEUI_ERR_STRUCTURENULL
The pointer to an OLEUIXXX structure passed into the function was NULL.

OLEUI_ERR_STRUCTUREINVALID
Insufficient permissions for read or write access to an OLEUIXXX structure.

OLEUI_ERR_CBSTRUCTINCORRECT
The *cbstruct* value is incorrect.

OLEUI_ERR_HWNDOWNERINVALID
The *hWndOwner* value is invalid.

OLEUI_ERR_LPSZCAPTIONINVALID
The *lpszCaption* value is invalid.

OLEUI_ERR_LPFNHOOKINVALID
The *lpfnHook* value is invalid.

OLEUI_ERR_HINSTANCEINVALID
The *hInstance* value is invalid.

OLEUI_ERR_LPSZTEMPLATEINVALID
The *lpszTemplate* value is invalid.

OLEUI_ERR_HRESOURCEINVALID
The *hResource* value is invalid.

Initialization Errors

OLEUI_ERR_FINDTEMPLATEFAILURE
Unable to find the dialog template.

OLEUI_ERR_LOADTEMPLATEFAILURE
Unable to load the dialog template.

OLEUI_ERR_DIALOGFAILURE
Dialog initialization failed.

OLEUI_ERR_LOCALMEMALLOC
A call to **LocalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_GLOBALMEMALLOC
A call to **GlobalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_LOADSTRING
Unable to **LoadString** localized resources from the library.

OLEUI_ERR_OLEMEMALLOC
A call to the standard *IMalloc* allocator failed.

Function Specific Errors

OLEUI_ERR_STANDARDMAX
Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

OLEUI_CSERR_LINKCNTRNULL
The *lpOleUILinkContainer* value is NULL.

OLEUI_CSERR_LINKCNTRINVALID
The *lpOleUILinkContainer* value is invalid.

OLEUI_CSERR_FROMNOTNULL
The *lpszFrom* value is not NULL.

OLEUI_CSERR_TONOTNULL
The *lpszTo* value is not NULL.

OLEUI_CSERR_SOURCEINVALID
The *lpszDisplayName* or *nFileLength* value is invalid, or cannot retrieve the link source.

OLEUI_CSERR_SOURCEPARSEERROR
The *nFilename* value is wrong.

Comments

The link source is not changed by the Change Source dialog itself. Instead, it is up to the caller to change the link source using the returned file and item strings. The Edit Links dialog typically does this for the caller.

See Also

OLEUICHANGESOURCE, **OleUIEditLinks**, **IOleUILinkContainer**

OleUIConvert

Invokes the standard Convert dialog box, allowing the user to change the type of a single specified object, or the type of all OLE objects of the specified object's class.

UINT OleUIConvert(
  LPOLEUICONVERT *lpCV* //Points to initialization structure
 );

Parameter

lpCV
 Points to an **OLEUICONVERT** structure that contains information used to initialize the dialog box.

Return Values

Standard Success/Error Definitions

OLEUI_FALSE
 Unknown failure (unused).

OLEUI_SUCCESS
 No error, same as OLEUI_OK

OLEUI_OK
 The user pressed the OK button.

OLEUI_CANCEL
 The user pressed the Cancel button.

Standard Field Validation Errors

OLEUI_ERR_STANDARDMIN
 Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

OLEUI_ERR_STRUCTURENULL
 The pointer to an OLEUIXXX structure passed into the function was NULL.

OLEUI_ERR_STRUCTUREINVALID
 Insufficient permissions for read or write access to an OLEUIXXX structure.

OLEUI_ERR_CBSTRUCTINCORRECT
 The *cbstruct* value is incorrect.

OLEUI_ERR_HWNDOWNERINVALID
 The *hWndOwner* value is invalid.

OLEUI_ERR_LPSZCAPTIONINVALID
 The *lpszCaption* value is invalid.

OLEUI_ERR_LPFNHOOKINVALID
 The *lpfnHook* value is invalid.

OLEUI_ERR_HINSTANCEINVALID
 The *hInstance* value is invalid.

OLEUI_ERR_LPSZTEMPLATEINVALID
 The *lpszTemplate* value is invalid.

OLEUI_ERR_HRESOURCEINVALID
 The *hResource* value is invalid.

Initialization Errors

OLEUI_ERR_FINDTEMPLATEFAILURE
 Unable to find the dialog template.

OLEUI_ERR_LOADTEMPLATEFAILURE
Unable to load the dialog template.

OLEUI_ERR_DIALOGFAILURE
Dialog initialization failed.

OLEUI_ERR_LOCALMEMALLOC
A call to **LocalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_GLOBALMEMALLOC
A call to **GlobalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_LOADSTRING
Unable to **LoadString** localized resources from the library.

OLEUI_ERR_OLEMEMALLOC
A call to the standard *IMalloc* allocator failed.

Function Specific Errors

OLEUI_ERR_STANDARDMAX
Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

OLEUI_CTERR_CLASSIDINVALID
A *clsid* value was invalid.

OLEUI_CTERR_DVASPECTINVALID
The *dvAspect* value was invalid. This member specifies the aspect of the object.

OLEUI_CTERR_CBFORMATINVALID
The *wFormat* value was invalid. This member specifies the data format of the object.

OLEUI_CTERR_STRINGINVALID
A string value (for example, *lpszUserType* or *lpszDefLabel*) was invalid.

Comments

OleUIConvert populates the Convert dialog's listbox with object classes by traversing the Registry and looking for entries in the Readable and ReadWritable keys. Every class that includes the original class' default file format in its Readable key is added to the Convert list, and every class that includes the original class' default file format in its ReadWritable key is added to the Activate As list. The Convert list is shown in the dialog's listbox when the Convert radio button is selected (the default selection), and the Activate As list is shown when Activate As is selected.

Note that you can change the type of all objects of a given class only when CF_CONVERTONLY is not specified.

The convert command, which invokes this function, should only be made available to the user if **OleUIConvertOrActivateAs** returns S_OK.

See Also

OLEUICONVERT, **OleUICanConvertOrActivateAs**

OleUIEditLinks

Invokes the standard Links dialog box, allowing the user to make modifications to a container's linked objects.

UINT OleUIEditLinks(
  LPOLEUIEDITLINKS *lpEL*   //Points to the initialization structure
 );

Parameter

lpEL
  Points to an **OLEUIEDITLINKS** structure that contains information used to initialize the dialog box.

Return Value

Standard Success/Error Definitions

OLEUI_FALSE
  Unknown failure (unused).

OLEUI_SUCCESS
  No error, same as OLEUI_OK.

OLEUI_OK
  The user pressed the OK button.

OLEUI_CANCEL
  The user pressed the Cancel button.

Standard Field Validation Errors

OLEUI_ERR_STANDARDMIN
  Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

OLEUI_ERR_STRUCTURENULL
  The pointer to an OLEUIXXX structure passed into the function was NULL.

OLEUI_ERR_STRUCTUREINVALID
  Insufficient permissions for read or write access to an OLEUIXXX structure.

OLEUI_ERR_CBSTRUCTINCORRECT
  The *cbstruct* value is incorrect.

OLEUI_ERR_HWNDOWNERINVALID
  The *hWndOwner* value is invalid.

OLEUI_ERR_LPSZCAPTIONINVALID
  The *lpszCaption* value is invalid.

OLEUI_ERR_LPFNHOOKINVALID
  The *lpfnHook* value is invalid.

OLEUI_ERR_HINSTANCEINVALID
  The *hInstance* value is invalid.

OLEUI_ERR_LPSZTEMPLATEINVALID
  The *lpszTemplate* value is invalid.

OLEUI_ERR_HRESOURCEINVALID
  The *hResource* value is invalid.

Initialization Errors

OLEUI_ERR_FINDTEMPLATEFAILURE
  Unable to find the dialog template.

OLEUI_ERR_LOADTEMPLATEFAILURE
Unable to load the dialog template.

OLEUI_ERR_DIALOGFAILURE
Dialog initialization failed.

OLEUI_ERR_LOCALMEMALLOC
A call to **LocalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_GLOBALMEMALLOC
A call to **GlobalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_LOADSTRING
Unable to **LoadString** localized resources from the library.

OLEUI_ERR_OLEMEMALLOC
A call to the standard *IMalloc* allocator failed.

Function Specific Errors

OLEUI_ERR_STANDARDMAX
Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

See Also

OLEUIEDITLINKS, **IOleUILinkContainer**

OleUIInsertObject

Invokes the standard Insert Object dialog box, which allows the user to select an object source and class name, as well as the option of displaying the object as itself or as an icon.

```
UINT OleUIInsertObject(
    LPOLEUIINSERTOBJECT lpIO    //Points to the in-out structure
  );
```

Parameter

lpIO
Points to the in-out structure for this dialog.

Return Values

Standard Success/Error Definitions

OLEUI_FALSE
Unknown failure (unused).

OLEUI_SUCCESS
No error, same as OLEUI_OK.

OLEUI_OK
The user pressed the OK button.

OLEUI_CANCEL
The user pressed the Cancel button.

Standard Field Validation Errors

OLEUI_ERR_STANDARDMIN
Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

OLEUI_ERR_STRUCTURENULL
The pointer to an OLEUIXXX structure passed into the function was NULL.

OLEUI_ERR_STRUCTUREINVALID
Insufficient permissions for read or write access to an OLEUIXXX structure.

OLEUI_ERR_CBSTRUCTINCORRECT
The *cbstruct* value is incorrect.

OLEUI_ERR_HWNDOWNERINVALID
The *hWndOwner* value is invalid.

OLEUI_ERR_LPSZCAPTIONINVALID
The *lpszCaption* value is invalid.

OLEUI_ERR_LPFNHOOKINVALID
The *lpfnHook* value is invalid.

OLEUI_ERR_HINSTANCEINVALID
The *hInstance* value is invalid.

OLEUI_ERR_LPSZTEMPLATEINVALID
The *lpszTemplate* value is invalid.

OLEUI_ERR_HRESOURCEINVALID
The *hResource* value is invalid.

Initialization Errors

OLEUI_ERR_FINDTEMPLATEFAILURE
Unable to find the dialog template.

OLEUI_ERR_LOADTEMPLATEFAILURE
Unable to load the dialog template.

OLEUI_ERR_DIALOGFAILURE
Dialog initialization failed.

OLEUI_ERR_LOCALMEMALLOC
A call to **LocalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_GLOBALMEMALLOC
A call to **GlobalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_LOADSTRING
Unable to **LoadString** localized resources from the library.

OLEUI_ERR_OLEMEMALLOC
A call to the standard *IMalloc* allocator failed.

Function Specific Errors

OLEUI_ERR_STANDARDMAX
Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

OLEUI_IOERR_LPSZFILEINVALID
The *lpszFile* value is invalid or user has insufficient write access permissions.. This *lpszFile* member points to the name of the file linked to or inserted.

OLEUI_IOERR_LPFORMATETCINVALID
The *lpFormatEtc* value is invalid. This member identifies the desired format.

OLEUI_IOERR_PPVOBJINVALID
The *ppvOjb* value is invalid. This member points to the location where the pointer for the object is returned.

OLEUI_IOERR_LPIOLECLIENTSITEINVALID
The *lpIOleClientSite* value is invalid. This member points to the client site for the object.

OLEUI_IOERR_LPISTORAGEINVALID
The *lpIStorage* value is invalid. This member points to the storage to be used for the object.

OLEUI_IOERR_SCODEHASERROR
The *sc* member of *lpIO* has additional error information.

OLEUI_IOERR_LPCLSIDEXCLUDEINVALID
The *lpClsidExclude* value is invalid. This member contains the list of CLSIDs to exclude.

OLEUI_IOERR_CCHFILEINVALID
The *cchFile* or *lpszFile* value is invalid. The *cchFile* member specifies the size of the *lpszFile* buffer. The *lpszFile* member points to the name of the file linked to or inserted.

Comments

OleUIInsertObject allows the user to select the type of object to be inserted from a listbox containing the object applications registered on the user's system. To populate that listbox, **OleUIInsertObject** traverses the registry, adding every object server it finds that meets the following criteria:

- The registry entry does not include the **NotInsertable** key.
- The registry entry includes a OLE 1.0 style **Protocol\\StdFileEditing\\Server** key.
- The registry entry includes the **Insertable** key.
- The object's CLSID is not included in the list of objects to exclude (the *lpClsidExclude* member of **OLEUIINSERTOBJECT**).

By default, **OleUIInsertObject** does not validate object servers; however, if the IOF_VERIFYSERVEREXIST flag is included in the *dwFlags* member of the **OLEUIINSERTOBJECT** structure, then **OleUIInsertObject** verifies that the server exists. If it does not exist, then the server's object is not added to the list of available objects. Server validation is a time-extensive operation and is a significant performance factor.

To free an HMETAFILEPICT returned from the Insert Object or Paste Special dialog box, delete the attached metafile on the handle, as follows:

```
void FreeHmetafilepict(HMETAFILEPICT hmfp)
{
    if (hmfp != NULL)
        {
        LPMETAFILEPICT pmfp = GlobalLock(hmfp);

        DeleteMetaFile(pmfp->hMF);
        GlobalUnlock(hmfp);
        GlobalFree(hmfp);
        }
}  // FreeHmetafilepict
```

See Also

OLEUIINSERTOBJECT

OpenFile **DeleteMetaFile**, **GlobalUnlock**, **GlobalFree** in Win32

OleUIObjectProperties

Invokes the Object Properties dialog, which displays General, View, Link information about an object.

```
UINT OleUIObjectProperties(
                                        //Points to the OLEUIOBJECTPROPS structure
LPOLEUIOBJECTPROPS
lpOP
   );
```

Parameter*lpOP*
Points to the **OLEUIOBJECTPROPS** structure.

Return Values

Standard Success/Error Definitions

OLEUI_FALSE
Unknown failure (unused).

OLEUI_SUCCESS
No error, same as OLEUI_OK.

OLEUI_OK
The user pressed the OK button.

OLEUI_CANCEL
The user pressed the Cancel button.

Standard Field Validation Errors

OLEUI_ERR_STANDARDMIN
Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

OLEUI_ERR_STRUCTURENULL
The pointer to an OLEUIXXX structure passed into the function was NULL.

OLEUI_ERR_STRUCTUREINVALID
Insufficient permissions for read or write access to an OLEUIXXX structure.

OLEUI_ERR_CBSTRUCTINCORRECT
The *cbstruct* value is incorrect.

OLEUI_ERR_HWNDOWNERINVALID
The *hWndOwner* value is invalid.

OLEUI_ERR_LPSZCAPTIONINVALID
The *lpszCaption* value is invalid.

OLEUI_ERR_LPFNHOOKINVALID
The *lpfnHook* value is invalid.

OLEUI_ERR_HINSTANCEINVALID
The *hInstance* value is invalid.

OLEUI_ERR_LPSZTEMPLATEINVALID
The *lpszTemplate* value is invalid.

OLEUI_ERR_HRESOURCEINVALID
The *hResource* value is invalid.

Initialization Errors

OLEUI_ERR_FINDTEMPLATEFAILURE
Unable to find the dialog template.

OLEUI_ERR_LOADTEMPLATEFAILURE

Unable to load the dialog template.

OLEUI_ERR_DIALOGFAILURE
Dialog initialization failed.

OLEUI_ERR_LOCALMEMALLOC
A call to **LocalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_GLOBALMEMALLOC
A call to **GlobalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_LOADSTRING
Unable to **LoadString** localized resources from the library.

OLEUI_ERR_OLEMEMALLOC
A call to the standard *IMalloc* allocator failed.

Function Specific Errors

OLEUI_ERR_STANDARDMAX
Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

OLEUI_OPERR_SUBPROPNULL
lpGP or *lpVP* is NULL, or *dwFlags* and OPF_OBJECTISLINK and *lpLP* are NULL.

OLEUI_OPERR_SUBPROPINVALID
Insufficient write-access permissions for the structures pointed to by *lpGP*, *lpVP*, or *lpLP*.

OLEUI_OPERR_PROPSHEETNULL
The *lpLP* value is NULL.

OLEUI_OPERR_PROPSHEETINVALID
Insufficient write-access for one or more of the structures used by OLEUIOBJECTPROPS.

OLEUI_OPERR_SUPPROP
The sub-link property pointer, *lpLP*, is NULL.

OLEUI_OPERR_PROPSINVALID
Insufficient write access for the sub-link property pointer, *lpLP*.

OLEUI_OPERR_PAGESINCORRECT
Some sub-link properties of the *lpPS* member are incorrect.

OLEUI_OPERR_INVALIDPAGES
Some sub-link properties of the *lpPS* member are incorrect.

OLEUI_OPERR_NOTSUPPORTED
A sub-link property of the *lpPS* member is incorrect.

OLEUI_OPERR_DLGPROCNOTNULL
A sub-link property of the *lpPS* member is incorrect.

OLEUI_OPERR_LPARAMNOTZERO
A sub-link property of the *lpPS* member is incorrect.

OLEUI_GPERR_STRINGINVALID
A string value (for example, *lplpszLabel* or *lplpszType*) is invalid.

OLEUI_GPERR_CLASSIDINVALID
The *clsid* value is invalid.

OLEUI_GPERR_LPCLSIDEXCLUDEINVALID
The *ClsidExcluded* value is invalid.

OLEUI_GPERR_CBFORMATINVALID
The *wFormat* value is invalid.

OLEUI_VPERR_METAPICTINVALID
The *hMetaPict* value is invalid.

OLEUI_VPERR_DVASPECTINVALID

The *dvAspect* value is invalid.

OLEUI_OPERR_PROPERTYSHEET
The *lpPS* value is incorrect.

OLEUI_OPERR_OBJINFOINVALID
The *lpObjInfo* value is NULL or the calling process doesn't have read access.

OLEUI_OPERR_LINKINFOINVALID
The *lpLinkInfo* value is NULL or the calling process doesn't have read access.

Comment

OleUIObjectProperties is passed an **OLEUIOBJECTPROPS** structure, which supplies the information needed to fill in the General, View, and Link tabs of the Object Properties dialog.

See Also

IOleUIObjInfo, **IOleUILinkInfo**, **OLEUIOBJECTPROPS**, **OLEUIGNRLPROPS**, **OLEUIVIEWPROPS**, **OLEUILINKPROPS**

OleUIPasteSpecial

Invokes the standard Paste Special dialog box, allowing the user to select the format of the clipboard object to be pasted or paste-linked.

```
UINT OleUIPasteSpecial(*
    LPOLEUIPASTESPECIAL lpPS     //In-out structure for this dialog
  );
```

Parameter

lpPS
LPOLEUIPASTESPECIAL pointing to the in-out structure for this dialog.

Return Values

Standard Success/Error Definitions

OLEUI_FALSE
Unknown failure (unused).

OLEUI_SUCCESS
No error, same as OLEUI_OK

OLEUI_OK
The user pressed the OK button.

OLEUI_CANCEL
The user pressed the Cancel button.

Standard Field Validation Errors

OLEUI_ERR_STANDARDMIN
Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

OLEUI_ERR_STRUCTURENULL
The pointer to an OLEUIXXX structure passed into the function was NULL.

OLEUI_ERR_STRUCTUREINVALID
Insufficient permissions for read or write access to an OLEUIXXX structure.

OLEUI_ERR_CBSTRUCTINCORRECT
The *cbstruct* value is incorrect.

OLEUI_ERR_HWNDOWNERINVALID
The *hWndOwner* value is invalid.

OLEUI_ERR_LPSZCAPTIONINVALID
The *lpszCaption* value is invalid.

OLEUI_ERR_LPFNHOOKINVALID
The *lpfnHook* value is invalid.

OLEUI_ERR_HINSTANCEINVALID
The *hInstance* value is invalid.

OLEUI_ERR_LPSZTEMPLATEINVALID
The *lpszTemplate* value is invalid.

OLEUI_ERR_HRESOURCEINVALID
The *hResource* value is invalid.

Initialization Errors

OLEUI_ERR_FINDTEMPLATEFAILURE
Unable to find the dialog template.

OLEUI_ERR_LOADTEMPLATEFAILURE
    Unable to load the dialog template.

OLEUI_ERR_DIALOGFAILURE
    Dialog initialization failed.

OLEUI_ERR_LOCALMEMALLOC
    A call to **LocalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_GLOBALMEMALLOC
    A call to **GlobalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_LOADSTRING
    Unable to **LoadString** localized resources from the library.

OLEUI_ERR_OLEMEMALLOC
    A call to the standard *IMalloc* allocator failed.

Function Specific Errors

OLEUI_ERR_STANDARDMAX
    Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

OLEUI_IOERR_SRCDATAOBJECTINVALID
    The *lpSrcDataObject* field of **OLEUIPASTESPECIAL** is invalid

OLEUI_IOERR_ARRPASTEENTRIESINVALID
    The *arrPasteEntries* field of **OLEUIPASTESPECIAL** is invalid.

OLEUI_IOERR_ARRLINKTYPESINVALID
    The *arrLinkTypes* field of **OLEUIPASTESPECIAL** is invalid

OLEUI_PSERR_CLIPBOARDCHANGED
    The clipboard contents changed while the dialog was displayed.

OLEUI_PSERR_GETCLIPBOAARDFAILED
    The *lpSrcDataObj* member is incorrect.

Comments

The design of the paste special dialog assumes the following: if you are willing to permit a user to link to an object, you are also willing to permit the user to embed that object. For this reason, if any of the OLEUIPASTE_LINKTYPE flags associated with the **OLEUIPASTEFLAG** enumeration are set, then the OLEUIPASTE_PASTE flag must also be set in order for the data formats to appear in the paste special box.

To free an HMETAFILEPICT returned from the Insert Object or Paste Special dialog box, delete the attached metafile on the handle, as follows:

```
void FreeHmetafilepict(HMETAFILEPICT hmfp)
{
    if (hmfp != NULL)
        {
        LPMETAFILEPICT pmfp = GlobalLock(hmfp);

        DeleteMetaFile(pmfp->hMF);
        GlobalUnlock(hmfp);
        GlobalFree(hmfp);
        }
}  // FreeHmetafilepict
```

See Also

OLEUIPASTESPECIAL, **OLEUIPASTEFLAG**

DeleteMetaFile, **GlobalUnlock**, **GlobalFree** in Win32

OleUIPromptUser

Displays a dialog box with the specified template and returns the response (button id) from the user. This function is used to display OLE warning messages, for example, Class Not Registered.

```
int __export FAR CDECLOleUIPromtUser(
    int nTemplate,            //Resource number of dialog
    HWND hwndParent       //Link identifier
  );
```

Parameters

nTemplate
Specifies the resource number of the dialog to display. See Comments.

hwndParent
The parent of the dialog box. Specifies zero or more optional arguments. These parameters are passed to **wsprintf** to format the message string.

Return Value

Returns the button id selected by the user (template dependent).

Standard Success/Error Definitions

OLEUI_FALSE
Unknown failure (unused).

OLEUI_SUCCESS
No error, same as OLEUI_OK

OLEUI_OK
The user pressed the OK button.

OLEUI_CANCEL
The user pressed the Cancel button.

Standard Field Validation Errors

OLEUI_ERR_STANDARDMIN
Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

OLEUI_ERR_STRUCTURENULL
The pointer to an OLEUIXXX structure passed into the function was NULL.

OLEUI_ERR_STRUCTUREINVALID
Insufficient permissions for read or write access to an OLEUIXXX structure.

OLEUI_ERR_CBSTRUCTINCORRECT
The *cbstruct* value is incorrect.

OLEUI_ERR_HWNDOWNERINVALID
The *hWndOwner* value is invalid.

OLEUI_ERR_LPSZCAPTIONINVALID
The *lpszCaption* value is invalid.

OLEUI_ERR_LPFNHOOKINVALID
The *lpfnHook* value is invalid.

OLEUI_ERR_HINSTANCEINVALID
The *hInstance* value is invalid.

OLEUI_ERR_LPSZTEMPLATEINVALID
The *lpszTemplate* value is invalid.

OLEUI_ERR_HRESOURCEINVALID

The *hResource* value is invalid.

Initialization Errors

OLEUI_ERR_FINDTEMPLATEFAILURE
Unable to find the dialog template.

OLEUI_ERR_LOADTEMPLATEFAILURE
Unable to load the dialog template.

OLEUI_ERR_DIALOGFAILURE
Dialog initialization failed.

OLEUI_ERR_LOCALMEMALLOC
A call to **LocalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_GLOBALMEMALLOC
A call to **GlobalAlloc** or the standard *IMalloc* allocator failed.

OLEUI_ERR_LOADSTRING
Unable to **LoadString** localized resources from the library.

OLEUI_ERR_OLEMEMALLOC
A call to the standard *IMalloc* allocator failed.

Function Specific Errors

OLEUI_ERR_STANDARDMAX
Errors common to all dialogs lie in the range OLEUI_ERR_STANDARDMIN to OLEUI_ERR_STANDARDMAX. This value allows the application to test for standard messages in order to display error messages to the user.

Comments

The following dialog templates are defined in *Windows Interface Guidelines--A Guide for Designing Software*. The *nTemplate* parameter must be a currently defined resource; however, additional templates can be added to prompt.dlg.

```
IDD_LINKSOURCEUNAVAILABLE
IDD_CANNOTUPDATELINK
IDD_SERVERNOTREG
IDD_CANNOTRESPONDVERB
IDD_SERVERNOTFOUND
IDD_UPDATELINKS
```

See Also

wsprintf in Win32

OleUIUpdateLinks

Updates all links in the link container and displays a dialog box that shows the progress of the updating process. The process is stopped if the user presses the Stop button or when all links are processed.

```
BOOL OleUIUpdateLink(
    LPOLEUILINKCONTAINER                     //Points to the link container
lpOleUILinkCntr,
    HWND hwndParent,                         //Dialog box's parent window
    LPTSTR lpszTitle,                        //Dialog box title
    int cLinks                               //Number of links
  );
```

Parameters

lpOleUILinkCntr
The pointer to the Link Container.

hwndParent
The parent window of the dialog.

lpszTitle
The title of the dialog box.

cLinks
The total number of links.

Return Values

TRUE
The links were successfully updated.

FALSE
Unable to update the links.

See Also

IOleUILinkContainer::GetLinkUpdateOptions, **IOleUILinkContainer::UpdateLink**

OleUninitialize QuickInfo

Closes the OLE library, freeing any resources that it maintains.

void OleUninitialize();

Comments

You should call this function on application shutdown, as the last call made to the OLE library. **OleUninitialize** calls the **CoUninitialize** function internally to shut down the OLE Component Object(COM) Library.

The **OleInitialize** and **OleUninitialize** calls must be balanced – if there are multiple calls to the **OleInitialize** function, there must be the same number of calls to **OleUninitialize**: Only the **OleUninitialize** call corresponding to the **OleInitialize** call that actually initialized the COM library can close it.

See Also

OleInitialize, **CoUninitialize**

ProgIDFromCLSID

Retrieves the ProgID for a given CLSID.

```
HRESULT ProgIDFromCLSID(
   REFCLSID clsid,                //The CLSID for which the ProgID is requested
   LPOLESTR * lplpszProgID        //Receives a pointer to the requested ProgID on return
  );
```

Parameters

clsid
Specifies the CLSID for which the ProgID is requested.

lplpszProgID
Receives a pointer to the requested ProgID on return.

Return Values

S_OK
Indicates the ProgID was returned successfully.

REGDB_E_CLASSNOTREG
Indicates class not registered in the registry.

REGDB_E_READREGDB
Indicates error reading registry.

Comments

Every OLE 2 object class listed in the Insert Object dialog box must have a *programmatic identifier* (ProgID), a string that uniquely identifies a given class. In addition to determining the eligibility for the Insert Object dialog box, the ProgID can be used as an identifier in a macro programming language to identify a class. Finally, the ProgID is also the class name used for an OLE 2 class when placed in an OLE 1 container.

The **ProgIDFromCLSID** function uses entries in the registry to do the conversion. OLE 2 application authors are responsible for correctly configuring the registry at application installation time.

The ProgID string must be different than the class name of any OLE 1 application, including the OLE 1 version of the same application, if there is one. In addition, a ProgID string must not contain more than 39 characters or start with a digit. Except for a single period, it cannot contain any punctuation (including underscores).

The ProgID must *never* be shown to the user in the user interface. If you need a short human-readable string for an object, call **IOleObject::GetUserType**.

The **CLSIDFromProgID** function can be called to create a CLSID from a given ProgID. CLSIDs can be freed with the task allocator (refer to the **CoGetMalloc** function).

See Also

CLSIDFromProgID

PropagateResult

```
#define PropagateResult(hrPrevious, scBase) ((HRESULT) scBase)
```

PropagateResult is a no-op

This macro is obsolete and should not be used.

ReadClassStg QuickInfo

Reads the CLSID from the storage object.

```
HRESULT ReadClassStg(
    IStorage * pStg,          //Points to the storage containing the CLSID to be retrieved
    CLSID * pclsid            //Points to location for returning the CLSID
  );
```

Parameters

pStg
Points to the storage object containing the CLSID to be returned.

pclsid
Points to location where the CLSID is to be returned. May return CLSID_NULL.

Return Values

S_OK
Indicates the CLSID was returned successfully.

E_OUTOFMEMORY
Indicates the CLSID could not be returned due to lack of memory.

See also the **IStorage::Stat** method for other error return values.

See Also

This function is simply a helper function that calls the **IStorage::Stat** method and retrieves the CLSID from the resulting **STATSTG** structure.

See Also

OleLoad, **WriteClassStg**

ReadClassStm QuickInfo

Reads the CLSID previously written to a stream object with the **WriteClassStm** method.

```
HRESULT ReadClassStm(
   IStream * pStm,          //Points to the stream from which the CLSID is to be read
   CLSID * pclsid           //Points to the location where the CLSID is returned
  );
```

Parameters

pStm
Points to the stream object containing the CLSID to be read. This CLSID must have been previously written to the stream object using **WriteClassStm**.

pclsid
Points to the location where the CLSID is to be returned.

Return Values

S_OK
Indicates the CLSID was successfully returned.

STG_E_READFAULT
Indicates end of file was reached.

See also the **IStream::Read** method for other error return values.

Comments

Most applications do not call the **ReadClassStm** method. OLE calls it before making a call to an object's **IPersistStream::Load** implementation.

See Also

WriteClassStm, **ReadClassStg**, **WriteClassStg**

ReadFmtUserTypeStg QuickInfo

Returns the clipboard format and user type previously saved with the **WriteFmtUserTypeStg** function.

```
HRESULT ReadFmtUserTypeStg(
    IStorage * pStg,              //Points to storage object for which the values are to be retreived
    CLIPFORMAT * pcf,             //Points to location for returning the clipboard format
    LPWSTR * lplpszUserType       //Points to location for returning the user type string
  );
```

Parameters

pStg
Points to the storage object from which the information is to be read.

pcf
Points to the location where the clipboard format is to be returned. It can be NULL, indicating the format is of no interest to the caller.

lplpszUserType
Points to the location where the user type string is to be returned. It can be NULL, indicating the user type is of no interest to the caller.

Return Values

S_OK
Indicates the requested information was read successfully.

E_OUTOFMEMORY
Indicates the information could not be retrieved due to a lack of memory.

E_FAIL
Indicates the **WriteFmtUserTypeStg** function was never called on the object.

See also the **IStream::Read** method for other error return values.

Comments

This function returns the clipboard format and the user type string from the specified storage object. The **WriteClassStg** function must have been called before calling the **ReadFmtUserTypeStg** function.

See Also

WriteFmtUserTypeStg

RegisterDragDrop QuickInfo

Registers the specified window as a window that can be the target of an OLE drag and drop operation and specifies the **IDropTarget** instance to use for drop operations.

```
HRESULT RegisterDragDrop(
   HWND hwnd,                      //Window that can accept drops
   IDropTarget * pDropTarget       //Points to the IDropTarget interface
  );
```

Parameters

hwnd
Specifies the handle to a window that can be a target for an OLE drag and drop operation.

pDropTarget
Specifies the **IDropTarget** instance to use for communicating OLE drag and drop information for the specified window.

Return Values

S_OK
Indicates the application was registered successfully.

DRAGDROP_E_INVALIDHWND
Indicates invalid handle returned in the *hwnd* parameter.

DRAGDROP_E_ALREADYREGISTERED
Indicates the specified window has already been registered as a drop target.

E_OUTOFMEMORY
Indicates out of memory.

Comments

If your application can accept dropped objects during OLE drag and drop operations, you must call the **RegisterDragDrop** function. Do this whenever one of your application windows is available as a potential drop target, i.e., when the window appears unobscured on the screen.

The **RegisterDragDrop** function only registers one window at a time, so you must call it for each application window capable of accepting dropped objects.

As the mouse passes over unobscured portions of the target window during an OLE drag and drop operation, the **DoDragDrop** function calls the specified **IDropTarget::DragOver** method for the current window. When a drop operation actually occurs in a given window, the **DoDragDrop** function calls **IDropTarget::Drop**.

The **RegisterDragDrop** function also calls the **IUnknown::AddRef** method for your drop target interface.

See Also

RevokeDragDrop

ReleaseStgMedium [QuickInfo]

Frees the specified storage medium.

```
void ReleaseStgMedium(
   STGMEDIUM * pmedium        //Storage medium to be freed
  );
```

Parameter

pmedium
Points to the storage medium that is to be freed.

Return Value

None.

Comments

The **ReleaseStgMedium** function frees the specified storage medium. Use this function during data transfer operations where storage medium structures are parameters, such as **IDataObject::GetData** or **IDataObject::SetData**. This structure specifies the appropriate **IUnknown::Release** method for releasing the storage medium when it is no longer needed.

The **ReleaseStgMedium** function calls the appropriate method or function to release the specified storage medium. Either the original provider of the medium or the receiver of the medium can call **ReleaseStgMedium**.

In cases where the original provider of the medium is responsible for freeing the medium, the provider calls **ReleaseStgMedium**. The **ReleaseStgMedium** function first makes a call as described in the following table, depending on the type of storage medium being freed.

Medium	ReleaseStgMedium Action
TYMED_HGLOBAL	None.
TYMED_GDI	None.
TYMED_ENHMF	None.
TYMED_MFPICT	None.
TYMED_FILE	Frees the filename string using standard memory management mechanisms.
TYMED_ISTREAM	Calls **IStream::Release**.
TYMED_ISTORAGE	Calls **IStorage::Release**.

After making the call described in the preceding table, **ReleaseStgMedium** then calls the **IUnknown::Release** method for the specified storage medium. This method is specified in the storage medium structure.

In cases where the receiver of the medium is responsible for freeing the medium, the storage medium structure specifies NULL for the **IUnknown::Release** method. In this case, the receiver calls **ReleaseStgMedium,** which makes a call as described in the following table depending on the type of storage medium being freed.

Medium	ReleaseStgMedium Action
TYMED_HGLOBAL	Calls **GlobalFree** on the handle.
TYMED_GDI	Calls **DeleteObject** on the handle.
TYMED_ENHMF	Deletes the enhanced metafile.
TYMED_MFPICT	The *hMF* that it contains is deleted with **DeleteMetaFile**; then the handle itself is

	passed to **GlobalFree**.
TYMED_FILE	Frees the disk file by deleting it. Frees the file name string by using the standard memory management paradigm.
TYMED_ISTREAM	**Calls IStream::Release**.
TYMED_ISTORAGE	**Calls IStorage::Release**.

In either case after the call to **ReleaseStgMedium**, the specified storage medium is invalid and can no longer be used.

ResultFromScode

```
#define ResultFromScode(sc) ((HRESULT) (sc))
```

Converts an **SCODE** into an **HRESULT**.

This macro is obsolete and should not be used.

RevokeDragDrop [QuickInfo]

Revokes the registration of the specified application window as a potential target for OLE drag and drop operations.

```
HRESULT RevokeDragDrop(
   HWND hwnd         //Window that can accept drops
  );
```

Parameter

hwnd
Specifies the handle to a window previously registered as a target for an OLE drag and drop operation.

Return Values

S_OK
Indicates registration as a target window was revoked successfully.

DRAGDROP_E_INVALIDHWND
Indicates invalid handle returned in the *hwnd* parameter.

DRAGDROP_E_NOTREGISTERED
Indicates an attempt was made to revoke a drop target that has not been registered.

E_OUTOFMEMORY
Out of memory.

Comments

When your application window is no longer available as a potential target for an OLE drag and drop operation, you must call **RevokeDragDrop**.

This function calls the **IUnknown::Release** method for your drop target interface.

See Also

RegisterDragDrop

SCODE_CODE

```
#define SCODE_CODE(sc) ((sc) & 0xFFFF)
```

Extracts the code part of the **SCODE**.

SCODE_FACILITY

```
#define SCODE_FACILITY(sc) (((sc) >> 16) & 0x1fff)
```

Extracts the facility from the **SCODE**.

SCODE_SEVERITY

```
#define SCODE_SEVERITY(sc) (((sc) >> 31) & 0x1)
```

Extracts the severity field from the **SCODE**.

SetConvertStg QuickInfo

Indicates whether the object is to be automatically converted to a different CLSID. It does this by setting the conversion bit in a storage object. The setting can be retrieved with a call to the **GetConvertStg** function.

```
HRESULT SetConvertStg(
  IStorage * pStg,         //Storage object where the conversion bit is to be set
  BOOL fConvert            //Indicates whether an object is to be converted
 );
```

Parameters

pStg
Specifies the storage object in which to set the conversion bit.

fConvert
If TRUE, sets the conversion bit for the object. If FALSE, clears the conversion bit.

Return Values

S_OK
Indicates the object's conversion bit was set successfully.

STG_E_ACCESSDENIED
Indicates access to the storage object is not available.

E_OUTOFMEMORY
Indicates the conversion bit was not set due to a lack of memory.

E_INVALIDARG
Indicates one or more arguments are invalid.

E_UNEXPECTED
Indicates an unexpected error occurred.

See the **IStorage::CreateStream**, **IStorage::OpenStream**, **IStream::Read**, and **IStream::Write** methods for possible storage and stream access errors.

Comments

As part of converting an object from one class to another, container applications call **SetConvertStg** to set the conversion bit in the storage object. The bit is set to TRUE, indicating that the object has been tagged for conversion to a new class the next time it is loaded.

To retrieve the value of the conversion bit, an object application calls the **GetConvertStg** function when it is loading the object. If the bit is set, the object application converts the object to the new CLSID. To reset an object's conversion bit, the object application calls the **SetConvertStg** function with the *fConvert* parameter set to FALSE.

See Also

GetConvertStg

StgCreateDocfile **QuickInfo**

Creates a new storage object using the OLE-provided compound file implementation for the **IStorage** interface.

```
HRESULT StgCreateDocfile(
    const WCHAR * pwcsName,      //Points to pathname of compound file to create
    DWORD grfMode,               //Specifies the access mode for opening the storage object
    DWORD reserved,              //Reserved; must be zero
    IStorage ** ppstgOpen        //Points to location for returning the new storage object
  );
```

Parameters

pwcsName
Points to the pathname of the compound file to create. Passed uninterpreted to the file system. This can be a relative name or NULL. If NULL, a temporary compound file is allocated with a unique name.

grfMode
Specifies the access mode to use when opening the new storage object. For more information, see the **STGM** enumeration.

reserved
Reserved for future use; must be zero.

ppstgOpen
Points to the location where the new storage object is placed.

Return Values

S_OK
Indicates the compound file was successfully created.

STG_E_ACCESSDENIED
Indicates the calling process does not have sufficient access. Attempt to open file with conflicting permissions to a simultaneous open.

STG_E_FILEALREADYEXISTS
Indicates the compound file already exists and *grfMode* is set to STGM_FAILIFTHERE.

STG_S_CONVERTED
Indicates the specified file was successfully converted to Storage format.

STG_E_INSUFFICIENTMEMORY
Indicates the compound file was not created due to a lack of memory.

STG_E_INVALIDNAME
Indicates bad name in the *pwcsName* parameter.

STG_E_INVALIDPOINTER
Indicates bad pointer in the *pwcsName* parameter or the *ppStgOpen* parameter.

STG_E_INVALIDFLAG
Indicates bad flag combination in the *grfMode* pointer.

STG_E_TOOMANYOPENFILES
Indicates the compound file was not created due to a lack of file handles.

See also any file system errors for other error return values.

Comments

This function creates a new storage object using the OLE-provided, compound-file implementation for the **IStorage** interface. The name of the open compound file can be retrieved by calling the **IStorage::Stat** method.

The **StgCreateDocfile** function creates the file if it does not exist. If it does exist, the use of the STGM_CREATE, STGM_CONVERT, and STGM_FAILIFTHERE flags in the *grfMode* parameter indicate how to proceed. See the **STGM** enumeration for more information on these values.

If the compound file is opened in transacted mode (the *grfMode* parameter specifies STGM_TRANSACTED) and a file with this name already exists, the existing file is not altered until all outstanding changes are committed. If the calling process lacks write access to the existing file (because of access control in the file system), the *grfMode* parameter can only specify STGM_READ and *not* STGM_WRITE or STGM_READWRITE. The resulting new open compound file can still be written to, but a commit operation will fail (in transacted mode, write permissions are enforced at commit time).

If the *grfMode* parameter specifies STGM_TRANSACTED and no file yet exists with the name specified by the *pwcsName* parameter, the file is created immediately. In an access-controlled file system, the caller must have write permissions in the file system directory in which the compound file is created.

The absence of STGM_TRANSACTED in the *grfMode* parameter indicates the file is to be created and opened in direct access mode. Any existing file with the same name is destroyed before creating the new file.

The **StgCreateDocfile** function can be used to create a temporary compound file by passing a NULL value for the *pwcsName* parameter. However, these files are temporary only in the sense that they have a system-provided unique name - likely one that is meaningless to the user. The caller is responsible for deleting the temporary file when finished with it, unless STGM_DELETEONRELEASE was specified for the *grfMode* parameter.

Note By default, a Windows application can have 20 file handles open at one time. Each storage object opened in transacted mode uses three file handles. To increase the number of file handles the application has available, applications often call the Windows **SetHandleCount** function. However, the **SetHandleCount** function affects C run-time libraries (those statically linked to the application) that are hard-coded to expect only 20 file handles. There is no check on overflow, so if the application already has a few compound files open and it then uses a C run-time library (for example, **fopen** or **sopen**), the application corrupts memory past the end of the file-handle array (low in its data segment) if the library gets a file handle above 20.

To solve this problem, applications should reserve some number of file handles on start up, making them available to the components that might be based on C run-time libraries. If your application does not use any dynamic link libraries that use the C run-time libraries, there is no need for any special file-handle reservation code.

See Also

StgCreateDocFileOnILockBytes

StgCreateDocfileOnILockBytes QuickInfo

Creates and opens a new storage object on top of an **ILockBytes** instance provided by the caller. The storage object uses the OLE-provided, compound-file implementation for the **IStorage** interface.

```
HRESULT StgCreateDocfileOnILockBytes(
   ILockBytes * plkbyt,          //Specifies the byte array object
   DWORD grfMode,                //Specifies the access mode
   DWORD reserved,               //Reserved; must be zero
   IStorage ** ppstgOpen         //Points to location for returning the new storage object
  );
```

Parameters

plkbyt
Points to the underlying byte array object on which to create a compound file.

grfMode
Specifies the access mode to use when opening the new compound file. For more information, see the **STGM** enumeration.

reserved
Reserved for future use; must be zero.

ppstgOpen
Points to the location where the new storage object is placed.

Return Values

S_OK
Indicates the compound file was successfully created.

STG_E_ACCESSDENIED
Indicates the calling process does not have sufficient access. Attempt was made to open an **ILockBytes** object with permissions that conflict with another current opening of the same object.

STG_E_FILEALREADYEXISTS
Indicates the compound file already exists and the *grfMode* parameter is set to STGM_FAILIFTHERE.

STG_S_CONVERTED
Indicates the compound file was successfully converted. The original byte array object was successfully converted to **IStorage** format.

STG_E_INSUFFICIENTMEMORY
Indicates the storage object was not created due to a lack of memory.

STG_E_INVALIDPOINTER
Indicates a bad pointer was in the *pLkbyt* parameter or the *ppStgOpen* parameter.

STG_E_INVALIDFLAG
Indicates a bad flag combination was in the *grfMode* parameter.

STG_E_TOOMANYOPENFILES
Indicates the storage object was not created due to a lack of file handles.

See also any file system errors for other error return values.

See also the **ILockBytes** interface for other error return values.

Comments

This function creates a storage object on top of a byte array object using the OLE-provided, compound-file implementation for the **IStorage** interface. The **StgCreateDocfileOnILockBytes** function can be used to store a document in a relational database. The byte array indicated by the *pLkbyt* parameter is used for the underlying storage in place of a disk file.

The **StgCreateDocfileOnILockBytes** function has almost exactly the same semantics as the **StgCreateDocfile** function. For more information, see the discussion of **StgCreateDocfile**.

The newly created compound file is opened according to the access modes in the *grfMode* parameter. For conversion purposes, the file is always considered to already exist. As a result, it is not useful to use the STGM_FAILIFTHERE value, because it always causes an error to be returned. However, STGM_CREATE and STGM_CONVERT are both still useful.

The ability to build a compound file on top of a byte array object is provided to support having the data (underneath an **IStorage** and **IStream** tree structure) live in persistent space, space that does not ultimately reside in the file system. Given this capability, there is nothing preventing a document that *is* stored in a file from using this facility. For example, a container might do this to minimize the impact on its file format caused by adopting OLE. However, it is recommended that OLE documents adopt the **IStorage** interface for their own outer-level storage. This has the following advantages:

- The storage structure of the document is the same as its storage structure when it is an embedded object, reducing the number of cases the application needs to handle.
- One can write tools to access the OLE embeddings and links within the document without special knowledge of the document's file format. An example of such a tool is a copy utility that copies all the documents included in a container containing linked objects. A copy utility like this needs access to the contained links to determine the extent of files to be copied.
- The **IStorage** instance addresses the problem of how to commit the changes to the file. An application using the **ILockBytes** byte array must handle these issues itself.
- Future file systems will likely implement the **IStorage** and **IStream** interfaces as their native abstractions, rather than layer on top of a byte array as is done in compound files. Such a file system could be built so documents using the **IStorage** interface as their outer level containment structure would get an automatic efficiency gain by having the layering flattened when files are saved on the new file system.

See Also

StgCreateDocfile

StgIsStorageFile [QuickInfo]

Indicates whether a particular disk file contains a storage object.

HRESULT StgIsStorageFile(
  const WCHAR * *pwcsName*   //Points to a pathname of the file to check
 );

Parameter

pwcsName
  Points to the name of the disk file to be examined. The *pwcsName* parameter is passed uninterpreted to the underlying file system.

Return Values

S_OK
  Indicates the file contains a storage object.

S_FALSE
  Indicates the file does not contain a storage object.

STG_E_INVALIDFILENAME
  Indicates a bad filename was passed in the *pwcsName* parameter.

STG_E_FILENOTFOUND
  Indicates the *pwcsName* parameter could not be determined.

See also any file system errors for other error return values.

Comments

At the beginning of the disk file underlying a storage object is a signature distinguishing a storage object from other file formats. The **StgIsStorageFile** function is useful to applications whose documents use a disk file format that might or might not use storage objects.

See Also

StgIsStorageILockBytes

StgIsStorageILockBytes QuickInfo

Indicates whether the specified byte array contains a storage object.

```
HRESULT StgIsStorageILockBytes(
   ILockBytes * plkbyt          //Points to the byte array to be examined
  );
```

Parameter

plkbyt
Points to the byte array to be examined.

Return Values

S_OK
Indicates the specified byte array contains a storage object.

S_FALSE
Indicates the specified byte array does not contain a storage object.

File system errors.

ILockBytes interface error return values.

Comments

At the beginning of the **ILockBytes** byte array underlying a storage object is a signature distinguishing an **IStorage** object from other file formats. The **StgIsStorageILockBytes** function is useful to applications whose documents use an **ILockBytes** byte array format that might or might not use storage objects.

See Also

StgIsStorageFile

StgOpenStorage QuickInfo

Opens an existing root storage object in the file system. You can use this function to open directories, files, compound files, and summary catalogs. Nested storage objects can only be opened using their parent's **IStorage::OpenStorage** method.

```
HRESULT StgOpenStorage(
   const WCHAR * pwcsName,        //Points to the pathname of the file containing storage object
   IStorage * pstgPriority,       //Points to a previous opening of a root storage object
   DWORD grfMode,                 //Specifies the access mode for the object
   SNB snbExclude,                //Points to an SNB structure specifying elements to be excluded
   DWORD reserved,                //Reserved; must be zero
   IStorage ** ppstgOpen          //Points to location for returning the storage object
  );
```

Parameters

pwcsName

Points to the pathname of the storage object to open. This parameter is ignored if the *pStgPriority* parameter is not NULL.

pstgPriority

Most often NULL. If not NULL, this parameter is used instead of the *pwcsName* parameter to specify the storage object to open. It points to a previous opening of a root storage object, most often one that was opened in priority mode.

After the **StgOpenStorage** function returns, the storage object specified in the *pStgPriority* parameter on function entry is invalid, and can no longer be used. Use the one specified in the *ppStgOpen* parameter instead.

grfMode

Specifies the access mode to use to open the storage object.

snbExclude

If not NULL, this parameter points to a block of elements in this storage that are to be excluded as the storage object is opened. This exclusion occurs independent of whether a snapshot copy happens on the open. May be NULL.

reserved

Indicates reserved for future use; must be zero.

ppstgOpen

Points to the location where the opened storage is placed.

Return Values

S_OK

Indicates the storage object was successfully opened.

STG_E_FILENOTFOUND

Indicates the specified file does not exist.

STG_E_ACCESSDENIED

Indicates insufficient access to open file. Exclusions specified without read-write permissions or attempt to open file with conflicting permissions to a simultaneous open.

STG_E_FILEALREADYEXISTS

Indicates the file exists but is not a storage object.

STG_E_TOOMANYOPENFILES

Indicates the storage object was not opened because there are too many open files.

STG_E_INSUFFICIENTMEMORY

Indicates the storage object was not opened due to a lack of memory.

STG_E_INVALIDNAME

Indicates bad name in the *pwcsName* parameter or the *snbExclude* parameter.

STG_E_INVALIDPOINTER
: Indicates bad pointer in one of the parameters: *snbExclude*, *pwcsName*, *pstgPriority*, or *ppStgOpen*.

STG_E_INVALIDFLAG
: Indicates bad flag combination in the *grfMode* parameter.

STG_E_INVALIDFUNCTION
: Indicates STGM_DELETEONRELEASE specified in the *grfMode* parameter.

STG_E_OLDDLL
: Indicates the DLL being used to open this storage object is a version prior to the one used to create it.

STG_E_OLDFORMAT
: Indicates the storage object being opened was created by the Beta 1 storage provider. This format is no longer supported.

File system error return values.

Comments

This function opens the specified root storage object according to the access mode in the *grfMode* parameter. A pointer to the opened storage object is returned through the *ppstgOpen* parameter.

Note Opening a storage object in read and/or write mode without denying writer permission to others (the *grfMode* parameter specifies STGM_SHARE_DENY_WRITE) can be be a time consuming operation since the **StgOpenStorage** call must make a snapshot of the entire storage object.

Applications will often try to open storage objects with the following access permissions:

```
STGM_READ_WRITE | STGM_SHARE_DENY_WRITE
     // transacted vs. direct mode omitted for exposition
```

If the application succeeds, it will never need to do a snapshot copy. If it fails, the application can revert to using the permissions and make a snapshot copy:

```
STGM_READ_WRITE
     // transacted vs. direct mode omitted for exposition
```

In this case, the application should prompt the user before doing a time consuming copy. Alternatively, if the document sharing semantics implied by the access modes are appropriate, the application could try to open the storage as follows:

```
STGM_READ | STGM_SHARE_DENY_WRITE
     // transacted vs. direct mode omitted for exposition
```

In this case, if the application succeeds, a snapshot copy will not have been made (because STGM_SHARE_DENY_WRITE was specified, denying others write access).

To reduce the expense of making a snapshot copy, applications can open storage objects in priority mode (*grfMode* specifies STGM_PRIORITY).

The *snbExclude* parameter specifies a set of element names in this storage object that are to be emptied as the storage object is opened: streams are set to a length of zero; storage objects have all their elements removed. By excluding certain streams, the expense of making a snapshot copy can be significantly reduced. Almost always, this approach will be used after first opening the storage object in priority mode, then completely reading the now-excluded elements into memory. This earlier priority mode opening of the storage object should be passed through the *pstgPriority* parameter to remove the exclusion implied by priority mode. The calling application is responsible for rewriting the contents of excluded items before committing. Thus, this technique is most likely only useful to applications whose documents do not require constant access to their storage objects while they are active.

StgOpenStorageOnILockBytes QuickInfo

Opens an existing storage object that does not reside in a disk file, but instead has an underlying byte array provided by the caller.

```
HRESULT StgOpenStorageOnILockBytes(
    ILockBytes * plkbyt,           //Specifies the underlying byte array
    IStorage * pStgPriority,       //Points to a previous opening of a root storage object
    DWORD grfMode,                 //Specifies the access mode for the object
    SNB snbExclude,                //Points to an SNB structure specifying elements to be excluded
    DWORD reserved,                //Reserved, must be zero
    IStorage ** ppstgOpen          //Points to location for returning the storage object
  );
```

Parameters

plkbyt
Points to the underlying byte array that contains the storage object to be opened.

pStgPriority
Most often NULL. If not NULL, this parameter is used instead of the *plkbyt* parameter to specify the storage object to open. It points to a previous opening of a root storage object, most often one that was opened in priority mode.

After the **StgOpenStorageOnILockBytes** function returns, the storage object specified in the *pStgPriority* parameter on function entry is invalid, and can no longer be used; use the one specified in the *ppStgOpen* parameter instead.

grfMode
Specifies the access mode to use to open the storage object.

snbExclude
May be NULL. If not NULL, this parameter points to a block of elements in this storage that are to be excluded as the storage object is opened. This exclusion occurs independent of whether a snapshot copy happens on the open. .

reserved
Indicates reserved for future use; must be zero.

ppstgOpen
Points to the location where the opened storage is placed.

Return Values

S_OK
Indicates the storage object was successfully opened.

STG_E_FILENOTFOUND
Indicates the specified byte array does not exist.

STG_E_ACCESSDENIED
Indicates insufficient access to open the byte array. Exclusions specified without read-write permissions or attempt to open the byte array with conflicting permissions to a simultaneous open.

STG_E_FILEALREADYEXISTS
Indicates the byte array exists but is not a storage object.

STG_E_TOOMANYOPENFILES
Indicates the storage object was not opened because there are too many open files.

STG_E_INSUFFICIENTMEMORY
Indicates the storage object was not opened due to a lack of memory.

STG_E_INVALIDNAME
Indicates bad name in the *pwcsName* parameter or the *snbExclude* parameter.

STG_E_INVALIDPOINTER
: Indicates bad pointer in one of the parameters: *snbExclude*, *pwcsName*, *pstgPriority*, or *ppStgOpen*.

STG_E_INVALIDFLAG
: Indicates bad flag combination in the *grfMode* parameter.

STG_E_INVALIDFUNCTION
: Indicates STGM_DELETEONRELEASE specified in the *grfMode* parameter.

STG_E_OLDDLL
: Indicates the DLL being used to open this storage object is a version prior to the one used to create it.

STG_E_OLDFORMAT
: Indicates the storage object being opened was created by the Beta 1 storage provider. This format is no longer supported.

File system error return values.

ILockBytes interface error return values.

Comments

This function opens the specified root storage object. The storage object is opened according to the access mode in the *grfMode* parameter; a pointer to the opened storage object is returned through the *ppstgOpen* parameter.

The storage object must have been previously created by the **StgCreateDocfileOnILockBytes** function.

The semantics of the **StgOpenStorageOnILockBytes** function are almost exactly the same as those of the **StgOpenStorage** function. For more information, see the discussion of the **StgOpenStorage** function.

See Also

StgOpenStorage

StgSetTimes QuickInfo

Sets the creation, access, and modification times of the indicated file, if supported by the underlying file system.

```
HRESULT StgSetTimes(
    WCHAR const * lpszName,         //Points to the name of the file to be changed
    FILETIME const * pctime,        //Points to the new value for the creation time
    FILETIME const * patime,        //Points to the new value for the access time
    FILETIME const * pmtime         //Points to the new value for the modification time
  );
```

Parameters

lpszName
Points to the name of the file to be changed.

pctime
Points to the new value for the creation time.

patime
Points to the new value for the access time.

pmtime
Points to the new value for the modification time.

Return Values

S_OK
Indicates time values successfully set.

STG_E_FILENOTFOUND
Indicates element does not exist.

STG_E_INVALIDNAME
Indicates bad name passed in the *lpszName* parameter, or a file system error.

STG_E_ACCESSDENIED
Indicates insufficient permissions to access storage.

File system error return values.

Comments

This function sets the time values for the specified file. Each of the time value parameters can be NULL, indicating no modification should occur.

It is possible that one or more of these time values are not supported by the underlying file system. This function sets the times that can be set and ignores the rest.

StringFromCLSID QuickInfo

Converts the CLSID into a string of printable characters so different CLSIDs always convert to different strings.

```
HRESULT StringFromCLSID(
   REFCLSID rclsid,          //The CLSID to be converted
   LPOLESTR * lplpsz         //Receives the resulting string on return
  );
```

Parameters

rclsid
Specifies the CLSID to be converted.

lplpsz
Receives the resulting string on return.

Return Values

S_OK
Indicates the CLSID was successfully converted and returned.

E_OUTOFMEMORY
Out of memory.

Comments

The **StringFromCLSID** function calls the **StringFromGuid2** function to convert a globally unique identifier (GUID) into a string of printable characters.

See Also

CLSIDFromString, **StringFromGuid2**

StringFromGUID2 QuickInfo

Converts a globally unique identifier (GUID) into a string of printable characters.

```
int StringFromGUID2(
   REFGUID rguid,           //The interface ID to be converted
   LPOLESTR lpsz,           //Receives the resulting string on return
   int cbMax                //The maximum size the returned string is expected to be
  );
```

Parameters

rguid
Specifies the interface ID to be converted.

lpsz
Receives the resulting string on return.

cbMax
Specifies the maximum size the returned string is expected to be.

Return Values

0 (zero)
Indicates buffer is too small for returned string.

Non-zero value
Specifies the number of characters in the returned string, including the null terminator.

Comments

The string that the *lpsz* parameter receives has a format like the following sample:

```
[c200e360-38c5-11ce-ae62-08002b2b79ef]
```

where the successive fields break the GUID into the form DWORD-WORD-WORD-WORD-WORD.DWORD covering the 128-bit GUID. The string includes enclosing braces, which are an OLE convention.

See Also

StringFromCLSID

StringFromIID QuickInfo

Converts an interface ID into a string of printable characters.

```
HRESULT StringFromIID(
   REFIID rclsid               //The interface ID to be converted
   LPOLESTR * lplpsz           //Receives a pointer to the resulting string on return
  );
```

Parameters

rclsid
Specifies the interface ID to be converted.

lplpsz
Receives a pointer to the resulting string on return.

Return Values

S_OK
Indicates the character string was successfully returned.

E_OUTOFMEMORY
Out of memory.

Comments

The string returned by the function is freed in the standard way, using the task allocator (refer to the **CoGetMalloc**function).

See Also

IIDFromString, **CoGetMalloc**

SUCCEDDED

```
#define SUCCEEDED(Status) ((HRESULT)(Status) >= 0)
```

Provides a generic test for success on any status value. Non-negative numbers indicate success.

WriteClassStg QuickInfo

Stores the specified CLSID in a storage object.

```
HRESULT WriteClassStg(
   IStorage * pStg,        //Points to the storage object
   REFCLSID rclsid         //Specifies the CLSID to be stored in the storage object
  );
```

Parameters

pStg
Points to the storage object that will get a new CLSID.

rclsid
Points to the CLSID to be stored with the object.

Return Values

S_OK
Indicates the CLSID was successfully written to the file.

STG_E_MEDIUMFULL
Indicates the CLSID could not be written due to lack of memory.

IStorage::SetClass method error return values.

Comments

This function writes a CLSID to the specified storage object so it can be read by the **ReadClassStg** function. Container applications typically call this function before calling the **IPersistStorage::Save** method.

See Also

OleSave, **ReadClassStg**

WriteClassStm QuickInfo

Stores the specified CLSID in the stream.

```
HRESULT WriteClassStm(
  IStream * pStm,          //Points to the stream object
  REFCLSID rclsid          //Specifies the CLSID to be stored in the stream object
 );
```

Parameters

pStm
Points to the stream into which the CLSID is to be written.

rclsid
Specifies the CLSID to write to the stream.

Return Values

S_OK
Indicates the CLSID was successfully written.

STG_E_MEDIUMFULL
The CLSID could not be written because there is no space left on device.

IStorage::SetClass method error return values.

Comments

This function writes a CLSID to the specified stream object so it can be read by the **ReadClassStm** function. Most applications do not call the **WriteClassStm** function. OLE calls it before making a call to an object's **IPersistStream::Save** method.

See Also

ReadClassStm, **WriteClassStg**, **ReadClassStg**

WriteFmtUserTypeStg QuickInfo

Writes a clipboard format and user type to the storage object.

```
HRESULT WriteFmtUserTypeStg(
    IStorage * pStg,                    //Points to the storage object
    CLIPFORMAT cf,                     //Specifies the clipboard format
    LPWSTR * lpszUserType       //Points to the current user type
  );
```

Parameters

pStg

Points to the storage object where the information is to be written.

cf

Specifies the clipboard format that describes the structure of the native area of the storage object. The format tag includes the policy for the names of streams and substorages within this storage object and the rules for interpreting data within those streams.

lpszUserType

Points to the object's current user type. It cannot be NULL. This is the type returned by the **IOleObject::GetUserType** method. If this function is transported to a remote machine where the object class does not exist, this persistently stored user type can be shown to the user in dialog boxes.

Return Values

S_OK

Indicates the information was written successfully.

STG_E_MEDIUMFULL

Indicates information could not be written due to lack of space on the storage medium.

IStream::Write method error return values.

Comments

The **WriteFmtUserTypeStg** function must be called in an object's implementation of the **IPersistStorage::Save** method. It must also be called by document-level objects that use structured storage for their persistent representation in their save sequence.

To read the information saved, applications call the **ReadFmtUserTypeStg** function.

See Also

IPersistStorage::Save, **ReadFmtUserTypeStg**

Glossary

A

Activation

The process of binding an object in order to put it into its running state. Also refers to invoking a particular operation on an object. *See also* Binding.

Absolute moniker

A moniker that specifies the absolute location of an object. An absolute moniker is analogous to a full pathname. *See also* Moniker*.*

Advisory sink

An object that can receive notifications of changes in an embedded object or linked object because it implements the IAdviseSink interface (and possibly the IAdviseSink2 interface). Notifications originate in the object application, where they are cached in an advise holder, which passes them to advisory sinks implemented by object handlers. For an embedded object, the corresponding object handler forwards the notification to the object's container. For a linked object, the object handler forwards the notification to the link object, which in turn passes it on to the container. *See also* Object handler, Embedded object, Link object, Linked object, Container application.

Aggregate object

A component object that is made up of one or more other component objects. One object in the aggregate is designated the control object, which controls which interfaces in the aggregate are exposed and which are private. The control object has a special implementation of **IUnknown** called the controlling unknown. All non-controlling objects in the aggregate must pass calls to **IUnknown** methods through the controlling unknown.

Aggregation

A composition technique for implementing component objects. It allows you to build a new object with one or more existing objects that support some or all of the new object's required interfaces.

Anti-moniker

The inverse of a file, item, or pointer moniker. An anti-moniker is added to the end of a file-, item-, or pointer moniker to nullify it. This is analogous to the way that ".." nullifies a directory component of a pathname (that is, **\work\art\..** equals **\work**). Anti-monikers are used in the construction of relative monikers. *See also* Relative Moniker.

Artificial reference counting

A technique used to safeguard an object before making a call that could result in its premature destruction by ensuring it won't be de-allocated. A process calls the **AddRef** method of **IUnknown** to increment the count before making the call that could otherwise free the object prematurely, and after the function returns, calls the **IUnknown::Release** method to decrement the count.

Asynchronous call

A function that allows the next instruction in the process to be executed as soon as the call is made, without waiting for the function to return. Most OLE functions and interface methods are synchronous; that is, the function must return before the next instruction is executed. OLE defines six asynchronous methods, five within the **IAdviseSink** interface – **OnDataChange**, **OnViewChange**, **OnRename**, **OnSave**, and **OnClose** – and one within the **IAdviseSink2** interface – **OnLinkSourceChange**.

Automation
See OLE Automation

B

Bind context
An object that implements the **IBindCtx** interface. Bind contexts are used in moniker operations. A bind context holds references to the objects activated when a moniker is bound, contains parameters that apply to all operations during the binding of a generic composite moniker, and provides the means by which the moniker implementation should retrieve information about its environment. *See* Binding, Moniker.

Binding
Generically, binding is the process of associating a name to its referent. When referring to monikers specifically, binding is the process of locating the object named by the moniker, activating it (loading it in memory) if it isn't already, and returning an interface pointer to it. Binding puts an object into its running state, allowing the operations it supports to be invoked. Objects can be bound at run time (also called late binding or dynamic binding) or at compile time (also called static binding). In OLE Automation, vtable-based binding is referred to as "early binding" and IDispatch-based binding is referred to as "late binding."

C

Cache
Generally, a cache is a (usually temporary) local store of information. In OLE, the term cache is used in connection with linked or embedded objects, in which the cache contains information that defines the presentation of a contained object when the container is opened.

Cache initialization
The process of filling an embedded object's cache with data formats. The **IOleCache** interface provides methods that a container object can call to control the data that gets cached for embedded objects.. The **IOleCache::InitCache** method fills the cache using data provided by a transfer from the Clipboard or from a drag-and-drop operation. The **IOleCache::SetData** method fills the cache using data in a storage object.

Class
In object-oriented programming, a class is a set of objects whose behavior is defined by the same set of code (they share the same implementation).
A COM class is not necessarily the same as a class in an object-oriented language. A COM class can be identified by a CLSID, though for most COM operations, it is not necessary to know an object's class.

Class factory
A object that you use to create one or more instances of a an object identified by a given CLSID (class identifier). A class factory object implements the **IClassFactory** interface. A class factory is the most frequently used type of class object in OLE. *See also CLSID.*

Class identifier (CLSID)
A unique identification tag (UUID) associated with an OLE class object. A class object that is intended to create more than one object registers its CLSID in a task table in the registration database to enable clients to locate and load the executable code associated with the object(s). Every OLE object application (or container that allows linking to its embedded objects) must register a CLSID for each supported object definition.

Class object
In COM, class objects are called class factories, and typically have no behavior except to create new instances of the class. In object-oriented programming, an object whose state is shared by all the objects in a class, and whose behavior acts on that class-wide state data. *See also* Class factory.

Client
In referring to OLE objects, an object that requests services from another object. *See also* Container.

Client site
The display site for an embedded or linked object within a compound document. The client site is the principal means by which an object requests services from its container.

CLSID
See Class identifier

Component object
An object that conforms to the OLE component object model (COM). Clients deal with a component object only through a pointer to an interface (a related set of functions called methods). With the pointer, clients can call the methods that perform operations or manipulate the data associated with the object. Component objects are instances of an object definition, which specifies an implementation of the interfaces on the object.

Composite moniker
A moniker that consists of two or more monikers composed together. A composite moniker can be non-generic, meaning that its component monikers have special knowledge of each other, or generic, meaning that its component monikers know nothing about each other except that they are monikers. *See* Generic composite moniker.

Commit
The act of persistently saving any changes made to an object since its storage was opened or since the last time changes were saved. *See also Revert*.

Component object model (COM)
The OLE object-oriented programming model that defines how objects interact within a single application or between applications. In COM, clients have access to an OLE object through a pointer to an interface (a related set of functions called methods) on the object.

Composite menu
A shared menu bar composed of menu groups from both an in-place container and an in-place object application. The object application must install and remove the menu from the container's frame window.

Compound document
A document that contains data of different formats, created by different applications. Compound documents are created in a container application, such as Word, and information from other applications (such as spreadsheets, sound clips, and bitmaps) is either embedded in or linked to the container application. When the compound document is saved, the container application saves it.

Compound file

An OLE-provided implementation of structured storage which includes the **IStorage**, the **IStream**, and the **ILockBytes** interfaces. You use a set of **Stg*Xxx*** API functions to create and use a compound file.

Container

See Container application.

Container application

An application that supports compound documents. Container applications provide storage for the object, a site for display, access to the display site, and an advisory sink for receiving notifications of changes in the object. *See also* Compound document, Client site, and Advisory sink.

Container/Object

An application that has implemented OLE interfaces such that the application supports the features and capabilities of both a container and object application.

Container/Server

See Container/Object.

Control object

The object within an Aggregate object that controls which interfaces within the aggregate object are exposed and which are private.. The control object has a special implementation of **IUnknown** called the controlling unknown, and all other objects in the aggregate must pass calls to **IUnknown** methods through the controlling unknown. *See* Aggregate object.

D

Data transfer object

An object that supports the **IDataObject** interface and contains data to be transferred from one object to another either through the Clipboard or drag-and-drop operations.

Default object handler

A DLL provided with the OLE SDK that provides a partial implementation of basic interfaces. It is a surrogate in the processing space of the container application for the real object.

With the default object handler, it is possible to look at an object's stored data without actually activating the object. The default object handler performs other tasks on behalf of a loaded object, such as rendering an object from its cached state when the object is loaded into memory.

Direct access mode

One of two access modes in which a storage object can be opened. In direct mode, all changes are immediately committed to the root storage object. *See also* Transacted access mode.

Drag and drop

OLE specifies a set of data transfer and drag-and drop-interface-specific interfaces that allow applications to implement drag and drop, an operation in which the end-user uses the mouse or other pointing device to drag data from one window and drop it into another location in the same window or into another window.

E

Embedded object

A compound-document object that is stored with the container application, but when it is running, lives in the process space of the server application, which creates and subsequently edits it. The default handler provides a surrogate in the processing space of the container application for the real object.

Explicit caching

One of two ways an object can cache its presentation data. Explicit caching requires the physical creation of the cache nodes needed to save the data formats of the object. *See also* Implicit caching.

F

File moniker

A moniker based on a path in the file system. File monikers can be used to identify objects that are saved in their own files. A file moniker is an object that represents an implementation of the **IMoniker** interface for the file class. *See also* Item moniker and Generic composite moniker.

G

Global memory

See Shared application memory.

Generic composite moniker

The generic composite moniker is a sequenced collection of other types of monikers, starting with a file moniker to provide the document-level path and continuing with one or more item monikers. *See also* Item moniker and File moniker.

H

Handler

See Object handler.

Helper function

A function that encapsulates other functions and interface methods publicly available in the OLE SDK.

HRESULT

An opaque result handle defined to be zero for a successful return from a function and non-zero if error or status information is to be returned. To convert an HRESULT into the more detailed SCODE, applications call **GetScode**(). *See* SCODE.

I

Implicit caching

The "implied" caching of presentation data by an object that is capable of rendering itself using its native data. Cache nodes are not created with implicit caching. *See also* Explicit caching.

In parameter

A parameter that is allocated, set, and freed by the caller of a function.

In/Out parameter
A parameter that is initially allocated by the caller of a function and set, freed, and reallocated if necessary by that which is called.

In-process server
An object server or application implemented as a DLL that runs in the process space of the object's container. *See also* Local server, Remote server.

Instance
An object for which memory is allocated or which is persistent.

Instantiate
The process of creating an activating an obnect based on its definition.

Interface
A group of related functions that provide access to OLE objects. The set of OLE interfaces define a contract that allow objects to interact according to the Component Object Model (COM). While OLE provides many interface implementations, most interfaces can also be implemented by developers designing OLE applications.

Interface identifier (IID)
A unique identification tag associated with each interface. Some functions take IIDs as parameters to allow the callers to specify what type of interface pointer should be returned.

Item moniker
A moniker based on a string that identifies an object in a container. Item monikers can be used to identify objects smaller than a file, including embedded objects in a compound document, or a pseudo-object (like a range of cells in a spreadsheet). *See also* File moniker and Generic composite moniker.

L

Link object
A component object that is instantiated when a linked compound document object is created or loaded. The link object implements the IOleLink interface and is provided by OLE.

Linked object
A compound-document object whose source data physically resides where it was initially created. Only a moniker that represents the source data and the appropriate presentation data is kept with the compound document. Changes made to the link source are automatically reflected in the linked compound-document object in the container(s).

Link source
The data that is the source of a linked compound document object. A link source may be a file or a portion of a file, such as an embedded object or a selected range within a file (also called a pseudo object).

Loaded state
The state of a compound-document object after its data structures have been loaded into

container memory. The data structures are created by the object handler. *See also* Passive state and Running state.

Local server

An server application implemented as an EXE running on the same machine as the client application using it. Because the server application is an EXE, it runs in its own process. *See also* In-Process server and Remote server.

Lock

OLE defines two types of locks that can be held on an object: *strong* and *weak*. A strong lock will keep an object in memory, a weak lock will not.

LRPC (Lightweight remote procedure call)

OLE's RPC-based protocol for interprocess communication. LRPC is "lightweight" in that it handles communication between processes on one machine only.

M

Marshaling

The process of packaging and sending interface parameters across process boundaries.

Moniker

An object that implements the **IMoniker** interface. A moniker acts as a name that uniquely identifies a COM object. You can think of a moniker as a generalization of a pathname; in the same way that a pathname identifies a file in the file system, a moniker identifies a COM object. Monikers support an operation known as "binding," which is the process of locating the object named by the moniker, activating it (loading it in memory) if it isn't already, and returning an interface pointer to it.

Moniker class

An implementation of the IMoniker interface. System-supplied moniker classes include file monikers, item monikers, generic composite monikers, anti-monikers, and pointer monikers.

Moniker client

An application that uses monikers to acquire interface pointers to objects managed by another application.

Moniker provider

An application that hands out monikers that identify the objects it manages, so that they are accessible to other applications.

Multiple Document Interface (MDI) Application

An application that can support multiple documents from one application instance. MDI object applications can simultaneously serve a user and one or more embedding containers. *See also* Single Document Interface (SDI) application.

N

Native data

The data used by an OLE server application when editing an embedded object. *See also* Presentation data.

Nested object

An OLE object that is embedded within another OLE object through the use of a combination container/server application. OLE objects can be arbitrarily nested to any level.

O

Object

Generally, an instance of an entity that embodies both specific data and the functions that manipulate it.

Specifically in object-oriented programming, an object is an entity that has state, behavior and identity. An object's state consists of its attributes and and the attributes' current values. An object's behavior consists of the operations that can be performed on it and the accompanying state changes. An object's identity is what you use to distinguish it from other objects.

In contrast, COM objects' behavior is defined by the interfaces it supports. A COM object's state is not explicitly specified, but is implied by its interfaces. A COM object's identity is defined by the ability to use **IUnknown::QueryInterface** to move between interfaces.

A COM object follows a specific model in which clients (those using an object's services) gain access to the object's data only through a pointer to an interface consisting of a set of methods (related functions). The client can then call these methods to perform desired operations. There is no direct access to an OLE object's data.

Object application

An OLE-aware application that can create compound document objects. Containers can then embed or link to these objects.

Object handler

A piece of class-specific code that is dynamically loaded into the address space of its container. Object handlers process requests for specific classes of objects, so that much of the communication can be in-process. This is more efficient because it reduces the need for remote procedure calls.

Object state

One of three relationships between a compound document object in its container and the application responsible for the object's creation: *passive*, *loaded*, and *running*. Passive objects are stored (on disk or in a database), and the object is not selected or active. In the loaded state, its data structures have been loaded into memory, but is not yet available for operations such as editing. Running objects are both loaded and available for all operations.

Object type name

A unique identification string that is stored as part of the information available for an object in the registration database—for example, Acme Drawing.

OLE Automation

A way to manipulate an application's objects from outside the application. OLE automation is typically used to create applications that expose objects to programming tools and macro languages, create and manipulate one application's objects from another applications, or to create tools for accessing and manipulating objects. For information on OLE Automation, refer to the O*LE Programmer's Reference*, Volume 2.

Out parameter

A function parameter that is allocated and set by whatever a function calls, and must be freed by the caller.

P

Passive state

The state of an OLE object when it is stored (on disk or in a database). The object is not selected or active. *See also* Loaded state and Running state.

Persistent storage

Storage of a file or object in a medium such as a file system or database so that the file can be closed and then re-opened at a later time. Data can be retrieved from the file until the underlying file is deleted.

Pointer moniker

A moniker that wraps an interface pointer to an object in memory. Whereas most monikers identify objects that can be saved to persistent storage, pointer monikers identify objects that cannot. They allow such objects to participate in a moniker binding operation.

Presentation data

The data used by an OLE container applicationto display embedded or linked objects. *See also* Native data.

Primary verb

The action associated with the most common, preferred operation users perform on an object; the primary verb is always defined as verb zero in the system registration database. An object's primary verb is executed by double-clicking on the object.

Proxy

An interface-specific object that packages parameters for that interface in preparation for a remote method call. A proxy runs in the address space of the sender and communicates with a corresponding stub in the receiver's address space. *See also* Stub, Marshaling, and Unmarshaling.

Proxy manager

Manages all the proxies for a single object.

Pseudo object

A selection of data within a document or embedded object that can be the source for a compound-document object.

R

Reference counting

Keeping a count of each interface pointer instance to ensure that an object is not destroyed before all references to it are released.

Relative moniker

A moniker that specifies the location of an object relative to the location of another object. A relative moniker is analogous to a relative pathname (such as "..\backup\report.old").

Remote Server

A server application, implemented as an EXE, running on a different machine from the client application using it. *See also* In-process server, Local server.

Revert

The act of discarding any change(s) made to an object since the last time the changes were committed or the object's storage was opened. *See also* Commit, Transacted access mode.

Root IStorage object

The outermost **IStorage** instance in a document; also called the root storage object. Compound-document objects are always saved as children of a root **IStorage** object.

Running state

The state of an OLE object when the object application is running and it is possible to edit the object, access its interfaces, and receive notification of changes. *See also* Loaded state and Passive state.

Running object table (ROT)

A globally-accessible table on each machine that keeps track of all the OLE objects that can be identified by a moniker and are currently running on the machine. Monikers use the Running Object Table. Registering an object in the running object table increments the object's reference count. Before the object can be destroyed, its moniker must be released from the running object table.

S

SCODE

A DWORD value that is used to pass detailed information to the caller of an interface method or function. *See also* HRESULT.

Server

See Object application.

Shared application memory

Memory that is primarily used between processes to optimize the data copying that occurs in LPRC calls.

Single Document Interface (SDI) Application

An application that can support only one document at a time. Multiple instances of an SDI application must be started to service both an embedded object and a user. *See also* Multiple Document Interface (MDI) application.

Single object application

An application that is capable of creating and manipulating one class of object. *See also* Multiple object application.

Stand-alone object application

An object application implemented as an executable (EXE) program, rather than as a DLL object application.

State

See Loaded state, Passive state, and Running state.

Static object

An object that contains only a presentation, with no native data. A container application can treat a static object as though it were a linked or embedded object, except that it is not possible to edit a static object.

A static object can result, for example, from the breaking of a link on a linked object; you don't want the linked object to be updated anymore.

Storage object

An object that implements the **IStorage** interface. A storage object contains nested storage objects or stream objects, resulting in a directory/file system within a single file. It provides the underlying storage for compound documents. Container applications provide a nested storage object for each of their embedded objects. The embedded object stores its data nested storage and stream objects within the storage object provided by its container. *See also* Root IStorage object and Stream object.

Stream object

An object that implements the **IStream** interface. A stream object is analogous to a file in a directory/file system. *See also* Storage object.

Structured storage model

A specification that defines a hierarchical method of storing objects. OLE provides an implementation of the structured storage model called Compound Files. *See* Compound file.

Stub

When a function's or interface method's parameters are marshaled across a process boundary, the stub is an interface-specific object that unpackages the marshaled parameters and calls the required method. The stub runs in the receiver's address space and communicates with a corresponding proxy in the sender's address space. *See* Proxy, Marshaling, and Unmarshaling.

Stub manager

Manages all of the stubs for a single object.

Synchronous call

A function that does not allow further instructions in the process to be executed until the function returns. Most OLE interface methods, except some of the **IAdviseSink** methods, are synchronous calls. *See also* Asychronous call.

T

Transacted access mode

One of two access modes in which a storage object can be opened. When opened in transacted mode, changes are stored in temporary buffers until the root **IStorage** object commits its changes. *See also* Direct access mode, Commit, Revert.

U

Uniform data transfer

A model for transferring data via the Clipboard, drag and drop, or through automation. Objects conforming to the model implement the **IDataObject** interface. This model replaces DDE (dynamic data exchange). *See* Data transfer object.

Unmarshaling

The processing of unpackaging parameters that have been sent across process boundaries.

V

Virtual Table (VTBL)
An array of pointers that point to interface method implementations.

Visual Editing
A term in end-user documents that refers to the user's ability to interact with a compound-document object in the context of its container. The term most often used by developers is in-place activation. *See also* In-place activation.

Compatibility with OLE 1 and 16:32-Bit Interoperability

Note In moving from OLE 1 to OLE 2, the following changes in terminology were made:

- The OLE 1 term "server application" has been changed to "object application."
- The OLE 1 term "client application" has been changed to "container application."

Compatibility implies that an OLE 1 client application can contain OLE 2 embedded and linked objects and that an OLE 1 server application can create objects to be embedded in and linked to by OLE 2 containers. OLE provides these capabilities by means of a built-in compatibility layer in the core code, which includes a set of functions for conversion.

Interoperability is not the same as OLE 1 to OLE 2 compatibility. Interoperability implies that a 16-bit OLE application can interact with a 32-bit application running on the same system. Interoperability between 16- and 32-bit implementations means that 16-bit and 32-bit implementations can call each other.

Compatibility with OLE 1

The following figure illustrates how OLE 1 and OLE 2 applications communicate. OLE 1 applications make calls to other OLE 1 applications and to OLE 2 applications using DDE.

Compatibility between OLE 2 and OLE 1 applications is achieved through the implementation of two types of special remoting objects, called stubs and proxies. The stub is instantiated on the object, or server, application's side of the process; the proxy is instantiated on the container, or client, application's side. These special stubs and proxies use DDE to communicate rather than LRPC. When an OLE 2 object makes a call to a function in an OLE 1 client application, for example, the stub intercepts the call and responds appropriately. For the most part, this response simulates the response that an OLE 2 object or container would make. However, in a few cases the behavior is different or special **HRESULT** (SCODE) values are returned.

This chapter discusses the issues that affect applications that must be compatible with an earlier or more recent version of OLE and describes the functions that promote this compatibility.

Working with OLE 1 Clients

This section describes some of the idiosyncrasies of working with OLE 1 clients.

A successful call to the **IOleClientSite::GetContainer** method returns a pointer to the container's **IOleContainer** interface. If the container does not support the **IOleContainer** interface, OLE_E_NOT_SUPPORTED is returned. All OLE 1 clients fall in this category, as do OLE 2 containers that do not support linking to their embedded objects.

The **IOleClientSite::ShowObject** method, a request to make the embedded or linked object visible, always returns OLE_E_NOT_SUPPORTED when called on an OLE 1 client**.** The purpose of this method is to help make the user model work smoothly; its failure does not effect OLE functionality.

When an OLE 1 client contains an OLE 2 object and the object is activated or the **OleUpdate** function is called, the aspect of the data returned will always be DVASPECT_CONTENT. This is because OLE 1 clients have no concept of a **FORMATETC** data structure. This situation may occur when an iconic OLE 2 object is pasted from an OLE 2 container into an OLE 1 container. When the object is first pasted, its presentation remains iconic. With the next update, however, the object's content picture is returned.

OLE 1 clients can link to OLE 2 objects only if the link source:

- is represented by a file moniker or a generic composite moniker consisting of a file moniker and one item moniker.
- is not an embedded OLE 2 object.

An OLE 1 client can contain an incompatible link when a linked object is pasted from an OLE 2 container into the OLE 1 client or when an OLE 2 container saves the data to an OLE 1 file, to allow the OLE 1 version of the application access to its data. When the OLE 1 client loads the incompatible link, the link is converted to an embedded object and assigned the class name "Ole2Link." The OLE 1 client cannot connect to the link source. However, if the newly embedded object is then pasted into an OLE 2 container using the Clipboard, or converted to an OLE 2 object using the **OleConvertOLESTREAMToIStorage** function, it will be converted back to its original state as an OLE 2 linked object.

When the link source for an OLE 1 linked object changes its name, the link can remain intact only if the file moniker for the link source has changed. That is, if the link source is a range of cells within an OLE 2 spreadsheet application and the name of the file that contains the cell range changes, OLE will track the link. However, if the name of the cell range changes, the link will break.

Pasting an OLE 2 linked object into an OLE 1 client document and then calling the **OleCopyFromLink** function to convert it to an embedded object will fail if the data transfer object provided by the link source does not support the **IPersistStorage** interface. Creating an embedded object always requires native data, and the **IPersistStorage** interface provides access to native data.

Working with OLE 1 Servers

This section describes some of the known idiosyncrasies of embedding or linking OLE 1 objects.

As with OLE 2 objects, either the **IPersistStorage::InitNew** method or the **IPersistStorage::Load** method must be called to properly initialize a newly instantiated OLE 1 object before any other OLE calls are made. The **InitNew** method should be called to initialize a newly created object; the **Load** method should be called for existing objects. If one of the **OleCreate** helper functions or the **OleLoad** function is being used, these functions make the **IPersistStorage** call, eliminating the need to make the call directly. When an OLE 2 container with an OLE 1 embedded or linked object calls the **IDataObject::GetData** method or the **IDataObject::GetDataHere** method, the container can anticipate support for a smaller set of formats and storage mediums than would be supported for an OLE 2 object. The following table lists the combinations that can be supported.

Tymed Formats	Data Formats
TYMED_MFPICT	CF_METAFILEPICT
TYMED_GDI	CF_BITMAP
TYMED_HGLOBAL	cfNative, CF_DIB, and other OLE 1 server formats

For the aspect value of DVASPECT_ICON, only TYMED_MFPICT with CF_METAFILEPICT is supported. The icon returned from the **IDataObject::GetData** or **IDataObject::GetDataHere** call will always be the first icon (index 0) in the executable object application.

Several methods typically called by containers have unique implementations for OLE 1. The **IPersistStorage::IsDirty** method is defined to return S_OK if the object has changed since its last save to persistent storage; S_FALSE if it has not changed. When an OLE 2 container with an OLE 1 embedded object calls the **IPersistStorage::IsDirty** method, the compatibility code always returns S_OK when the server is running, because there is no way to determine if the object has in fact changed until the File Close or File Update command is selected. S_FALSE is returned when the server is not running.

An OLE 2 implementation of **IOleObject::IsUpToDate** can return either S_OK if the object is up-to-date, S_FALSE if it is not up-to-date, or OLE_E_UNAVAILABLE if the object cannot determine whether it is up-to-date. The OLE 1 implementation always returns either E_NOT_RUNNING, if the object is in the loaded state, or S_FALSE, if the server is running.

The OLE 1 implementation of the **IOleItemContainer::EnumObjects** method always returns OLE_E_NOTSUPPORTED because it is not possible for an OLE 1 server to enumerate its objects.

The **IOleObject::Close** method takes a save option as a parameter that indicates whether the object should be saved before the close occurs. For OLE 2 objects, there are three possible save options: OLECLOSE_SAVEIFDIRTY, OLECLOSE_NOSAVE, and OLECLOSE_PROMPTSAVE. The OLE 1 implementation of the **IOleObject::Close** method treats OLECLOSE_PROMPTSAVE as equivalent to OLECLOSE_SAVEIFDIRTY, because it is not possible to require an OLE 1 server to prompt the user.

OLE 2 containers cannot expect an OLE 1 object to activate in-place; all OLE 1 objects support activation in a separate, open window.

OLE 1 servers do not support linking to their embedded objects. It is up to OLE 2 containers with OLE 1 embedded objects to prevent a possible link from occurring. Containers can call the **CoIsOle1Class** function to determine at Clipboard copy time if a data selection being copied is an OLE 1 object. If the **CoIsOle1Class** function returns TRUE, indicating that the selection is an OLE 1 object, the container should not offer the Link Source format. Link Source must be available for a linked object to be created.

OLE 2 containers can store multiple presentations for an OLE 1 object. However, only the first presentation format is sent to the container when the OLE 1 server closes. After that, the server is in the process of closing down and cannot honor requests for any more formats. Therefore, only the first presentation cache will be updated. The rest will be out of date (perhaps blank) if the object has changed since the last update.

Because OLE 1 servers do not update the cache for every change to an embedded object until the user selects the File Update command, an OLE 2 container may not be obtaining the latest data from the server. By calling the **IOleObject::Update** method, the container can obtain the latest object data.

An OLE 1 embedded (not linked) object does not notify its container that its data has changed until the user chooses File Update or File Close. Therefore, if an OLE 2 container registers for a data-change notification on an OLE 2 object in a particular format, it should be aware that it will not be notified immediately when the data changes.

When an OLE 1 object is inserted into a container document and then closed without an update being invoked, the container document is not saved. Neither are the correct streams for the object written into storage. Any subsequent loading of the object by the container will fail. To protect against this, containers can keep data available after the object closes without updating by implementing the following:

```
OleCreate();                \\ to insert the object
OleRun();                   \\ if OLERENDER_NONE was specified
IOleObject::Update();       \\ to get snapshot of data
OleSave();
IOleObject::DoVerb();
```

Upgrading Applications

When an OLE 1 server is upgraded to an OLE 2 object application, several issues arise. A primary issue is whether the OLE 2 application will replace the OLE 1 application or both versions will coexist. If only the newer version will be available to the user, it is best to convert objects from the older version of the application automatically to the new version format. Objects can be converted on a global basis, where all objects of a specific class are converted, or on a more selective basis, where only some objects are converted. Conversion can be either automatic, under programmatic control, or under the control of a user.

The ability to detect whether an object is from an OLE 1 server is helpful for implementing conversion functionality. The OLE 2 implementation of the **IPersistStorage::Load** method can check for a stream named "\1Ole10Native." The "\1Ole10Native" stream contains a DWORD header whose value is the length of the native data that follows. The existence of this stream indicates that the data is coming from an OLE 1 server. Applications can check whether a storage object contains an object in an OLE 1 format by calling the **ReadFmtUserTypeStg** method and examining the contents of *pcfFormat.* This is where the OLE 1 class name would appear.

In the **IPersistStorage::Save** method, objects that are being permanently converted should be written back to storage in the new format and the "\1Ole10Native" stream should be deleted. The conversion bit in the storage should also be cleared once the conversion to the new format is complete.

To allow manual conversion of an old OLE 1 object to the new OLE 2 version, the OLE 2 object application must put the OLE 1 server's ProgID (OLE 1 server class name) into the registry under the **CLSID\{...}\Conversion\Readable\Main** entry. This entry indicates that the OLE 2 application can read its OLE 1 data format; the 'Clipboard format' of the OLE 1 data is the ProgID (that is, the class name) of the OLE 1 object.

To get a CLSID for an OLE 1 server, the **CLSIDFromProgId** function or the **CLSIDFromString** function must be called. That is, an OLE 1 application cannot be assigned a CLSID from an OLE 2 application with **uuidgen.exe**, **CoCreateGuid**, or by using a GUID from a range assigned by Microsoft. Because all OLE 1 CLSIDs are expected to fall in a specific range, OLE 1 CLSIDs are assigned with the **CLSIDFromProgId** function.

Refer to the appendix called "Registering Object Applications" for detailed information on the required registry entries for upgraded applications.

Functions to Support Compatibility

The following functions enable applications to determine whether an object class is from OLE 1, and to support conversion between OLE 1 and OLE 2 storage formats.

OLE 1 Compatibility Functions	Description
CoIsOle1Class	Determines if a given CLSID represents an OLE 1 object.
OleConvertIStorageToOLESTRAM	Converts the specified storage object from OLE 2 structured storage to the OLE 1 storage model.
OleConvertIStorageToOLESTREAMEx	Converts the specified storage object from OLE 2 structured storage to the OLE 1 storage model.
OleConvertOLESTREAMToIStorage	Converts the specified object from the OLE 1 storage model to an OLE 2 structured storage object.
OleConvertOLESTREAMToIStorageEx	Converts the specified object from the OLE 1 storage model to an OLE 2 structured storage object.

16:32 Bit Interoperability

The 32-bit implementation of OLE 2 is compatible with OLE 1 applications in the manner described above. Interoperability is achieved by converting calls made on objects in one model into calls suitable in another model. For example, if a call is made by a 16-bit application to an object that is a 32-bit implementation, the OLE 16:32 interoperability (thunk) layer takes care of marshalling and converting parameters between the two models.

Thunk Layer Operation

The thunk layer replaces the 16-bit implementations of OLE 2 with a new set of binaries that allow the 16-bit implementations to call the updated 32-bit implementation of OLE 2. These new binaries, collectively known as the thunk layer, forward all OLE 2 method and function calls to the 32-bit implementation of OLE. The OLE thunk layer inserts itself between implementations of different models. Whenever an interface is passed between implementation types, the system provides a thunk proxy and stub.

For example, a call to the **CreateFileMoniker** function returns a pointer to a moniker. When a 16-bit implementation calls the **CreateFileMoniker** function, the system thunks the call to the 32-bit OLE 2 implementation and creates a file moniker using a 32-bit implementation. When the call returns through the OLE thunk layer, the moniker pointer is translated into a 16/32 proxy.

A 16/32 proxy sets up an object in the 16-bit address space that, when called, converts each parameter for the specific method into the appropriate values for calling the 32-bit implementation. It then performs a model switch from 16-bit to 32-bit. The call is made on the 32-bit implementation, and then another model switch is made from the 32-bit to 16-bit implementation. In the process of switching back, any output parameters are converted into their 16-bit counterparts.

The system keeps track of the model of each pointer. For example, if a 16-bit application calls a 32-bit application, it passes a 16-bit pointer to an object. The thunk layer will create a 32/16 proxy to let the 32-bit application call the 16-bit application's object. In response, the 32-bit application calls back to the 16-bit application, passing the pointer it received. In this direction, the pointer being held by the 32-bit application holds is to a 32/16 proxy. The thunk layer also detects that the 32-bit pointer being passed in is a proxy, and passes the original pointer it started with to the 16-bit application. Thus, the 16-bit application sees the same pointer it passed in. This works when passing from 16 to 32 to 16, or from 32 to16 to 32. For more information about thunking, see the *Win32 SDK*.

Interoperability Using Standard Interfaces (OLE 2 Defined)

In its 32-bit implementation, OLE 2 is provided by the operating system and can translate OLE-defined interfaces and methods. Existing OLE 2 applications will continue to work as before. The following combinations of interoperability using the standard interfaces defined by OLE 2 are supported:

- 16 to 16 (in-process server)
- 16 to 16 (local server)
- 16 to 32 (local server)
- 32 to 32 (in-process server)
- 32 to 32 (local server)
- 32 to 16 (local server)

32- to16-bit (in-process server) interoperability using the standard interfaces is not supported.

Interoperability Using Custom Interfaces (either MIDL or Manually Written Marshalling)

Due to major architectural changes in the OLE communication layer, 16-bit applications that require remoting of custom interfaces are not supported at this time. However, 16-bit applications that use custom interfaces to an inprocess server should still function correctly. The following combinations of interoperability using custom interfaces are supported:

- 16 to 16 (inprocess server)
- 32 to 32 (inprocess server)
- 32 to 32 (local server using MIDL-generated stubs)

The following combinations of interoperability using custom interfaces are not supported:

- 16 to 16 (local server)
- 16 to 32 (all combinations)
- 32 to 16 (all combinations)

Programmers who have written applications that provide custom interfaces must rewrite the custom interface marshalling code to conform to the RPC-based communication layer. However, information about writing a 16-bit custom-marshalling library is not available at this time. For more information about RPC, see the *RPC Programmer's Guide and Reference*.

The Component Object Model

The Component Object Model (COM) is the base technology of OLE, a broad set of object-oriented technology standards. OLE includes, besides COM, object design standards at a higher level. Among these are standards for OLE Structured Storage, OLE Compound Documents, and OLE Controls. COM is the binary standard that defines the means for applications to interact within these technology standards.

To understand COM (and therefore all OLE technologies), it is crucial to bear in mind that it is not an object-oriented language, but a standard. Nor does COM specify how an application should be structured. Language, structure, and implementation details are left to the application programmer. COM does specify an object model and programming requirements that enable COM objects (also called OLE Components, or sometimes simply *objects*) to interact with other objects. These objects can be within a single process, in other processes, even on remote machines. They can have been written in other languages, and may be structurally quite dissimilar. That is why COM is referred to as a binary standard – it is a standard that applies after a program has been translated to binary machine code.

The only language requirement for COM is that code is generated in a language that can create structures of pointers and, either explicitly or implicitly, call functions through pointers. Object-oriented languages such as C++ and Smalltalk provide programming mechanisms that simplify the implementation of COM objects, but languages such as C, Pascal, Ada, and even BASIC programming environments can create and use OLE objects.

COM defines the essential nature of an OLE Component. In general, a software object is made up of a set of data and the functions that manipulate the data. An OLE Component is one in which access to an object's data is achieved exclusively through one or more sets of related functions. These function sets are called *interfaces*, and the functions of an interface are called *methods.* Further, COM requires that the only way to gain access to the methods of an interface is through a pointer to the interface.

Besides specifying the basic binary object standard, COM defines certain basic interfaces that provide function common to all COM-based technologies. It also provides a small number of API functions that all components require.

OLE Components and Interfaces

COM is a technology that allows objects to interact across process and machine boundaries as easily as objects within a single process interact. COM enables this by specifying that the only way to manipulate the data associated with an object is through what is called an *interface on the object.* When this term is used, it refers to an implementation in code of a COM binary-compliant interface that is associated with an object.

Talking about an object that *implements an interface* means that the object uses code that implements each method of the interface and provides COM binary-compliant pointers to those functions to the COM library. COM then makes those functions available to any client who asks for a pointer to the interface, whether the client is inside or outside of the process that implements those functions.

Interfaces and Interface Implementations

COM makes a fundamental distinction between interface definitions and their implementations. An *interface* is actually a contract that consists of a group of related function *prototypes* whose usage is defined but whose implementation is not. These function prototypes are equivalent to pure virtual base classes in C++ programming. An interface definition specifies the interface's member functions, called *methods,* their return types, the number and types of their parameters, and what they must do. There is no implementation associated with an interface.

An *interface implementation* is the code a programmer supplies to carry out the actions specified in an interface definition. Implementations of many of the interfaces a programmer could use in an object-based application are included in the OLE libraries. Programmers are, however, free to ignore these implementations and write their own. An interface implementation is to be associated with an object when an instance of that object is created, and provides the services that the object offers.

For example, a hypothetical interface named **IStack** (many interface names begin with the letter "I") might define two *methods,* named **Push** and **Pop**, specifying that successive calls to the **Pop** method return, in reverse order, values previously passed to the **Push** method. This interface definition, however, would not specify how the functions are to be implemented in code. In implementing the interface, however, one programmer might implement the stack as an array and implement the **Push** and **Pop** methods in such a way as to access that array; while another programmer might prefer to use a linked list and would implement the methods accordingly. Regardless of a particular implementation of the **Push** and **Pop** methods, however, the in-memory representation of a pointer to an **IStack** interface, and therefore its use by a client, is completely defined by the interface definition.

Simple objects may support only a single interface. More complicated objects, such as embeddable objects, typically support several interfaces. Clients have access to an OLE object only through a pointer to one of its interfaces, which, in turn, allows the client to call any of the methods that make up that interface. These methods determine how a client can use the object's data.

Interfaces, in fact, define a contract between an object and its clients. The contract specifies the methods that must be associated with each interface, and what the behavior of each of the methods must be in terms of input and output. The contract generally does not define *how* to implement the methods in an interface. Another important aspect of the contract is that if an object supports an interface, it must support all of its methods in some way. Not all of the methods in an implementation need to do something−if an object does not support the function implied by a method, its implementation may be a simple return, or perhaps the return of a meaningful error message−but the methods must *exist.*

COM uses the word "interface" in a sense different from that typically used in C++ programming. A C++ interface refers to *all* of the functions that a class supports and that clients of an object can call to interact with it. A COM interface refers to a predefined group of related functions that a COM class implements, but does not necessarily represent *all* the functions that the class supports.

Interface Pointers and Interfaces

An instance of an interface implementation is actually a pointer to an array of pointers to methods (a function table that refers to an implementation of all of the methods specified in the interface). Objects with multiple interfaces can provide pointers to more than one function table. Any code that has a pointer through which it can access the array can call the methods in that interface.

Speaking precisely about this multiple indirection is inconvenient, so instead, the pointer to the interface function table that another object must have to call its methods is called simply an *interface pointer*. You can manually create function tables in a C application or almost automatically with C++ (or other object-oriented languages that support COM).

With appropriate compiler support (which is inherent in C and C++), a client can call an interface method through its name, not its position in the array. Because an interface is a type, given the names of methods the compiler can check the types of parameters and return values of each interface method call. In contrast, such type-checking is not available even in C or C++ if a client uses a position-based calling scheme.

Each interface–the immutable contract of a functional group of methods–is referred to at runtime with a globally-unique interface identifier, an "IID". This IID, which is a specific instance of a GUID (a Globally Unique IDentifier supported by OLE) allows a client to ask an object precisely if it supports the semantics of the interface without unnecessary overhead and without the confusion that could arise in a system from having multiple versions of the same interface with the same name.

To summarize, it is important to understand what an interface is, and is not:

- An interface is not the same as a C++ class.
- An interface is not an object.
- Interfaces are strongly typed.
- Interfaces are immutable.

An interface is not a C++ class–the pure virtual definition carries no implementation. If you are a C++ programmer, you can, however, define your implementation of an interface as a class, but this falls under the heading of implementation details, which COM does not specify. An instance of an object that implements an interface must be created for the interface actually to exist. Furthermore, different object classes may implement an interface differently yet be used interchangeably in binary form, as long as the behavior conforms to the interface definition.

An interface is not an object – it is simply a related group of functions and is the binary standard through which clients and objects communicate. The object can be implemented in any language with any internal state representation, as long as it can provide pointers to interface methods.

Interfaces are strongly typed – every interface has its own interface identifier (a GUID), which eliminates the possibility of duplication that could occur with any other naming scheme.

Interfaces are immutable contracts – you cannot define a new version of an old interface and give it the same identifier. Adding or removing methods of an interface, or changing semantics creates a new interface, not a new version of an old interface. Therefore a new interface cannot conflict with an old interface. Objects can, of course, support multiple interfaces simultaneously, and can expose interfaces that are successive revisions of an interface, with different identifiers. Thus, each interface is a separate contract, so system-wide objects need not be concerned about whether the version of the interface they are calling is the one they expect. The interface ID (IID) defines the interface contract explicitly and uniquely.

IUnknown and Interface Definition Inheritance

Inheritance in COM does not mean code reuse. Because no implementations are associated with interfaces, interface inheritance does not mean code inheritance. It means only that the contract associated with an interface is inherited in a C++ pure-virtual base-class fashion and modified – either by adding new methods or by further qualifying the allowed usage of methods. There is no selective inheritance in COM: If one interface inherits from another, it includes all the methods that the other interface defines.

Inheritance is used sparingly in the predefined COM interfaces. All predefined interfaces inherit their definitions from the important interface **IUnknown**, which contains three vital methods: **QueryInterface**, **AddRef**, and **Release**. All COM objects must implement the **IUnknown** interface, because it provides the means to move freely between the different interfaces that an object supports with **QueryInterface**, and to manage its lifetime with **AddRef** and **Release**. More on these methods is discussed later in this chapter.

Any single object usually requires only a single implementation of the **IUnknown** methods. This means that by implementing any interface on an object you must completely implement the **IUnknown** functions. You do not generally need to explicitly inherit from nor implement **IUnknown** as its own interface. When a client of the object queries for it, it is usually necessary only to typecast another interface pointer into an **IUnknown**, as this interface makes up the first three entries in the function table already.

In some specific situations, notably in creating an object that supports aggregation, you may need to implement one set of **IUnknown** functions for all interfaces as well as a stand-alone **IUnknown** interface. This is described in the following section. In any case, any object implementor will implement **IUnknown** methods.

While there are a few interfaces that inherit their definitions from a second interface, in addition to **IUnknown**, the majority are simply the **IUnknown** interface methods plus the methods defined in the interface. This makes most interfaces relatively compact and easy to encapsulate.

Reusing Objects

An important goal of any object model is that object authors can reuse and extend objects provided by others as pieces of their own implementations. One way to do this in C++ and other languages is implementation inheritance, which allows an object to inherit ("subclass") some of its functions from another object while overriding other functions.

The problem for system-wide object interaction using traditional implementation inheritance is that the contract (the interface) between objects in an implementation hierarchy is not clearly defined. In fact, it is implicit and ambiguous. When the parent or child object changes its implementation, the behavior of related components may become undefined, or unstably implemented. In any single application, where the implementation can be managed by a single engineering team, who update all of the components at the same time, this is not always a major concern. In an environment where the components of one team are built through black-box reuse of other components built by other teams, this type of instability jeapordizes reuse. Additionally, implementation inheritance usually works only within process boundaries. This makes traditional implementation inheritance impractical for large, evolving systems composed of software components built by many engineering teams.

The key to building reusable components is to be able to treat the object as a black box. This means that the piece of code attempting to reuse another object knows nothing, and needs to know nothing, about the internal structure or implementation of the component being used. In other words, the code attempting to reuse a component depends upon the *behavior* of the object and not the exact *implementation*.

To achieve black-box reusability, COM adopts other established reusability mechanisms: *containment*/*delegation* and *aggregation*. For convenience in describing them, the object being reused is called the *inner object* and the object making use of that inner object is the *outer object*.

It is important to remember in both these mechanisms how the outer object appears to its clients. As far as the clients are concerned, both objects implement any interfaces to which the client can get a pointer. The client treats the outer object as a black box, and thus does not care, nor does it need to care, about the internal structure of the outer object – the client cares only about behavior.

Containment/Delegation

The first, and most common, mechanism is called containment/delegation. This type of reuse is a familiar concept found in most object-oriented languages and systems. The outer object acts as an object client to the inner object. The outer object "contains" the inner object and when the outer object requires the services of the inner object, the outer object explicitly delegates implementation to the inner object's methods. Thus, the outer object uses the inner object's services to implement itself.

It is not necessary for the outer and inner objects to support the same interfaces, although it certainly is reasonable to contain an object that implements an interface that the outer object does not, and implement the methods of the outer object simply as calls to the corresponding methods in the inner object. When the complexity of the outer and inner objects differs greatly, however, the outer object may implement some of the methods of its interfaces by delegating calls to interface methods implemented in the inner object.

It is simple to implement containment for an outer object. The outer object creates the inner objects it needs to use as any other client would. This is nothing new – the process is like a C++ object that itself contains a C++ string object that it uses to perform certain string functions, even if the outer object is not considered a "string" object in its own right. Then, using its pointer to the inner object, a call to a method in the outer object generates a call to an inner object method.

Aggregation

Aggregation is the other, richer, reuse mechanism, in which the outer object exposes interfaces from the inner object as if they were implemented on the outer object itself. This is useful when the outer object would always delegate every call to one of its interfaces to the same interface in the inner object. Aggregation is actually a specialized case of containment/delegation, and is available as a convenience to avoid extra implementation overhead in the outer object in these cases.

Aggregation is almost as simple to implement as containment is, except for the three **IUnknown** functions: **QueryInterface***,* **AddRef***,* and **Release**. The catch is that from the client's perspective, any **IUnknown** function on the outer object must affect the outer object. That is, **AddRef** and **Release** affect the outer object and **QueryInterface** exposes all the interfaces available on the outer object. However, if the outer object simply exposes an inner object's interface as its own, that inner object's **IUnknown** members called through that interface will behave differently than those **IUnknown** members on the outer object's interfaces, an absolute violation of the rules and properties governing **IUnknown**.

The solution is for the outer object to somehow pass the inner object some **IUnknown** pointer to which the inner object can re-route (that is, delegate) **IUnknown** calls in its own interfaces, and yet there must be a method through which the outer object can access the inner object's **IUnknown** functions that affect only the inner object. COM provides specific support for this solution. Briefly, aggregation requires an explicit implementation of **IUnknown** on the inner object and delegation of the **IUnknown** methods of any other interface to the outer object's **IUnknown** methods.

The COM Library

In addition to being a specification, COM also implements some important services in the COM Library. Provided as a DLL in Microsoft® Windows®, the COM Library includes:

- A small number of fundamental API functions that facilitate the creation of COM applications, both client and server. For clients, COM supplies basic functions for creating objects. For servers, COM supplies the means of exposing their objects.
- Implementation-locator services through which COM determines from a unique class identifier (CLSID) which server implements that class and where that server is located. This service includes support for a level of indirection, usually a system registry, between the identity of an object class and the packaging of the implementation such that clients are independent of the packaging, which can change in the future.
- Transparent remote procedure calls when an object is running in a local or remote server.
- A standard mechanism to allow an application to control how memory is allocated within its process, particularly memory that needs to be passed between cooperating objects such that it can be freed properly.

To use the basic services that COM supplies, all COM applications must call the **CoInitialize** function before calling any other COM function. This function initializes the COM library, allowing the application to use its services. OLE Compound Document applications call the **OleInitialize** function, which calls **CoInitialize** and also does some initialization required for compound documents.

It is also important to uninitialize the library. For each call to **CoInitialize** (or **OleInitialize**, there must be a corresponding call to **CoUninitialize** (or **OleUninitialize**).

OLE Class Objects and CLSIDs

A COM class is an implementation of a group of interfaces in code executed whenever you interact with a given object. It is important to make the distinction between a C++ class and a COM class. In C++, a class is a type. A COM class is simply a definition of the object, and carries no type, although a C++ programmer might implement it using a C++ class. COM is designed to allow a class (object definition) to be used by different applications, including applications written without knowledge of that particular class's existence. Therefore, class code for a given type of object exists either in a dynamic linked library (DLL) or in another application (EXE). COM specifies a mechanism by which the class code can be used by many different applications.

A COM class is identified by a unique 128-bit CLSID. This CLSID is used to associate an object class with the DLL or EXE in the file system whose code implements the class. A CLSID, like an interface identifier, is a GUID, so no other class, no matter who writes it, has a duplicate CLSID. Server implementors generally obtain CLSIDs through the **CoCreateGUID** function in COM. This function is guaranteed to produce unique CLSIDs, so server implementors across the world can independently develop and deploy their software without fear of accidental collision with software written by others.

Using unique CLSIDs avoids the possibility of name collisions among classes because CLSIDs are in no way connected to the names used in the underlying implementation. So, for example, two different vendors can write classes called "StackClass," but each would have a unique CLSID and therefore could not be confused.

On its host system, COM maintains a database (it makes use of the system registration database, or "registry" on Windows platforms) of all the CLSIDs for the servers installed on the system. This is a mapping between each CLSID and the location of the DLL or EXE that houses the code for that CLSID. COM consults this database whenever a client wants to create an instance of a COM class and use its services. The client, however, that needs to know only the CLSID, is independent of the specific installation location of the DLL or EXE on the particular machine.

When a COM application passes a CLSID to COM and asks for an object instance in return, it is a COM client. No matter where the server is located (in-process, local, or remote), a COM client always asks COM to create an object instance in exactly the same way.

The basic way to implement a class is through a COM *class object*. This is a simply an intermediate object that supports functions common to creating new instances of a given class. All class objects used to create objects from a CLSID support the **IClassFactory** interface, an interface that includes the important method **CreateInstance**. You implement an **IClassFactory** interface for each class of object that you offer to be instantiated.

When a client wants to create an instance of a class, it calls the COM function **CoGetClassObject**, supplying to it the CLSID of the desired class. This call instantiates the class object for the class and returns a pointer to the class object's **IClassFactory** interface implementation. The client would then call the returned interface's **CreateInstance** method, which creates an instance of the class and returns a pointer to a requested interface on the object.

At that point the client has interface pointers for *two separate objects*, the class object and an object of that class, each with their own reference counts. This is an important distinction.

The simplest way to create one instance is to call the COM function **CoCreateInstance.** It is a convenient helper function for clients that want to create a single instance of a class. The **CoCreateInstance** function internally calls **CoGetClassObject** itself, encapsulates the process just described, and additionally releases the class object for you.

In summary, a COM Client, in addition to its responsibilities as a COM application, must use COM to obtain a class object, ask the class object to create an object, initialize the object, and to call that object's (and the class object's) **Release** function when the client is finished with it.

Registering a COM Server

After you have defined a class in code and given it a CLSID and a class object, it is important to make OLE aware of the object by associating a DLL or EXE server with the CLSID.   By doing so, clients can create instances of it. There are two ways of doing this, depending on whether your class is available only dynamically (while some process or activity is running), or at any time (by running code that has been installed in the system).

In the former case, you can register a dynamic class with the system by using **CoRegisterClassObject**, passing to that function a CLSID and a pointer to an **IClassFactory** interface. Subsequent clients who call **CoCreateInstance** or **CoGetClassObject** with this CLSID will retrieve a reference to this **IClassFactory** pointer.

In the latter case, you can register information in the class database (within the registry) during installation of your component, telling COM what type of object you are defining and how it is to be instantiated. For example, classes implemented in dynamic-link libraries can specify that COM will load their DLL and call a well-known entry-point (**DllGetClassObject**) to retrieve an **IClassFactory** interface. Similarly, classes implemented in executables can specify that COM should execute their process and wait for the process to register their class object's **IClassFactory** through a call to the **CoRegisterClassObject** function.

Details about the information required in the registry can be found in Appendix A.

Getting a Pointer to an Object

Because OLE does not have a strict class model, there are several ways to instantiate or to get a pointer to an interface on an object. There are, in fact, four methods through which a client obtains its first interface pointer to a given object:

- Call a COM Library API function that creates an object of a pre-determined type – that is, the function will only return a pointer to one specific interface for a specific object class.
- Call a COM Library API function that can create an object based on a class identifier (CLSID) and that returns any type of interface pointer requested.
- Call a method of some interface that creates another object (or connects to an existing one) and returns an interface pointer on that separate object.
- Implement an object with an interface through which other objects pass their interface pointer to the client directly.

There are numerous OLE functions that return specific pointers to specific functions, such as **CoGetMalloc**, which returns a pointer to the standard OLE memory allocator.

The COM API functions **CoGetClassObject** and **CoCreateInstance**, described earlier, create a class object and return a pointer to a requested object. A special case of this kind of API is the OLE Compound Document helper API **OleCreate**, which creates an instance of an embeddable object based on a CLSID.

Among the many interface methods that return a pointer to a separate object are several that create and return a pointer to an enumerator object, which allows you to determine how many items of a given type an object maintains. OLE defines interfaces for enumerating a wide variety of items, such as strings, several structures important in various OLE technologies, monikers, and **IUnknown** interface pointers. The typical way to create an enumerator instance and get a pointer to its interface is to call a method from another interface. For example, the **IDataObject** interface defines two methods, **EnumDAdvise** and **EnumFormatEtc**, that return pointers to interfaces on two different enumeration objects. There are many other examples in OLE of methods that return pointers to objects, such as the OLE Compound Document interface **IOleObject::GetClientSite**, which, when called on the embedded or linked object, returns a pointer to the container object's implementation of **IOleClientSite**.

The fourth way to get a pointer to an object is used when two objects, such as an OLE Compound Document container and server need bi-directional communication. Each implements an object containing an interface method to which other objects can pass interface pointers. In the case of containers and servers, each object then passes its pointer to the other object. The implementing object, which is also the client of the created object, can then call the method and get the pointer that was passed.

QueryInterface: Navigating in an Object

Once you have an initial pointer to an interface on an object, OLE has a very simple mechanism to find out whether the object supports another specific interface, and, if so, to get a pointer to it. This mechanism is the **QueryInterface** method of the **IUnknown** interface. If the object supports the requested interface, the method must return a pointer to that interface. This permits an object to navigate freely through the interfaces that an object supports. **QueryInterface** separates the request "Do you support a given contract?" from the high-performance use of that contract once negotiations have been successful.

When a client initially gains access to an object, that client will receive, at a minimum, an **IUnknown** interface pointer (the most fundamental interface) through which it can control the lifetime of the object – tell the object when it is done using the object – and invoke **QueryInterface**. The client is programmed to ask each object it manages to perform some operations, but the **IUnknown** interface has no functions for those operations. Instead, those operations are expressed through other interfaces. The client is thus programmed to negotiate with objects for those interfaces. Specifically, the client will ask an object – by calling **QueryInterface** – for an interface through which the client may invoke the desired operations.

Since the object implements **QueryInterface**, it has the ability to accept or reject the request. If the object accepts the client's request, **QueryInterface** returns a new pointer to the requested interface to the client. Through that interface pointer, the client has access to the methods of that interface. If, on the other hand, the object rejects the client's request, **QueryInterface** returns a null pointer – an error – and the client has no pointer through which to call the desired functions. In this case, the client must deal gracefully with that possibility. For example, suppose a client has a pointer to interface A on an object, and asks for interfaces B and C. While the object supports interface B, it does not support interface C. The object returns a pointer to B, and reports that C is not supported.

A key point is that when an object rejects a call to **QueryInterface**, it is impossible for the client to ask the object to perform the operations expressed through the requested interface. A client *must* have an interface pointer to invoke methods in that interface. There is no alternative. If the object refuses to provide one, a client must be prepared to do without, simply not doing whatever it had intended to do with that object. Had the object supported that interface, the client might have done something useful with it. This works well in comparison with other object-oriented systems in which you cannot know whether a function will work until you call that function, and even then, handling failure is uncertain. **QueryInterface** provides a reliable and consistent way to know whether an object supports an interface before attempting to call its methods.

The **QueryInterface** method also provides a robust and reliable way for an object to indicate that it does *not* support a given contract. That is, if in a call to **QueryInterface** one asks an "old" object whether it supports a "new" interface (one, for example, that was invented after the old object had been shipped), the old object will reliably and , without causing a crash, answer "no." The technology that supports this is the algorithm by which IIDs are allocated. While this may seem like a small point, it is extremely important to the overall architecture of the system, and the ability to inquire of old things about new functionality is, surprisingly, a feature not present in most other object architectures.

Managing Object Lifetimes through Reference Counting

In traditional object systems, the life cycle of objects – the issues surrounding the creation and deletion of objects – is handled implicitly by the language (or the language runtime) –instances go explicitly out of scope as parameters or global variables, for example– or explicitly by application programmers.

In an evolving, decentrally constructed *system* made up of reused components, it is no longer true that any one or client, or even programmer, always "knows" how to deal with a component's lifetime. For a client with the right security privileges, it is still relatively easy to create objects through a simple request. But object deletion is another matter entirely. It is not necessarily clear when an object is no longer needed and should be deleted. Even when the original client is done with the object, it cannot simply shut the object down, because some other client (or clients) in the system may still have a reference to it.

One way to ensure that an object is no longer needed, so it can be deleted, is to depend entirely upon an underlying communication channel to inform the system when all connections to a cross-process or cross-channel object have disappeared. Schemes that use this method are unacceptable for several reasons. One problem is that it could require a major difference between the cross-process/cross-network programming model and the single-process programming model. In the cross-process/cross-network programming model, the communication system would provide the hooks necessary for component lifetime management, while in the single-process programming model, objects are directly connected without any intervening communications channel. Another problem is that this scheme could also result in a layer of system-provided software that would interfere with component performance in the in-process case. Furthermore, such a mechanism based on explicit monitoring would not tend to scale towards many thousands or millions of objects.

COM offers a scalable and distributed approach to this set of problems. Clients tell an object when they are using it and when they are done, and objects delete themselves when they are no longer needed. This approach mandates that all objects count references to themselves.

Just as an application must free memory it has allocated once that memory is no longer in use, a client of an object is responsible for freeing its references to the object when that object is no longer needed. In an object-oriented system, the client can only do this by giving the object an instruction to free itself.

It is important that an object be deallocated when it is no longer being used. The difficulty lies in determining when it is safe to deallocate an object. This is easy with automatic variables (those allocated on the stack) – they cannot be used outside the block in which they're declared, so the compiler deallocates them when the end of the block is reached. For COM objects, which are dynamically allocated, it is up to the clients of an object to decide when they no longer need to use the object – especially local or remote objects that may be in use by multiple clients at the same time. The object must wait until *all* clients are finished with it before freeing itself. Because COM objects are manipulated through interface pointers and can be used by objects in different processes or on other machines, the system cannot keep track of an object's clients.

COM's method of determining when it is safe to deallocate an object is manual reference counting. Each object maintains a 32-bit reference count that tracks how many clients are connected to it, that is, how many pointers exist to any of its interfaces in any client.

Implementing Reference Counting

Reference counting requires work on the part of both the implementor of a class and the clients who use objects of that class. When you implement a class, you must implement the **AddRef** and **Release** methods as part of the **IUnknown** interface. These two functions have simple implementations:

1. **AddRef** increments the object's internal reference count.
2. **Release** first decrements the object's internal reference count; then it checks whether the reference count has fallen to zero. If it has, that means no one is using the object any longer, so the **Release** function deallocates the object.

A common implementation approach for most objects is to have only one implementation of these functions (along with **QueryInterface**), which are shared between all interfaces, and therefore a reference count which applies to the entire object. Architecturally, however, from a client's perspective, reference counting is strictly and clearly a *per-interface-pointer* notion, and objects may be implemented which take advantage of this capability by dynamically constructing, destroying, loading, or unloading portions of their functionality based on the currently extant interface pointers

Whenever a client calls a method (or API function) that returns a new interface pointer, such as **QueryInterface**, the method being called is responsible for incrementing the reference count through the returned pointer. For example, when a client first creates an object, it receives an interface pointer to an object that, from the client's point of view, has a reference count of one. If the client then calls **AddRef** on the interface pointer, the reference count becomes two. The client must call **Release** twice on the interface pointer to drop all of its references to the object.

An illustration of how reference counts are strictly per-interface-pointer occurs when a client calls **QueryInterface** on the first pointer for either a new interface or the same interface. *In either of these cases* the client is required to call **Release** once for each pointer. COM does not require that an object return the same pointer when asked for the same interface multiple times. (The only exception to this is a query to **IUnknown**, which acts as an object's *identity* to COM.) This allows the object implementation to manage resources efficiently.

Rules for Managing Reference Counts

The elegance of using reference counting for object lifetime management is that independent objects can obtain and release access to a single object, and not have to coordinate with each other over lifetime management. In a sense, the object provides this management, so long as the client objects conform to the rules of use. Within a single object that is completely under the control of a single development organization, that organization can adopt whatever strategy it chooses. The following rules are about how to manage and communicate interface instances between objects, and are a reasonable starting point for a policy within an object.

The conceptual model is that interface pointers are thought of as living in pointer variables, which, for this discussion, include variables in memory locations and in internal processor registers, and both programmer- and compiler-generated variables. In short, it includes all internal computation state that holds an interface pointer. Assignment to or initialization of a pointer variable involves creating a *new copy* of an already existing pointer. Where there was one copy of the pointer in some variable (the value used in the assignment/initialization), there are now two. An assignment to a pointer variable *destroys* the pointer copy presently in the variable, as does the destruction of the variable itself (that is, the scope in which the variable is found, such as the stack frame, is destroyed).

Rule 1:

AddRef must be called for every new copy of an interface pointer, and **Release** called for every destruction of an interface pointer except where subsequent rules explicitly permit otherwise.

This is the default case. Unless special knowledge permits otherwise, the worst case must be assumed. The exceptions to Rule 1 all involve knowledge of the relationships of the lifetimes of two or more copies of an interface pointer. In general, they fall into two categories: 1) nested lifetimes and 2) overlapping lifetimes.

In a nested lifetime relationship, the first copy of the interface pointer is created, followed by the creation of the second copy. The second copy is destroyed before the end of the first copy's lifetime. The calls to **AddRef** and **Release** for the second copy can be omitted. The overlapping lifetime relationship begins like the nested lifetime relationship – the second interface pointer is created after the first. In overlapping lifetimes, however, the first pointer copy is destroyed before the second pointer copy. The calls to **AddRef** for the second copy and **Release** for the first copy can be omitted.

Rule 2:

If the piece of code has special knowledge of the relationships of the beginnings and the endings of the lifetimes of two or more copies of an interface pointer, it allows you to omit **AddRef**/**Release** pairs. The following rules call out specific common cases of Rule 2. The first two of these rules are particularly important, as they are especially common.

In-parameters to functions

The copy of an interface pointer which is passed as an actual parameter to a function has a lifetime which is nested in that of the pointer used to initialize the value. The actual parameter, therefore, need not be separately reference counted.

Out-parameters from functions, including return values

This situation falls under Rule 2. To set the out parameter, the function itself, by Rule 1, must have a stable copy of the interface pointer. On exit, the responsibility for releasing the pointer is transferred from the one called to the caller. Thus, the out-parameter need not be separately reference counted.

Local variables

A method implementation clearly has omniscient knowledge of the lifetimes of each of the pointer variables allocated on the stack frame. It can, therefore, use this knowledge to omit redundant **AddRef**/**Release** pairs.

Backpointers

Some data structures are contain two components, A and B, each with a pointer to the other. If the lifetime of one component (A) is known to contain the lifetime of the other (B), then the pointer from the second component back to the first (from B to A) need not be reference counted. Often, avoiding the cycle that would otherwise be created is important in maintaining the appropriate deallocation behavior. However, such non-reference counted pointers should be used *with extreme caution.* In particular, as the remoting infrastructure cannot know about the semantic relationship in use here, such backpointers cannot be remote references. In almost all cases, an alternative design of having the backpointer refer a second "friend" object of the first, rather than the object itself (thus avoiding the circularity), is a superior design.

The following rules call out common situations that follow from Rule 1.

In-Out parameters to functions

The caller must **AddRef** the actual parameter, since it will be released (with a call to **Release**) by the one called when the out-value is stored on top of it.

Fetching a global variable

The local copy of the interface pointer fetched from an existing copy of the pointer in a global variable must be independently reference counted since called functions might destroy the copy in the global while the local copy is still alive.

New pointers synthesized out of "thin air."

A function that synthesizes an interface pointer using special internal knowledge rather than obtaining it from some other source must do an initial **AddRef** on the newly synthesized pointer. Important examples of such routines include instance creation routines, implementations of **IUnknown::QueryInterface**, etc.

Returning a copy of an internally stored pointer

Once the pointer has been returned, the one called has no idea how its lifetime relates to that of the internally stored copy of the pointer. Thus, the one called must call **AddRef** on the pointer copy before returning it.

Finally, when implementing or using reference counted objects, a technique called *artificial reference counts* sometimes proves useful. In implementing a method of an interface, if you invoke functions that have even the remotest chance of decrementing your reference count, this may cause you to release before it returns to the method when it is called. The subsequent code in the method would crash.

A robust way to protect yourself from this is to insert an **AddRef** at the beginning of the method implementation which is paired with a **Release** just before the method returns.

These "artificial" reference counts guarantee object stability while processing is done.

Managing Memory Allocation

In COM, many, if not most, interface methods and APIs are called by code written by one programming organization and implemented by code written by another. Many of the parameters and return values of these functions are of types that can be passed around by value. Sometimes, however, it is necessary to pass data structures for which this is not the case, so it is necessary for both caller and called to have a compatible allocation and de-allocation policy. COM defines a universal convention for memory allocation, both because it is more reasonable than defining case-by-case rules, and so the COM remote procedure call implementation can correctly manage memory.

The methods of COM interface always provide memory management of pointers to the interface by calling the **AddRef** and **Release** functions found in the **IUnknown** interface, from which all other COM interfaces derive. This section describes only how to allocate memory for parameters that are not passed by value – *not* pointers to interfaces, but more mundane things like strings, pointers to structures, etc.

The OLE Memory Allocator

The COM Library provides an implementation of a memory allocator that is thread-safe (cannot cause problems in multi-threaded situations). A call to the API function **CoGetMalloc** provides a pointer to the OLE allocator, which is an implementation of the **IMalloc** interface. Whenever ownership of an allocated chunk of memory is passed through a COM interface or between a client and the COM library, you must use this allocator to allocate the memory. Allocation internal to an object can use any allocation scheme desired, but the COM memory allocator is a handy, efficient, and thread-safe allocator.

Allocation Rules for Non-by-value Parameters

Each parameter and the return value of a function can be classified into one of three groups: an **in** parameter, an **out** parameter (which includes return values), or an **in-out** parameter. In each class of parameter, the responsibility for allocating and freeing non-by-value parameters is the following:

- **in parameter** must be allocated and freed by the caller.
- **out parameter** must be allocated by the one called; freed by the caller.
- **in-out parameter** is initially allocated by the caller, then freed and re-allocated by the one called if necessary. As is true for out parameters, the caller is responsible for freeing the final returned value.

In the latter two cases, one piece of code allocates the memory and a different piece of code frees it. COM, again, requires the use of the COM allocator to ensure that the two pieces of code are using the same allocation methods.

Another area that needs special attention is the treatment of out and in-out parameters in failure conditions. If a function returns a status code which is a failure code, then, in general, the caller has no way to clean up the *out* or *in-out* parameters that are returned. This leads to a few additional rules:

out parameter

In error returns, the out parameters must *always* be reliably set to a value that will be cleaned up without any action by the caller. Further, it is the case that all out pointer parameters *must* explicitly be set to NULL. These are usually passed in a pointer-to-pointer parameter, but can also be passed as a member of a structure that the caller allocates and the called code fills. The most straightforward way to ensure this is (in part) to set these values to NULL on function entry. This rule is strong to promote more robust application interoperability.

in-out parameter

In error returns, all in-out parameters must either be left alone by the one called (and thus remaining at the value to which it was initialized by the caller; if the caller didn't initialize it, then it is an out-parameter, not an in-out-parameter) or be explicitly set as in the out-parameter error return case.

Remember that these memory management conventions for COM applications apply only across public interfaces and APIs – there is no requirement at all that memory allocation strictly internal to a COM application need be done using these mechanisms.

Debugging Memory Allocations

OLE provides an interface for developers to use to debug their memory allocations: the **IMallocSpy** interface. For each method in **IMalloc**, there are two methods in **IMallocSpy**, a "pre" method and a "post" method. Once a developer implements it and publishes it to the system, the system calls the **IMallocSpy** "pre" method just before the corresponding **IMalloc** method, effectively allowing the debug code to "spy" on the allocation operation, and calling the "post" method to release the spy.

For example, when OLE detects that the next call is a call to **IMalloc::Alloc**, it would call **IMallocSpy::PreAlloc**, executing whatever debug operations the developer wants during the **Alloc** execution, and then, when the **Alloc** call returns, call **IMallocSpy::PostAlloc** to release the spy and return control to the code.

Transparent Inter-object Communication

COM is designed to allow clients to communicate *transparently* with objects, regardless of where those objects are running – the same process, the same machine, or a different machine. This provides a *single programming model* for all types of objects, both object clients and object servers.

From a client's point of view, all objects are accessed through interface pointers. A pointer must be in-process. In fact, any call to an interface function always reaches *some* piece of in-process code first. If the object is in-process, the call reaches it directly, with no intervening system-infrastructure code. If the object is out-of-process, the call first reaches what is called a "proxy" object provided either by COM itself or by the object (if the implementor wishes). The proxy packages the interface pointer and generates the appropriate remote procedure call (or other communication mechanism in the case of custom generated proxies) to the other process or the other machine where the object implementation is located. This process of packaging pointers for transmission across process boundaries is called *marshaling*.

From a server's point of view, all calls to an object's interface functions are made through a pointer to that interface. Again, a pointer only has context in a single process, and so the caller must always be some piece of in-process code. If the object is in-process, the caller is the client itself. Otherwise, the caller is a "stub" object provided either by COM or by the object itself. The stub receives the remote procedure call (or other communication mechanism in the case of custom generated proxies) from the "proxy" in the client process, *unmarshals* the parameters, and calls the appropriate interface on the server object. From the points of view of both clients and servers, they always communicate directly with some other in-process code.

OLE provides an implementation of marshaling, referred to as *standard marshaling*. This implementation works very well for most objects, and greatly reduces programming requirements, making the marshaling process effectively transparent.

The clear separation of interface from implementation of OLE's process transparency can get, however, in the way for some situations when performance is of critical concern. The design of an interface that focuses on its function from the client's point of view can sometimes lead to design decisions that conflict with efficient implementation of that interface across a network. In cases like this, what is needed is not pure process transparency, but "process transparency, unless you need to care." COM provides this capability by allowing an object implementor to support *custom marshaling.* Standard marshaling is, in fact, an instance of custom marshaling–it is the default implementation used when an object does not require custom marshaling.

You can implement custom marshaling to allow an object to take different actions when used from across a network than it takes under local access – and it is completely transparent to the client. This architecture makes it possible to design client/object interfaces without regard to network performance issues, then, later, to address network performance issues without disrupting the established design.

COM does not specify how components are structured; it specifies how they interact. COM leaves the concern about the internal structure of a component to programming languages and development environments. Conversely, programming environments have no set standards for working with objects outside of the immediate application. C++, for example, works extremely well to manipulate objects inside an application, but has no support for working with objects outside the application. Generally, all other programming languages are the same in this regard. Therefore COM, through language-independent interfaces, picks up where programming languages leave off to provide the network-wide interoperability.

The double indirection of the *vtbl* structure means that the pointers in the table of function pointers do not need to point directly to the real implementation in the real object. This is the heart of process transparency.

For in-process servers, where the object is loaded directly into the client process, the function pointers in the table point directly to the actual implementation. In this case, a function call from the client to an interface method directly transfers execution control to the method. This cannot, however, work for local, let alone remote, objects because pointers to memory cannot be shared between processes. Nevertheless, the client must be able to call interface methods *as if it were calling the actual implementation*. Thus, the client uniformly transfers control to a method in some object by making the call.

A client always calls interface methods in some in-process object. If the actual object is local or remote, the call is made to a proxy object which then makes a remote procedure call to the actual object.

So what method is actually executed? The answer is that whenever there is a call to an out-of-process interface, each interface method is implemented by a proxy object. The proxy object is always an in-process object that acts on behalf of the object being called. This proxy object knows that the actual object is running in a local or remote server.

The proxy object packages up the function parameters in some data packets and generates an RPC call to the local or remote object. That packet is picked up by a stub object in the server's process on the local or a remote machine, which unpacks the parameters and makes the call to the real implementation of the method. When that function returns, the stub packages up any out-parameters and the return value, sends it back to the proxy, which unpacks them and returns them to the original client.

Thus, client and server always talk to each other as if everything was in-process. All calls from the client and all calls to the server are, at some point, in-process. But because the *vtbl* structure allows some agent, like COM, to intercept all function calls and all returns from functions, that agent can redirect those calls to an RPC call as necessary. Although in-process calls are, naturally, faster than out-of-process calls, the process differences are completely transparent to the client and server.

Concurrency and Synchronization Issues

Concurrency deals with things that can occur at the same time in the system. Whenever there is concurrency, synchronization is an issue. In OLE, there are two primary topics that raise issues in these categories: threading and calls that are not synchronous.

Processes and Threads

COM now permits you to create processes that contain more than one thread of execution, rather than the previous single-thread-per-process model, using a threading model called the *apartment model*. This multithreading model offers a message-based paradigm for dealing with multiple objects running concurrently. It allows you to write more efficient code by allowing a thread that is waiting for some time-consuming operation to allow another thread to be executed.

A thread of execution is the serial execution of code within a process. When there is only a single thread per process, each process is, effectively, a thread. Processes communicate through messages, using RPC to pass information between processes.

It is important to clarify what a COM object really is. Internal to any object server, a COM object is simply a pointer to data whose first piece of data is a pointer to a table of functions. Inside an object server there is really no difference between an **this** interface pointer and an instance of a C++ class derived from a pure-virtual base. Therefore, for a particular object server, there are really no OLE objects distinguishable from any other programming construct, as opposed to a window, which has operating-system-enforced behavior and existence. COM objects really only exist as the result of communication between two components.

A more exact way of stating this is that a COM object only appears as a result of a call to **CoCreateInstance**, being returned from another OLE interface, or as a programming construct being passed as an in-parameter to an interface. Thus, unless an application uses an OLE mechanism to create an instance of the interface, the only threading behavior is the Microsoft® Win32® base threading behavior. This provides a certain threading model with multiple threads of execution within a completely shared memory space, synchronized by various system implementations of semaphore primitives.

The big issue in programming with a multithreaded model is to ensure the code is *thread-safe*, so messages intended for a particular thread go only to that thread, and access to threads is protected somehow. The issues become more complicated when, for example, a single-threaded application loads a DLL server that creates multiple threads, and begins accessing an object of the single-threaded server from multiple threads at the same time. The apartment model addresses these issues.

The term *apartment* is used to indicate a logical grouping of related objects that all execute on the same thread, so must have synchronous execution. In addition, under the apartment model, while there can be multiple objects on a thread, an object cannot reside on more than one thread. This model means that calls to objects in other threads must be made on the thread that owns the object, in the same way that calls to objects in other processes must be made within the context of the owning process.

The inter-process and inter-thread models are similar. When it is necessary to pass an interface pointer to an object in another apartment (on another thread) within the same process, you use the same marshaling model that objects in different processes use to pass pointers across process boundaries. By getting a pointer to the standard marshaling object, you can marshal interface pointers across thread boundaries (between apartments) in the same way you do between processes.

Apartment threading rules are simple, but you must follow them carefully:

- Every object should live on only one thread (within a single apartment).
- You must call **OleInitialize** or **CoInitialize** to initialize each thread.
- All pointers to objects must be marshaled between apartments (ensuring that all calls to an object are made on its thread).
- Each apartment with objects in it must have a message queue to handle calls from other processes and apartments within the same process.
- Apartment-aware in-process objects must be registered as such in the system registry.
- Apartment-aware objects must write DLL entry points carefully.

While multiple objects can live on a single thread, no object can live on more than one thread. In addition, it is necessary to call **OleInitialize** or **CoInitialize** on each thread, because each thread requires the services of the OLE library

To facilitate marshaling interface pointers between apartments, OLE provides two helper functions. **CoMarshalInterThreadInterfaceInStream**, which marshals an interface into a stream object that is returned to the caller, and **CoGetInterfaceAndReleaseStream**, which unmarshals an interface pointer from a stream object and releases it. These functions are simply helpers that use **CoMarshalInterface** and **CoUnmarshalInterface**, which require the use of the flag MSHCTX_INPROC flag.

To set up a message queue for a thread, all that is necessary is that the thread's work function must have a **GetMessage**/**DispatchMessage** loop. If you use other synchronization primitives to communicate between threads, you can use the Win32 function **MsgWaitForMultipleObjects** to wait both for messages and for thread synchronization procedures (the Win32 SDK documentation for this function has an example of this sort of combination loop).

For thread-aware DLL-based or in-process objects, you need to set the threading model in the registry. The default model when you do not specify a threading model is single-thread-per-process. To specify a model, you add the **ThreadingModel** subkey to the **InprocServer32** key in the registration database.

To specify the apartment model, set **ThreadingModel=Apartment**. This permits in-process objects that can be created in multiple threads of a client application to interoperate correctly with older in-process objects that are not aware of multithreading. In-process objects not marked in this way are always created in the first initialized apartment (called the main apartment) in the process and other apartments automatically get proxied pointers to them.

You can also set **ThreadingModel=Both**. This subkey allows you to design a DLL that is thread-safe, but can run in both an apartment model server, and, when free-threading is supported in future releases of COM, a free-threading server.

DLLs that support instantiation of a class object must implement and export the functions **DllGetClassObject** and **DllCanUnloadNow**. When another object wants an instance of the class the DLL supports, a call to **CoGetClassObject** (either directly or through a call to **CoCreateInstance**) calls **DllGetClassObject** to get a pointer to its class object when the object is implemented in a DLL. As its name implies, **DllCanUnloadNow** is called to determine whether the DLL that implements it is in use, so the caller can safely unload it if it is not.

An apartment-aware object that calls **CoCreateInstance** on an apartment-aware object will call **DllGetClassObject** from its thread. **DllGetClassObject** should therefore be able to give away multiple class objects or a single thread-safe object (essentially just using **InterlockedIncrement**/**InterlockedDecrement** on their internal reference count). Calls to **CoFreeUnusedLibraries** from any thread always route through the main apartment's thread to call **DllCanUnloadNow**.

OLE also provides another helper function, **CoCreateFreeThreadedMarshaler**, which provides a pointer to an aggregatable object that implements code. This code allows direct access to an object's VTBL from multiple threads within the same process, but uses standard marshaling if a reference to the object is sent to another process.

Call Synchronization

OLE applications must be able to deal correctly with user input while processing one or more calls from OLE or the operating system. OLE calls between processes fall into three categories:

- Synchronous calls
- Asynchronous notifications
- Input-synchronized calls

Most of the communication that takes place within OLE is synchronous. When making *synchronous calls,* the caller waits for the reply before continuing and can receive incoming messages while waiting. OLE enters a modal loop to wait for the reply, receiving and dispatching other messages in a controlled manner.

When sending *asynchronous notifications,* the caller does not wait for the reply. OLE uses **PostMessage** or high-level events to send asynchronous notifications, depending on the platform. OLE defines five asynchronous methods:

- **IAdviseSink::OnDataChange**
- **IAdviseSink::OnViewChange**
- **IAdviseSink::OnRename**
- **IAdviseSink::OnSave**
- **IAdviseSink::OnClose**

While OLE is processing an asynchronous call, synchronous calls cannot be made. For example, a container application's implementation of **IAdviseSink::OnDataChange** cannot contain a call to **IPersistStorage::Save**.

When making *input-synchronized calls,* the object called must complete the call before yielding control. This ensures that focus management works correctly and that data entered by the user is processed appropriately. These calls are made by OLE through the Windows **SendMessage** function, without entering a modal loop. While processing an input-synchronized call, the object called must not call any function or method (including synchronous methods) that might yield control.

The following methods are input synchronized:

- **IOleWindow::GetWindow**
- **IOleInPlaceActiveObject::OnFrameWindowActivate**
- **IOleInPlaceActiveObject::OnDocWindowActivate**
- **IOleInPlaceActiveObject::ResizeBorder**
- **IOleInPlaceUIWindow::GetBorder**
- **IOleInPlaceUIWindow::RequestBorderSpace**
- **IOleInPlaceUIWindow::SetBorderSpace**
- **IOleInPlaceFrame::SetMenu**
- **IOleInPlaceFrame::SetStatusText**
- **IOleInPlaceObject::SetObjectRects**

To minimize problems that can arise from asynchronous message processing, the majority of OLE method calls are synchronous. With synchronous communication, there is no need for special code to dispatch and handle incoming messages. When an application makes a synchronous method call, OLE enters a modal wait loop that handles the required replies and dispatches incoming messages to applications capable of processing them.

OLE manages method calls by assigning an identifier called a *logical thread ID*. A new one is assigned when a user selects a menu command or when the application initiates a new OLE operation.

Subsequent calls that relate to the initial OLE call are assigned the same logical thread ID as the initial call.

Monikers

A moniker in COM is not only, as the name implies, a way to identify an object. A moniker is also implemented as an object. This object provides services allowing a component to obtain a pointer to the object identified by the moniker. This process is referred to as *binding*.

Monikers are objects that implement the **IMoniker** interface, and are generally implemented in DLLs as component objects. There are two ways of viewing the use of monikers: as a *moniker client*, a component that uses a moniker to get a pointer to another object, and as a *moniker provider*, a component that supplies monikers identifying its objects to moniker clients.

OLE uses monikers, for example, to identify, connect to, and run OLE Compound Document link objects. In this case, the link source acts as the moniker provider, and the container holding the link object acts as the moniker client.

Moniker Clients

Moniker clients must start by getting a moniker. There are several ways for a moniker client to get a moniker. For example, in OLE Compound Documents, when the end-user creates a linked item (either using Insert Object dialog, the clipboard, or drag-and drop), a moniker is embedded as part of the linked item. In that case, the programmer has minimal contact with monikers. Programmatically, if you have an interface pointer to an object that implements the **IMoniker** interface, you can use that to get a moniker, and there are methods on other interfaces that are defined to return monikers.

There are different kinds of monikers, which are used to identify different kinds of objects, but to a moniker client, all monikers look the same. A moniker client simply calls **IMoniker::BindToObject** on a moniker and gets an interface pointer to the object that the moniker identifies. No matter whether the moniker identifies an object as large as an entire spreadsheet or as small as a single cell within a spreadsheet, calling **IMoniker::BindToObject** will return a pointer to that object. If the object is already running, **IMoniker::BindToObject** will find it in memory. If the object is stored passively on disk, **IMoniker::BindToObject** will locate a server for that object, run the server, and have the server bring the object into the running state. All the details of the binding process are hidden from the moniker client. Thus, for a moniker client, using the moniker is very simple.

Moniker Providers

In general, a component should be a moniker provider when it allows access to one of its objects, while still controlling the object's storage. If a component is going to hand out monikers that identify its objects, it must be capable of performing the following tasks:

- On request, create a moniker that identifies an object.
- Enable the moniker to be bound when a client calls **IMoniker::BindToObject** on it.

A moniker provider must create a moniker of an appropriate *moniker class* to identify an object. The moniker class refers to a specific implementation of the **IMoniker** interface that defines the type of moniker created. While you can create a new moniker class through an implementation of the **IMoniker** interface, it is frequently unnecessary because OLE provides five different moniker classes, each with its own CLSID:

- File moniker
- Composite Moniker
- Item Moniker
- Anti-moniker
- Pointer moniker

The file, composite, and item monikers are the most frequently used monikers, as they can be used to make nearly any object in any location. Anti- and pointer monikers are primarily used inside OLE, but have some application in implementing custom monikers.

File Monikers

File monikers are the simplest moniker class. File monikers can be used to identify any object that is stored in its own file. A file moniker acts as a wrapper for the path name the native file system assigns to the file. Calling **IMoniker::BindToObject** for this moniker would cause this object to be activated and then would return an interface pointer to the object. The source of the object named by the moniker must provide an implementation of the **IPersistFile** interface to support binding a file moniker. File monikers can represent either a complete or a relative path.

For example, the file moniker for a spreadsheet object stored as the file C:\WORK\MYSHEET.XLS would contain information equivalent to that path name. The moniker would not necessarily consist of the same string, however. The string is just its *display name*, a representation of the moniker's contents that is meaningful to an end user. The display name, which is available through the **IMoniker::GetDisplayName** method, is used only when displaying a moniker to an end-user. This method gets the display name for any of the moniker classes. Internally, the moniker may store the same information in a format that is more efficient for performing moniker operations, but isn't meaningful to users. Then, when this same object is bound through a call to the **BindToObject** method. the object would be activated, probably by loading the file into the spreadsheet.

OLE offers moniker providers the helper API **CreateFileMoniker** that creates a file moniker object and returns its pointer to the provider.

Composite Monikers

One of the most useful features of monikers is that you can concatenate or *compose* monikers together. A composite moniker is a moniker that is a composition of other monikers, and can determine the relation between the parts. This lets you assemble the complete path to an object given two or more monikers that are the equivalent of partial paths. You can compose monikers of the same class (like two file monikers) or of different classes (like a file moniker and an item moniker). If you were to write your own moniker class, you could also compose your monikers with file or item monikers. The basic advantage of a composite is that it gives you one piece of code to implement every possible moniker that is a combination of simpler monikers.   That reduces tremendously the need for specific custom moniker classes.

Because monikers of different classes can be composed with one another, monikers provide the ability to join multiple namespaces. The file system defines a common namespace for objects stored as files because all applications understand a file-system path name. Similarly, a container object also defines a private namespace for the objects that it contains, because no container understands the names generated by another container. Monikers allow these name spaces to be joined because file monikers and item monikers can be composed. A moniker client can search the namespace for *all objects* using a single mechanism. The client simply calls **IMoniker::BindToObject** on the moniker, and the moniker code handles the rest. A call to **IMoniker::GetDisplayName** on a composite creates a name using the concatenation of all the individual monikers' display names.

Furthermore, because you can write your own moniker class, moniker composition allows you to add customized extensions to the namespace for objects.

Sometimes two monikers of specific classes can be combined in a special way. For example, a file moniker representing an incomplete path and another file moniker representing a relative path can be combined to form a single file moniker representing the complete path. For example, the file monikers **c:\work\art** could be composed with the relative file moniker **..\backup\myfile.doc** to equal **c:\work\backup\myfile.doc**. This is an example of "non-generic" composition.

"Generic" composition, on the other hand, permits the connection of any two monikers, no matter what their classes. For example, you could compose an item moniker onto a file moniker, though not, of course, the other way around.

Because a non-generic composition depends on the class of the monikers involved, its details are defined by a the implementation of a particular moniker class. You can define new types of non-generic compositions if you write a new moniker class. By contrast, generic compositions are defined by OLE. Monikers created as a result of generic composition are called generic composite monikers.

These three classes -- file monikers, item monikers, and generic composite monikers -- all work together, and they are the most commonly used classes of monikers.

Moniker clients should call **IMoniker::ComposeWith** to create a composite on moniker with another. The moniker it is called on internally decides whether it can do a generic or non-generic composition. If the moniker implementation determines that a generic composition is usable, OLE provides the **CreateGenericComposite** API function to facilitate this.

Item Monikers

Another OLE-implemented moniker class is the item moniker, which can be used to identify an object contained in another object. One type of contained object is an OLE object embedded in a compound document. A compound document could identify the embedded objects it contains by assigning each one an arbitrary name, such as "embedobj1," "embedobj2," and so forth. Another type of contained object is a user selection in a document, such as a range of cells in a spreadsheet or a range of characters in a text document. An object that consists of a selection is called a *pseudo-object* because it isn't treated as a distinct object until a user marks the selection. A spreadsheet might identify a cell range using a name such as "1A:7F", while a word processing document might identify a range of characters using the name of a bookmark.

An item moniker is useful primarily when concatenated -- or "composed" -- with another moniker, one that identifies the container. An item moniker is usually created, then composed onto (for example) a file moniker to create the equivalent of a complete path to the object. For example, you can compose the file moniker "C:\work\report.doc" (which identifies the container object) with the item moniker "embedobj1" (which identifies an object within the container) to form the moniker "C:\work\report.doc\embedobj1," which uniquely identifies a particular object within a particular file. You can also concatenate additional item monikers to identify deeply nested objects. For example, if "embedobj1" is the name of a spreadsheet object, then to identify a certain range of cells in that spreadsheet object you could append another item moniker to create a moniker that would be the equivalent of "C:\work\report.doc\embedobj1\1A:7F."

When combined with a file moniker, an item moniker forms a complete path. Item monikers thus extend the notion of path names beyond the file system, defining path names to identify individual objects, not just files.

There is a significant difference between an item moniker and a file moniker. The path contained in a file moniker is meaningful to anyone who understands the file system, while the partial path contained in an item moniker is meaningful only to *a particular container*. Everyone knows what "c:\work\report.doc" refers to, but only one particular container object knows what "1A:7F" refers to. One container cannot interpret an item moniker created by another application; the only container that knows which object is referred to by an item moniker is the container that assigned the item moniker to the object in the first place. For this reason, the source of the object named by the combination of a file and item moniker must not only implement **IPersistFile** to facilitate binding the file moniker, but also **IOleItemContainer** to facilitate resolving the name of the item moniker into the appropriate object, in the context of a file

The advantage of monikers is that someone using a moniker to locate an object doesn't need to understand the name contained within the item moniker, as long as the item moniker is part of a composite. Generally, it would not make sense for an item moniker to exist on its own. Instead, you would compose an item moniker onto a file moniker. You would then call **IMoniker::BindToObject** on the composite, which binds the individual monikers within it, interpreting the names.

To create an item moniker object and return its pointer to the moniker provider, OLE provides the helper API **CreateItemMoniker**. This function creates an item moniker object and returns its pointer to the provider.

Anti-monikers

OLE provides an implementation of a special type of moniker called an *anti-moniker*. You use this moniker in the creation of new moniker classes. You use it as the inverse of the moniker that it is composed onto, effectively canceling that moniker, in much the same way that the ".." operator moves up a directory level in a file system command.

It is necessary to have an anti-moniker available, because once a composite moniker is created, it is not possible to delete parts of the moniker if, for example, an object moves. Instead, you use an anti-moniker to remove one or more entries from a composite moniker.

Anti-monikers are a moniker class explicitly intended for use as an inverse. COM defines an API function named **CreateAntiMoniker**, which returns an anti-moniker. You generally use this function to implement the **IMoniker::Inverse** method.

An anti-moniker is *only* an inverse for those types of monikers that are implemented to treat anti-monikers as an inverse. For example, if you want to remove the last piece of a composite moniker, you should not create an anti-moniker and compose it to the end of the composite. You cannot be sure that the last piece of the composite considers an anti-moniker to be its inverse. Instead, you should call **IMoniker::Enum** on the composite moniker, specifying **FALSE** as the first parameter. This creates an enumerator that returns the component monikers in reverse order. Use the enumerator to retrieve the last piece of the composite, and call **IMoniker::Inverse** on that moniker. The moniker returned by **IMoniker::Inverse** is what you need to remove the last piece of the composite.

Pointer Monikers

A pointer moniker identifies an object that can exist only in the active or running state. This differs from other classes of monikers, which identify objects that can exist either in the passive or active state.

Suppose, for example, an application has an object that has no persistent representation. Normally, if a client of your application needs access to that object, you could simply pass the client a pointer to the object. However, suppose your client is expecting a moniker. The object cannot be identified with a file moniker, since it isn't stored in a file, nor with an item moniker, since it isn't contained in another object.

Instead, your application can create a pointer moniker, which is a moniker that simply contains a pointer internally, and pass that to the client. The client can treat this moniker like any other. However, when the client calls **IMoniker::BindToObject** on the pointer moniker, the moniker code does not check the Running Object Table (ROT) or load anything from storage. Instead, the moniker code simply calls **IUnknown::QueryInterface** on the pointer stored inside the moniker.

Pointer monikers allow objects that exist only in the active or running state to participate in moniker operations and be used by moniker clients. One important difference between pointer monikers and other classes of monikers is that pointer monikers cannot be saved to persistent storage. If you do, calling the **IMoniker::Save** method returns an error. This means that pointer monikers are useful only in specialized situations. You can use the **CreatePointerMoniker** API function if you need to use a pointer moniker.

Error Handling

At the source code level, all error values consist of three parts, separated by underscores. The first part is the prefix that identifies the facility associated with the error, the second part is E for error, and the third part is a string that describes the actual condition. For example, STG_E_MEDIUMFULL is returned when there is no space left on a hard disk. The STG prefix indicates the storage facility, the E indicates that the status code represents an error, and the MEDIUMFULL provides specific information about the error. Many of the values that you might want to return from an interface method or function are defined in winerror.h.

Success, warning, and error values are returned using a 32-bit number known as a result handle, or HRESULT. HRESULTs are really not handles to anything, but merely 32-bit values with several fields encoded in the value. A zero result indicates success and a non-zero result indicates failure.

HRESULTs work differently depending on the platform you are using. On 16-bit platforms, an HRESULT is generated from a 32-bit value known as a status code, or SCODE. On 32-bit platforms, an HRESULT is the same as an SCODE; they are synonymous. In fact, the SCODE is no longer used. 32-bit OLE uses only HRESULTs.

Structure of OLE Error Codes

SCODES on 16-bit platforms are divided into four fields: a severity code, a context field, a facility field, and an error code. The format of an SCODE on a 16-bit platform is shown below; the numbers indicate bit positions:

HRESULTs on 32-bit platforms have the following format:

The severity code in the 16-bit SCODE and the high order bit in the HRESULT indicates whether the return value represents success or failure. If set to zero, SEVERITY_SUCCESS, the value indicates success. If set to 1, SEVERITY_ERROR, it indicates failure.

In the 16-bit version of the SCODE, the context field is reserved; this field does not exist in the 32-bit version. The R, C, N, and r bits are reserved in both.

The facility field in both versions indicates the area of responsibility for the error. There are currently five facilities: FACILITY_NULL, FACILITY_ITF, FACILITY_DISPATCH, FACILITY_RPC, and FACILITY_STORAGE. If new facilities are necessary, Microsoft allocates them because they need to be unique. Most SCODEs and HRESULTs set the facility field to FACILITY_ITF, indicating an interface method error. The following table describes the various facility fields:

Facility	**Description**
FACILITY_NULL	For broadly applicable common status codes such as S_OK. This facility code has a value of zero.
FACILITY_ITF	For most status codes returned from interface methods, value is defined by the interface. That is, two SCODEs or HRESULTs with exactly the same 32-bit value returned from two different interfaces might have different meanings. This facility has a value of 4.
FACILITY_DISPATCH	For late binding **IDispatch** interface errors. This facility has a value of 2.
FACILITY_RPC	For status codes returned from remote procedure calls. This facility has a value of 1.
FACILITY_STORAGE	For status codes returned from **IStorage** or **IStream** method calls relating to structured storage. Status codes whose code (lower 16 bits) value is in the range of DOS error codes (that is, less than 256) have the same meaning as the corresponding DOS error. This facility has a value of 3.
FACILITY_WINDOWS	Used for additional error codes from Microsoft-defined interfaces.
FACILITY_WIN32	Used to provide a means of handling error codes from functions in the Win32 API as an HRESULT. Error codes in 16-bit OLE that duplicated Win32 error codes have also been changed to FACILITY_WIN32.

The code field is a unique number that is assigned to represent the error or warning.

By convention, **HRESULTs** generally have names in the following form:

```
<Facility>_<Severity>_<Reason>
```

where <*Facility*> is either the facility name or some other distinguishing identifier, <*Severity*> is a single letter, S or E, that indicates the severity of the error, and <*Reason*> is an identifier that describes the meaning of the code. For example, the status code STG_E_FILENOTFOUND indicates a storage-related error has occurred; specifically, a requested file does not exist. Status codes from FACILITY_NULL omit the <*Facility*>_ prefix.

Error codes are defined within the context of an interface implementation. Once defined, success codes cannot be changed or new success codes added. However, new failure codes can be written since they generally only provide hints at what might have gone wrong. Microsoft reserves the right to define new failure codes (but not success codes) for the interfaces described in this book in FACILITY_ITF or in new facilities.

Codes in FACILITY_ITF

These **HRESULTSs** with facilities such as FACILITY_NULL and FACILITY_RPC have universal meaning because they are defined at a single source: Microsoft. However, **HRESULTs** in FACILITY_ITF are determined by the interface method (or function) from which they are returned. That is, the same 32-bit value in FACILITY_ITF returned from two different interface methods might have different meanings.

The reason **HRESULTs** in FACILITY_ITF can have different meanings in different interfaces is that **HRESULTs** are kept to an efficient data type size of 32 bits. Unfortunately 32 bits is not large enough for the development of an allocation system for error codes that avoid conflict between codes allocated by different non-communicating programmers at different times in different places (unlike the handling of interface IDs and CLSIDs). As a result, the 32-bit **HRESULT** is structured in a way that Microsoft can define some universally-defined error codes, while allowing other programmers to define new error codes without fear of conflict. The status code convention is as follows:

1. Status codes in facilities *other than* FACILITY_ITF can only be defined by Microsoft.
2. Status codes in facility FACILITY_ITF are defined solely by the developer of the interface or API that returns the status code. To avoid conflicting error codes, whoever defines the interface is responsible for coordinating and publishing the FACILITY_ITF status codes associated with that interface.

All the OLE-defined FACILITY_ITF codes have a code value in the range of 0x0000 - 0x01FF. While it is legal to use any codes in FACILITY_ITF, it is recommended that only code values in the range of 0x0200 – 0xFFFF be used. This recommendation is made as a means of reducing confusion with any OLE-defined errors.

It is also recommended that developers define new functions and interfaces to return error codes as defined by OLE and in facilities other than FACILITY_ITF. In particular, interfaces that have any chance of being remoted using RPC in the future should define the FACILITY_RPC codes as legal. E_UNEXPECTED is a specific error code that most developers will want to make universally legal.

Using Macros for Error Handling

OLE defines a number of macros that make it easier to work with SCODEs on 16-bit platforms and HRESULTs on both platforms. Some of the macros and functions below provide conversion between the two data types and are quite useful in code that runs only on 16-bit platforms, code that runs on both 16-bit and 32-bit platforms, and 16-bit code that is being ported to a 32-bit platform. These same macros are meaningless in 32-bit environments and are available in order to provide compatibility and make porting easier. Newly written code should use the **HRESULT** macros and functions.

Error handling macros are listed below. Refer to *The OLE Programmer's Reference* for a complete description of each.

Macro	Description
GetScode	(Obsolete) Returns an SCODE given an HRESULT.
ResultFromScode	(Obsolete) Returns an HRESULT given an SCODE.
PropagateResult	(Obsolete) Generates an HRESULT to return to a function in cases where an error is being returned from an internally called function.
MAKE_HRESULT	Returns an HRESULT given an SCODE that represents an error.
MAKE_SCODE	Returns an SCODE given an HRESULT.
HRESULT_CODE	Extracts the error code part of the HRESULT.
HRESULT_FACILITY	Extracts the facility from the HRESULT.
HRESULT_SEVERITY	Extracts the severity bit from the SEVERITY.
SCODE_CODE	Extracts the error code part of the SCODE.
SCODE_FACILITY	Extracts the facility from the SCODE.
SCODE_SEVERITY	Extracts the severity field from the SCODE.
SUCCEEDED	Tests the severity of the SCODE or HRESULT - returns TRUE if the severity is zero and FALSE if it is one.
FAILED	Tests the severity of the SCODE or HRESULT - returns TRUE if the severity is one and FALSE if it is zero.
IS_ERROR	Provides a generic test for errors on any status value.
FACILITY_NT_BIT	Defines bits so macros are guaranteed to work.
RESULT_FROM_WIN32	Maps a Win32 error value into an HRESULT. This assumes that Win32 errors fall in the range -32k to 32k.
HRESULT_FORM_NT	Maps an NT status value into an HRESULT.

Note Calling **MAKE_HRESULT** for S_OK verification carries a performance penalty. You should not routinely use **MAKE_HRESULT** for successful results.

Error Handling Strategies

Because interface methods are virtual, it is not possible to know, as a caller, the full set of values that may be returned from any one call. One implementation of a method may return five values; another may return eight. The *OLE Programmer's Reference* lists common values that may be returned for each method; these are the values that you must check for and handle in your code because they have special meanings. Other values may be returned, but since they are not meaningful, you do not need to write special code to handle them. A simple check for zero or not zero is adequate.

HRESULTs

The values of some **HRESULT**s have been changed in 32-bit OLE to eliminate all duplication and overlapping with Win32 error codes. Those that duplicate Win32 error codes have been changed to FACILITY_WIN32 and those that overlap remain in FACILITY_NULL. Following is a list of their 32-bit values:

HRESULT	Value
E_UNEXPECTED	0x8000FFFF
E_NOTIMPL	0x80004001
E_OUTOFMEMORY	0x8007000E
E_INVALIDARG	0x80070057
E_NOINTERFACE	0x80004002
E_POINTER	0x80004003
E_HANDLE	0x80070006
E_ABORT	0x80004004
E_FAIL	0x80004005
E_ACCESSDENIED	0x80070005

Handling Error Information

Keep in mind that it is legal to return a status code only from the implementation of an interface method sanctioned as legally returnable. Failure to observe this rule invites the possibility of conflict between returned error code values and those sanctioned by the application. In particular, pay attention to this potential problem when propagating error codes from internally-Qcalled functions.

Applications that call interfaces should treat any unknown returned error code (as opposed to a success code) as synonymous with E_UNEXPECTED. This practice of handling unknown error codes is required by clients of the OLE-defined interfaces and APIs. Because typical programming practice is to handle a few specific error codes in detail and treat the rest generically, this requirement of handling unexpected or unknown error codes is easily met.

The following code sample shows the recommended way of handling unknown errors:

```
HRESULT hrErr;
hrErr = xxMethod();

switch (GetScode(hrErr))  {
     case NOERROR:
       //success
       break;

     case x1:
       .
       .
       break;

     case x2:
       .
       .
       break;

     case E_UNEXPECTED:
     default:
       //general failure
       break;
}
```

The following error check is often used with those routines that don't return anything special (other than S_OK or some unexpected error):

```
if (xxMethod() == NOERROR)
     //success
else
     //general failure;
```

Compound Documents

OLE compound documents enable users working within a single application to manipulate data written in various formats and derived from multiple sources. For example, a user might insert into a word processing document a graph created in a second application and a sound object created in a third application. Activating the graph causes the second application to load its user interface, or at least that part containing tools necessary to edit the object. Activating the sound object causes the third application to play it. In both cases, a user is able to manipulate data from external sources from within the context of a single document.

OLE compound document technology rests on a foundation consisting of COM, structured storage, and .uniform data transfer. As summarized below, each of these core technologies plays a critical role in OLE compound documents:

COM

A compound document object is essentially a COM object that can be embedded in, or linked to, an existing document. As a COM object, a compound document object exposes the **IUnknown** interface, through which clients can obtain pointers to its other interfaces, including several, such as **IOleObject**, **IOleLink**, **and** **IViewObject2**, that provide special features unique to compound document objects.

Structured Storage

A compound document object must implement the **IPersistStorage** or, optionally, **IPersistStream** interfaces to manage its own storage. A container used to create compound documents must supply the **IStorage** interface, through which objects store and retrieve data. Containers almost always provide instances of IStorage obtained from OLE's Compound Files implementation. Containers must also use an object's IPersistStorage and/or IPersistStream interfaces.

Uniform Data Transfer

Applications that support compound documents must implement **IDataObject** because embedded objects and linked objects begin as data that has been transferred using special OLE clipboard formats, rather than standard Microsoft® Windows® clipboard formats. In other words, formatting data as an embedded or linked object is simply one more option provided by OLE's uniform data transfer model.

OLE's compound document technology benefits both software developers and users alike. Instead of feeling obligated to cram every conceivable feature into a single application, software developers are now free, if they like, to develop smaller, more focused applications that rely on other applications to supply additional features. In cases where a software developer decides to provide an application with capabilities beyond its core features, the developer can implement these additional services as separate DLLs, which are loaded into memory only when their services are required. Users benefit from smaller, faster, more capable software that they can mix and match as needed, manipulating all required components from within a single master document.

Containers and Servers

Compound document applications are of two basic types: container applications and server applications. OLE container applications provide users with the ability to create, edit, save, and retrieve compound documents. OLE server applications provide users with the means to create documents and other data representations that can be contained as either links or embeddings in container applications. An OLE application can be a container application, a server application, or both.

OLE server applications also differ in whether they are implemented as *in-process servers* or *local servers*. An in-process server is a dynamic link library (DLL) that runs in the container application's process space. You can run an in-process server *only* from within the container application.

Note Future releases of OLE will enable linking and embedding across machine boundaries, so that a container application on one computer will be able to use a compound document object provided by a *remote server* running on another computer. From a container application's point of view, any OLE server application that runs in its own process space, whether on the same computer or a remote machine, is an *out-of-process server.*

Linking and Embedding

Users can create two types of compound-document objects: *linked* or *embedded*. The difference between the two types lies in how and where the object's source data is stored. Where the object resides affects, in turn, the object's portability and methods of activation, how data updates are performed, and the size and structure of its container.

Linked Objects

When a link to an object is inserted in a compound document, the source data, or *link source*, continues to reside wherever it was initially created, usually in another document. The compound document contains only a reference, or *link*, to the actual data stored at the link source, along with information about how to present that data to the user. Currently, moving a link source breaks the link unless both source and client maintain their relative positions in the directory tree. Eventually, link tracking by Windows will allow a link source to be moved independently of its client without breaking the link.

Activating a link runs the link source's server application, which the user requires in order to edit or otherwise manipulate the link data. Linking keeps the size of a compound document small. It is also useful when the data source is maintained by someone else and must be shared among many users. If the person maintaining the link source changes the data, the change is automatically updated in all documents containing a link to that data. In addition to creating simple links, users can nest links and combine linked and embedded objects to create complex documents.

Embedded Objects

An embedded object is physically stored in the compound document, along with all the information needed to manage the object. In other words, the embedded object is actually a part of the compound document in which it resides. This arrangement has a couple of disadvantages. First, a compound document containing embedded objects will be larger than one containing the same objects as links. Second, changes made to the source of an embedded object will not be automatically replicated in the embedded copy, and changes in the source will not be reflected in the source, as they are with a link.

Still, for certain purposes, embedding offers several advantages over links. First, users can transfer compound documents with embedded objects to other computers, or other locations on the same computer, without breaking a link. Second, users can edit embedded objects without changing the content of the original. Sometimes, this separation is precisely what is required. Third, embedded objects can be activated in place, meaning that the user can edit or otherwise manipulate the object without having to work in a separate window from that of the object's container. Instead, when the object is activated, the container application's user interface changes to expose those tools that are necessary to manage or modify the object.

Object Handlers

If an OLE server application is a local server, meaning that it runs in its own process space, communication between container and server must occur across process boundaries. Since this process is expensive, OLE relies on a surrogate object loaded into the container's process space to act on behalf of a local server application. This surrogate object, known as an *object handler* services container requests that do not require the attention of the server application, such as requests for drawing. When a container requests something that the object handler cannot provide, the handler communicates with the server application using COM's out-of-process communication mechanism.

An object handler is unique to an object class. When you create an instance of a handler for one class, you cannot use it for another. When used for a compound document, the object handler implements the container-side data structures when objects of a particular class are accessed remotely.

OLE provides a default object handler that local server applications can use. For applications that require special behaviors, developers can implement a custom handler that either replaces the default handler or uses it to provide certain default behaviors.

An object handler is a DLL containing several interacting components. These components include remoting pieces to manage communication between the handler and its server application, a cache for storing an object's data, along with information on how that data should be formatted and displayed, and a controlling object that coordinates the activities of the DLL's other components. In addition, if an object is a link, the DLL also includes a linking component, or *linked object,* which keeps track of the name and location of the link source.

The *cache* contains data and presentation information sufficient for the handler to display a loaded, but not running, object in its container. OLE provides an implementation of the cache used by OLE's default object handler and the link object. The cache stores data in formats needed by the object handler to satisfy container draw requests. When an object's data changes, the object sends a notification to the cache so that an update can occur. For more information on the cache, see "View Caching" later in this chapter.

The Default Handler and Custom Handlers

The default handler, an implementation provided by OLE, is used by most applications as the handler. An application implements a custom handler when the default handler's capabilities are insufficient. A custom handler can either completely replace the default handler or use parts of the functionality it provides where appropriate. In the latter case, the application handler is implemented as an aggregate object composed of a new control object and the default handler. Combination application/default handlers are also known as *in-process handlers*. The *remoting handler* is used for objects that are not assigned a CLSID in the system registry or that have no specified handler. All that is required from a handler for these types of objects is that they pass information across the process boundary.

In-Process Servers

If you implement an OLE server application as an in-process server – a DLL running in the process space of the container application – rather than as a local server – an EXE running in its own process space – communication between container and server is simplified because communication between the two can take the form of normal function calls. Remote procedure calls are not required because the two applications run in the same process space. As you would expect, the objects that manage the marshaling of parameters are also unnecessary , although they may be aggregated within the DLL without interfering with the communication between container and server.

When an OLE server application is implemented as an in-process server, a separate object handler is not required because the server itself lives in the client's process space. The main difference between an in-process server and object handler is that the server is able to manage the object in a running state while the handler cannot. One consequence of this difference is that a server must provide a user interface for manipulating the running object, while a handler delegates this requirement to the object's server. In creating an in-process server, you can aggregate on the OLE default handler, letting it handle basic chores, such as display, storage, and notifications while you implement only those services that the handler either does not provide or does not implement in the way you require

Advantages

The advantages of implementing your application as an in-process server are speed and combining some the advantages of an object handler and a local server. In-process servers are faster than local servers for several reasons. First, because they are smaller and run in the process space of the container application, they load more quickly. Second, they are optimized to perform certain tasks. Third, communication between container and server does not rely on remote procedure calls.

Disadvantages

In-process servers provide the speed and size advantage of an object handler with the editing capability of a local server. So why would you ever choose to implement your OLE application as a local server rather than an in-process server? There are several reasons:

- Security. Only a local server has its address space isolated from that of the client. An in-process server shares the address space and process context of the client and can therefore be less robust in the face of faults or malicious programming.
- Granularity. A local server can host multiple instances of its object across many different clients, sharing server state between objects in multiple clients in ways that would be difficult or impossible if implemented as an in-process server, which is simply a DLL loaded into each client.
- Compatibility. If you choose to implement an in-process server, you relinquish compatibility with OLE 1, which does not support such servers. This will not be a consideration for many developers, but if it is, then it is of critical concern.
- Inability to support links. An in-process server cannot serve as a link source. Since a DLL cannot run by itself, it cannot create a file object to be linked to.

Despite these disadvantages, an in-process server can be an excellent choice for its speed and size – if it fits all your other requirements.

Linked Objects

Linked objects, like embedded objects, rely on an object handler to communicate with server applications.The linked object itself, however, manages the naming and tracking of link sources.The linked object acts like an in-process server. For example, when activated, a linked object locates and launches the OLE server application that is the link source.

A linked object's handler is made up of two main components: the handler component and the linking component. The handler component contains the controlling and remoting pieces and functions much like a handler for an embedded object. The linking component has its own controller and cache and provides access to the object's structured storage. The linking component's controller supports source naming through the use of monikers, and binding, the process of locating and running the link source. (For more information on monikers and binding, see "The Component Object Model.")

When a user initially creates a linked object or loads an existing one from storage, the container loads an instance of the linking component into memory, along with the object handler. The linking component supplies interfaces – most notably **IOleLink** – that identify the object as a link and enable it to manage the naming, tracking, and updating of its link source.

By implementing the **IOleLink** interface, a linked object provides its container with functions that support linking. Only linked objects implement **IOleLink**, and by querying for this interface a container can determine whether a given object is embedded or linked. The most important function provided by **IOleLink** enables a container to binding to the source of the linked object, that is, to activate the connection to the document that stores the linked object's native data. **IOleLink** also defines functions for managing information about the linked object, such as cached presentation data and the location of the link source.

When a compound document containing a linked object is saved, the link's data is saved with the link source, not with the container. Only information about its name and location is saved along with the compound document. This behavior is in contrast to that of an embedded object, whose data is stored along with that of its container.

Container applications can provide information about their embedded objects such that the latter, or portions thereof, can act as link sources. By implementing support for linking to your container's embedded objects, you make nested embeddings possible, relieving the user of having to track down the originals of every embedding object to which a link is desired. For example, if a user wants to embed a Microsoft® Excel worksheet in Microsoft® Word, and the worksheet contains a bitmap created in Paintbrush™, the user can link to a copy of the bitmap contained in the worksheet rather than the original.

Notifications

Notifications are callbacks generated by an object when it detects a change in its name, state, data, or presentation. Containers and other clients require notifications to respond appropriately to these changes. A container registers to receive notifications by setting up an advisory connection to an object of interest. Other interested clients can do the same. The container also creates an advisory sink to receive the notifications. Using the connection established by the container, an object experiencing a change notifies the advisory sink. Upon receiving a notification, the container takes whatever action has been defined for the type of change that has occurred.

Types of Notifications

Notifications fall into three groups: compound document, data, and view. An object sends compound document notifications in response to being renamed, saved, closed or, in the case of a link, having its link source renamed. As you would expect, objects send data notifications in response to changes in their data and send view notifications in response to changes in their presentation. Container applications must register separately for each of these notification types, but all can be handled by a single advisory sink.

All containers, the object handler, and the linked object register for compound document notifications. The typical container also registers for view notifications. Data notifications are usually sent to both the linked object and object handler. A special purpose container, such as one that renders the data itself, might benefit from receiving data notifications instead of view notifications. For example, an embedded chart container with a link to a table can register for data notifications. Because a change to the table affects the chart, the receipt of a data notification would direct the container to get the new tabular data.

How Notifications Work

Notifications originate in the object application and flow to the container by way of the object handler. If the object is a linked object, the linked object intercepts the notifications from the object handler and notifies the container directly.

An object application must manage registration requests, keeping track of where to send which notifications and sending those notifications when appropriate. OLE provides two component objects to simplify this task: the OleAdviseHolder for compound document notifications and the DataAdviseHolder for data notifications.

Object applications determine the conditions that prompt the sending of each specific notification and the frequency with which each notification should be sent. When it is appropriate for multiple notifications to be sent, it does not matter which notification is sent first; they can be sent in any order.

The timing of notifications affects the performance and coordination between an object application and its containers. Whereas notifications sent too frequently slow processing, notifications sent too infrequently result in an out-of-sync container. Notification frequency can be compared with the rate at which an application repaints. Therefore, using similar logic for the timing of notifications (as is used for repainting) is wise.

Compound Document Interfaces

The following tables list the interfaces implemented by OLE containers, OLE servers, and compound document objects. The required interfaces *must* be implemented on the components for which they are listed. For example, containers must implement the **IOleClientSite** and **IAdviseSink** interfaces. All other features are optional. If you want to include a particular feature in your application, however, you must implement the interfaces shown for that feature in the table below. All other interfaces are required only if you are including a particular feature. For example, if you want your application to do message filtering (recommended), you must implement **IMessageFilter**.

The following table lists required and optional behaviors for OLE containers and which interfaces you must implement for each.

OLE Containers

Behavior	Interfaces
Required Behaviors	IOleClientSite
	IAdviseSink
Message Filtering	IMessageFilter
Linking	none
Linking to Embedded Objects	IOleItemContainer
	IPersistFile
	IClassFactory
In-Place Activation	IOleInPlaceSite
	IOleInPlaceActive-Frame
	IOleInPlaceUIObject
Drag and Drop	IDropSource
	IDropTarget
	IDataObject

The following table lists required and optional behaviors for OLE servers and their compound document objects and which interfaces you must implement for each. The table distinguishes OLE servers and their objects in order to clarify which component implements which interfaces. The table also notes the different requirements of objects provided by out-of-process versus in-process servers.

Feature	OLE Server	Compound Document Object	
		Out-of-Process	**In-Process**
Required Behaviors	IClassFactory	IOleObject	IOleObject
		IDataObject	IDataObject
		IPersistStorage	IPersistStorage
			IViewObject2
			IOleCache2
Message Filtering	IMessageFilter		
Linking	IOleItemContainer		IOleLink
	IPersistFile		IExternalConnection
In-Place-		IOleInPlaceObjec	IOleInPlaceObjec

Activation		t IOleInPlaceActiveObject	t IOleInPlaceActiveObject
Drag and Drop	IDropSource IDropTarget IDataObject		

Object States

A compound object exists in one of three states: passive, loaded, or running. A compound-document object's state describes the relationship between the object in its container and the application responsible for its creation. The following table summarizes these states:

Object State	Description
Passive	The compound-document object exists only in storage, either on disk or in a database. In this state, the object is unavailable for viewing or editing.
Loaded	The object's data structures created by the object handler are in the container's memory. The container has established communication with the object handler and there is cached presentation data available for rendering the object. Calls are processed by the object handler. This state, because of its low overhead, is used when a user is simply viewing or printing an object.
Running	The objects that control remoting have been created and the OLE server application is running. The object's interfaces are accessible, and the container can receive notification of changes. In this state, an end user can edit or otherwise manipulate the object.

Entering the Loaded State

When an object enters the loaded state, the in-memory structures representing the object are created so that operations can be invoked on it. The object's handler or in-process server is loaded. This process, referred to as *instantiation,* occurs when an object is loaded from persistent storage (a transition from the passive to the loaded state) or when an object is being created for the first time.

Internally, instantiation is a two-phase process. An object of the appropriate class is created, after which a method on that object is called to perform initialization and give access to the object's data. The initialization method is defined in one of the object's supported interfaces. The particular initialization method called depends on the context in which the object is being instantiated and the location of the initialization data.

Entering the Running State

When an embedded object makes the transition to the running state, the object handler must locate and run the server application in order to utilize the services that only the server provides. Embedded objects are placed in the running state either explicitly through a request by the container, such as a need to draw a format not currently cached, or implicitly by OLE in response to invoking some operation, such as when a user of the container double-clicks the object.

When a linked object makes the transition into the running state, the process is known as *binding*. In the process of binding, the object handler asks its stored moniker to locate the link's data, then runs the server application.

At first glance, binding a linked object appears to be no more complicated than running an embedded object. However, the following points complicate the process:

- A link can refer to an object, or a portion thereof, that is embedded in another container. This capability implies a potential for nested embeddings. Resolving references to such a hierarchy requires recursively traversing a *composite moniker*, beginning with the rightmost member.
- When the link source is running, OLE binds to the running instance of the object rather than running another instance. In the case of nested embedded objects, one of which is the link source, OLE must be able to bind to an already running object at any point.
- Running an object requires accessing the storage area for the object. When an embedded object is run, OLE receives a pointer to the storage during the load process, which it passes on to the OLE server application. For linked objects, however, there is no standard interface for accessing storage. The OLE server application may use the file system interface or some other mechanism.

Entering the Passive State

Object closure forces an embedded or linked object into the passive state. It is typically initiated from the OLE server application's user interface, such as when the user selects the File Close command. In this case, the OLE server application notifies the container, which releases its reference count on the object. When all references to the object have been released, the object can be freed. When all objects have been freed, the OLE server application can safely terminate.

A container application can also initiate object closure. To close an object, the container releases its reference count after completing an optional save operation. You can design containers to release objects when they are deactivating after an in-place activation session, allowing the user to click outside the object without losing the active editing session.

Implementing In-Place Activation

In-place activation enables a user to interact with an embedded object without leaving the container document. When a user activates the object, a *composite menu bar* comprising elements from both the container application's and server application's menu bars replaces the container's main menu bar. Commands and features from both applications are thus available to the user, including context sensitive help for the active object. When a user begins working with some non-object portion of the document, the object is deactivated, causing the container document's original menu to replace the composite menu.

This capability originally went by the name of *in-place editing.* The name was changed because editing is only one way for a user to interact with a running object. Sound clips, for example, can be listened to instead of editing. Video clips can be viewed instead of editing. In-place activation is particularly apt in the case of video clips because it allows them to run in place, without calling up a separate window. This could be critical if the video were to be viewed, say, in conjunction with adjacent text data in the container document.

Implementing in-place activation is strictly optional for both container and server applications. OLE still supports the model in which activating an object causes the server application to open a separate window. Linked objects always open in a separate window to emphasize that they reside in a separate document.

In-place activation begins with the object in response to an **IOleObject::DoVerb** call from its container. This call usually happens in response to a user double-clicking the object or selecting a command (verb) from the container application's Edit menu.

The in-place window receives keyboard and mouse input while the embedded object is active. When a user selects commands from the composite menu bar, the command and associated menu messages are sent to the container or object application, depending on which owns the particular drop-down menu selected. Input by means of an object's rulers, toolbars, or frame adornments go directly to the embedded object, which owns these windows.

An in-place-activated embedded object remains active until either the container deactivates it in response to user input or the object voluntarily gives up the active state, as a video clip might do, for example. A user can deactivate an object by clicking inside the container document but outside the object's in-place-activation window, or simply by clicking another object. An in-place-activated object remains active, however, if the user clicks the container's title bar, scroll bar or, in particular, menu bar.

You can implement an in-place-activation-object server either as an in-process server (DLL) or a local server (EXE). In both cases, the composite menu bar contains items (typically drop-down menus) from both the server and container processes. In the case of a in-process server, the in-place activation window is simply another child window in the container's window hierarchy, receiving its input through the container application's message pump.

In the case of a local server, the in-place activation window belongs to the embedded object's server application process, but its parent window belongs to the container. Input for the in-place-activation window appears in the server's message queue and is dispatched by the server's message loop. The OLE libraries are responsible for seeing to it that menu commands and messages are dispatched correctly.

Creating Linked and Embedded Objects from Existing Data

A user typically assembles a compound document by using either the clipboard or drag and drop to copy a data object from its server application to the user's container application. With applications that support OLE, the user can initiate the transfer from either the server or the container. For example, the server can copy data to the clipboard in the server application, then switch to the container application and choose Paste Special/Embedded Object or an equivalent menu command to create a new embedded object from the selected data. Or, the user can drag the data from one application to the other. The process is similar for creating a linked object.

Note An application that functions as both OLE server and container can use a selection of its own data to create an embedded or linked object at a new location within the same document.

Data transfer between OLE server and container applications is built on uniform data transfer, as described in Chapter 4, "Data Transfer." OLE servers and object handlers implement **IDataObject** in order to make their data available for transfers using either the clipboard or drag and drop. OLE objects support all the usual clipboard formats. In addition, they support six clipboard formats that support the creation of linked and embedded objects from a selected data object.

OLE clipboard formats describe data objects that, upon being dropped or pasted in OLE containers, are to become embedded or linked compound-document objects. The data object presents these formats to container applications in order of their fidelity as descriptions of the data. In other words, the object presents first the format that best represents it, followed by the next best format, and so on. This intentional ordering encourages a container application to use the best possible format.

View Caching

A container application must be able to obtain a presentation of an object for the purpose of displaying or printing it for users when the document is open but the object's server application is not running or is not installed on the user's machine. To assume, however, that the servers for all the objects that might conceivably be found in a document are installed on every user's machine and can always run on demand is unrealistic. The default object handler, which *is* available at all times, solves this dilemma by caching object presentations in the document's storage and manipulating these presentations on any platform regardless of the availablility of the object server on any particular installation of the container.

Containers can maintain drawing presentations for one or more specific target devices in addition to the one required to maintain the object on screen. Moreover, if you port the object from one platform to another, OLE automatically converts the object's data formats to ones supported on the new platform. For example, if you move an object from Windows to the Macintosh, OLE will convert its metafile presentations to PICT formats.

In order to present an accurate representation of an embedded object to the user, the object's container application initiates a dialog with the object handler, requesting data and drawing instructions. To be able to fulfill the container's requests, the handler must implement the **IDataObject**, **IViewObject2**, and **IOleCache** interfaces.

IDataObject enables an OLE container application to get data from and send data to its embedded or linked objects. When data changes in an object, this interface provides a way for the object to make its new data available to its container and provides the container with a way to update the data in its copy of the object. (For a discussion of data transfer in general, see Chapter 4, "Data Transfer.")

The **IViewObject2** interface is very much like the **IDataObject** interface except that it asks an object to draw itself on a device context, such as a screen, printer, or metafile, rather than move or copy its data to memory or some other transfer medium. The purpose of the interface is to enable an OLE container to obtain alternative pictorial representations of its embedded objects, whose data it already has, thereby avoiding the overhead of having to transfer entirely new instances of the same data objects simply to obtain new drawing instructions. Instead, the **IViewObject2** interface enables the container to ask an object to provide a pictorial representation of itself by drawing on a device context specified by the container.

When calling the **IViewObject2** interface, a container application can also specify that the object draw itself on a target device different than the one on which it will actually be rendered. This enables the container, as needed, to generate different renderings from a single object. For example, the caller could ask the object to compose itself for a printer even though the resulting drawing will be rendered on screen. The result, of course, would be a print-preview of the object.

The **IViewObject2** interface also provides methods that enable containers to register for view-change notifications. As with data and OLE advisories, a view advisory connection enables a container to update its renderings of an object at its own convenience rather than in response to a call from the object. For example, if a new version of an object's server application were to offer additional views of the same data, the object's default handler would call each container's implementation of **IAdviseSink::OnViewChange** to let them know that the new presentations were available. The container would retrieve this information from the advise sink only when needed.

Because Windows device contexts have meaning only within a single process, you cannot pass **IViewObject2** pointers across process boundaries. As a result, OLE local and remote servers have no need whatsoever to implement the interface, which wouldn't work properly even if they did. Only object handlers and in-process servers implement the **IViewObject2** interface. OLE provides a default implementation, which you can use in your own OLE in-process servers and object handlers simply by aggregating the OLE default handler. Or you can write your own implementation of **IViewObject**.

An object implements the **IOleCache** interface in order to let the handler know what capabilities it should cache. The object handler also owns the cache and ensures it is kept up to date. As the embedded object enters the running state, the handler sets up appropriate advisory connections on the server object, with itself acting as the sink. The **IDataObject** and **IViewObject2** interface implementations operate out of data cached on the client side. The handler's implementation of **IViewObject2** is responsible for determining what data formats to cache in order to satisfy client draw requests. The handler's implementation of **IDataObject** is responsible for getting data in various formats, etc., between memory and the underlying **IStorage** instance of the embedded object. Custom handlers can use these implementations by aggregating on the default handler.

Note The **IViewObject2** interface is a simple functional extension of **IViewObject** and should be implemented instead of the latter interface, which is now obsolete. In addition to providing the **IViewObject** methods, the **IViewObject2** interface provides a single additional member, **GetExtent**, which enables a container application to get the size of an object's presentation from the cache without first having to move the object into the running state with a call to **IOleObject::GetExtent**.

OLE Custom Interfaces

As an OLE developer, you implement and use objects that are based on the Component Object Model (COM). COM interfaces specify a contract between the interface implementor and its user. The contract applies to every COM object that supports that interface.

OLE provides a standard set of interfaces, but these OLE-provided interfaces may not be perfectly suited to your needs. Through custom interfaces, you can create new ones tailored to fulfill the specific needs of your application. Custom interfaces are extensions to the COM standard; they let you extend the standard behavior and still take advantage of the services provided by the base interface. Like all COM interfaces, custom interfaces derive from the **IUnknown** interface and must contain the three methods of **IUnknown**. Otherwise, they can support a wide range of methods and parameters. (Note that, currently, custom interfaces cannot include asynchronous methods. Asynchronous methods may be supported in future versions of OLE.)

Once the interface is written, anyone can use it. Calls to methods in COM interfaces can cross process boundaries as long as both processes are running on the same machine.

This appendix introduces custom interfaces and explains how to implement them. OLE experience is a prerequisite, as is a basic knowledge of RPC. For more information about OLE and COM technology, see the *OLE Reference Guide* and the other parts of this *OLE Programmer's Guide*. Refer to the *RPC Programmer's Guide and Reference* for information about the RPC programming environment.

How Do Client/Server Objects Work?

A client application, such as a compound document container, makes a call to methods in a COM interface. An object application (sometimes called the server) implements an object that provides the interface and methods that are called by the client. A reference to the object application is stored in the registry. OLE locates the object application through this reference and runs it.

The object application is a piece of code that implements one or more COM interfaces. It can be an in-process server in a DLL or a local executable that runs in another process. Making a call to an interface method in an object in another process involves the cooperation of several components. The standard proxy is a piece of interface-specific code that resides in the client's process space and prepares the interface parameters for transmittal. It packages, or marshals, them in such a way that they can be re-created and understood in the receiving process. The standard stub, also a piece of interface-specific code, resides in the server's process space and reverses the work of the proxy. The stub unpackages, or unmarshals, the sent parameters and forwards them to the object application. It also packages reply information to send back to the client.

Note Readers more familiar with RPC than OLE may be used to seeing the terms client stub and server stub. These terms are analogous to proxy and stub.

The following diagram shows the flow of communication between the components involved. On the client side of the process boundary, the client's method call goes through the proxy and then onto the channel. Note that the channel is part of the COM library. The channel sends the buffer containing the marshalled parameters to the RPC run-time library, which transmits it across the process boundary. The RPC run-time and the COM libraries exist on both sides of the process. Note also that the distinction between the channel and the RPC run-time is a characteristic of this implementation and is not part of the programming model or the conceptual model for OLE client/server objects. OLE object applications see only the proxy or stub and, indirectly, the channel. Future implementations may use different layers below the channel or no layers.

{ewc msdncd, EWGraphic, group10502 0 /a "SDK.wmf"}

The following sections discuss some of the issues involved in Custom Interface technology and describe these components in greater detail.

Connecting to the Object Application

An internal OLE component ensures that when a client request is made, the appropriate object is connected and ready to receive the request. OLE keeps a database of class information based on the system registry that the client caches locally through the COM library.

When the server is an in-process DLL, OLE simply passes the name of the DLL to the COM library. The COM library then loads the server and makes the method call. When the server is an executable in another process, OLE either creates the server process and starts the server or contacts it directly if it is already running.

Marshalling

Marshalling is the process of packaging and unpackaging parameters so a remote procedure call can take place. Different parameter types are marshalled in different ways. For example, marshalling an integer parameter involves simply copying the value into the message buffer (although even in this simple case, there are issues such as byte ordering to deal with with cross-machine calls). Marshalling an array, however, is a more complex process. Array members are copied in a specific order so that the other side can reconstruct the array exactly. When a pointer is marshalled, the data that the pointer is pointing to is copied following rules and conventions for dealing with nested pointers in structures. Unique functions exist to handle the marshalling of each parameter type.

There are two types of OLE marshalling: standard and custom.

With standard marshalling, the proxies and stubs are system-wide resources for the interface, and they interact with the channel through a standard protocol. Standard marshalling can be used by both standard OLE interfaces and custom interfaces:

- In the case of most OLE interfaces, the proxies and stubs for standard marshalling are themselves in-process component objects which are loaded from a system-wide DLL provided by OLE, OLE32.DLL.
- In the case of custom interfaces, the proxies and stubs for standard marshalling are generated by the interface designer, typically with MIDL. These proxies and stubs are statically configured in the registry, so any potential client can use the custom interface across process boundaries. These proxies and stubs are loaded from a DLL that is located via the system registry using the interface ID (IID) for the custom interface they marshal.

Standard marshalling is the default method for the provision of marshalling code when passing interface pointers. However, because multiple objects may implement a single interface any number of ways, standard marshalling may not be sufficient for a particular implementation. Therefore, any object may override the default and provide custom marshalling for its interfaces. Other implementations of the interface are not affected.

As an alternative to standard marshalling, an interface (standard or custom) can use custom marshalling. With custom marshalling, an object dynamically implements the proxies at run-time for each interface that it supports. For any given interface, the object can select OLE-provided standard marshalling or custom marshalling. This choice is made by the object on an interface by interface basis. Once the choice is made for a given interface, it remains in effect during the object's lifetime. However one interface on an object can use custom marshalling while another uses standard marshalling.

Custom marshalling is inherently unique to the object application that implements it. It uses proxies implemented by the object and provided to the system on request at run-time. Objects that custom-marshal must implement the **IMarshal** interface, whereas objects that support standard marshalling do not. For complete documentation of the **IMarshal** interface, see the *OLE Reference Guide*. Also, see the chapter "The Component Object Model" for more information on marshalling.

Marshalling support is provided by the system for all OLE-defined interfaces. This support is via component objects in the file OLE32.DLL. If you decide to write a custom interface, you must provide marshalling support for it. Typically, you will provide a standard marshalling DLL for the interface you design. You can use the tools contained in the *Win32 SDK* CD to create the proxy/stub code and the proxy/stub DLL.

Proxy

A proxy resides in the address space of the calling process and acts as a surrogate for the remote object. From the perspective of the calling object, the proxy is the object. Typically, the proxy's role is to package the interface parameters for calls to methods in its object interfaces. The proxy packages the parameters into a message buffer and passes the buffer onto the channel, which handles the transport between processes. The proxy is implemented as an aggregate, or composite, object. It contains a system-provided, manager piece called the proxy manager and one or more interface-specific components called interface proxies. The number of interface proxies equals the number of object interfaces that have been exposed to that particular client. To the client complying with the component object model, the proxy appears to be the real object.

Note With custom marshalling, the proxy can be implemented similarly or it can communicate directly to the object without using a stub.

Each interface proxy is a component object that implements the marshalling code for one of the object's interfaces. The proxy represents the object for which it provides marshalling code. Each proxy also implements the **IRpcProxyBuffer** interface. Although the object interface represented by the proxy is public, the **IRpcProxyBuffer** implementation is private and is used internally within the proxy. The proxy manager keeps track of the interface proxies and also contains the public implementation of the controlling **IUnknown** interface for the aggregate. Each interface proxy can exist in a separate DLL that is loaded when the interface it supports is materialized to the client.

The following diagram shows the structure of a proxy that supports the standard marshalling of parameters belonging to two interfaces: **IFoo1** and **IFoo2**. Each interface proxy implements **IRpcProxyBuffer** that is used for internal communication between the aggregate pieces. When the proxy is ready to pass its marshalled parameters across the process boundary, it calls methods in the **IRpcChannelBuffer** interface, which is implemented by the channel. The channel in turn forwards the call to the RPC run-time library so that it can reach its destination in the object.

{ewc msdncd, EWGraphic, group10502 1 /a "SDK.wmf"}

Stub

The stub, like the proxy, is made up of one or more interface pieces and a manager. Each interface stub provides code to unmarshal the parameters and code that calls one of the object's supported interfaces. Each stub also provides an interface for internal communication. The stub manager keeps track of the available interface stubs.

There are, however, some differences between the stub and the proxy:

- The most important difference is that the stub represents the client in the object's address space.
- The stub is not implemented as an aggregate object since there is no requirement that the client be viewed as a single unit; each piece in the stub is a separate component.
- The interface stubs are private, rather than public.
- The interface stubs implement **IRpcStubBuffer**, not **IRpcProxyBuffer**.
- Instead of packaging parameters to be marshalled, the stub unpackages them after they have been marshalled and then packages the reply.

The following diagram shows the structure of the stub. Each interface stub is connected to an interface on the object. The channel dispatches incoming messages to the appropriate interface stub. All the components talk to the channel through **IRpcChannelBuffer**, the interface that provides access to the RPC run-time library.

{ewc msdncd, EWGraphic, group10502 2 /a "SDK.wmf"}

Channel

The channel has the responsibility of transmitting all messages between client and object across the process boundary. The channel has been designed to work transparently with different channel types, is compatible with OSF DCE standard RPC, and supports single and multi-threaded applications.

Microsoft RPC

RPC is a model for programming in a distributed computing environment. The goal of RPC is to provide transparent communication so that the client appears to be directly communicating with the server. Microsoft's implementation of RPC is compatible with the Open Software Foundation (OSF) Distributed Computing Environment (DCE) RPC.

You can configure RPC to use one or more transports, one or more name services, and one or more security servers. The interface to those providers are handled by RPC. Because Microsoft RPC is designed to work with multiple providers, you can choose the providers that work best for your network. The transport is responsible for transmitting the data across the network. The name service takes an object name, such as a moniker, and finds its location on the network. The security server offers applications the option of denying access to specific users and/or groups. Refer to the next section "Writing a Custom Interface" for more detailed information about application security.

In addition to the RPC run-time libraries, Microsoft RPC includes the Interface Definition Language (IDL) and its compiler. Although the IDL file is a standard part of RPC, Microsoft has enhanced it to extend its functionality to support custom COM interfaces. The Microsoft Interface Definition Language (MIDL) compiler uses the IDL file that describes your custom interface to generate several files discussed in the section "Building a Proxy/Stub DLL".

Writing a Custom Interface

In order to perform the tasks discussed in the following section, you must obtain a Microsoft *Win32 Software Development Kit (SDK).* It contains a toolkit and all of the libraries you need for building a custom interface project. It provides more detailed information about how to write a custom interface using the MIDL compiler.

After installing the toolkit, you can design and write your custom interface using the MIDL compiler, and build the standard-marshalling proxy/stub DLL for your interface from the files generated by the compiler. You will then install the DLL in the system registry.

Designing Efficient Interfaces

There are some special considerations to keep in mind when designing a custom interface. If you follow these guidelines, your marshalling code can be used across the network when support for remotable objects is provided in a future release.

First of all, because data is shipped across address spaces, using architecture-dependent types such as **int** may prohibit the data from being re-created correctly in the object application's address space. The size of an integer varies from architecture to architecture. Thus, the data type **int** should not be used for structure data members or interface method parameters. Specifying data types unambiguously to the MIDL compiler will allow your interface to be more easily transferred to remote machines with different architectures and addressing schemes. The MIDL compiler requires all integer variables that may be remoted to be explicitly declared as short, long, or their unsigned equivalents.

A frequently used data type for pointers, the **void *** construct, allows the callee of a method to interpret the data pointed to according to the need. While local interfaces can use this construct, distributed applications cannot. This is because the MIDL compiler must know exactly what types of data are being transmitted so that the data can be accurately re-created on the receiving side. The **void *** construct is too vague.

Pointers to data must be used carefully. To re-create the data in the callee's address space, RPC must know the exact size of the data. If, for example, a **char *** parameter points to a buffer of characters rather than to a single character (as is implied), the data cannot be correctly re-created. Use the syntax available with MIDL to accurately describe the data structures represented by your type definitions.

Initialization is essential for pointers that are embedded in arrays and structures and passed across process boundaries. Uninitialized pointers may work when passed to a program in the same process space, but proxies and stubs assume that all pointers are initialized with valid addresses or are null.

Be careful when aliasing pointers (allowing pointers to point to the same piece of memory). If the aliasing is intentional, these pointers should be declared aliased in the IDL file. Pointers declared as non-aliased should never alias each other.

The MIDL has directional attributes that let you specify the data flow direction. The proxies and stubs use these attributes to determine whether to send the data from the client to the object or from the object to the client, respectively. Data labeled as *out* only is uninitialized on the way to the interface stub; data labeled as *in* only does not affect the data structures upon return. The *in, out* attribute indicates that data is sent to the object initialized and the object will change it before sending it back.

It is also imperative to realize the importance of defining status codes. This must be coordinated by the definer of the interface to avoid conflicting error codes.

Finally, the designer of an interface must develop an understanding of how the interface will be used by client applications. In particular, the frequency of method calls across the interface boundary and the amount of data to be transferred to complete a given method call together determine whether the interface will be efficient across process and machine boundaries. Although OLE makes cross-process and, eventually, cross-network calls transparent to programs, it cannot make high-frequency and high-bandwidth calls efficient across address spaces. In some cases, it is more appropriate to design interfaces that will normally be implemented only as in-process servers while other interfaces are more appropriate for remote use.

Creating a Custom Interface

Writing an application using a custom interface is similar to writing an OLE application using standard interfaces, with two important differences. First, in addition to registering the class identifier (CLSID) for your object, you also must register the unique interface identifier (IID) for your custom interface. Second, you must provide OLE with proxy and stub components capable of remoting the interface. In most cases, you will use the MIDL compiler to generate code to create the proxy/stub DLL for your custom interface.

As an OLE developer, you should already be familiar with the concept of using CLSIDs and interface identifiers (IIDs) to uniquely identify class objects and interfaces. The MIDL compiler refers to universally unique identifiers (UUIDs) and uses UUIDs to uniquely represent its interfaces. In this case, a UUID is the same as an IID. Every custom interface must have a IID; the UUIDGEN.EXE tool provided with the toolkit can create one for you.

MIDL is a rich, complex language that allows you to define interface parameters carefully in terms of their direction and type. Although the MIDL compiler offers much, you will need only a small subset of its attributes to define your interface.

The IDL file consists of two parts: the interface header and the interface body. The interface header, delimited by brackets, contains information about the interface as a whole, such as its IID. A prerequisite to completing the interface header is running the RPC utility UUIDGEN.EXE to create the IID. Because IIDs must be unique, never reuse one of them by copying it from one IDL file to another. The header also contains the keyword **object**, indicating that this interface is a COM-style interface (that is, derived from **IUnknown**). Whereas standard RPC interfaces have the version attribute in their interface headers, object interfaces do not.

The following excerpt is from an actual IDL file:

```
[
    object,
    uuid(7ACC12C3-C4BB-101A-BB6E-0000C09A6549),
    pointer_default(unique)
]
```

The **uuid** attribute precedes the unique identifier for the interface. The **pointer_default** attribute specifies the default IDL type for all pointers except for those included in parameter lists. Parameter list pointers must be explicitly declared with the pointer attribute. The default pointer type may either be **unique**, **ref**, or **ptr**. For more information on using IDL with pointers, see the *RPC Programmer's Guide and Reference*.

The interface body contains declarations for data members, prototypes for all methods, and other information such as directives to the preprocessor and include statements for other IDL files. The following example is the interface body for **ICustomInterface**:

```
interface ICustomInterface : IUnknown
{
    import "unknwn.idl";

    HRESULT CustomReport(void);
}
```

The **import** attribute allows an interface to reference constructs defined in other IDL files. Because **ICustomInterface** derives from **IUnknown**, the UNKNWN.IDL file must be included. **ICustomInterface** has only one method that takes no parameters; few interfaces will be this simple.

Building a Proxy/Stub DLL

Completed IDL files must be compiled with the IDL compiler, MIDL.EXE. The IDL compiler takes each IDL file and generates proxy/stub code (IDLNAME_P.C), an interface header file (IDLNAME.H), and an interface identifier file (IDLNAME_I.C). The header files contain both C and C++ class definitions.

By default, MIDL.EXE generates names based on your IDL file's name. However, you can use the following command line switches to override the default:

Command line switch	Description
-header	Specifies the name of the interface header file.
-proxy	Specifies the name of the proxy source file.
-iid	Specifies the name of the interface identifier file.

After a successful compile of the IDL file, the generated files are run through the standard C/C++ compile and link steps. These source files implement helper functions for marshalling and an implementation of **DllGetClassObject** for the proxy/stub libraries, among other things. Note that the RPC4RT.LIB library includes an implementation of the **DllGetClassObject** function. In this provided implementation, the CLSID of the proxy/stub has to be the same as the IID of your custom interface. If you want to use a CLSID that differs from the IID then you must provide your own implementation of the **DllGetClassObject** function.

The following diagram shows all the pieces involved in a build of a custom proxy/stub DLL. The IDL file, ITF.IDL, is fed into the IDL compiler. Three files are generated: ITF_P.C, ITF.H, and ITF_I.C. These three files are compiled and linked and the result is PROXSTUB.DLL.

{ewc msdncd, EWGraphic, group10502 3 /a "SDK.wmf"}

Registering a Proxy/Stub DLL

Before a client can use your custom interface, the proxy/stub DLL for the interface must be installed in the system registry. Registering the proxy/stub DLL involves creating a .REG file and running REGINI. In the following entries, notice that the IID for **ICustomInterface** is the same as the CLSID for the proxy/stub DLL.

```
 \Registry\MACHINE\SOFTWARE\Classes\Interface\
     {7ACC12C3-C4BB-101A-BB6E-0000C09A6549}
         = ICustomInterface
\Registry\MACHINE\SOFTWARE\Classes\Interface\
     {7ACC12C3-C4BB-101A-BB6E-0000C09A6549}\ProxyStubClsid32
         = {7ACC12C3-C4BB-101A-BB6E-0000C09A6549}
\Registry\MACHINE\SOFTWARE\Classes\CLSID\
     {7ACC12C3-C4BB-101A-BB6E-0000C09A6549}
         = ICustomInterface_PSFactory
\Registry\MACHINE\SOFTWARE\Classes\CLSID\
     {7ACC12C3-C4BB-101A-BB6E-0000C09A6549}\InprocServer32
         = proxstub.dll
```

Using a Custom Interface

The client code is the user of the custom interface. To use any interface, custom or standard, a client must know its IID. In the following function, **CustomRpt**, the driver that calls **CustomRpt** passes it the name of the object that is converted to a wide-character format. The object name is fed to **CreateFileMoniker** so that a file moniker can be created and the client can bind to the running object. Once the object is running, **CustomRpt** can access a pointer to either an interface in the standard proxy/stub, such as **IPersistFile**, or to the custom interface, **ICustomInterface**.

```
void CustomRpt(char *pszObject)
{
    HRESULT          hr;
    WCHAR                  szObject[128];
    WCHAR               wszMsg[128] = {L"Your Message Here...\n"};
    IMoniker                  *pmkObject = NULL;
    IUnknown            *pIUnk = NULL;
    IPersistFile        *pIPersistFile = NULL;
    ICustomInterface   *pICustomInterface = NULL;

    // Create a wide-character version of the object's file name.
    wsprintf(wszObject, L"%hs", pszObject);

    // Get a file moniker for the object (a *.smp file).
    hr = CreateFileMoniker(wszObject, &pmkObject);

    if(FAILED(hr))
    {
         printf("Client: CreateFileMoniker for Object failed");
         return;
    }

    // BindMoniker is equivalent to calling CreateBindCtx() followed by
    // a call to BindToObject(). It has the net result of binding the
    // interface (specified by the IID) to the moniker.

    hr = BindMoniker(pmkObject, 0, IID_IUnknown, (void **)&pIUnk);
    if (FAILED(hr))
    {
         printf("Client: BindMoniker failed (%x)\n", hr);
         return;
    }

    // Try a couple QueryInterface calls into the object code, first a
    // QueryInterface to IPersistFile...

    hr = pIUnk->QueryInterface(IID_IPersistFile,
       (void **)&pIPersistFile);

    if (FAILED(hr)) {
        printf("Client: QueryInterface IPersistFile failed (%x)\n", hr);
        pIUnk->Release();
        return;
    }

    // Followed by a QueryInterface to ICustomInterface.
    hr = pIUnk->QueryInterface(IID_ICustomInterface,
                    (void **)&pICustomInterface);

    if (FAILED(hr)) {
        printf("Client: QueryInterface failed (%x)\n", hr);
        pIUnk->Release();
        pIPersistFile->Release();
        return;
    }

    // CustomReport() is the object function that displays the time and
    // date information on the object.
    hr = pICustomInterface->CustomReport();

    if (FAILED(hr))
    {
        printf("Client: pICustomInterface->CustomReport failed (%x)\n",
             hr);
        pIUnk->Release();
        pIPersistFile->Release();
        return;
    }

    // Clean up resources by calling release on each of the interfaces.
    pIPersistFile->Release();
    pICustomInterface->Release();
    pIUnk->Release();
    return;
}
```

Data Transfer

The Component Object Model (COM) provides a standard mechanism for transferring data between applications. This mechanism is the *data object*, which is simply any COM object that implements the **IDataObject** interface. Some data objects, such as a piece of text copied to the clipboard, have **IDataObject** as their sole interface. Others, such as compound document objects, expose several interfaces, of which **IDataObject** is simply one. Data objects are fundamental to the working of compound documents, although they also have widespread application outside that OLE technology.

By exchanging pointers to a data object, providers and consumers of data can manage data transfers in a uniform manner, regardless of the format of the data, the type of medium used to transfer the data, or the target device on which it is to be rendered. You can include support in your application for basic clipboard transfers, drag and drop transfers, and OLE compound document transfers with a single implementation of **IDataObject**. Having done that, the amount of code required to accommodate the special semantics of each protocol is minimal.

Data Transfer Interfaces

The **IDataObject** interface provides consumers of data with methods for getting and setting an object's data, determining which formats the object supports, and registering for and receiving notifications when data in the object changes. When obtaining data, a caller can specify the format in which it wants to render the data. The source of the data, however, determines the storage medium, which it returns in an out parameter provided by the caller.

By itself, **IDataObject** supplies all the tools you need to implement Microsoft® Windows® clipboard transfers or compound document transfers in your applications. If you also want to support drag and drop transfers, you need to implement the **IDropSource** and **IDropTarget** **interfaces**along with **IDataObject**.

The **IDataObject** interface combined with OLE clipboard APIs provide all the capabilities of the Microsoft® Win32® clipboard APIs. Using both the platform's clipboard APIs and OLE's is usually redundant and unnecessary. Suppliers of data that support either drag and drop transfers or OLE compound documents *must* implement the **IDataObject** interface. If your application supports only clipboard transfers now, but you intend to add drag and drop or compound documents in later releases, you may want to implement **IDataObject** and the OLE clipboard APIs now in order to minimize the amount of time spent recoding and debugging later. You may also want to implement **IDataObject** in order to utilize transfer media other than global memory.

The following table summarizes which ones to use, depending on what types of data transfer you want to support:

If You Want to Support	You Must Use
Compound documents	**IDataObject**
Drag and drop transfers	**IDataObject**, **IDropSource**, **IDropTarget**, **DoDragDrop** (or the equivalent)
Windows clipboard transfers using global memory exclusively	Windows clipboard APIs
Windows clipboard transfers using exchange mediums other than global memory.	**IDataObject**
Clipboard transfers now but drag and drop or compound documents later	**IDataObject** and the interfaces and function listed above for "Drag and drop transfers"

When a user initiates a data transfer operation, the source application creates an instance of **IDataObject** and through it makes the data available in one or more formats. In a clipboard transfer, the application calls the **OleSetClipboard** function to pass a data-object pointer to OLE. (**OleSetClipboard** also offers standard clipboard data formats for both OLE version 1 and non-OLE applications.) In a drag and drop transfer, the application calls the **DoDragDrop** function instead.

On the receiving side of the transfer, the destination receives the **IDataObject** pointer either as an argument to an invocation of **IDropTarget::Drop** or by calling the **OleGetClipboard** function, depending on whether the transfer is by means of drag and drop or the clipboard. Having obtained this pointer, the destination calls **IDataObject::EnumFormatEtc** to learn what formats are available for retrieval and on what types of media they can be obtained. Armed with this information, the receiving application requests the data with a call to **IDataObject::GetData**.

Data Formats and Transfer Media

Most platforms, including Windows, define a standard protocol for transferring data between applications, based on a set of functions called the clipboard. Applications using these functions can share data even if their native data formats are wildly different. Generally, these clipboards have two significant shortcomings that COM has overcome.

First, data descriptions use only a format identifier, such as the single 16-bit clipboard format identifier on Windows, which means the clipboard can only describe the structure of its data, that is, the ordering of the bits. It can report, "I have a bitmap" or "I have some text," but it cannot specify the target devices for which the data is composed, which views or aspects of itself the data can provide, or which storage media are best suited for its transfer. For example, it cannot report, "I have a string of text that is stored in global memory and formatted for presentation either on screen or on a printer" or "I have a thumbnail sketch bitmap rendered for a 100 dpi dot-matrix printer and stored as a disk file."

Second, all data transfers using the clipboard generally occur through global memory. Using global memory is reasonably efficient for small amounts of data but horribly inefficient for large amounts, such as a 20 MB multimedia object. Global memory is slow for a large data object, whose size requires considerable swapping to virtual memory on disk. In cases where the data being exchanged is going to reside mostly on disk anyway, forcing it through this virtual-memory bottleneck is highly inefficient. A better way would skip global memory entirely and simply transfer the data directly to disk.

To alleviate these problems, COM introduces two new data structures: FORMATETC and STGMEDIUM.

The FORMATETC Structure

The FORMATETC structure is a generalized clipboard format, enhanced to encompass a target device, an *aspect* or view of the data, and a storage medium. A data consumer, such as an OLE container application, passes the FORMATETC structure as an argument in calls to **IDataObject** to indicate the type of data it wants from a data source, such as a compound document object. The source uses the FORMATETC structure to describe what formats it can provide. FORMATETC can describe virtually any data, including other objects such as monikers. A container can ask one of its embedded objects to list its data formats by calling **IDataObject::EnumFormatetc**, which returns an enumerator object that implements the **IEnumFormatEtc** interface. Instead of replying merely that it has "text and a bitmap," the object can provide a detailed description of the data, including the device (normally screen or printer) for which it is rendered, the aspect to be presented to the user (full contents, thumbnail, icon, or formatted for printing), and the storage medium containing the data (global memory, disk file, storage object, or stream). This ability to tightly describe data will, in time, result in higher quality printer and screen output as well as more efficiency in data browsing, where a thumbnail sketch is much faster to retrieve and display than a fully detailed rendering.

The following table lists fields of the FORMATETC data structure and the information they specify:

Field	**Specifies**
cfFormat	The format in which the data is to be rendered, which can be a standard clipboard format, a proprietary format, or an OLE format. For more information on OLE formats, see Chapter 5, "Compound Documents."
ptd	A DVTARGETDEVICE structure, which contains enough information about a Windows target device, such as a screen or printer, so that a handle to its device context (hDC) can be created using the Windows **CreateDC** function.
dwAspect	The aspect or view of the data to be rendered; can be the full contents, a thumbnail sketch, an icon, or formatted for printing.
lindex	The part of the aspect that is of interest; for the present, the value must be -1, indicating that the entire view is of interest.
tymed	The data's storage medium, which can be global memory, disk file, or an instance of one of COM's structured-storage interfaces.

The STGMEDIUM Structure

Just as the FORMATETC structure is an enhancement of the Windows clipboard format identifier, so the STGMEDIUM structure is an improvement of the global memory handle used to transfer the data. The STGMEDIUM structure includes a flag, *tymed,* which indicates the medium to be used, and a union comprising pointers and a handle for getting whichever medium is specified in *tymed.*

The STGMEDIUM structure enables both data sources and consumers to choose the most efficient exchange medium on a per-rendering basis. If the data is so big that it should be kept on disk, the data source can indicate a disk-based medium in its preferred format, only using global memory as a backup if that's the only medium the consumer understands. Being able to use the best medium for exchanges as the default improves overall performance of data exchange between applications. For example, if some of the data to be transferred is already on disk, the source application can move or copy it to a new destination, either in the same application or in some other, without having first to load the data into global memory. At the receiving end, the consumer of the data does not have to write it back to disk.

Drag and Drop

"Drag and drop" refers to data transfers in which a mouse or other pointing device is used to specify both the data source and its destination. In a typical drag and drop operation, a user selects the object to be transferred by moving the mouse pointer to it and holding down either the left button or some other button designated for this purpose. While continuing to hold down the button, the user initiates the transfer by dragging the object to its destination, which can be any OLE container. Drag and drop provides exactly the same functionality as the OLE Clipboard copy and paste but adds visual feedback and eliminates the need for menus. In fact, if an application supports Clipboard copy and paste, little extra is needed to support drag and drop.

During an OLE drag and drop operation, the following three separate pieces of code are used:

Drag-and-drop code source	**Implementation and use**
IDropSource interface	Implemented by the object containing the dragged data, referred to as the *drag source*.
IDropTarget interface	Implemented by the object that is intended to accept the drop, referred to as the *drop target*.
DoDragDrop function	Implemented by OLE and used to initiate a drag and drop operation. Once the operation is in progress, it facilitates communication between the drag source and the drop target.

The **IDropSource** and **IDropTarget** interfaces can be implemented in either a container or in an object application. The role of drag source or drop target is not limited to any one type of OLE application.

The OLE function **DoDragDrop** implements a loop that tracks mouse and keyboard movement until such time as the drag is canceled or a drop occurs. **DoDragDrop** is the key function in the drag and drop process, facilitating communication between the drag source and drop target.

During a drag and drop operation, three types of feedback can be displayed to the user:

Type of feedback	**Description**
Source feedback	Provided by the drag source, the source feedback indicates the data is being dragged and does not change during the course of the drag. Typically, the data is highlighted to signal it has been selected.
Pointer feedback	Provided by the drag source, the pointer feedback indicates what happens if the mouse is released at any given moment. Pointer feedback changes continually as the user moves the mouse and/or presses a modifier key. For example, if the pointer is moved into a window that cannot accept a drop, the pointer changes to the "not allowed" symbol.
Target feedback	Provided by the drop target, target feedback indicates where the drop is to occur.

Although the Windows 3.1 File Manager does not use OLE to implement drag and drop transfers, it is nevertheless an example of an application that acts as both a drag source and drop target and that provides all three types of feedback. When a user selects a file to copy, File Manager supplies source feedback by indicating the selection. As the selection is dragged and modifier keys are pressed, the mouse pointer changes. For example, if the user presses the CTRL key, a plus sign is added, indicating that the drop would result in a copy rather than moving the original. When the file is dragged over an area that is not a drop target, the pointer changes appropriately. Target feedback, a line drawn around a file or directory, is provided when the pointer is over an area that is a drop target.

Drag Source Responsibilities

The drag source is responsible for the following tasks:

- Providing a data-transfer object for the drop target that exposes the **IDataObject** and **IDropSource** interfaces.
- Generating pointer and source feedback.
- Determining when the drag operation has been canceled or a drop operation has occurred.
- Performing any action on the original data caused by the drop operation, such as deleting the data or creating a link to it.

The main task is creating a data-transfer object that exposes the **IDataObject** and **IDropSource** interfaces. The drag source might or might not include a copy of the selected data. Including it is not mandatory, but doing so safeguards against inadvertent changes and allows the Clipboard operations code to be identical to the drag and drop code.

While a drag operation is in progress, the drag source is responsible for setting the mouse pointer and, if appropriate, for providing additional source feedback to the user. The drag source cannot provide any feedback that tracks the mouse position other than by actually setting the real pointer (see the Windows **SetCursor** function). This rule must be enforced to avoid conflicts with the feedback provided by the drop target. (A drag source can also be a drop target. When dropping on itself, the source/target can, of course, provide target feedback to track the mouse position. In this case, however, it is the drop target tracking the mouse, not the source.) Based on the feedback offered by the drop target, the source sets an appropriate pointer.

Data Notification

Objects that consume data from an external source sometimes need to be informed when data in that source changes. For example, a stock ticker tape viewer that relies on data in some spreadsheet needs to be notified when that data changes so it can update its display. Similarly, a compound document needs information about data changes in its embedded objects so that it can update its data caches. In cases such as this, where dynamic updating of data is desirable, sources of data require some mechanism of notifying data consumers of changes as they occur without obligating the consumers to drop everything in order to update their data. Ideally, having been notified that a change has occurred in the data source, a consuming object can ask for an updated copy at its leisure.

COM's mechanism for handling asynchronous notifications of this type is an object called an advise sink, which is simply any COM object that implements an interface called **IAdviseSink**.   Consumers of data implement the **IAdviseSink**. They register to receive notifications by handing a pointer to the data object of interest.

The **IAdviseSink** interfaces exposes the following methods for receiving asynchronous notifications:

Method	**Notifies the Advise Sink that**
OnDataChange	Calling object's data has changed.
OnViewChange	Instructions for drawing the calling object have changed.
OnRename	Calling object's moniker has changed.
OnSave	Calling object has been saved to persistent storage.
OnClose	Calling object has been closed.

As the table indicates, the **IAdviseSink** interface exposes methods for notifying the advise sink of events other than changes in the calling object's data. The calling object can also notify the sink when the way in which it draws itself changes, or it is renamed, saved, or closed. These other notifications are used mainly or entirely in the context of compound documents, although the notification mechanism is identical. For more information on compound-document notifications, see "Compound Documents."

In order to take advantage of the advise sink, a data source must implement **IDataObject::DAdvise, IDataObject::DUnadvise,** and **IDataObject::EnumDAdvise.** A data consumer calls the **DAdvise** mothod to notify a data object that it wishes to be notified when the object's data changes. The consuming object calls the **DUnadvise** method to tear down this connection. Any interested party can call the **EnumAdvise** method to learn the number of objects having an advisory connection with a data object.

When data changes at the source, the data object calls **IAdviseSink::OnDataChange** on all data consumers that have registered to receive notifications. To keep track of advisory connections and manage the dispatch of notifications, data sources rely on an object called a *data advise holder*. You can create your own data advise holder by implementing the **IDataAdviseHolder** interface. Or, you can let COM do it for you by calling the helper function **CreateDataAdviseHolder**.

Overview

Today's software applications are more capable and easier to use than ever before. Yet as their feature sets have grown in complexity and size, they have become increasingly difficult and costly to engineer, maintain, and upgrade. Adding new features always runs the risk of introducing new, sometimes intractable bugs and endangers backward compatibility with earlier versions. In addition, most applications are monolithic, providing rich sets of features but no easy way to add missing features or remove unneeded ones. Applications that do support adding and removing features in the form of plug-in components may expose their services to sibling applications from the same vendor but rarely to those from entirely different vendors. As a result, applications from different vendors typically work together poorly or not at all.

Operating systems have a related set of problems. Because most operating systems are not sufficiently modular, upgrading, replacing, or overriding existing services in a clean, flexible way is difficult. Like application developers, systems vendors face the difficult choice of foregoing or compromising needed improvements in order to maintain backward compatibility, or disenfranchising existing users in order to move the technology forward. These are choices that serve no one.

Software developers have recognized for some time that object-oriented programming offers promising solutions to several of these problems. By encapsulating data and functions in a single entity, then providing access to this rich content through a single reference, or pointer, object-oriented programming makes it possible to break down monolithic applications into functional modules, or *components,* that can be added or removed as they are needed. Moreover, object-oriented programming can potentially reduce the expense of implementing new objects by enabling them, through mechanisms such as inheritance, polymorphism, and aggregation, to make use of the capabilities built into already-existing objects.

Even so, object-oriented programming has yet to reach its full potential because no standard framework has existed through which software objects created by different vendors could interact with one another within the same address space, much less across process, machine, and network boundaries. Once you have broken down software into manageable, interchangeable components, you need some way for those components to communicate. That's where OLE comes in.

OLE provides a standard conceptual framework for creating, managing, and accessing object-based components that provide services to other objects and applications. OLE components can exist as parts of an operating system or application, or as stand-alone entities, and the services they provide can be most anything that operating systems and applications currently supply. You can use OLE components to expose both data and services to programs created by other vendors and write your own applications in such a way as to take advantage of services provided by others. You can also extend or customize basic OLE services to suit the needs of your objects and applications while guaranteeing that they continue to work with other OLE components.

Explaining the Component Object Model

The bedrock on which OLE rests is the Component Object Model (COM), which provides both the programming model and binary standard for OLE components. COM defines and implements mechanisms that enable software components, such as applications, data objects, controls, and services, to interact as *objects*. A software object comprises some body of data, along with one or more functions for accessing and using that data. An OLE component is one in which access to an object's data is achieved exclusively through one or more sets of related functions called *interfaces.*

An object provider, or *server*, makes an OLE component available by implementing one or more interfaces. A prospective user, or *client,* of an OLE component gains access to the object by obtaining a pointer to one of its interfaces. With this pointer, the client can reliably use the object without understanding its implementation and with the assurances that the object will always behave the same way. In this sense, an object's interface is a contract defining its behavior for prospective clients. The object will honor that contract even if its client lives in a different process or on a different machine, runs under a different operating system, is written by a different software developer using a different computer language, or represents an earlier or later version than the client used before.

By defining interfaces as contracts between objects and their clients, COM effectively solves the versioning problem. Here's how. To create a new version of an object, you simply add new interfaces while leaving the older ones in place. You define the new interfaces in such a way that clients are permitted to query for either the new version of the interface, or the old version, but not for both. Because of this restriction, adding new interfaces, or new versions of older interfaces, in no way interferes with the way existing clients work with an object.

Because OLE components conform to a binary standard, you can implement them using nearly any programming language. Object-oriented languages such as C++ and Smalltalk are ideal for this purpose, but any procedural language, such as C or Pascal, that supports the notion of a reference, or pointer, to an object will do. OLE components implemented in one programming language will work, without modification, with applications or other objects implemented in another language. In addition, because OLE components are language-independent, they are not constrained by language-based requirements to operate in the same address space as the objects or applications using them. As a result, OLE components provide the basis for sharing information among separate objects in different processes on the same machine or on different networked computers.

OLE Information Management

Built on top of COM are core facilities for storing and naming objects and for transferring data between them. Together, these facilities form the core of system-wide information management. You can take advantage of these facilities by using OLE's supplied implementations, or you can extend or replace them with your own. These core facilities include structured storage, monikers, and uniform data transfer. Together they provide the infrastructure required for objects to interrelate across process, machine, and network boundaries.

OLE's structured storage model defines an architecture for storing and retrieving objects that reside inside files or other containers. The basic problem addressed by the structured storage model is how to write an object to and retrieve an object from any persistent storage when that object resides inside the flat file space of some container. OLE solves the problem by providing interfaces that support *storage* and *stream* objects. A storage object provides a hierarchical structure for mapping the location of storage and stream objects just as directories provide a way of locating files on disk. A stream object contains the object's data, or contents. Using structured storage is mandatory if you want to support compound documents or other forms of containment. OLE also provides a default implementation of structured storage called *compound files.*

OLE supports the persistent naming of objects through a mechanism called a *moniker.* A moniker provides a way for clients of an object to locate that object even when it resides in a different process or on a different computer. Objects have a file moniker, which is similar to their absolute pathnames, but their clients normally bind to them using a relative moniker*,* which indicates the object's location relative to the client. In this way, if client and object are both relocated, they can maintain their connection as long as their locations relative to each other remain the same. In general, a moniker also enables clients to locate and connect to some portion of an object−a certain range of cells in a spreadsheet, for example−by providing an *item moniker,* by which that *pseudo-object* is always known.

OLE's uniform data transfer model provides a single, standard mechanism for transferring objects and data between applications. Through a single interface, OLE supports standard Microsoft® Windows® clipboard transfers, as well as drag and drop, and embedded objects. Uniform data transfer greatly enhances the clipboard model by providing rich structures for describing the format of the data to be transferred and the storage medium in which a transfer is to occur. For example, instead of always having to pass data in a global memory handle, you can choose from among several storage media, including a disk-based file and pointers to OLE storage and stream interfaces.

Built on top of COM, structured storage, monikers, and uniform data transfer are the three technologies for which OLE is best known: compound documents, OLE automation, and OLE controls. For many software vendors, the desire to incorporate one or both of these technologies in their applications, or to make sure their applications work easily with operating systems and other applications that do, has been the prime motivation for turning to OLE.

Describing Compound Documents

A *compound document* is one that contains, along with its native data, one or more objects that were created in other applications and therefore have different data formats. Such objects are called *compound document objects.* An application used to create compound document objects is called an *OLE server application.* An application whose documents act as object clients is called a *container application* or, simply, a *container*. Any given application can be one, or the other, or both.

Compound document objects are essentially OLE components that also implement interfaces that support object linking and embedding. A linked object includes information about the location of its data and the formats necessary to present it on screen to the user, but the data itself is stored elsewhere. An embedded object is one whose data is stored along with that of its container. Applications that support linking and embedding can share data without having knowledge of one another's data structures and without having to implement separate protocols for every other application with which they communicate. Users of OLE compound document applications can embed data created in one application in a document created in another, and can view, edit, or otherwise manipulate that data without having to exit the application in which they are working. Users can also create links in one document to data in another so that changes to data in the source document are updated in the link.

Defining OLE Automation

OLE automation is the ability for one application or tool to manipulate programmatically objects exposed by another application. Using OLE automation, you can also create tools that access and manipulate objects. Such tools can include embedded macro languages, programming tools, object browsers, and compilers. Like compound documents, OLE automation is based on COM, but applications can implement OLE automation independently of compound documents or other OLE technologies.

Where to Find Additional Information

The current release of *OLE Programmer's Guide* examines the architecture and capabilities of OLE as a whole, as well as of each of its core facilities: COM, structured storage, and uniform data transfer. A separate chapter is devoted to compound documents, the technology that inspired the original development of OLE and laid the groundwork for its evolution into a systemwide, object-based service architecture. A subsequent release of the *OLE Programmer's Guide* will include detailed instructions for developing both OLE client and server applications for compound documents that work under both Microsoft® Windows NT™ and Microsoft® Windows® 95. For descriptions of specific interfaces, helper functions, data structures, and enumerations, refer to the *OLE Programmer's Reference,* Volume 1. For information on OLE automation and developing OLE controls, see the *OLE Programmer's Reference,* Volume 2 *Automation*.

Guide

Appendixes

Programming Considerations

While detailed application-specific information is beyond the scope of this appendix, this section contains some helpful guidelines for developing OLE applications.

Designing Component Objects

Component objects contain data specific to the object and one or more interface implementations. The data is private and inaccessible from outside the object, while the interface implementations are public and you can access them through pointers.

Because interfaces are a binary standard, interface implementation is language independent. However, C++ is the preferred language because it supports many of the object-oriented concepts inherent in OLE. Using a procedural language such as C involves extra work, as summarized below:

- You must explicitly initialize VTBLs either at compile-time or run-time and you should not change them later. Once initialized with function pointers, those pointers should remain unchanged until the application shuts down. VTBLs in C++ are declared as constants to prevent them from being modified inadvertently. However, in C there is no way to ensure that a VTBL will remain unchanged.

  To overcome this difference, OLE provides a mechanism that lets C developers declare VTBLs as constants. To do so, place the following statement before the *#include* statement for the "ole2.h" header file:

  ```
  #define CONST_VTABLE
  ```

- Each method requires a pointer to the object that owns the method. In C++, all members are implicitly dereferenced off the *this* pointer. In C, there must be an additional first parameter passed to each method that is a pointer to the interface in which the method is declared.
- Methods in C++ can have identical names because methods are actually known by a name that is the result of concatenating the method name to the class name. Methods in C must have a unique name to designate the object with which they are associated.

For example, the following C++ code sample defines an implementation of **IUnknown::QueryInterface**. The method name is **QueryInterface** and there are two parameters: a REFIID and a pointer to where to return the requested interface instance.

```
CUnknown::QueryInterface (REFIID riid, LPVOID * ppvObj);
```

A similar C implementation would require a more complex name and an additional first parameter to indicate the object owning the method:

```
IUnknown_Doc_QueryInterface (LPUNKNOWN pUnk, REFIID riid,
     LPVOID * ppvObj);
```

The following sections demonstrate how to declare a component object in a few typical ways: using C nested data structures, C++ nested classes, and C++ multiple inheritance. The demonstration object, called *CObj*, derives from **IUnknown** and supports two interfaces that also derive from **IUnknown**, **InterfaceA** and **InterfaceB**. *CObj's* private data includes a pointer to another component object in the application (*m_pCDoc*), a count of all the external references to the object (*m_ObjRefCount*), and pointers to two interfaces implemented by other component objects and used by *CObj (m_pOleObj* and *m_pStg)*. All object members use the *m_* prefix to make it easy to distinguish between member variables and other variables.

Component Objects: C Nested Structures

C interface implementations comprise data structures nested within the object's data structure. Each interface structure contains a VTBL pointer as its first member (*pVtbl*), a pointer to the object (*pCObj*), and a count of the external references to the interface (*m_RefCount*). The order of the members in the interface structures is identical, to facilitate code sharing.

```
typedef struct CObj {
  ULONG            m_ObjRefCount;
  LPSTORAGE        m_pStg;
  LPOLEOBJECT      m_pOleObj;
  struct CDOC    *  m_pCDoc;

  struct InterfaceA  {
    LPVTBL        pVtbl;
    struct CObj  *  pCObj;
    ULONG        m_RefCount;
  } m_InterfaceA;

  struct InterfaceB {
    LPVTBL       pVtbl;
    struct Obj  *   pCObj;
    ULONG        m_RefCount;
  } m_InterfaceB;

} COBJ;
```

Component Objects: C++ Nested Classes

The next example shows how the same object is declared and initialized using the C++ nested class approach. As in the C example, the nested class declaration includes one data structure for each interface and four private data members: an object-level reference count, two interface pointers, and a pointer to the enclosing object. The private implementations of the **IUnknown** methods are called by the implementations declared for the derived interfaces. For each interface implementation, there is a structure containing a public constructor and destructor, private declarations of the interface methods, a private pointer to *CObj*, and an interface-level reference counter for debugging purposes. To allow the nested interface classes to access the private members of the outer class, each interface class is made a friend of the outer class.

The benefits of implementing with C++ nested classes lie in the ability to include initialization code and method implementation inline. However, inline declaration is for the convenience of illustration and is not required.

```
class  CObj  {

private:
  ULONG          m_ObjRefCount;
  LPSTORAGE      m_pStg;
  LPOLEOBJECT    m_pOleObj;
  CDOC    *      m_pCDoc;

public:
  CObj();
  ~CObj();

  struct CUnknown : IUnknown
{
  private:
  ULONG          m_RefCount;
  CObj  *        m_pCObj;

  public:
  CUnknown(CObj (pCObj)
  { m_pCObj = pCObj; m_RefCount = 0; }
  HRESULT QueryInterface(REFIID riid, LPVOID  * ppvObj)
  ULONG AddRef(void) { return ++m_ObjRefCount; }
  ULONG Release(void);
}
friend CUnknown;
CUnknown m_Unknown;

struct InterfaceA : InterfaceA
{
  private:
  ULONG          m_RefCount;
  CObj  *        m_pCObj;

  public:
  CInterfaceA(CObj *pCObj)
  { m_pCObj = pCObj; m_RefCount = 0; }
  HRESULT QueryInterface(REFIID riid, LPVOID  * ppvObj)
  ULONG AddRef(void) { return ++m_ObjRefCount; }
  ULONG Release(void);
  HRESULT MethodA1(LPVOID  * ppvObj);
  HRESULT MethodA2(DWORD dwArg);
}
friend CInterfaceA;
CInterfaceA m_InterfaceA;

struct InterfaceB : InterfaceB
{
  private:
  ULONG         m_RefCount;
  CObj  *       m_pCObj;

  public:
  CInterfaceB(CObj *pCObj)
  { m_pCObj = pCObj; m_RefCount = 0; }
  HRESULT QueryInterface(REFIID riid, LPVOID  * ppvObj)
  ULONG AddRef(void) { return ++m_ObjRefCount; }
  ULONG Release(void);
  HRESULT MethodB1(void);
  HRESULT MethodB2(DWORD dwArg1, DWORD dwArg2);
}
friend CInterfaceB;
CInterfaceB m_InterfaceB;
```

Multiple Inheritance

The next example illustrates the use of C++ multiple inheritance. There are two disadvantages to using multiple inheritance with OLE. First, it is not possible to have an interface-level reference count. For more information about reference counting, see Chapter 2, "The Component Object Model." Second, there is the potential for confusion over the interpretation of the class statement. A standard C++ multiple inheritance declaration implies the "is a" relationship where an object inherits implementations. In OLE, however, interfaces are attributes of the object and implementations are not inherited.

The main advantage to using multiple inheritance lies in its simplicity. Only the prototypes for each of the interface methods are listed; no interface data structures or class definitions are necessary.

Because both InterfaceA and InterfaceB inherit from **IUnknown**, it is not necessary to list **IUnknown** explicitly in the class statement. A single implementation of the **IUnknown** methods (**QueryInterface**, **AddRef**, and **Release**) is sufficient.

```
class  CObj : public InterfaceA, public InterfaceB
{
private:
  ULONG          m_ObjRefCount;
  LPSTORAGE      m_pStg;
  LPOLEOBJECT    m_pOleObj;
  CDOC  *        m_pCDoc;

public:
  CObj();
  ~CObj();

  HRESULT QueryInterface(REFIID riid, LPVOID  * ppvObj)
  ULONG AddRef(void) { return ++m_ObjRefCount; }
  ULONG Release(void);

  HRESULT MethodA1(LPVOID  * ppvObj);
  HRESULT MethodA2(DWORD dwArg);

  HRESULT MethodB1(void);
  HRESULT MethodB2(DWORD dwArg1, DWORD dwArg2);
};
```

Aggregating Objects

An aggregate is an object like any other object in that it implements one or more interfaces. Internal to the aggregate, the implementation of certain interfaces is actually provided by one or more contained objects. Users of the object are unaware of this internal structure. They cannot tell and do not care that the object is an aggregate: aggregation is purely an implementation technique.

The figure below shows an aggregate object consisting of a control object that implements Interface A and Interface B, and a noncontrol object that implements Interface C. All these interface implementations are exposed publicly, as indicated by the line and circle extending to the outside of the aggregate.

{ewc msdncd, EWGraphic, group10505 0 /a "SDK.wmf"}

In the aggregation model, the *control object* determines how an aggregate behaves and operates, making decisions about which interfaces are exposed outside of the object and which interfaces remain private. The control object has a special instance of the **IUnknown** interface known as the *controlling unknown*. The controlling unknown must always be implemented as part of the new code written when the aggregate is put together from other objects. However, when the controlling unknown is passed in, it does not increment the reference count; instead it is a non-incrementing pointer.

The other objects can be implemented at any time. These noncontrol objects can be instantiated separately or as part of the aggregate. However, if these objects are to be capable of aggregation, they must be written to cooperate with the control object by forwarding their **QueryInterface**, **AddRef**, and **Release** calls to the controlling unknown. A reference count for the aggregate as a whole is maintained by the control object so that it is kept alive if there are one or more references to any of the interfaces supported by either the control or noncontrol objects.

It is possible for an object to call a method that may cause the object to be released. A technique known as "artificial reference counting" can be used to guard against this untimely release. The object calls **IUnknown::AddRef** before the potentially destructive method call and **IUnknown::Release** after it. If the object in question can be aggregated, it must call the controlling unknown's implementations of **AddRef** and **Release** (*pUnkOuter*->**AddRef** and *pUnkOuter*->**Release**) to artificially increment the reference count rather than its own implementations. For more information about reference counting, see Chapter 2, "The Component Object Model."

The next figure shows how the coordination between the control and noncontrol object works. The control object exposes the controlling unknown, Interface A and Interface B. The noncontrol object supports **IUnknown** and Interface C. All three interfaces derive from **IUnknown**.

The control object holds a pointer to the noncontrol object's **IUnknown** implementation so it can call the noncontrol methods when appropriate. The noncontrol object holds a pointer to the controlling unknown for the same reason. The pointer to the controlling unknown is supplied when the noncontrol object is instantiated. When the controlling unknown pointer is saved by the noncontrol object, **AddRef** is not called to increment the reference count as you might expect. This is because of the reference-counting cycle it would create. Instead, the noncontrol object just keeps a copy of the pointer and delegates calls to it as described in this section.

Referring again to the next figure, whereas the controlling unknown is available to the outside, as is indicated by the line and circle identifying it extending past the bounds of the aggregate, the noncontrol object's **IUnknown** implementation works locally and can't be obtained from outside the aggregate. It is called solely by the controlling unknown to obtain instances of the noncontrol interfaces, such as Interface C, to expose to the outside.

{ewc msdncd, EWGraphic, group10505 1 /a "SDK.wmf"}

Aggregation and Cached Interface Pointers

Interface-based reference counting has generally been a preferable way of keeping OLE objects stable, although 16-bit aggregates have proven troublesome. However, 32-bit OLE applications can overcome this difficulty with relative ease. For example, when a control object holds an interface pointer to a noncontrol object and the reference count has not been incremented, the pointer is said to be cached. In 16-bit OLE applications, this is achieved by calling **QueryInterface** to get the cached pointer and then releasing it in the following manner:

```
// 16-bit method for caching pointers to interfaces on noncontrol
// objects
IInterface1 *m_pi1;
m_pUnkInner->QueryInterface(IID_Interface1, &m_pi1);
m_pi1->Release();
```

To facilitate the use of per-interface reference counting in 32-bit OLE, the method for caching interface pointers has changed, as shown in the following:

```
// 32-bit method for caching pointers to interfaces on noncontrol
// objects.
IInterface1 *m_pi1;
m_pUnkInner->QueryInterface(IID_Interface1, &m_pi1);
m_pUnkOuter->Release();       \\ Uses the same m_pUnkOuter given to
                              \\ the aggregated object.
```

The difference is in the third line of code. Furthermore, when a cached pointer is released, include the following lines (no equivalent actions are done for 16-bit applications at the same point):

```
m_pUnkOuter->Addref();
m_pi1->Release();
```

Note These two approaches are not compatible; all 16-bit aggregate control objects must use the first method and all 32-bit aggregate control objects must use the second one.

Note that interface-based reference counting works best when control objects delay the acquisition of cached pointers and release them when there is no possible use for them.

During the destruction of any aggregate, even those in which the control object does not cache pointers, the control object must guard the destructor by means of artifical reference counting, as shown below:

```
ULONG Release_ObjectXXX(ObjectXXX *this)
{
     if(- -this->m_refs;
     {
           ++this->m_refs;
           ...destroy the instance..
     return 0;
     }
     return this->m_refs;
}
```

This code must be included because the sequence **m_pUnkOuter->AddRef()** followed by **m_pi1->Release** increments and decrements the reference count of the aggregate as a whole. Since this may occur in the destruction of the aggregate itself, the destructor must be guarded.

Note You cannot use this technique for noncontrol objects to hold cached pointers to objects higher up in the aggregate. The noncontrol object has no way of determining whether it has the same *pUnkOuter* as any other object in the aggregate. Although this is usually the case, it cannot be guaranteed.

Aggregation is entirely optional, and you should decide whether to use it as you design each object. However, providing support for aggregation costs little in time and complexity. The reward is larger than the cost in that aggregation enables the effective re-use of interface implementations. The key in implementing an aggregate object is making the correct delegation decision in both the aggregated and non-aggregated situations.

The following code example is an aggregate object implemented using C++ nested classes. The constructor takes as an argument the pointer to the control object's **IUnknown** implementation, or controlling unknown. If this pointer is NULL, indicating that the new object is not being aggregated, the *m_pUnkOuter* member is set to the address of *m_Unknown*, the local implementation of **IUnknown**. If *pUnkOuter* is not NULL, indicating that the new object is being aggregated, *m_pUnkOuter* is set to *pUnkOuter*. The Interface A implementations of the **IUnknown** methods forward their calls to the controlling unknown implementations. The controlling unknown **IUnknown** implementation manages queries for interfaces for the aggregate and maintains a master reference count.

```
class CAggregateObj
{
public:
  static HRESULT CreateObj(LPUNKNOWN pUnkOuter, REFIID riid,
        LPVOID  * ppvObj)
  {
    HRESULT hr;
    CAggregateObj *pObj = new CAggregateObj(pUnkOuter);

    if (pObj == NULL)
    {
      *ppvObj = NULL;
      return E_OUTOFMEMORY;
    }

    if ((hr = pObj->m_Unknown.QueryInterface(riid, ppvObj)) !=
NOERROR)
          // QI call above will set *ppvObj on success
          // and NULL it on error
delete pObj;

    return hr;
  }

private:
  CAggregateObj(LPUNKNOWN pUnkOuter)
    : m_Unknown(this), m_InterfaceA(this)
  {
    if (pUnkOuter == NULL)
          // not aggregated; make like the
          // aggregated case
      pUnkOuter = &m_Unknown;
    m_pUnkOuter = pUnkOuter;
    m_cRefs = 0;
  }
  ~CAggregateObj();

  struct CUnknown : IUnknown
  {
    CUnknown(CAggregateObj  * pAgg)
    { m_pAgg = pAgg; }

    HRESULT QueryInterface(REFIID riid, LPVOID  * ppvObj);

    ULONG AddRef(void)
    { return ++m_pAgg->m_ObjRefCount; }

    ULONG Release(void)
    {
      if (--m_pAgg->m_cRefs  == 0)
      {
          // don't need guard here since this object
          // doesn't aggregate another one
        delete m_pAgg;
        return 0;
      }

      return m_pAgg->m_cRefs;
    }
    private:
    CAggregateObj  * m_pAgg;
  }
  friend CUnknown;
  CUnknown m_Unknown;

  struct CInterfaceA : InterfaceA
  {
    CInterfaceA(CAggregateObj  * pAgg)
    { m_pAgg = pAgg; }

    HRESULT QueryInterface(REFIID riid, LPVOID  * ppvObj)
    { return m_pAgg->m_pUnkOuter->QueryInterface(riid, ppvObj); }

    ULONG AddRef(void)
    { return m_pAgg->m_pUnkOuter->AddRef(); }

    ULONG Release(void)
    { return m_pAgg->m_pUnkOuter->Release(); }

    HRESULT MethodA1(LPVOID  * ppvObj);
    HRESULT MethodA2(DWORD dwArg);

    private:
    CAggregateObj  * m_pAgg;
  }
  friend CInterfaceA;
  CInterfaceA m_InterfaceA;

private:
  ULONG      m_cRefs;
  LPUNKNOWN    m_pUnkOuter;
}
```

Grouping Interfaces

Interface relationships are based on the ability to successfully obtain interface pointers by using **IUnknown::QueryInterface**. Although interface implementors are advised to design objects in such a way that interfaces are accessible from many other interfaces, there is generally no guarantee to a user that a specific interface is accessible from another interface. Implementations of **QueryInterface** are application-specific, so it usually cannot be assumed that it returns a pointer to another specific interface.

Two interfaces can be related in one of three possible ways. Each of these is set forth in the following list, using Interfaces named A and B as examples. Notice the importance of reference counting in managing the three types of relationships.

1. Given a pointer to Interface A, it **should** be possible to obtain a pointer to Interface B. Interface A and Interface B are managed by a single reference count. OLE defines the following interfaces that fall in this category:
   - **IOleObject**, **IDataObject**, **IViewObject**, **IOleCache**, **IOleInPlaceObject**, and **IPersistStorage**. If the object is a link, **IOleLink** should also be available.
   - **IPersistFile** and **IOleItemContainer**
   - **IOleClientSite** and **IOleInPlaceSite**
2. Given a pointer to Interface A, it **must not** be possible to obtain a pointer to Interface B. Interface A and Interface B can be managed by either a single reference count or separate reference counts. **IOleInPlaceActiveObject** and **IOleInPlaceObject** fall in this category. **IStorage** and **IStream**, typically implemented by OLE, also fall in this category, simply because they point to different objects.
3. Given a pointer to Interface A, it might be possible to obtain a pointer to Interface B, depending on the implementing application. Interface A and Interface B can be managed by a single reference count. Clients of the interfaces **cannot** assume it is always possible to obtain the same set of interfaces in this category from all implementing applications. The decision to group these interfaces is up to the implementor, and there is no guarantee they will be implemented in the same manner by all applications.

All OLE interfaces not specifically mentioned as belonging to the first or second groups belong to the third type. These interfaces can be grouped in a manner that works best for the application. It is wise to implement interfaces together whenever possible, because doing so leads to greater flexibility and interconnectivity. Implementing interfaces together implies they are within a single object supported by one piece of memory. The interfaces share one **IUnknown::QueryInterface** implementation that recognizes all related interfaces. Pointers to the interfaces are managed by a single reference count.

An object that has multiple groups of interfaces, each managed by its own reference count, is kept alive if there is at least one reference to any interface in one of the groups.

There are two possible approaches to interfaces that cannot be implemented together. They can be implemented either in separate objects, or in a single object that manages the interfaces independently. In the single object approach, there can be one **IUnknown::QueryInterface** implementation or several, depending on the number of interface groups in the object. In the single **QueryInterface** implementation, explicit steps must be taken to return pointers only to related interfaces.

Converting Mapping Modes

In Windows, mapping modes define how numbers relating to object sizes are to be passed and interpreted. The 32-bit version of OLE uses only one mapping mode, MM_HIMETRIC. The other Win32 mapping modes include MM_HIENGLISH, MM_LOENGLISH, MM_LOMETRIC, TWIPS, and PIXEL Modes (MM_ISOTROPIC and MM_ANISOTROPIC being PIXEL modes). When working with the visual presentation of OLE data, it is important to be aware of the mapping mode that OLE uses and how this mapping mode affects an application.

The mapping modes communicate physical sizes. For example, if an application using MM_HIMETRIC is to display a line ten centimeters long, the number of units would be 10,000. However, the line drawn on the screen would be ten centimeters long, regardless of the size of the video display area. The printed output would also be a line ten centimeters long.

Note For those applications that use a mapping mode other than MM_HIMETRIC, the sample user interface library provides some functions that can be used to convert objects to and from MM_HIMETRIC units.

Because people read display screens from a greater distance than they do printed copy, most applications written for Windows display text in a larger size than they print it, using what is commonly referred to as *logical resolution*. For example, a ten-point font is easy enough to read on the printed page, but generally appears too small on a screen for comfortable reading. To afford more comfortable viewing, applications typically expand the size of the displayed text to some logical size. Using this approach, a column of text that is physically six inches wide might be eight inches wide on the screen, yet still print as a six-inch column.

While this display-enlargement scheme works well from the user's point of view, a problem can occur when pasting objects into container documents. It is possible to lose the correct size ratio between the pasted object and the text owned and displayed by the container. The result is that the container's text is scaled up for readability but the pasted object might not be. Consequently, applications must preserve the relative size and position of text and objects, meaning that if text uses logical resolution, it should scale objects accordingly.

In the examples shown in the figure below, a chart object has been pasted from a source application that uses physical size into documents of two different containers that use logical resolution for display of the text. The container displaying the object on the left has scaled up the chart object to the logical size of the adjacent text to maintain the object/text size ratio. That is, it has been enlarged from its physical size by an amount that maintains its proportion to the text of the container document. In the document on the right, the application displays the chart object at its physical size, with no scaling to logical resolution for display. Both documents print with the correct object-to-text-size ratio.

{ewc msdncd, EWGraphic, group10505 2 /a "SDK.wmf"}

In OLE, the units for specifying the size of drawn objects is MM_HIMETRIC, which means object sizes are in physical units. However, containers need not use the MM_HIMETRIC mapping mode to draw pasted objects to the display. Rather, they should should map objects to the screen in the same manner as text. That is, if the container application displays text to the screen using a mapping mode that enlarges it, objects should be mapped to the display in the same manner. Using the same mapping mode for both the text and objects is required to establish the correct object-to-text ratio as shown in the document on the left side of the preceding figure. Because most Windows applications use logical resolution for this type of display mapping, we suggest that OLE containers also use logical resolution and set up their mapping modes and coordinate transforms accordingly. This allows objects to be moved from one application to another without changing their displayed size.

OLE Property Sets

Property sets store information in such a way that any conforming program can later manipulate the information. Examples of property sets are the character-formatting properties in a word processor or rendering attributes of an element in a drawing program. OLE does not provide any code or interfaces for manipulating property sets, it only specifies a standard data format structure.

Structure of Property Sets

Property sets are made up of a tagged section of values, with the section uniquely identified by a Format Identifier (FMTID). Every property consists of a *property identifier* and a *type indicator* that represents a *value*. Each value stored in a property set has a unique property identifier that names the property. The type indicator describes the representation of the data in the value. For example, if a specific property in a property set holds an animal's scientific name, that name could be stored as a zero-terminated string. Stored along with the name would be a type indicator to indicate the value is a zero-terminated string. The representation of properties is shown below:

Property Identifier	**Type Indicator**	**Value Represented**
PID_ANIMALNAME	VT_LPSTR	Zero-terminated string
PID_LEGCOUNT	VT_I2	WORD

Any application that recognizes the property set format can look at the property with an ID of PID_ANIMALNAME, determine it is a zero-terminated string, and read and write the value. Of course, the given application might not have any information about the property. The standard structure defined by OLE for property sets is generic as it relates to the semantics of the properties being represented; it says nothing about what they mean.

Each property set can have a dictionary associated with it to provide a displayable name for each property ID in the set. An application can read and write this dictionary and can allow end users to choose the displayable names for property IDs.

Examples of Using Property Sets

To illustrate the diverse functionality of property sets, this section presents two samples. The first uses the OLE Document Summary Information property set to show how property sets can be stored within files to allow common access to the information represented. The second example shows how property sets can be transferred between applications or OLE objects as an effective means of communication.

Storing Document Summary Information

The Document Summary Information property set is one of the simplest and most commonly used property sets. Most documents created by applications have a common set of attributes that are useful to users of those documents. These attributes include the name of the document's author, the subject of the document, when it was created, and so on.

In Windows 3.1, each application has a different way of storing this information within its documents. To examine the summary information for a given document, the user has to run the application that created the document, open it, and invoke the application's Summary Information dialog box. For example, the summary information dialog boxes that Microsoft Word 2.0 displays for its documents are shown in the following drawings:

{ewc msdncd, EWGraphic, group10506 0 /a "SDK.wmf"}

{ewc msdncd, EWGraphic, group10506 1 /a "SDK.wmf"}

Unfortunately, no application but Microsoft Word 2.0 can display the summary information for a Microsoft Word 2.0 document. However, with OLE property sets, this limitation can be overcome. This appendix describes one standard property set, the OLE Document Summary Information Property Set. Any application that understands the data format for this property set will be able to access the summary information contained in any document created by an OLE application that also understands the format.

For example, Microsoft Word 6.0 and many other OLE-enabled applications now save their documents using OLE structured storage and the property set standard described here. Thus, applications other than Word 6.0 are able to display the summary information property set for such a file, as long as that file is an OLE structured storage file, and the creating application saved the information in the OLE Property Set format.

For more information about the Document Summary Information property set and implementing it in your applications, see "Guidelines for Implementing the Document Summary Information Property Set" later in this appendix.

Transferring Data Contained in Property Sets

Storing information about the contents of a document is useful, and a standard that would allow all files to share the same information with all applications would be even better. However, this is only one example of how property sets can be used.

Another use for them is in the transfer of data between OLE objects or applications. To better understand the process of transferring data between two OLE objects, consider the following example. A trader on Wall Street relies on large amounts of data to make important financial decisions regarding trades of securities. This market data must be delivered to the trader's workstation in real time (or at least very soon after it was generated).

Assume that, at any given moment during an open day on Wall Street, the trader is interested in the opening, high, low, current, and last closing price of an instrument (a specific stock), and the volume of trading on that stock so far that day. The symbol for the stock will also be needed.

As noted earlier, a property in a property set has three attributes: the property identifier, a value type indicator, and a value. In this example, the following properties can be defined:

Property Identifier	Type Indicator	Value Represented
PID_SYMBOL	VT_LPSTR	Zero-terminated string
PID_OPEN	VT_CY	Currency value
PID_CLOSE	VT_CY	Currency value
PID_HIGH	VT_CY	Currency value
PID_LOW	VT_CY	Currency value
PID_LAST	VT_CY	Currency value
PID_VOLUME	VT_I4	32-bit unsigned integer

Without worrying about how the data is transferred between the data server and the trader's application, think about the data transferred and its format. Every property set must have a FMTID associated with it.

A single data element representing the stock of XYZ during the day might look like this:

```
PID_SYMBOL  contains "XYZ"
PID_OPEN    contains 42-3/4
PID_CLOSE   contains 42-3/4
PID_HIGH    contains 49-1/8
PID_LOW     contains 40-7/8
PID_LAST    contains 47-3/8
PID_VOLUME  contains 123,032
```

During a trading day on Wall Street only a few of the properties of a single instrument change. The opening price doesn't change all day, and the low price may go unchanged for several hours. Using property sets as the data transfer format allows applications to transfer only the data that has changed. For example, at some point during the day the Data Server may update the Traders Application by transferring the following StockQuote property set:

```
PID_SYMBOL  contains "XYZ"
PID_HIGH    contains 49-1/8
PID_LAST    contains 49-1/8
PID_VOLUME  contains 23,321
```

This example demonstrates how property sets can be used as a data transfer format, allowing sparse data representation. Transferring only the changed data enhances the overall performance of the trader's workstation.

Other Uses for Property Sets

Just about any object in the system can be tagged with properties. For example, a printer object can have a property set that includes such information as Location. A user object can have a property set including information like First Name, Last Name, Office, and Phone.

Applications can be written to query a system-wide set of objects based on their properties, for example, displaying all printers located in a certain building.

These examples illustrate the potential power of using property sets.

Notes on Using Property Sets

OLE property sets were designed to store data that is suited to representation as a moderately-sized collection of fine-grained values. Data sets that are too large for this to be feasible should be broken into separate streams and/or storages. The OLE property set data format was not meant to provide a substitute for a database of millions of tiny objects.

OLE Property Set Specification

All data elements within a property set are stored in Intel representation (that is, in little-endian byte order).

OLE defines a standard, serialized data format for property sets, which gives them the following characteristics:

- Property sets allow for different applications to create their own independent property sets to serve the application's needs.
- Property sets can be stored in a single **IStream** instance or in an **IStorage** instance containing multiple streams. Indeed, in the abstract, property sets are simply another data type that can be stored in many different forms of an in-memory or on-disk storage. For recommended conventions on creating the string name for the storage object, see the section "Naming Property Sets" later in this appendix.
- Property sets can be transmitted using the **IDataObject** interface and/or the Clipboard. Because property sets are self-identifying, they are ideal for transferring data between applications.
- Property sets allow for a dictionary of displayable names to be included to further describe the contents. A set of conventions for choosing property names are recommended. For more information on this optional dictionary, see "Property ID Zero" later in this appendix.

The figure below shows the overall structure of a property set:

The property set stream is divided into three major parts:

- Header
- FORMATID/offset pair
- Section containing the actual property set values

The overall length of the property set stream is 256Kb. The following sections of this chapter describe the individual components that make up the property set data format as shown in the previous figure.

Note Previous versions of this document described extensions to the property set stream with more than one section allowed, but that has been revised now to provide for one section in the property stream.

Property Set Header

At the beginning of the property set stream is a header. It consists of a byte-order indicator, a format version, the originating operating system version, the CLSID, and a reserved field. Each offset is the distance in bytes from the start of the whole stream to where the section begins.

The following pseudo-structure illustrates the header:

```
typedef struct tagPROPERTYSETHEADER
{
     // Header
     WORD  wByteOrder ;  // Always 0xFFFE
     WORD  wFormat ;     // Always 0
     WORD  dwOSVer ;     // System version
     CLSID  clsID ;      // Application CLSID
     DWORD  reserved ;   // Should be 1
} PROPERTYSETHEADER;
```

Byte-Order Indicator

The byte-order indicator is a WORD and should always hold the value 0xFFFE. This is the same as the Unicode byte-order indicator. This value is always written in Intel byte order and, thus, appears in the file or stream as 0xFE, 0xFF.

Format Version

The Format Version is a WORD used to indicate the format version of this stream. It should always be zero. The format-version indicator should be checked when reading the property set. If it is not zero, then the stream was written to a different specification and cannot be read by code developed according to the OLE 2 specification.

Originating OS Version

This DWORD should hold the kind of operating system in the high word and the version number of the operating system in the low word. Possible values for the operating system are:

Operating System	Value
32-Bit Windows (Win32)	0x0002
Macintosh	0x0001
16-Bit Windows (Win16)	0x0000

For Windows, the operating system version is the low order word returned by the **GetVersion** function. On Windows, the following code would correctly set the version of the originating operating system:

```
#ifdef WIN32
dwOSVer = (DWORD)MAKELONG( LOWORD(GetVersion()), 2 ) ;
#else
dwOSVer = (DWORD)MAKELONG( LOWORD(GetVersion()), 0 ) ;
#endif
```

CLSID

The CLSID is that of a class that can display and/or provide programmatic access to the property values. If there is no such class, it is recommended that you set this value the same as the Format ID (see below). Alternately, you can set this value to all zeroes; however, using the Format ID allows more flexibility in the future.

Reserved

This DWORD is reserved for future use. Writers of property sets should set this value to 1; readers of property sets should ensure that this value is at least 1.

Format Identifier/Offset Pair

The second part of the property set stream contains one Format Identifier (FMTID)/Offset Pair. The FMTID is the name of the property set; it uniquely identifies how to interpret the contents of the following section. The Offset is the distance of bytes from the *start of the whole stream* to where the section begins.

The following structure is helpful in dealing with Format Identifier/Offset Pairs:

```
typedef struct tagFORMATIDOFFSET
{
     FMTID  fmtid ;     // The name of the section.
     DWORD  dwOffset ;  // The offset for the section
} FORMATIDOFFSET;
```

Format Identifiers

Property set values are stored in a section that is tagged with a unique format identifier. For example, the FMTID for the OLE Document Summary Information property set is:

```
F29F85E0-4FF9-1068-AB91-08002B27B3D9
```

To define a FMTID for the Document Summary Information property set, you would use the **DEFINE_GUID** macro in an include file for the code that manipulates the property set:

```
DEFINE_GUID(FormatID_SummaryInformation, 0xF29F85E0, 0x4FF9, 0x1068, 0xAB,
0x91, 0x08, 0x00, 0x2B, 0x27, 0xB3, 0xD9);
```

Anywhere in your code you need to use the FMTID for the Document Summary Information property set, you can access it through the FormatID_SummaryInformation variable.

When storing property sets in **IStorage** instances, you need to convert the FMTID to a string name for the storage object. For more information on recommended conventions for creating the string name, see the section "Naming Property Sets" later in this appendix.

Allocating Format Identifiers

FMTIDs are created and represented in the same way as OLE CLSIDs and interface IDs. To create a unique FMTID, use the UUIDGEN.EXE program included in the *Win32 SDK*.

Section

This is the third part of the property set stream, as shown in Figure C.2. A section contains:

- byte count for the section (which is inclusive of the byte count itself)
- count of the property values in the section
- array of 32-bit Property ID/Offset pairs
- array of property Type Indicators/Value pairs

Offsets are the distance from the start of the section to the start of the property (type, value) pair. This allows a section to be copied as an array of bytes without any translation of internal structure.

The following pseudo-structures illustrate the format of a section:

```
typedef struct tagPROPERTYSECTIONHEADER
{
     DWORD  cbSection ;    // Size of Section
     DWORD  cProperties ;  // Count of Properties in section
} PROPERTYSECTIONHEADER;

typedef struct tagPROPERTYIDOFFSET
{
     DWORD  propid;   // Name of property
     DWORD  dwOffset; // Offset from start of section to that property
} PROPERTYIDOFFSET;

typedef struct tag SERIALIZEDPROPERTYVALUE
{
     DWORD  dwType;  // Property Type
     BYTE  rgb[];    // Property Value
} SERIALIZEDPROPERTYVALUE ;
```

Size of Section

This DWORD indicates the size of the section. Because the section size is the first four bytes, you can copy sections as an array of bytes.

For example, an empty section (one with zero properties in it) would have a byte count of eight (the DWORD byte count and the DWORD count of properties). The section itself would contain the eight bytes:

```
08 00 00 00 00 00 00 00
```

Count of Properties

This DWORD gives a count of the property values in the section. A property set may contain any number of property values. Readers must be able to handle the case where there are zero properties.

Property ID/Offset Pairs

Following the Count of Properties is an array of Property ID/Offset Pairs. Property IDs are 32-bit values that uniquely identify a property within a section. The Offsets indicate the distance from the start of the section to the start of the property Type/Value Pair. If the offsets are relative to the section, sections can be copied as an array of bytes.

Property IDs are not sorted in any particular order. Properties can be omitted from the stored property set; readers must not rely on a specific order or range of property IDs.

Type Indicators

After the table of Property ID/Offset Pairs comes the actual properties. Each property is stored as a DWORD type followed by the data value.

Type indicators and their associated values are defined in the OLE header files that are shipped with the *Win32 SDK*.

All Type/Value pairs must begin on a 32-bit boundary. Thus values may be followed with null bytes to align the subsequent pair on a 32-bit boundary. Given a count of bytes, the following code will calculate how many bytes are needed to align on a 32-bit boundary:

```
cbAdd = (((cbCurrent + 3) >> 2) << 2) - cbCurrent ;
```

Within a vector of values, each repetition of value must align with its natural alignment rather than with a 32-bit alignment. In practice, this is only significant for types VT_I2 and VT_BOOL (which have two-byte natural alignment). All other types have four-byte natural alignment. Therefore, a property value with type indicator VT_I2 | VT_VECTOR would be:

- A DWORD element count, followed by
- A sequence of packed two-byte integers with no padding between them.

A property value of type identifier VT_LPSTR | VT_VECTOR would be:

- A DWORD element count (**DWORD** *cch*), followed by
- A sequence of strings (**char** *rgch*[]), each of which may be followed by null padding to round to a 32-bit boundary.

The following table lists the standard OLE-defined property type indicators and their meaning:

Type Indicator	Code	Value Representation
VT_EMPTY	0	None. A property set with a type indicator of VT_EMPTY has no data associated with it; that is, the size of the value is zero.
VT_NULL	1	None. This is like a pointer to NULL.
VT_I2	2	Two bytes representing a 2-byte signed int value. This value will be zero-padded to a 32-bit boundary.
VT_I4	3	Four bytes representing a 4-byte signed int value.
VT_R4	4	Four bytes representing a 32-bit IEEE floating point value.
VT_R8	5	Eight bytes representing a 64-bit IEEE floating point value.
VT_CY	6	Eight-byte two's complement integer (scaled by 10,000). This type is commonly used for currency amounts.
VT_DATE	7	Time format used by many applications, it is a 64-bit floating point number representing days since December 31, 1899. This is stored in the same representation as VT_R8. For example, January 1, 1900 is 2.0, while January 2, 1900 is 3.0, and so on.

VT_BSTR	8	Counted, zero-terminated binary string; represented as a DWORD byte count (including the terminating null character) followed by the bytes of data.
VT_BOOL	11	Two bytes representing a Boolean (WORD) value containing 0 (FALSE) or -1 (TRUE). This type must be zero-padded to a 32-bit boundary.
VT_VARIANT	12	Four-byte indicator followed by the corresponding value. This is only used in conjunction with VT_VECTOR.
VT_I8	20	Eight bytes representing a signed integer.
VT_LPSTR	30	Same as VT_BSTR; this is the representation of most strings.
VT_LPWSTR	31	A counted and zero-terminated Unicode string; a DWORD character count (where the count includes the terminating null character) followed by that many Unicode (16-bit) characters. Note that the count is not a byte count, but a WORD count.

VT_FILETIME	64	64-bit **FILETIME** structure, as defined by Win32:
		typedef struct_FILETIME{
		  DWORD dwLowDateTime;
		  DWORD dwHighDateTime;
		}FILETIME;

VT_BLOB	65	DWORD count of bytes, followed by that many bytes of data. The byte count does not include the four bytes for the length of the count itself; an empty BLOB would have a count of zero, followed by zero bytes. This is similar to VT_BSTR but does not guarantee a null byte at the end of the data.
VT_STREAM	66	A VT_LPSTR (DWORD count of bytes followed by a zero-terminated string that many bytes long) that names the stream containing the data. The real value for this property is stored in a stream object, which is a sibling to the Contents stream. This type is only valid for property sets stored in the Contents stream of a storage object.
VT_STORAGE	67	A VT_LPSTR (DWORD count of bytes followed by a zero-terminated string that many bytes long) that names the storage containing the data. The real value for this property is stored in a storage object, which is a sibling to the Contents stream that contains the property set. This type is only valid for property sets stored in the Contents stream of a storage object.

VT_STREAMED_OBJECT	68	Same as VT_STREAM, but indicates that the stream object named in this property contains a serialized object, which is a CLSID followed by initialization data for the class. The named stream is a sibling to the Contents stream that contains the property set. This type is only valid for property sets stored in the Contents stream of a storage object.
VT_STORED_OBJECT	69	Same as VT_STORAGE, but indicates that the storage object named in this property contains an object. This type is only valid for property sets stored in the Contents stream of a storage object.
VT_BLOB_OBJECT	70	An array of bytes containing a serialized object in the same representation as would appear in a VT_STREAMED_OBJECT (VT_LPSTR). The only significant difference between this type and VT_STREAMED_OBJECT is that VT_BLOB_OBJECT does not have the system-level storage overhead as VT_STREAMED_OBJECT. VT_BLOB_OBJECT is more suitable for scenarios involving numerous small objects.
VT_CF	71	An array of bytes containing a Clipboard format identifier followed by the data in that format. That is, following the VT_CF identifier is the data in the format of a VT_BLOB. This is a DWORD count of bytes followed by that many bytes of data in the following format: a LONG followed by an appropriate Clipboard identifier and a property whose value is plain text should use VT_LPSTR, not VT_CF to represent the text. Notice also that an application should choose a single Clipboard format for a property's value when using VT_CF. For more information, see "Clipboard Format Identifiers," later in this appendix.
VT_CLSID	72	A CLSID, which is a DWORD, two WORDs, and eight bytes.
VT_VECTOR	0x1000	If the type indicator is one of the previous values in addition to this bit being set, then the value is a DWORD count of elements, followed by that many repetitions of the value. When VT_VECTOR is combined with VT_VARIANT (VT_VARIANT *must* be combined with VT_VECTOR) the value contains a DWORD element count, a

DWORD type indicator, the first value, a DWORD type indicator, the second value, and so on.

Examples:

VT_LPSTR | VT_VECTOR has a DWORD element count, a DWORD byte count, the first string data, a DWORD byte count, the second string data, and so on.

VT_I2 | VT_VECTOR has a DWORD element count followed by a sequence of two-byte integers, with no padding between them.

New Type Indicators

There is no provision in the OLE specification for adding new type indicators to the list above. However, new types can be defined by using the VT_VARIANT type combined with the VT_VECTOR flag. For example, assume you want to store the following packed structure in a property set:

```
typedef struct tagPACKED
{
     DWORD  dwValue1 ;  // 32 bit value
     WORD   wFlag ;     // 16 bits of flags
     WORD   wValue2 ;   // 16 bit value
} PACKED ;
```

This 64-bit structure could be stored using VT_VARIANT | VT_VECTOR as follows:

```
DWORD  // dwTypeIndicator = VT_VARIANT | VT_VECTOR ;
DWORD  // dwElementCount = 3 ;
DWORD  // dwTypeIndicator = VT_I4 ;
DWORD  // dwValue1 ;
DWORD  // dwTypeIndicator = VT_I2 ;
WORD   // wFlag ;
DWORD  // dwTypeIndicator = VT_I2 ;
WORD   // wValue2 ;
```

Reserved Property IDs

As a designer of property sets you can use any Property ID for your properties except 0, 1, 0xFFFFFFFE, and all negative values. These Property ID values are reserved for use by applications as follows.

Property ID Zero

To enable users of property sets to attach meaning to properties beyond those provided by the type indicator, property ID zero is reserved for an optional dictionary of displayable names for the property set.

The dictionary contains a count of entries in the list followed by a list of dictionary entries.

```
typedef struct tagDICTIONARY
{
     DWORD  cbEntries ;                    // Count of entries in the list
     ENTRY  rgEntry[ cbEntries ] ; // Property ID/String pair
} DICTIONARY ;
```

Each dictionary entry in the list is a Property ID/String pair. This can be illustrated using the following pseudo-structure definition for a dictionary entry (it's a pseudo-structure because the sz[] member is variable in size):

```
typedef struct tagENTRY
{
     DWORD  dwPropID ; // Property ID
     DWORD  cb ;       // Count of characters in the string
     char  sz[cb];     // Zero-terminated string
} ENTRY ;
```

Note the following about property set dictionaries:

- Property ID Zero does not have a type indicator. The DWORD that indicates the count of entries sits in the type indicator position.
- The count of characters in the string (cb) includes the zero character that terminates the string.
- The dictionary is entirely optional. Not all the names of properties in the set need appear in the dictionary. The dictionary can omit entries for properties assumed to be universally known by clients that manipulate the property set. Typically, names for the base property sets for widely accepted standards are omitted, but special purpose property sets may include dictionaries for use by browsers.
- Property names in the dictionary are stored in the code page indicated by Property ID One (see below). For all code pages other than the Unicode code page (1200), each name is byte aligned. Thus, there is no padding between property names with Property ID Zero. If the Unicode code page is specified, the property name strings must be aligned on 32-bit boundaries.
- Property names that begin with the binary Unicode characters 0x0001 through 0x001F are reserved for future use.

The dictionary is stored as a VT_BLOB | VT_VECTOR value. Each BLOB in the vector contains the prerequisite **DWORD** count of bytes followed by a **DWORD** indicating the Property ID followed by the name (a zero-terminated string).

Example Dictionary

The stock market data transfer example (see "Transferring Data Contained in Property Sets," earlier in this appendix) might include a displayable name of "Stock Quote" for the entire set, and "Ticker Symbol" for PID_SYMBOL. If a property set contained just a symbol and the dictionary, the property set section would have a byte stream that looked like the following:

```
Offset     Bytes
; Start of section
0000    43 01 00 00      ; DWORD size of section
0004    03 00 00 00      ; DWORD number of properties in section

; Start of PropID/Offset pairs
0008    00 00 00 00      ; DWORD Property ID (0 == dictionary)
000C    20 00 00 00      ; DWORD offset to property ID
0010    01 00 00 00      ; DWORD Property ID (1 == code page)
0014    D0 00 00 00      ; DWORD offset to property ID
0018    07 00 00 00      ; DWORD Property ID (i.e., PID_SYMBOL)
001C    34 01 00 00      ; DWORD offset to property ID

; Start of Property 0 (which is really the dictionary
0020    08 00 00 00    ; DWORD Number of entries in dictionary
                       ; (Note: No type indicator)
0024    00 00 00 00        ; DWORD dwPropID = 0
0028    18 00 00 00    ; DWORD cb = (wstrlen("Stock Quote")+1)
                       ;      * sizeof(WCHAR) == 24
002C    L"Stock Quote\0"    ; char sz[24]
0044    03 00 00 00     ; DWORD dwPropID = 3 (PID_SYMBOL)
0048    1C 00 00 00     ; DWORD cb = (wstrlen("Ticker Symbol") + 1
                        ;      * sizeof(WCHAR) == 28
004C    L"Ticker Symbol\0"    ; char sz[28]
     ; The dictionary would continue, but may not contain entries for
     ; every possible property. Also entries do not need to be in order
     ; (except dwPropID == 0 must be first).

; Start of Property 1 (code page indicator)
00D0    02 00 00 00    ; DWORD type indicator (VT_12)
00D4    B0 04 00 00    ; USHORT codepage (0x04B0 == 1200 == Unicode)

; Start of Property 7 (PID_SYMBOL)
0134     1E 00 00 00     ; DWORD type indicator (VT_LPWSTR == 1F)
0138     0A 00 00 00     ; DWORD count of bytes
013C    L"MSFT\0"        ; WCHAR sz[5] (incl 2 bytes for the NULL)
```

Property ID One

Property ID One is reserved as an indicator of which code page (Windows) or Script (Macintosh) to use when interpreting the strings in the property set. All string values in the property set must be stored with the same code page. The originating operating system value in the property set header (PROPERTYSETHEADER::dwOSVer) determines whether the code page indicator corresponds to a Windows code page or Macintosh script.

When an application that is not the author of a property set changes a property of type string in the set, it should examine the code page indicator and take one of the following actions:

- Write the new value using the code page found in the code page indicator.
- Rewrite all string values in the property set using the new code page (including the new value), and modify the code page indicator to reflect the new code page.

If an application cannot understand this indicator, it should not modify the property. All writers of property sets must write a code page indicator; however, if the code page indicator is not present, the prevailing code page on the reader's machine must be assumed.

Possible values for the code page are given in the Window 32 API (see the **GetACP** function) and *Inside Macintosh Volume VI,* §*14-111*. For example, the code page US ANSI is represented by 0x04E4 (1252 in decimal) while the code page for Unicode is 0x04B0 (1200 in decimal).

It is recommended that the Unicode code page be used whenever possible. Using the same code page for all property sets is the only way to achieve interoperable property sets on a worldwide basis. In the Unicode code page, note that the count at the start of a VT_LPSTR or VT_BSTR or other eight-bit character quantities in the property set is a **byte** count and not a **character** count. This byte count includes the two zero bytes at the end of the string.

Property ID one is a VT_12 type and thus starts with a DWORD containing the value VT_12 followed by a USHORT indicating the code page.

Property ID 0xFFFFFFFE

Property ID 0xFFFFFFFE (Locale Indicator) is reserved as an indication of the locale for which the property set is written. The default locale for a property set is the system's default locale (LOCALE_SYSTEM_DEFAULT). See the Win32 SDK for more information on LOCALE_SYSTEM_DEFAULT. The default is used in the event that the locale indicator does not exist in the property set.

Applications can choose to support the locale or just get the default behavior. It is recommended that applications allow users to specify a working locale. Such applications should write the user-specified locale identifier to the property. Applications that use the user's default locale (LOCALE_USER_DEFAULT) should write the user's default locale identifier to the property. See the Win32 SDK for more information on LOCALE_USER_DEFAULT.

Applications should also handle the case of a foreign object which is one where the locale is not the application's locale, the user's locale, or the system's locale.

The locale indicator property is of type VT_U4, and therefore consists of a DWORD containing VT_U4 followed by a DWORD containing the Locale Identifier (LCID) as defined in the Win32 SDK.

Other Reserved Property IDs

Property IDs with the high bit set (that is, negative values) are reserved for future use by Microsoft.

Clipboard Format Identifiers

There are five kinds of Clipboard FMTIDs that can occur in VT_CF values: Windows Clipboard Format Values, Macintosh Format Values, Format Identifiers, Clipboard Format Names, and No Format Name. The following table illustrates the representation of each of these five types:

First Four Bytes	Following Value Size	Meaning
-1L	4 bytes (DWORD)	Windows built-in Clipboard Format Value (CF_TEXT).
-2L	4 bytes (DWORD)	Macintosh Format Value (4-byte tag).
-3L	16 bytes (Format ID)	FMTID.
Length of String	Variable	Clipboard format name that has been registered by **RegisterClipboardFormat** (or some Macintosh equivalent, if any). The string length includes the null terminator.
0L	Zero	No format name.

Therefore, the format of a VT_CF value is:

```
DWORD cb ;               // count of bytes that follow (4 + cbTag +
                        // cbData)
DWORD cftag ;               // contains one of the five cftag values
                        // (0,-1,-2,-3, or positive)
BYTE rgcftag[ cbTag ] ;      // cbTag bytes representing the FMTID
BYTE rgData[ cbData ] ; // clipboard data in the specified format
```

For example, if a VT_CF property contained a 4235 byte bitmap that was stored in the Windows CF_DIB (0x08) Clipboard format, the count of bytes would be 4243 (4235 + 4 + 4) or 0x1093. The following stream of bytes would be stored:

```
93 10 00 00 FF FF FF FF 08 00 00 00 ?? ?? ?? ?? ?? ?? ...
|--- cb ---|-- cftag --|- rgcftag -|--- rgData (4235 bytes )...
```

Storing Property Sets

Applications can expose some of the state of their documents so that other applications can locate and read that information. Some examples are a property set describing the author, title, and keywords of a document created with a word processor, or the list of fonts used in a document. This facility is not restricted to documents; it can also be used on embedded objects.

Note If you are storing a property set that is internal to your application, you might not want to follow the guidelines described below. If you want to expose your property set to other applications, you need to follow the guidelines.

To store a property set in a compound file:

1. Create an **IStorage** or **IStream** instance in the same level of the storage structure as its data streams. Prepend the name of your **IStorage** or **IStream** instance with "\005." Stream and storage names that begin with 0x05 are reserved for common property sets that can be shared among applications. Also, streams beginning with that value are limited to 256Kb. The names can be selected from either published names and formats or by assigning the property set a FMTID and deriving the name from the FMTID according to the conventions described in the section "Naming Property Sets".
2. A property set may be stored in a single **IStream** instance or in an **IStorage** instance containing multiple streams. In the case of an **IStorage** instance, the contained stream named "Contents" is the primary stream containing property values, where some values may be names of other streams or storage instances within the storage for this property set.
3. Specify the CLSID of the object class that can display and/or provide programmatic access to the property values. If there is no such class, the the CLSID should be set to CLSID_NULL. For a property set that uses an **IStorage** instance, either set the CLSID of the **IStorage** instance to be the same as that stored in the Contents stream or to CLSID_NULL (the value in a newly created **IStorage** instance).
4. You have the option of specifying displayable names that form the contents of the dictionary.

Some applications can read only implementations of property sets stored as **IStream** instances. Applications should be written to expect that a property set may be stored in either an **IStorage** or **IStream** instance, unless the property set definition indicates otherwise. For example, the Document Summary Information property set's definition says that it can only be stored in a named **IStream** instance. In cases where you are searching for a property set and you don't know whether it is a storage or stream, look for an **IStream** instance with your property set name first. If that fails, look for an **IStorage** instance.

To better understand storing property sets in an **IStorage** implementation, suppose there is a class of applications that edit information about animals. First, a CLSID (CLSID_AnimalApp) is defined for this set of applications so they can indicate that they understand property sets containing animal information (FormatID_AnimalInfo) and others containing medical information (FormatID_MedicalInfo).

```
IStorage (File): "C:\OLE\REVO.DOC"
  IStorage: "\005AnimalInfo", CLSID = CLSID_AnimalApp
    IStream: "Contents"
      WORD dwByteOrder, WORD wFmtVersion, DWORD dwOSVer,
      CLSID CLSID_AnimalApp, DWORD cSections...
      ...
      FormatID = FormatID_AnimalInfo
      Property: Type = PID_ANIMALTYPE, Type = VT_LPSTR, Value = "Dog"
      Property: Type = PID_ANIMALNAME, Type = VT_LPSTR, Value = "Revo"
      Property: Type = PID_MEDICALHISTORY, Type = VT_STREAMED_OBJECT,
              Value = "MedicalInfo"
      ...
    IStream: "MedicalInfo"
      WORD dwByteOrder, WORD wFmtVersion, DWORD dwOSVer,
      CLSID CLSID_AnimalApp, DWORD cSections...
      ...
      FormatID = CLSID_MedicalInfo
      Property: Type = PID_VETNAME, Type = VT_LPSTR, Value = "Dr. Woof"
      Property: Type = PID_LASTEXAM, Type = VT_DATE, Value = ...
```

Note that the class IDs of the **IStorage** and both property sets is CLSID_AnimalApp. This identifies any application that can display and/or provide programmatic access to these property sets. Any application can read the information within the property sets (the point behind property sets), but only applications identified with the class ID of CLSID_AnimalApp can understand the meaning of the data in the property sets.

Naming Property Sets

Property sets are named with FMTIDs, but in **IStorage** and **IStream** instances, they are named with strings with a maximum length of 32 characters. As a result, you need to create string names corresponding to the FMTID.

To create a string name corresponding to the FMTID, first check to determine if the FMTID is one of a fixed set of established values. If it is an already established value, then use the corresponding string name as follows:

FMTID	String Name	Meaning
F29F85E0-4FF9-1068- AB91-08002B27B3D9	"\005SummaryInformation"	OLE Summary Information.

With the exception of the SummaryInformation property set (described in the following section), all other stream and storage objects used to store property sets should be named using the algorithm described in the following steps. The algorithm maps the 16-byte FMTID of the property set to a 31-character string. Storage and stream objects are restricted to 32-character names.

1. Convert the FMTID to little-endian byte order if necessary.
2. Take the 128 bits of the FMTID and treat them as one long bit string by concatenating each of the bytes together. The first bit of the 128 bit value is the most significant bit of the first byte in memory of the FMTID. The last bit of the 128 value is the least significant bit of the last byte in memory of the FMTID.
3. Extend these 128 bits to 130 bits by adding two zero bits to the end.
4. Divide the 130 bits into groups of five bits. There will be 26 groups of five bits each.
5. Consider each of the 26 groups as an integer value and use it as the index into the following array of 32 characters:

   ```
   ABCDEFGHIJKLMNOPQRSTUVWXYZ012345
   ```

   This results a sequence of 26 Unicode characters (one character from each of the 26 groups). Only the uppercase alphabet and numerals are used. No two characters in this range will compare as equal in any locale even when case is ignored.
6. Concatenate "\005" with the resulting 26-character string from step 5 for a total string length of 27 characters.

Note To be compatible with future versions of the operating system, you need to follow the mapping scheme described here.

The following code illustrates this process:

```
#define CCH_MAP         (1 << CBIT_CHARMASK)          // 32
#define CHARMASK        (CCH_MAP - 1)                 // 0x1f
#define CBIT_BYTE             8
#define CBIT_CHARMASK   5
#define CBIT_GUID       (CBIT_BYTE * sizeof(GUID))
#define WC_PROPSET0     ((WCHAR) 0x0005)
#define CWC_PROPSET     (1 + (CBIT_GUID + CBIT_CHARMASK-1)/CBIT_CHARMASK)
#define CWC_PROPSETSZ   (CWC_PROPSET + 1)            // allow null

CHAR awcMap[CCH_MAP + 1] = "abcdefghijklmnopqrstuvwxyz012345";
#define CALPHACHARS           ('z' - 'a' + 1)
```

```
GUID guidSummary =
    { 0xf29f85e0,
      0x4ff9, 0x1068,
      { 0xab, 0x91, 0x08, 0x00, 0x2b, 0x27, 0xb3, 0xd9 } };

WCHAR wszSummary[] = L"SummaryInformation";

// ---------------------------------------------------------------------
// Function:    GuidToPropertySetName
//
// Synopsis:    Map property set GUID to null-terminated UNICODE name
string.
//
//    The awcname parameter is assumed to be a buffer with room for
//    CWC_PROPSETSZ (28) UNICODE characters.  The first character
//    is always WC_PROPSET0 (0x05), as specified by the OLE Appendix
//    C documentation.  The colon character normally used as an NT
//    stream name separator is not written to the caller's buffer.
//
// Arguments:   IN GUID *pguid      -- pointer to GUID to convert
//    OUT WCHAR awcname[] -- output string buffer
//
// Returns:     count of non-NULL characters in the output string buffer
//----------------------------------------------------------------------

ULONG
GuidToPropertySetName(
    IN GUID const *pguid,
    OUT WCHAR awcname[])
{
    BYTE *pb = (BYTE *) pguid;
    BYTE *pbEnd = pb + sizeof(*pguid);
    ULONG cbitRemain = CBIT_BYTE;
    WCHAR *pwc = awcname;

    *pwc++ = WC_PROPSET0;

    // Note: CWC_PROPSET includes the WC_PROPSET0, and sizeof(wsz...)
    // includes the trailing L'\0', so sizeof(wsz...) is ok because the
    // WC_PROPSET0 character compensates for the trailing NULL character.

    ASSERT(CWC_PROPSET >= sizeof(wszSummary)/sizeof(WCHAR));
    if (*pguid == guidSummary)
    {
  CopyMemory(pwc, wszSummary, sizeof(wszSummary));
  return(sizeof(wszSummary)/sizeof(WCHAR));
    }

     while (pb < pbEnd)
    {
  ULONG i = *pb >> (CBIT_BYTE - cbitRemain);

  if (cbitRemain >= CBIT_CHARMASK)
  {
      *pwc = MapChar(i);
      if (cbitRemain == CBIT_BYTE && *pwc >= L'a' && *pwc <= L'z')
      {
    *pwc += (WCHAR) (L'A' - L'a');
      }
      pwc++;
      cbitRemain -= CBIT_CHARMASK;
      if (cbitRemain == 0)
      {
    pb++;
    cbitRemain = CBIT_BYTE;
      }
  }
  else
  {
      if (++pb < pbEnd)
      {
    i |= *pb << cbitRemain;
      }
            *pwc++ = MapChar(i);
      cbitRemain += CBIT_BYTE - CBIT_CHARMASK;
  }
    }
    *pwc = L'\0';
    return(CWC_PROPSET);
}


// ---------------------------------------------------------------------
// Function:    PropertySetNameToGuid
//
// Synopsis:    Map non null-terminated UNICODE string to a property set
GUID.
//
//    If the name is not properly formed as per
//    GuidToPropertySetName(), STATUS_INVALID_PARAMETER is
//    returned.  The pguid parameter is assumed to point to a buffer
//    with room for a GUID structure.
//
// Arguments:   IN ULONG cwcname   -- count of WCHARs in string to convert
//    IN WCHAR awcname[] -- input string to convert
//    OUT GUID *pguid     -- pointer to buffer for converted GUID
//
// Returns:     ULONG status
//----------------------------------------------------------------------

ULONG
PropertySetNameToGuid(
    IN ULONG cwcname,
    IN WCHAR const awcname[],
    OUT GUID *pguid)
{
    ULONG Status = ERROR_INVALID_PARAMETER;
    WCHAR const *pwc = awcname;

    if (pwc[0] == WC_PROPSET0)
    {
  // Note: cwcname includes the WC_PROPSET0, and sizeof(wsz...)
  // includes the trailing L'\0', but the comparison excludes both
  // the leading WC_PROPSET0 and the trailing L'\0'.

  if (cwcname == sizeof(wszSummary)/sizeof(WCHAR) &&
      wcsnicmp(&pwc[1], wszSummary, cwcname - 1) == 0)
  {
      *pguid = guidSummary;
      return(NO_ERROR);
  }

  if (cwcname == CWC_PROPSET)
  {
      ULONG cbit;
      BYTE *pb = (BYTE *) pguid - 1;

      ZeroMemory(pguid, sizeof(*pguid));
      for (cbit = 0; cbit < CBIT_GUID; cbit += CBIT_CHARMASK)
      {
    ULONG cbitUsed = cbit % CBIT_BYTE;
    ULONG cbitStored;
    WCHAR wc;

    if (cbitUsed == 0)
    {
        pb++;
    }
    wc = *++pwc - L'A';        // assume upper case
    if (wc > CALPHACHARS)
    {
        wc += (WCHAR) (L'A' - L'a');    // oops, try lower case
        if (wc > CALPHACHARS)
        {
      wc += L'a' - L'0' + CALPHACHARS;  // must be a digit
      if (wc > CHARMASK)
      {
          goto fail;      // invalid character
      }
        }
    }
    *pb |= (BYTE) (wc << cbitUsed);

    cbitStored = min(CBIT_BYTE - cbitUsed, CBIT_CHARMASK);

    // If the translated bits wouldn't all fit in the current byte

    if (cbitStored < CBIT_CHARMASK)
    {
        wc >>= CBIT_BYTE - cbitUsed;
        if (cbit + cbitStored == CBIT_GUID)
        {
      if (wc != 0)
      {
          goto fail;      // extra bits
      }
      break;
        }
        pb++;
        *pb |= (BYTE) wc;
    }
      }
      Status = NO_ERROR
  }
    }
fail:
    return(Status);
}
```

The Document Summary Information Property Set

OLE defines a standard common property set for storing summary information about documents. The Document Summary Information property set must be stored in an **IStream** instance off of the root storage object; it is not valid to store the property set in the "Contents" stream of a named **IStorage** instance.

All shared property sets are identified by a stream or storage name prepended with "\005" (or 0x05) to show it is a property set shareable among applications and the Document Summary Information property set is no exception. The name of the stream that contains the Document Summary Information property set is:

```
"\005SummaryInformation"
```

The FMTID for the Document Summary Information property set is:

```
  F29F85E0-4FF9-1068-AB91-08002B27B3D9
```

Use the DEFINE_GUID macro to define the FMTID for the property set:

```
DEFINE_GUID(FormatID_SummaryInformation, 0xF29F85E0, 0x4FF9, 0x1068, 0xAB,
0x91, 0x08, 0x00, 0x2B, 0x27, 0xB3, 0xD9);
```

On an Intel byte-ordered machine, the FMTID has the following representation:

```
  E0 85 9F 4F 68 10 AB 91 08 00 2B 27 B3 D9
```

The following table shows the property names for the Document Summary Information property set, along with the respective property IDs and type indicators.

Property Name	Property ID	Property ID Code	Type
Title	PID_TITLE	0x00000002	VT_LPSTR
Subject	PID_SUBJECT	0x00000003	VT_LPSTR
Author	PID_AUTHOR	0x00000004	VT_LPSTR
Keywords	PID_KEYWORDS	0x00000005	VT_LPSTR
Comments	PID_COMMENTS	0x00000006	VT_LPSTR
Template	PID_TEMPLATE	0x00000007	VT_LPSTR
Last Saved By	PID_LASTAUTHOR	0x00000008	VT_LPSTR
Revision Number	PID_REVNUMBER	0x00000009	VT_LPSTR
Total Editing Time	PID_EDITTIME	0x0000000A	VT_FILETIME
Last Printed	PID_LASTPRINTED	0x0000000B	VT_FILETIME
Create Time/Date (*)	PID_CREATE_DTM	0x0000000C	VT_FILETIME
Last saved Time/Date (*)	PID_LASTSAVE_DTM	0x0000000D	VT_FILETIME
Number of Pages	PID_PAGECOUNT	0x0000000E	VT_I4
Number of Words	PID_WORDCOUNT	0x0000000F	VT_I4
	PID_CHARCOUNT	0x00000010	VT_I4

Number of Characters			
Thumbnail	PID_THUMBNAIL	0x00000011	VT_CF
Name of Creating Application	PID_APPNAME	0x00000012	VT_LPSTR
Security	PID_SECURITY	0x00000013	VT_I4

* Some methods of file transfer (such as a download from a BBS) do not maintain the file system's version of this information correctly.

Guidelines for Implementing the Document Summary Information Property Set

The following guidelines pertain to the Document Summary Information property set described in the preceding section:

- PID_TEMPLATE refers to an external document containing formatting and styling information. The means by which the template is located is implementation defined.
- PID_LASTAUTHOR is the name stored in User Information by the application. For example, suppose Mary creates a document on her machine and gives it to John, who then modifies and saves it. Mary is the author, John is the last saved by value.
- PID_REVNUMBER is the number of times the File/Save command has been called on this document.
- PID_CREATE_DTM is a read-only property; this property should be set when a document is created, but should not be subsequently changed.
- For PID_THUMBNAIL, applications should store data in CF_DIB or CF_METAFILEPICT format. CF_METAFILEPICT is recommended.
- PID_SECURITY is the suggested security level for the document. By noting the security level on the document, an application other than the originator of the document can adjust its user interface to the properties appropriately. An application should not display any information about a password protected document or allow modifications to enforced read-only or locked-for-annotations documents. Applications should warn the user about read-only recommended if the user attempts to modify properties:

Security Level	**Value**
None	0
Password protected	1
Read-only recommended	2
Read-only enforced	3
Locked for annotations	4

Registering Object Applications

To become functional, object applications must register in the Microsoft® Windows® registry. The registry contains information about the application that is needed to support OLE features. This chapter discusses registering object applications and describes OLE registry entries and the subkeys under which they are entered.

Your installation/setup program must add information to the registry, and its associated subkeys, if it is to perform any of the three following types of installations:

- Installing an object application.
- Installing a container/object application.
- Installing a container application *that allows linking to its embedded objects*.

In all three cases, you must register OLE 2 library (DLL) information as well as application-specific information.

The subkeys are contained in the following registry keys:

HKEY_CLASSES_ROOT\CLSID
HKEY_CLASSES_ROOT\Interface

Note that **HKEY_CLASSES_ROOT** provides compatibility with Windows 3.1 and Windows 95.

For information about OLE registry functions, refer to the following API functions:

- **CoGetTreatAsClass**
- **CoTreatAsClass**
- **OleDoAutoConvert**
- **OleGetAutoConvert**
- **OleSetAutoConvert**
- **SetConvertStg**
- **GetConvertStg**
- **OleRegGetUserType**
- **OleRegEnumFormatEtc**
- **OleRegGetMiscStatus**
- **OleRegEnumVerbs**

For information on the registry, the structure of registry entries, and registry functions, refer to the *Microsoft Win32 Programmer's Reference*.

New Registry Subkeys for 32-bit OLE

This section describes changes made to the Windows NT™ registry to allow it to support 32-bit OLE. The Windows NT registry introduces new subkeys that are specific to 32-bit applications. Otherwise, a 32-bit application is registered in exactly the same manner as a 16-bit application.

The subkeys that are support 16-bit OLE applications are:

LocalServer	Full path to the application '16- or 32-bit'.
InprocServer	Registers a 16-bit in-process server DLL.
InprocHandler	Registers a 16-bit handler DLL.
ProxyStubClsid	Maps an IID to a CLSID in a 16-bit proxy DLL.

For 32-bit OLE applications, the following registry subkeys are added:

LocalServer32	Full path to the application '32-bit'.
InprocServer32	Registers a 32-bit in-process server DLL.
InprocHandler32	Registers a 32-bit handler DLL.
ProxyStubClsid32	Maps an IID to a CLSID in a 32-bit proxy DLL.

Registry Support for 32-bit OLE

The Insertable subkey is a required subkey for 32-bit OLE applications whose objects can be inserted into existing 16-bit applications. Existing 16-bit applications look in the registry for the Insertable key. This key informs the application that the server supports embeddings. If the Insertable key exists, 16-bit applications may also attempt to verify that the server exists on the machine. 16-bit applications typically will retrieve the value of the LocalServer key from the class, and check to see if it is a valid file on the system. Therefore, in order for a 32-bit implementation to be insertable by a 16-bit application, the 32-bit application should register the LocalServer subkey in addition to registering LocalServer32.

For 32-bit local servers, the required entries are InprocHandler32, LocalServer, LocalServer32, and Insertable. For a 32-bit InprocServer, the required entries are InprocHandler32, InprocServer, InprocServer32, and Insertable. Note that the LocalServer entry and the InprocServer entry provide backward compatibility. If they are missing, the class still works, but can't be inserted in 16-bit applications. The InprocServer32 key has a ThreadingModel subkey to support more than one thread per process. See the section "The CLSID Key and Subkeys" later in this appendix for more information.

If a container is searching the registry for a local server, a 32-bit local server has priority over a 16-bit local server. With InprocHandlers and InprocServers, the 16-bit versions have priority with a 16-bit container, and 32-bit versions have priority with a 32-bit container.

Note that ProxyStubClsid32 is required because the IID-to-CLSID mapping may be different for 16- and 32- bit interfaces. The IID-to-CLSID mapping depends on the way the interface proxies are packaged into a set of proxy DLLs.

OLE Registry Entries

This section describes the following OLE registry entries:

- Programmatic Identifiers
- The ProgID Key and Subkeys
- The CLSID Key and Subkeys
- The Version-Independent ProgID Key and Subkeys
- The File Extension Key
- The (Non-Compound) FileType Key and Subkeys
- The New Interface Key and Subkeys

Programmatic Identifiers

Every OLE 2 object class that is to appear in an Insert Object dialog box (also referred to as an "insertable class") must have a programmatic identifier or *ProgID*. This string uniquely identifies a given class and is intended to be in a form that can be read by users. Programmatic identifiers are not guaranteed to be universally unique so they can be used only where name collisions are manageable, such as in achieving compatibility with OLE 1. Also, the ProgID is the "class name" used for an OLE 2 class when it is placed in an object application (OLE 1 server).

The ProgID string must:

- Have no more than 39 characters.
- Contain no punctuation (including underscores) except one or more periods.
- Not start with a digit.
- Be different from the class name of any OLE 1 application, *including the OLE 1 version of the same application*, if there is one.

The user must never see the ProgID in the user interface. If you need a short displayable string for an object, call the **IOleObject::GetUserType** method.

Because it is necessary to make a conversion between the ProgId and the CLSID, it is important to note that there are two kinds of ProgIDs. One depends on the version of the object application (the version-dependent ProgId); and one does not (the version-independent ProgId).

Version-dependent Identifiers

The version-dependent ProgID is the string used when OLE 1 is trying to contact OLE 2 using DDE. Version-dependent ProgID-to-CLSID conversions must be specific, well-defined, and one-to-one.

Version-independent Identifiers

The version-independent ProgID, on the other hand, can be used when a container application creates a chart or table with a toolbar button. In this situation, the application can use the version-independent ProgID to determine the latest version of the needed object application.

The version-independent ProgID is stored and maintained solely by application code. When given the version-independent ProgID, the **CLSIDFromProgID** function returns the CLSID of the current version. The **CLSIDFromProgID** function works on the version-independent ProgID because the subkey, CLSID, is the same as it is for the version-dependent one.

You can call the **IOleObject::GetUserType** method or the **OleRegGetUserType** function to change the identifier to a displayable string.

Converting Identifiers

The **CLSIDFromProgID** and **ProgIdFromCLSID** API functions can be called to convert back and forth between the two representations. These functions use information stored in the registry to perform the conversion.

The ProgID-to-CLSID conversion is done using the **HKEY_LOCAL_MACHINE\SOFTWARE\Classes**\<*ProgID*>\CLSID key. The reverse translation is done using the **HKEY_LOCAL_MACHINE\SOFTWARE\Classes**\CLSID\<*clsid*>\ProgID subkey. This means that the ProgID subkey under **HKEY_LOCAL_MACHINE\SOFTWARE\Classes**\CLSID\<*clsid*> is version-dependent.

The ProgID Key and Subkeys

OLE 2 classes that belong in the Insert Object dialog box each have a ProgID. The value assigned to this key is the name displayed in the dialog box; it should be the same as the "*MainUserTypeName*" of the class. If this class is insertable in an OLE 2 container, the ProgID key must have an immediate subkey named **Insertable**, which must have no value assigned to it.

Note Because OLE 2 provides a built-in OLE 1/OLE 2 compatibility layer, rarely will an OLE 2 class that is insertable in an OLE 2 container not be insertable in an OLE 1 container.

Conventions Used in Examples

In the registry examples in this appendix, **boldface** indicates a literal standard key or subkey, <*italics*> indicates an application-supplied string or value, and <***boldface-italics***> indicates an application-supplied key or subkey. In the first example, "OLE1ClassName," "OLE1UserTypeName," and "CLSID" are all supplied by the application.

Inserting an OLE 2 Object in an OLE 1 Application

If a particular class is insertable in an OLE 1 container, the "ProgID" *root* key will contain a **Protocol\StdFileEditing** subkey with appropriate subkeys **Verb**, **Server**, and so on, as in OLE 1. The **Server** that should be registered here is the full path to the executable file of the OLE 2 object application. An OLE 1 container uses the path and executable file names to launch the OLE 2 object application. The initialization of this application, in turn, loads the OLE 2 compatibility layer. This layer handles subsequent interactions with the OLE 1 container (client), turning them into OLE 2-like requests to the OLE 2 application. An OLE 2 object application doesn't have to take any special action beyond setting up these registry entries to make objects insertable into an OLE 1 container.

The ProgID key and its subkeys appear in the registry as shown in the following example, where "***<Progid>***" is the key, and "**Insertable**," "**Protocol**," "**StdFileEditing**," "**Verb**," and so on are subkeys.

```
<ProgId> = <MainUserTypeName>
        Insertable      // class is insertable in OLE 2 containers
        Protocol
                StdFileEditing  // OLE 1 compatibility info; present if, and only if,
                                // objects of this class are insertable in OLE 1 containers.
                        Server = <full path to the OLE 2 object application>
                        Verb
                                0 = <verb 0>    // Verb entries for the OLE 2
                                                // application must start with zero as the
                                1 = <verb 1>    // primary verb and run consecutively.
        CLSID = <CLSID>         // The corresponding CLSID. Needed by GetClassFile.
        Shell   // Windows 3.1 File Manager Info
                Print
                Open
                Command = <appname.exe> %1
```

To summarize, any root key that has either an **Insertable** or a **Protocol\StdFileEditing** subkey is the ProgID (or OLE 1 class name) of a class that should appear in the Insert Object dialog box. The value of that root key is the name displayed in the Insert Object dialog box.

The values of each key in the example below are used for registering the "Ole 2 In-Place Server Outline" sample application. Set these values as required and used by your application.

Long Form String Subkey Entry

The entry to register the string (long form), such as that used in the Insert New dialog box, is as follows.

```
HKEY_CLASSES_ROOT\OLE2ISvrOtl = Ole 2 In-Place Server Outline
```

The recommended maximum length for the string is 40 characters.

Information for OLE 1 Applications Subkey Entries

To maintain compatibility with OLE 1, you must include specific OLE 1 information. The "server" key entry should contain a full path to the application. The entries for verbs must start with 0 as the primary verb and be consecutively numbered.

```
HKEY_CLASSES_ROOT\OLE2ISvrOtl\protocol\StdFileEditing\server =
     c:\samp\isvrotl.exe
HKEY_CLASSES_ROOT\OLE2ISvrOtl\protocol\StdFileEditing\verb\0 = &Edit
HKEY_CLASSES_ROOT\OLE2ISvrOtl\protocol\StdFileEditing\verb\1 = &Open
```

Windows 3.1 Shell Subkey Entries

These entries are for Windows 3.1 shell printing and File Open use. They should contain the path and filename of the object application. The examples below contain simple entries only; more complicated ones could include DDE entries.

```
HKEY_CLASSES_ROOT\OLE2ISvrOtl\Shell\Print\Command = c:\svr\isvrotl.exe %1
HKEY_CLASSES_ROOT\OLE2ISvrOtl\Shell\Open\Command = c:\svr\isvrotl.exe %1
```

Insertable Subkey Entry

This entry indicates that the object application should appear in the Insert New dialog box's list box when used by OLE 2 container applications.

```
HKEY_CLASSES_ROOT\OLE2ISvrOtl\Insertable
```

Entry Point Subkey

This entry points to the application's OLE 2 information in the registry. To assign a CLSID for your application, run the UUIDGEN.EXE found in the \TOOLS directory of the OLE 2 Toolkit.

```
HKEY_CLASSES_ROOT\OLE2ISvrOtl\CLSID = {00000402-0000-0000-C000-000000000046}
```

The CLSID Key and Subkeys

Most of the OLE 2 object application information is stored in subkeys under the **CLSID** *root* key. The immediate subkey of the **CLSID** key is a string version CLSID. Subkeys of this version of the CLSID indicate where the code that services this class is found. For more information about converting the CLSID, see the **StringFromCLSID** function.

Most of the **CLSID** information is used by the default OLE 2 handler to return various information about the class when it is in the loaded state. Examples of this include the **Verb**, the **AuxUserType**, and the **MiscStatus** entries. The **Insertable** subkey appears under both this key and the **ProgId** key.

```
CLSID
<CLSID> = <Main User Type Name>
        LocalServer = <path to 16- or 32-bit exe>       // local (same machine) server; see "Server ="
                                                                // under ProgID key.
        LocalServer32 = <path to 32-bit exe>            // local (same machine) server; see "Server ="
                                                                // under ProgID key.
        InprocServer = <path to dll>     // in process server; relatively rare for insertable classes.
        InprocServer32 = <path to 32-bit dll>    // in process server; relatively rare for insertable classes.
        InprocHandler = <path to dll>    // in process handler. "OLE2.DLL" for the default OLE 2 handler
        InprocHandler32 = <path to 32-bit dll>  // in process handler. "OLE2.DLL" for the default OLE 2
                                                        // handler
        Verb            // info returned in IOleObject::EnumVerbs().
                verb number = <name, menu flags, verb flags>
        // several examples follow:
                 0 = &Edit, 0, 2  // primary verb; often Edit; on menu; possibly dirties object,
                                        // MF_STRING | MF_UNCHECKED | MF_ENABLED == 0.
                 1 = &Play, 0, 3  // other verb; appears on menu; leaves object clean
                -3 = Hide, 0, 1   // pseudo verb for hiding window; not on menu, opt.
                -2 = Open, 0, 1  // pseudo verb for opening in sep. window; not on menu, opt.
                -1 = Show, 0, 1  // pseudo verb for showing in preferred state; not on menu, opt.
        AuxUserType   // auxiliary user types (main user type above)
                <form of type> = <string>         // See IOleObject::GetUserType(); for example:
                2 = <ShortName>         // key 1 should not be used
                3 = <Application name> // Contains the displayable name of the application. Used when
                                // the actual name of the app is needed (for example in
                                // the Paste Special dialog's result field) Example: Acme Draw
        MiscStatus = <default> // def status used for all aspects; see IOleObject::GetMiscStatus
                <aspect> = <integer>    // exceptions to above;   for example:
                        4 = 1                   // DVASPECT_ICON = OLEMISC_RECOMPOSEONRESIZE
        DataFormats
                DefaultFile = <format>  // default main file/object format of objects of this class.
                GetSet // list of formats for default impl. of EnumFormatEtc; very similar
                                // to Request/SetDataFormats in OLE 1 entries
                        <n> = <format ,aspect, medium, flag>     // in this line,
                                // n  is a zero-based integer index;
                                // format is clipboard format;
                                // aspect is one or more of DVASPECT_*, -1 for "all";
                                // medium is one or more of TYMED_*;
                                // flag is one or more of DATADIR_*.
                                // three examples follow:
                        0 = 3, -1, 32, 1              // CF_METAFILE, all aspects, TYMED_MFPICT,
                                // DATADIR_GET
                        1 = Biff3, 1, 15, 3           // this example shows
                                // Microsoft Excel's Biff format version 3,
                                // DVASPECT_CONTENT,
                                // TYMED_HGLOBAL | TYMED_FILE |
                                // TYMED_ISTREAM | TYMED_ISTORAGE,
                                // (DATADIR_SET | DATADIR_GET)
                        2 = Rich Text Format, 1,1,3
        Insertable      // when present, the class appears in the Insert Object dialog.
                                // (not present for internal classes like the moniker classes)
        ProgID = <ProgID>               // the programmatic identifier for this class.
        TreatAs = <CLSID>                               // see CoGetTreatAs()
        AutoTreatAs = <CLSID>                   //see CoTreatAsClass
        AutoConvertTo = <CLSID>                 // see OleGetAutoConvert()
        Conversion                      // support for Change Type dialog box
                Readable
                        Main = <format,format,format,format, ...>
                Readwritable
                        Main = <format,format,format,format, ...>
        DefaultIcon = <path to exe, index>      // parameters passed to ExtractIcon
        Interfaces = <IID, IID, ...>    // optional. If this key is present, then its values are the
                                             // totality of the interfaces supported by this class: if the
                                             // IID is not in this list, then the interface is never supported
                                             // by an instance of this class.
        VersionIndependentProgID = <VersionIndependentProgID>
```

Existing 16-bit applications look in the registry for the Insertable keyword. This key informs the application that the server supports embeddings. If the Insertable keyword exists, 16-bit applications may also attempt to verify that the server exists on the machine. 16-bit applications typically will retrieve the value of the LocalServer key from the class, and check to see if it is a valid file on the system. Therefore, to be insertable by 16-bit applications, 32-bit applications should register LocalServer in addition to registering LocalServer32.

The InprocServer32 key has a ThreadingModel subkey to support more than one thread per process. You can set the ThreadingModel key to Apartment to support the apartment model threading or to Both to support both the apartment model and free threading. See the chapter "The Component Object Model" for more information on apartment model threading and free threading.

CLSID (Object Class ID) Subkey Entry

The following information creates a *CLSID* subkey under the **CLSID** key. To obtain a CLSID for your application, run the UUIDGEN.EXE found in the \TOOLS directory of the OLE 2 Toolkit.

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046} = Ole 2 In-Place Server Outline
```

LocalServer Subkey Entry

This subkey designates the application's location. The LocalServer subkey should contain a full path. The entry can contain command-line arguments. Note that OLE 2 appends the "-Embedding" flag to the string, so the application that uses flags must parse the whole string and check for the -Embedding flag.

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\LocalServer = 
c:\samp\isvrotl.exe
```

To run an OLE object server in a separate memory space (Windows NT 3.5 and above only), change the LocalServer key in the registry for the CLSID to the following:

```
cmd /c start /separate <path.exe
```

LocalServer32 Subkey Entry

This subkey designates the location of the 32-bit version of the application. The LocalServer32 subkey should contain a full path. The entry can contain command-line arguments. Note that OLE 2 appends the "-Embedding" flag to the string, so the application that uses flags will need to parse the whole string and check for the -Embedding flag.

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\LocalServer32
= c:\samp\isvrotl.exe
```

When OLE starts a 32-bit local server, the server must register a class object within an elapsed time set by the user. By default, the elapsed time value must be at least five minutes, in milliseconds, but cannot exceed the number of milliseconds in 30 days. Applications typically should not set this value which is in the \HKEY_CLASSES_ROOT\Software\Microsoft\OLE2\ServerStartElapsedTime entry.

InprocHandler Subkey Entry

This subkey designates whether the application uses a custom handler. If no custom handler is used, the entry should be set to OLE2.DLL, as shown in the following example.

```
HKEY_-CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\InprocHandler = ole2.dll
```

InprocHandler32 Subkey Entry

This subkey designates whether the application uses a custom handler. If no custom handler is used, the entry should be set to OLE32.DLL, as shown in the following example.

```
HKEY_-CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\InprocHandler32 = ole32.dll
```

Verb Subkey Entry

The verbs to be registered for the application must be numbered consecutively. The first value after the verb string describes how the verb is appended by **AppendMenu** function call. See the *Win32 SDK* documentation for details.

The second value indicates whether the verb will dirty the object. It also indicates whether the verb should appear in the menu (as defined by OLEVERBATTRIB_). For more information, see the **IOleObject::EnumVerbs** method and the **OleRegEnumVerbs** function.

Verb 0: "Edit", MF_UNCHECKED | MF_ENABLED, no OLEVERATTRIB flags

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\Verb\0 =
&Edit,0,0
```

Verb 1: "Open", MF_UNCHECKED | MF_ENABLED, no OLEVERATTRIB flags

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\Verb\1 =
&Open,0,0
```

AuxUserType Subkey Entry

This key describes the short and actual displayable names of the application. See also the **IOleObject::GetUserType** method. The short name is used in the menus, including pop-ups, and the recommended maximum length for the string is 15 characters. A short name example follows.

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\AuxUserType\2
= In-Place Outline
```

The long displayable name of the application is used in the Results field of the Paste Special dialog box. This string should contain the actual name of the application (such as "Acme Draw 2.0").

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\AuxUserType\3
= Ole 2 In-Place Server
```

MiscStatus Subkey Entry

The following is an example of a MiscStatus subkey. Refer to the **IOleObject::GetMiscStatus** method description for information on the different settings.

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\MiscStatus =
0
```

DataFormats Subkey Entry

The DataFormats subkey lists the default and main data formats supported by the application. This entry is used by the **IDataObject::GetData**, **IDataObject::SetData** and **IDataObject::EnumFormatEtc** methods.

```
The values defined in the following example entry are CF_TEXT,
DVASPECT_CONTENT, TYMED_HGLOBAL, and DATADIR_GET | DATADIR_SET.
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\DataFormats\GetSet\0 = 1,1,1,3
```

The values defined in the following entry are: CF_METAFILEPICT DVASPECT_CONTENT, TYMED_MFPICT, DATADIR_GET.

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\DataFormats\GetSet\1 = 3,1,32,1
```

The values defined in the following entry are: 2 = cfEmbedSource, DVASPECT_CONTENT, TYMED_ISTORAGE, and DATADIR_GET.

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\DataFormats\GetSet\2 = Embed Source,1,8,1
```

The values defined in the following entry are: 3 = cfOutline, DVASPECT_CONTENT, TYMED_HGLOBAL, and DATADIR_GET | DATADIR_SET.

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\DataFormats\GetSet\3 = Outline,1,1,3
```

The following entry declares that the default File Format supported by this application is CF_OUTLINE.

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\DataFormats\DefaultFile = Outline
```

Insertable Subkey

The Insertable entry indicates that this application should appear in the Insert New dialog box's list box when used by OLE 2 container applications. Note that this is a duplicate entry of the one above but is required for future use.

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\Insertable
```

ProgID Subkey Entry

Every insertable object class has a "programmatic identifier" or ProgID. For information on the creation and syntax of programmatic identifiers, see "**Programmatic Identifiers**" earlier in this chapter.

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\ProgID =
OLE2ISvrOtl
```

Conversion Subkey Entry

Conversion information is used by the Convert dialog box to determine what formats the application can read and write. A comma-delimited file format is indicated by a number if it is one of the Clipboard formats defined in WINDOWS.H. A string indicates the format is not one defined in WINDOWS.H (private). Note that in this case the readable and writeable format is CF_OUTLINE (private).

Shown below is an entry that registers a file format the application can read (convert from).

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\Conversion\Readable\Main = Outline,1
```

Shown below is an entry that registers a file format the application can read and write (activate as).

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-
000000000046}\Conversion\Readwritable\Main = Outline,1
```

DefaultIcon Subkey Entry

The DefaultIcon subkey provides default icon information for iconic presentations of objects. This entry contains the full path to the executable name of the object application and the index of the icon within the executable. Applications can use this information to obtain an icon handle with **ExtractIcon**.

```
HKEY_CLASSES_ROOT\CLSID\{00000402-0000-0000-C000-000000000046}\DefaultIcon =
c:\samp\isvrotl.exe,0
```

The Version-Independent ProgID Key and Subkeys

An application must register a version-independent programmatic identifier in the "Version-Independent ProgID" key, that is, a second ProgID referring to its class, one that would not change from version to version, but would remain constant across all versions. This key provides a constant name, referred to as the *MainUserTypeName*. It is used with macro languages and refers to the currently installed version of the application's class. The *MainUserTypeName* must correspond to the name of the latest version of the object application.

```
<VersionIndependentProgID> = <MainUserTypeName>
        CLSID = <CLSID>        // the class id of the newest installed version of that class
        CurVer = <ProgID>      // the ProgID of the newest installed version of that class
```

In addition to the preceding entry, you should add the following entry to the **CLSID** *root* key.

```
CLSID
<CLSID> = <Main User Type Name>
        VersionIndependentProgID = <VersionIndependentProgID>
```

The File Extension Key

If your OLE 2 application can handle requests from the Windows 3.1 File Manager, create the file extension *root* key.

<.ext> = <*ProgID*>   // used by File Manager and by GetClassFile (and thus File Monikers).

The (Non-Compound) FileType Key and Subkeys

Entries under the FileType key are used by the **GetClassFile** function to match patterns against various file bytes in a non-compound file. FileType has *CLSID* subkeys, each of which has a series of subkeys 0, 1, 2, ... These values contain a pattern that, if matched, should yield the indicated *CLSID*. See also the **GetClassFile** function.

```
FileType                    // used by GetClassFile()
        <CLSID>
                <n> = <offset, cb, mask, value> // Offset and cb are limited to 16 bits. Offset can
                                //be negative for file end.
                                // As above, offset and byte count are decimal unless preceded by "0x",
in which
                                // case they are hex. The mask and the pattern are always hex and
cannot be
                                // preceded by "0x". Mask can be omitted, implying a value of all ones.
                                // An example:
                0 = 0, 4, FFFFFFFF, ABCD1234 // This requires that the first 4 bytes be AB CD
                                             // 12 34, in that order,
                1 = 0, 4, FFFFFFFF, 9876543  // or requires that they match 9876543.
                2 = -4, 4, FEFEFEFE          // The last four bytes in the file must be FEFEFEFE
```

The Interface Key and Subkeys

If your application adds a new interface, the Interface key must be completed for OLE 2 to register the new interface. There must be one IID subkey for each new interface.

```
Interface
        <IID> = <textual name of interface>      // for example: "IOleObject"
                ProxyStubClsID = <CLSID       >          // Maps an IID to a CLSID in a 16-bit proxy DLL.
                                                         //Used internally by OLE 2 for interprocess
                                                 // communication.
                ProxyStubClsID32 = <CLSID     >          // Maps an IID to a CLSID in a 32-bit proxy DLL.
                                                         //Used internally by OLE 2 for interprocess
                                                         // communication.
                NumMethods = <integer>           // Number of methods in the interface.
                BaseInterface = <IID>    // Interface from which this was derived. Absence
                                         // of key means IUnknown. Key present but empty value
                                         // means derived from nothing.
```

Note that you must use ProxyStubClsid32 because the IID-to-CLSID mapping may be different for 16- and 32-bit interfaces. The IID-to-CLSID mapping depends on the way the interface proxies are packaged into a set of proxy DLLs.

OLE 1 Compatibility Subkeys

To handle two-way compatibility, the OLE 2 compatibility layer creates OLE 2-style entries for OLE 1 classes it discovers and places them under the CLSID key. When installing an OLE 2 object application on a system that contains an OLE 1 version of the same application, it might be necessary to add "AutoConvertTo" or "TreatAs" subkeys to the *original* OLE 1 application portion of the registry. For more information, see "When the OLE 1 Version is Overwritten."

CLSID
<*CLSID*> = <*OLE1UserTypeName*> // This entry is generated automatically by OLE 2 for an
OLE 1 class.
OLE1Class = <*OLE 1 class name*> // This entry is created the first time an object of
that class
// is inserted in an OLE 2 container**.**
ProgId = <*OLE 1 class name*> // Allows OLE 2 to convert CLSID back to an OLE 1
class name.

OLE 1 Application Entries

When an OLE 1 class is inserted into an OLE 2 container for the first time, a new subkey, **CLSID**, is added *to the original OLE 1 registration information* by the OLE 2 compatibility layer. The value given to this key is a CLSID assigned by OLE 2 to this OLE 1 class as shown in the following portion of the registry.

```
<OLE1ClassName> = <OLE1UserTypeName>
        Protocol
                StdFileEditing
                Verb
        CLSID = <CLSID>
```

Registering OLE 2 Libraries

The OLE 2 libraries require that many internal interfaces be registered. Thus, if your installation program installs the OLE 2 libraries on a machine that does not already have them, it should register OLE 2-specific information during installation.

If your installation program does not install OLE 2 libraries when it finds more recent libraries on the user's drive, it should not register the OLE 2 interface information.

Note When checking the version stamp (using VER.DLL) on existing OLE 2 libraries to determine whether or not to replace them on the user's hard drive, do so on a per file basis.

Accommodating OLE 1 Versions of the Object Application

When the OLE 1 version of an object application is superseded by an OLE 2 version and the OLE 2 version is to be installed in the user's system, such as when upgrading, two situations can arise:

- An OLE 1 version of the application is present on the user's system and the installation process overwrites the OLE 1 executable with the OLE 2 version.
- An OLE 1 version of the application is present on the user's system and the user chooses not to overwrite it with the OLE 2 version.

Note Even if the OLE 1 object application is not on the user's system, the install/setup program for the OLE 2 object application should register the application as capable of servicing its OLE 1 objects. To do this, follow the guidelines presented under "When the OLE 1 Version is Overwritten," and add the following entry to the *CLSID root key*:

<**OLE 1 class name**>/**CLSID =** <*CLSID of OLE 1 application*>

When the OLE 1 Version is Overwritten

When the OLE 1 object application is replaced by the OLE 2 version, do the following. (For the purpose of illustration, the OLE 1 object application is referred to as Ole 1 In-Place Server Outline while the OLE 2 version is Ole 2 In-Place Server Outline, as shown in the dialog box illustrations that follow).

1. Register Ole 2 In-Place Server Outline.
2. Modify (with your install/setup program) the *original* registry entries of Ole 1 In-Place Server Outline by changing the executable path to point to the Ole 2 In-Place Server Outline executable.

   For example, the Server subkey for the OLE 1 executable, named **svrapp.exe** in this example, changes from

   OLE1ISvrOtl\Protocol\StdFileEditing\Server = svrapp.exe

   to

   OLE1ISvrOtl\Protocol\StdFileEditing\Server = isvrotl.exe

   where **isvrotl.exe** is the name of the OLE 2 object application.

   Next, proceed to either step 3 or step 4, depending on whether or not you want OLE 1 objects converted automatically to the OLE 2 format.
3. If Ole 1 In-Place Server Outline objects will be converted automatically to the Ole 2 In-Place Server Outline format when the application is saved, create or modify the following registration database entries:
   a. Modify the *original registry entry* of Ole 1 In-Place Server Outline by changing the "Value of the ProgID = Main User Type Name" key of the entry to Ole 2 In-Place Server Outline.

      For example, where **OLE1ISvrOtl** is the ProgID of the OLE 1 application,

      OLE1ISvrOtl = Ole 1 In-Place Server Outline

      becomes

      OLE1ISvrOtl = Ole 2 In-Place Server Outline
   b. Modify the *original registry entry* of Ole 1 In-Place Server Outline by adding a **NotInsertable** subkey under the ProgID key. For example:

      ```
      HKEY_CLASSES_ROOT\OLE1SvrOtl\NotInsertable
      ```

Note Since the original registry entries for the OLE 1 server application remain (but with a pointer to the OLE 2 object application as shown in the preceding *step 3a*), the OLE 1 ProgID will appear in the Insert Object dialog box of any OLE 2 container application installed on the system. The **NotInsertable** subkey mentioned in *step 3b* prevents the ProgID of the OLE 1 application from appearing in the Insert Object dialog box of OLE 2 containers. The **NotInsertable** subkey overrides any **Insertable** subkey entries for that ProgID key.

   c. Set the "AutoConvertTo = CLSID" subkey entry for Ole 1 In-Place Server Outline *under the CLSID key* to the CLSID of Ole 2 In-Place Server Outline. (See also "OLE 1 Compatibility Subkeys").

      CLSID\{*CLSID of OLE 1 app.***}\AutoConvertTo = {***CLSID of OLE 2 app.***}**

Note You can obtain the CLSID of the OLE 1 object application for inclusion in registration entry file by calling **CLSIDFromProgID**.

   d. Modify the *original registry entry* of Ole 1 In-Place Server Outline by setting the verbs to those of Ole 2 In-Place Server Outline.

      For example, change

      OLE1ISvrOtl\Protocol\StdFileEditing\Verb\0 = &Edit

      to

OLE1ISvrOtl\Protocol\StdFileEditing\Verb\0 = &Edit

OLE1ISvrOtl\Protocol\StdFileEditing\Verb\1 = &Open

4. If the user is allowed to open Acme Draw 1.0 objects and save them back to disk in the Acme Draw 1.0 format:
   a. Set the "TreatAs = CLSID" entry to the CLSID of Ole 2 In-Place Server Outline using the following form.

      CLSID\{*CLSID of OLE 1 app.***}\TreatAs = {***CLSID of OLE 2 app.***}**

   b. Set the Ole 1 In-Place Server Outline verbs to those of Ole 2 In-Place Server Outline as described in the preceding step 3.

When the OLE 1 Version is Not Overwritten

You can install your program so the OLE 1 object application (Ole 1 In-Place Server Outline) is *not* replaced by the OLE 2 version (Ole 2 In-Place Server Outline). Instead, the user is allowed to open Ole 1 In-Place Server Outline objects with the Ole 2 In-Place Server Outline and save them back to disk in the Ole 1 In-Place Server Outline format. To do this, set the "TreatAs = CLSID" entry (Ole 1 In-Place Server Outline's portion of the registry) to the CLSID of Ole 2 In-Place Server Outline (as in step 4 above).

If the OLE 1 version of the application is not overwritten, or if the user does not want to set the "Treat As" option, register the OLE 2 version as a separate and new application.

{ewc msdncd, EWGraphic, group10507 0 /a "SDK.wmf"}

Checking Registration During Run Time

The application should check its registration at the application load time, noting three particular issues:

- If the CLSID(s) that the application services is not present in the registry, the application should register as it does during the original setup.
- If the application's CLSID is present, but has no OLE 2-related information in it, the application should register as it does during the original setup.
- If the path containing server entries (LocalServer and LocalServer32, InprocServer and InprocServer32, and DefaultIcon entries) does not point to the location in which the application is currently installed, the application should rewrite the path entries to point to its current location.

Using the Registry for Localization

It is possible to address a pair of product-localization issues by adding a key to the registry. This key, described below, allows functions to return a specified string instead of a default value or "Unknown."

This section includes the following topics:

- Specifying Unregistered Verbs
- Specifying Unknown User Types

Specifying Unregistered Verbs

It is possible to register OLE 1 object applications (servers) with no specified verbs. In such cases, the application has a single implied verb that is understood to be "edit" by OLE 1 containers. When an OLE 2 application calls the **IOleObject::EnumVerbs** method (or the **OleRegEnumVerbs** function) on an object of this class, one verb is enumerated. By default, the name of the verb is "Edit." To avoid having the string "Edit" as the enumerated verb, a key can be included in the registry. This key is

\Software\Microsoft\OLE1\UnregisteredVerb = <*verbname*>

(where *verbname* is the value that will be returned from the enumeration) and allows for localization of the default verb to a specified string to accommodate a specific language.

Note Do not attempt to register a verb (under ProgID\Protocol\StdFileEditing) for an application that did not register the verb itself.

Specifying Unknown User Types

The OLE 2 default handler's implementation of the **IOleObject::GetUserType** method first examines the registry by calling the **OleRegGetUserType** method. If the object's class is not found in the registry, the User Type from the object's **IStorage** instance is returned. If the class is not found in the object's **IStorage** instance, the string "Unknown" is returned.

By inserting the

\Software\Microsoft\OLE2\UnknownUserType = <*usertype*>

key in the registry, the **IOleObject::GetUserType** method returns the value of the string specified by <*usertype*>. This string can be localized for a different language user-type name, instead of using the "Unknown" string, for the User Type.

Structured Storage

Traditional file systems face challenges when they try to efficiently store multiple kinds of objects in one document. OLE provides a solution: a file system *within* a file. OLE structured storage defines how to treat a single file entity as a structured collection of two types of objects – storages and streams – that act like directories and files. This scheme is called structured storage. The purpose of structured storage is to reduce the performance penalties and overhead associated with storing separate objects in a flat file

The Evolution of File Systems

Years ago, in the days before disk operating systems, each computer was built to run a single, proprietary application, which had complete and exclusive control of the entire machine. The application would write its persistent data directly to a disk, or drum, by sending commands directly to the disk controller. The application was responsible for managing the absolute locations of data on the disk, making sure that it was not overwriting data that was already there, but since the application was the only one running on the machine, this task was not too difficult.

The advent of computer systems that could run more than one application required some sort of mechanism to make sure that applications did not write over each other's data. Application developers addressed this problem by adopting a single standard for marking which disk sectors were in use and which were free. In time, these standards coalesced to become a disk operating system, which provided various services to the applications, including a file system for managing persistent storage. With the advent of a file system, applications no longer had to deal directly with the physical storage medium. Instead, they simply told the file system to write blocks of data to the disk and let the file system worry about how to do it. In addition, the file system allowed applications to create data hierarchies through the abstraction known as a directory. A directory could contain not only files but other directories. which in turn could contain their own files and directories, and so on.

The file system provided a single level of indirection between applications and the disk, and the result was that every application saw a file as a single contiguous stream of bytes on the disk even though the file system was actually storing the file in discontiguous sectors. The indirection freed the applications from having to care about the absolute position of data on a storage device.

Today, virtually all system APIs for file input and output provide applications with some way to write information into a flat file that applications see as a single stream of bytes that can grow as large as necessary until the disk is full. For a long time these APIs have been sufficient for applications to store their persistent information. Applications have made significant innovations in how they deal with a single stream of information to provide features like incremental "fast" saves.

In a world of component objects, however, storing data in a single flat file is no longer efficient. Just as file systems arose out of the need for multiple applications to share the same storage medium, so now, component objects require a system that allows them to share storage within the conceptual framework of a single file. Even though it is possible to store the separate objects using conventional flat file storage, should one of the objects increase in size, or should you simple add another object, it becomes necessary to load the entire file into memory, insert the new object, and then save the whole file. Such a process can be extremely time-consuming.

The solution provided by OLE is to implement a second level of indirection: a file system *within* a file. Instead of requiring that a large contiguous sequence of bytes on the disk be manipulated through a single file handle with a single seek pointer, OLE structured storage defines how to treat a single file system entity as a structured collection of two types of objects – storages and streams – that act like directories and files.

Storages and Streams

OLE provides a set of services collectively called structured storage. The purpose of these services is to reduce the performance penalties and overhead associated with storing separate objects in a flat file. Instead, OLE stores the separate objects in a single, structured file consisting of two main elements: storage objects and stream objects. Together, they function like a file system within a file.

A storage object is analogous to a file system directory. Just as a directory can contain other directories and files, a storage object can contain other storage objects and stream objects. Also like a directory, a storage object tracks the locations and sizes of the storage objects and stream objects nested beneath it.

A stream object is analogous to the traditional notion of a file. Like a file, a stream contains data stored as a consecutive sequence of bytes.

An OLE compound file consists of a root storage object containing at least one stream object representing its native data along with one or more storage objects corresponding to its linked and embedded objects. The root storage object maps to a filename in whatever file system it happens to reside in. Each of the objects inside the document also is represented by a storage object containing one or more stream objects, and perhaps also containing one or more storage objects. In this way, a document can consist of an unlimited number of nested objects.

Structured storage solves the performance problem because whenever a new object is added to a compound file, or an existing object increases in size, the file does not have to be totally rewritten to storage. Instead, the new data is written to the next available location in permanent storage, and the storage object updates the table of pointers it maintains to track the locations of its storage objects and stream objects. At the same time, structured storage enables end users to interact and manage a compound file as if it were a single file rather than a nested hierarchy of separate objects.

Structured storage also provides additional benefits:

- Incremental access. If a user needs access to an object within a compound file, the user can load and save only that object, rather than the entire file.
- Multiple use. More than one end user or application can concurrently read and write information in the same compound file.
- Transaction processing. Users can read or write to OLE compound files in transacted mode, where changes made to the file are buffered and can subsequently either be committed to the file or reversed.
- Low memory saves. Structured storage provides facilities for saving files in low memory situations.

Compound Files

Although you can implement your own structured storage objects and interfaces, OLE provides a standard implementation called Compound Files. Using Compound Files saves you the work of coding your own implementation of structured storage and confers several additional benefits derived from adhering to a defined standard. These benefits include the following:

- File-system and platform independence. Since OLE's Compound Files implementation runs on top of existing flat file systems, compound files stored in FAT, NTFS, or the Macintosh file systems can be opened by applications using any one of the others.
- Browsability. Since the separate objects in a compound file are saved in a standard format and can be accessed using standard OLE interfaces and APIs, any browser utility using these interfaces and APIs can list the objects in the file, even though the data within a given object may be in a proprietary format.
- Access to certain internal data. Since the Compound Files implementation provides standard ways of writing certain types of data – summary information, for example – applications can read this data using OLE interfaces and APIs.

Structured Storage Elements

Structured storage provides the equivalent of a complete file system for storing objects through the following elements:

- The **IStorage**, **IStream**, **ILockBytes**, and **IRootStorage** interfaces, along with a set of related interfaces – **IPersistStorage** , **IPersist**, **IPersistFile**, and **IPersistStream**.
- Helper APIs that facilitate your implementation of structured storage, as well as a second set of APIs for compound files, OLE's implementation of the structured storage interfaces. OLE also provides a set of supporting structures and enumeration values used to organize parameters for interface methods and APIs.
- A set of access modes for regulating access to compound files.

Interfaces

Structured storage services are organized into three categories of interfaces. Each set represents a successive level of indirection, or abstraction, between a compound file, the objects it contains, and the physical media on which these individual components are stored.

The first category of interfaces consists of **IStorage**, **IStream**, and **IRootStorage**. The first two interfaces define how objects are stored *within* a compound file. These interfaces provide methods for opening storage elements, committing and reverting changes, copying and moving elements, and reading and writing streams. These interfaces do not understand the native data formats of the individual objects and therefore have no methods for saving those objects to persistent storage. The **IRootStorage** interface has a single method for associating a compound document with an underlying file system name. Clients are responsible for implementing these interfaces on behalf of their compound files.

The second category of interfaces consists of the **IPersist** interfaces, which objects implement to manage their persistent data. These interfaces provide methods to read the data formats of individual objects and therefore know how to store them. Objects are responsible for implementing these interfaces because clients do not know the native data formats of their nested objects. These interfaces, however, have no knowledge of specific physical storage media.

A third category consists of a single interface, **ILockBytes**, which provides methods for writing files to specific physical media, such as a hard disk or tape drive. OLE provides an **ILockBytes** interface for the operating system's file system.

API Functions

Some of the API functions for structured storage are helper functions that perform a sequence of calls to other API functions and interface methods. You can use these helper functions as short cuts.

Other API functions provide access to OLE's implementation of structured storage, Compound Files.

Still another set of API functions enable you to convert OLE 1 objects to structured storage. You can use these functions to determine if an object class is from OLE 1 and to convert objects between OLE 1 and current OLE storage formats.

Finally, there are API functions for converting and emulating other objects. These functions provide services for one server application to work with data from another application. The data can be converted to the native format of the server application by reading the data from its original format but writing it in the native format of the object application. Or the data can remain in its original format while the server application both reads and writes the data in its original format.

Access Modes

In a world where multiple processes and users can access an object simultaneously, mechanisms for controlling that access are essential. OLE provides these mechanisms by defining access modes for both storage and stream objects. The access mode specified for a parent storage object is inherited by its children, though you can place additional restrictions on the child storage or stream. A nested storage or stream object can be opened in the same mode or in a more restricted mode than that of its parent, but it cannot be opened in a less restricted mode than that of its parent.

You specify access modes by using the values listed in the **STGM** enumeration. These values serve as flags to be passed as arguments to methods in the **IStorage** interface and associated API functions. Typically, several flags are combined in the parameter *grfMode,* using a Boolean OR operation.

The flags fall into six groups: transaction flags, storage creation flags, temporary creation flag, priority flag, access permission flags, and shared access flags. Only one flag from each group can be specified at a time.

Transaction Flags

An object can be opened in either direct or transacted mode. When an object is opened in direct mode, changes are made immediately and permanently. When an object is opened in transacted mode, changes are buffered so they can be explicitly committed or reverted once editing is complete. Committed changes are saved to the object while reverted changes are discarded. Direct mode is the default access mode.

Transacted mode is not required on a parent storage object in order to use it on a nested element. A transaction for a nested element, however, is nested within the transaction for its parent storage object. Therefore, changes made to a child object cannot be committed until those made to the parent are committed, and both are not committed until the root storage object (the top-level parent) is actually written to disk. In other words, the changes move outward: inner objects publish changes to the transactions of their immediate containers.

Storage Creation Flags

Storage creation flags specify what OLE should do if an existing storage, stream, or lockbytes object has the same name as a new storage or stream object that you are creating. The default is to return an error message and not create the new object. You can use only one of these flags in a given creation call.

Temporary Creation Flag

The temporary creation flag indicates that the underlying file is to be automatically destroyed when the root storage object is released. This capability is most useful for creating temporary files.

Priority Flag

The priority flag opens a storage object in priority mode. When it opens an object, an application usually works from a snapshot copy because other applications may also be using the object at the same time. When opening a storage object in priority mode, however, the application has exclusive rights to commit changes to the object.

Priority mode enables an application to read some streams from storage before opening the object in a mode that would require the system to make a snapshot copy to be made. Since the application has exclusive access, it doesn't have to make a snapshot copy of the object. When the application subsequently opens the object in a mode where a snapshot copy is required, the application can exclude from the snapshot the streams it has already read, thereby reducing the overhead of opening the object.

Since other applications cannot commit changes to an object while it is open in priority mode, applications should keep it in that mode for as short a time as possible.

Access Permission Flags

Access permission flags specify the type of access a client application has to an open object: read, write, or read/write. Read permission enables an application to read the contents of a stream but not to write changes to the stream. Write permission enables an application to call a function that commits changes to an object's storage but not to read the contents of the stream. Read/write permission enables an application to either read the object's streams or write to the object's storage.

Shared Access Flags

The shared access flags specify the degree of access other applications can have to an object that your application is opening. You can deny read access, deny write access, or deny all access. You can also specify explicitly that no type of access be denied.

Storage Object Naming Conventions

Storage and stream objects are named according to a set of conventions.

The name of a root storage object is the actual name of the file in the underlying file system. It obeys the conventions and restrictions the file system imposes. Filename strings passed to storage-related methods and functions are passed on, uninterpreted and unchanged, to the file system.

The name of a nested element contained within a storage object is managed by the implementation of the particular storage object. All implementations of storage objects must support nested element names 32 characters in length (including the NULL terminator), although some implementations might support longer names. Whether the storage object does any case conversion is implementation-defined. As a result, applications that define element names must choose names that are acceptable in either situation. The OLE implementation of compound files supports names up to 32 characters in length, and does not perform any case conversion.

IAdviseSink

The **IAdviseSink** interface enables containers and other interested objects to receive notifications of data changes, view changes, and compound-document changes occurring in objects of interest. Container applications, for example, require such notifications to keep cached presentations of their linked and embedded objects up-to-date.

For an advisory connection to exist, the object that is to receive notifications must implement **IAdviseSink**, and the objects in which it is interested must implement one or more of the following: **IOleObject::Advise**, **IDataObject::DAdvise**, **or** **IViewObject::SetAdvise**. Objects implementing **IOleObject** must support all of these methods.

As shown in the following table, an object that has implemented an advise sink registers its interest in receiving certain types of notifications by calling the appropriate method:

Call This Method	**To Register for These Notifications**
IOleObject::Advise	When a document is saved, closed, or renamed.
IDataObject::DAdvise	When a document's data changes.
IViewObject::SetAdvise	When a document's presentation changes.

When an event occurs that applies to a registered notification type, the object application calls the appropriate **IAdviseSink** method. For example, when an embedded object closes, it calls the **IAdviseSink::OnClose** method to notify its container. These notifications are asynchronous, occurring after the events that trigger them

When to Implement

Objects, such as container applications, compound documents, and client sites implement **IAdviseSink** to receive notification of changes in data, presentation, name, or state of their linked and embedded objects. Implementors register for one or more types of notification, depending on their needs.

Notifications for an embedded object originate in the object application and flow to the container by way of the object handler. If the object is a linked object, the OLE link object intercepts the notifications from the object handler and notifies the container directly. All containers, the object handler, and the OLE link object register for OLE notifications. The typical container also registers for view notifications. Data notifications are usually sent to the OLE link object and object handler.

When to Use

Objects call **IAdviseSink** to notify other objects with whom they have an advisory connection of changes in data, view, name, or state.

Note Making synchronous calls within asynchronous methods is not valid, so you cannot make synchronous calls within any of the the **IAdviseSink** interface's methods. For example, an implementation of **IAdviseSink::OnDataChange** cannot contain a call to **IDataObject::GetData**.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IAdviseSink Methods **Description**

OnDataChange	Advises that data has changed.
OnViewChange	Advises that view of object has changed.
OnRename	Advises that name of object has changed.
OnSave	Advises that object has been saved to disk.
OnClose	Advises that object has been closed.

See Also

IAdviseSink2, **IDataAdviseHolder**, **IOleAdviseHolder**, **IOleObject::Advise**, **IOleObject::Unadvise**, **IOleObject::EnumAdvise**

IAdviseSink::OnClose

Notifies all registered advisory sinks that the object has changed from the running to the loaded state.

Void OnClose();

Comments

OnClose notification indicates that an object is making the transition from the running to the loaded state, so that appropriate measures to ensure orderly shutdown should be taken. For example, an object handler must release its pointer to the object.

If the object that is closing is the last open object supported by its OLE server application, then the application can also shut down.

In the case of a link object, the notification that the object is closing should always be interpreted to mean that the connection to the link source has broken.

IAdviseSink::OnDataChange

Notifies a data object's currently registered advise sinks that data in the object has changed.

```
void OnDataChange(
   FORMATETC * pFormatetc,          //Pointer to format information
   STGMEDIUM * pStgmed              //Pointer to storage medium
 );
```

Parameters

pFormatetc
[in] Points to the **FORMATETC** structure, which describes the format, target device, rendering, and storage information of the calling data object.

pStgmed
[in] Points to the **STGMEDIUM** structure, which defines the storage medium (global memory, disk file, storage object, stream object, GDI object, or undefined) and ownership of that medium for the calling data object.

Comments

Object handlers and link objects implement **IAdviseSink::OnDataChange** to receive data-change notifications. They also must call **IDataObject::DAdvise** to set up an advisory connection with the data objects in which they are interested. (See **IDataObject::DAdvise** for more information on how to specify an advisory connection for data objects.)

Containers that take advantage of OLE's caching support do not need to register for data-change notifications, because the information necessary to update the container's presentation of the object, including any changes in its data, are maintained in the object's cache.

Notes to Implementors

If you implement **IAdviseSink::OnDataChange** for a container, remember that this method is asynchronous and that making synchronous calls within asynchronous methods is not valid. Therefore, you cannot call **IDataObject::GetData** to obtain the data you need to update your object. Instead, you either post an internal message or invalidate the rectangle for the changed data by calling **InvalidateRect** and then wait for a WM_PAINT message, at which point you are free to get the data and update the object.

The data itself, which is valid only for the duration of the call, is passed using the storage medium pointed to by *pmedium.* Since the medium is owned by the caller, the advise sink should not free it. Also, if *pmedium* points to an **IStorage** or **IStream**, the sink must not increment the reference count.

See Also

IDataObject::DAdvise

IAdviseSink::OnRename

Notifies all registered advisory sinks that the object has been renamed.

```
Void OnRename(
    IMoniker * pmk          //Points to the IMoniker interface
  );
```

Parameter

pmk
Pointer to the new full moniker of the object.

Comments

OLE link objects normally implement **IAdviseSink::OnRename** to receive notification of a change in the name of a link source or its container. The object serving as the link source calls **OnRename** and passes its new full moniker to the object handler, which forwards the notification to the link object. In response, the link object must update its moniker. The link object, in turn, forwards the notification to its own container.

IAdviseSink::OnSave

Notifies all registered advisory sinks that the object has been saved.

Void OnSave();

Comments

Object handlers and link objects normally implement **IAdviseSink::OnSave** to receive notifications of when an object is saved to disk, either to its original storage (through a Save operation) or to new storage (through a Save As operation). Object Handlers and link objects register to be notified when an object is saved for the purpose of updating their caches, but then only if the advise flag passed during registration specifies ADVFCACHE_ONSAVE. Object handlers and link objects forward these notifications to their containers.

IAdviseSink::OnViewChange

Notifies an object's registered advise sinks that its view has changed.

```
Void OnViewChange(
   DWORD dwAspect,        //A value specifying aspect of object
   LONG lindex            //Currently must be -1
  );
```

Parameters

dwAspect
[in] Specifies the aspect, or view, of the object. Contains a value taken from the enumeration, **DVASPECT**.

lindex
[in] Identifies which piece of the view has changed. Currently only -1 is valid.

Comments

Containers register to be notified when an object's view changes by calling **IViewObject::SetAdvise**. Once registered, the object will call the sink's **IAdviseSink::OnViewChange** method when appropriate. **OnViewChange** can be called when the object is in either the loaded or running state.

Even though **DVASPECT** values are individual flag bits, *dwAspect* may only represent one value. That is, *dwAspect* cannot contain the result of an OR operation combining two or more **DVASPECT** values.

The *lindex* member represents the part of the aspect that is of interest. The value of *lindex* depends on the value of *dwAspect*. If *dwAspect* is either DVASPECT_THUMBNAIL or DVASPECT_ICON, *lindex* is ignored. If *dwAspect* is DVASPECT_CONTENT, *lindex* must be -1, which indicates that the entire view is of interest and is the only value that is currently valid.

See Also

IViewObject::SetAdvise

IAdviseSink2

The **IAdviseSink2** interface is an extension of **IAdviseSink** and was defined to prevent overloading the **IAdviseSink::OnRename** method. Without **IAdviseSink2**, embedded objects and link sources, upon being renamed, would both have to notify their containers by calling **IAdviseSink::OnRename**. In applications where different blocks of code might execute depending on which of these two similar events has occurred, using the same method for both events complicates testing and debugging. With **IAdviseSink2**, embedded objects whose name has changed still call **IAdviseSink::OnRename**, but the link object maps this notification to **IAdviseSink2::OnLinkSrcChange**. Making this distinction explicit makes it easier to test and maintain your code.

When to Implement

If your application supports links, you should definitely implement **IAdviseSink2**. Even if your application does not support links, it may do so in future releases. Since the **IAdviseSink2** interface inherits all the methods of **IAdviseSink** without additional overhead, it makes sense to implement **IAdviseSink2** in all cases.

When to Use

When a link source is renamed, the link object should notify its container by calling **IAdviseSink::OnLinkSrcChange**.

Methods in Vtable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IAdviseSink Methods	Description
OnDataChange	Advises that data has changed.
OnViewChange	Advises that view of object has changed.
OnRename	Advises that name of object has changed.
OnSave	Advises that object has been saved to disk.
OnClose	Advises that object has been closed.

IAdviseSink2 Method	Description
OnLinkSrcChange	Advises that link source has changed.

IAdviseSink2::OnLinkSrcChange

Notifies the container that registered the advise sink that a link source has changed, enabling the container to update the link's moniker.

```
void OnLinkSrcChange(
    IMoniker *pmk        //A pointer to the new link source
  );
```

Parameter

pmk
[in] Points to the new link source contained within the link object.

Comments

A container of linked objects implements this method to receive notification in the event of a change in the moniker of its link source.

IAdviseSink2::OnLinkSrcChange is called by the OLE link object when it receives the **OnRename** notification from the link-source (object) application. The link object updates its moniker and sends the **OnLinkSrcChange** notification to containers that have implemented **IAdviseSink2**.

Notes to Implementors

Nothing prevents a link object from notifying its container of the moniker change by calling **IAdviseSink::OnRename** instead of **OnLinkSrcChange**. In practice, however, overloading **OnRename** to mean either that a link object's moniker has changed or that the corresponding link source's moniker has changed makes it difficult for applications to determine which event has, in fact, occurred. If the two events trigger different processing, as will often be the case, then using a different method for each makes the job of determining which event occurred much easier.

See Also

IAdviseSink::OnRename

IBindCtx

The **IBindCtx** interface provides access to a bind context, which is an object that stores information about a particular moniker binding operation. You pass a bind context as a parameter when calling many methods of **IMoniker** and in certain API functions related to monikers.

A bind context includes the following information:

- A **BIND_OPTS** structure containing a set of parameters that do not change during the binding operation. When a composite moniker is bound, each component uses the same bind context; a bind context is thus a mechanism for passing the same parameters to each component of a composite moniker.
- A set of pointers to objects that the binding operation has activated. The bind context holds pointers to these bound objects, keeping them loaded and thus eliminating redundant activations if the objects are needed again during subsequent binding operations.
- A pointer to the Running Object Table on the machine of the process that started the bind operation. Moniker implementations that need to access the Running Object Table should use the **IBindCtx::GetRunningObjectTable** method rather than using the **GetRunningObjectTable** API function. This allows future enhancements to the system's **IBindCtx** implementation to modify binding behavior.
- A table of interface pointers, each associated with a string key. This capability enables moniker implementations to store interface pointers under a well-known string so that they can later be retrieved from the bind context. For example, OLE defines several string keys (e.g., "ExceededDeadline", "ConnectManually") that can be used to store a pointer to the object that caused an error during a binding operation.

When to Implement

You do not need to implement this interface. The system provides an **IBindCtx** implementation, accessible using the **CreateBindCtx** API function, that is suitable for all situations.

When to Use

The primary users of **IBindCtx** are those who write their own implementation of the **IMoniker** interface; they must call **IBindCtx** methods from the implementation of several **IMoniker** methods. Another category of **IBindCtx** users are moniker providers (that is, those who hand out monikers that identify their objects). They may need to call **IBindCtx** methods from their implementations of the **IOleItemContainer** or **IParseDisplayName** interfaces.

Moniker clients (that is, those who use monikers to acquire interface pointers to objects) typically don't call many **IBindCtx** methods; they simply pass a bind context as a parameter to the **IMoniker** method they're using. Moniker clients typically perform the following steps:

1. Call the **CreateBindCtx** API function to create a bind context.
2. Optionally, call the **IBindCtx::SetBindOptions** method in order to specify the bind options. Moniker clients rarely perform this step.
3. Pass the bind context as a parameter to the desired **IMoniker** method (most commonly **IMoniker::BindToObject**).
4. Call **IUnknown::Release** on the bind context to release it.

The most common example of moniker clients are applications that act as link containers, that is, container applications that allow their documents to contain linked objects. However, link containers are a special case in that they rarely call **IMoniker** methods directly. Generally, they manipulate linked objects through the **IOleLink** interface; the system-supplied linked object implements this interface and calls the appropriate **IMoniker** methods as needed. Consequently, the linked object is the one that passes bind contexts to the **IMoniker** methods.

Methods in Vtable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IBindCtx Methods	Description
RegisterObjectBound	Registers an object with the bind context.
RevokeObjectBound	Revokes an object's registration.
ReleaseBoundObjects	Releases all registered objects.
SetBindOptions	Sets the binding options.
GetBindOptions	Retrieves the binding options.
GetRunningObjectTable	Retrieves a pointer to the Running Object Table.
RegisterObjectParam	Associates an object with a string key.
GetObjectParam	Returns the object associated with a given string key.
EnumObjectParam	Enumerates all the string keys in the table.
RevokeObjectParam	Revokes association between an object and a string key.

See Also

CreateBindCtx, IMoniker, IOleItemContainer, IParseDisplayName

IBindCtx::EnumObjectParam

Returns a pointer to an enumerator that can return the keys of the bind context's string-keyed table of pointers.

```
HRESULT EnumObjectParam(
  IEnumString **ppenum        //Receives a pointer to an enumerator
 );
```

Parameter

ppenum
[out] Receives an **IEnumString** pointer to the enumerator. If an error occurs, **ppenum* is set to NULL. If **ppenum* is non-NULL, the implementation calls **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK
An enumerator was successfully returned.

E_OUTOFMEMORY
Indicates insufficient memory.

Comments

This method eturns an **IEnumString** pointer to an enumerator that can return the keys of the bind context's string-keyed table of pointers. The keys returned are the ones previously specified in calls to **IBindCtx::RegisterObjectParam**.

Notes to Callers

A bind context maintains a table of interface pointers, each associated with a string key. This enables communication between a moniker implementation and the caller that initiated the binding operation. One party can store an interface pointer under a string known to both parties so that the other party can later retrieve it from the bind context.

See Also

IBindCtx::RegisterObjectParam, **IEnumString**

IBindCtx::GetBindOptions

Returns the binding options stored in this bind context.

```
HRESULT GetBindOptions(
   BIND_OPTS *pbindopts        //Points to a BIND_OPTS structure
  );
```

Parameter

pbindopts
[in, out] Points to a **BIND_OPTS** structure that receives the current binding parameters. For more information on the **BIND_OPTS** structure, see the "Data Structures" section.

Return Values

S_OK
The stored binding options were successfully returned.

E_UNEXPECTED
Indicates an unexpected error.

Comments

A bind context contains a block of parameters, stored in a **BIND_OPTS** structure, that are common to most **IMoniker** operations and that do not change as the operation moves from piece to piece of a composite moniker.

Notes to Callers

You typically call this method if you are writing your own moniker class (that is, implementing the **IMoniker** interface). You call this method to retrieve the parameters specified by the moniker client.

You must provide the **BIND_OPTS** structure that is filled in by this method. Before calling this method, you must initialize the **cbStruct** field of the structure to the size of the **BIND_OPTS** structure.

See Also

IBindCtx::SetBindOptions

IBindCtx::GetObjectParam

Retrieves the pointer associated with the specified key in the bind context's string-keyed table of pointers.

```
HRESULT GetObjectParam(
   LPOLESTR pszKey,        //Points to the key to be used
   IUnknown **ppunk        //Receives a pointer to the object associated with the key
  );
```

Parameters

pszKey

[in] Points to a zero-terminated string containing the key to search for. For Win32 applications, the **LPOLESTR** type indicates a wide character string (two bytes per character); otherwise, the string has one byte per character. Key string comparison is case sensitive.

ppunk

[out] Receives a pointer to the object associated with *pszKey*. If an error occurs, **ppunk* is set to NULL. If **ppunk* is non-NULL, the implementation calls **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK

The pointer associated with the specified key was successfully returned.

E_FAIL

Indicates that no object has been registered with the specified key.

Comments

A bind context maintains a table of interface pointers, each associated with a string key. This enables communication between a moniker implementation and the caller that initiated the binding operation. One party can store an interface pointer under a string known to both parties so that the other party can later retrieve it from the bind context.

The pointer this method retrieves must have previously been inserted into the table using the **IBindCtx::RegisterObjectParam** method.

Notes to Callers

You can call **IBindCtx::GetObjectParam** if you are either a moniker implementor or a sophisticated moniker client.

If you are a moniker client, you can call this method if a binding operation failed to get specific information about the error that occurred. Depending on the error, you may be able to correct the situation and retry the binding operation. See **IBindCtx::RegisterObjectParam** for more information.

If you are a moniker implementor, you can call this method to retrieve information specified by the caller that initiated the binding operation. By convention, you should use key names that begin with the string form of the CLSID of your moniker class (see the **StringFromCLSID** API function).

See Also

IBindCtx::RegisterObjectParam, **IBindCtx::EnumObjectParam**

IBindCtx::GetRunningObjectTable

Returns an interface pointer to the Running Object Table (ROT) for the machine on which this bind context is running.

```
HRESULT GetRunningObjectTable(
   IRunningObjectTable **pprot         //Receives a pointer to the Running Object Table
  );
```

Parameter

pprot

[out] Receives a pointer to the Running Object Table. If an error occurs, **pprot* is set to NULL. If **pprot* is non-NULL, the implementation calls **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK

A pointer to the ROT was returned successfully.

E_OUTOFMEMORY

Indicates insufficient memory.

E_UNEXPECTED

Indicates an unexpected error.

Comments

The Running Object Table is a globally-accessible table on each machine. It keeps track of all the objects that are currently running on the machine.

Notes to Callers

You typically call this method if you are writing your own moniker class (that is, implementing the **IMoniker** interface). You can call this method from your implementation of **IMoniker::BindToObject** or **IMoniker::IsRunning** to check whether a given object is currently running. You can also call this method from your implementation of **IMoniker::GetTimeOfLastChange** to learn when a running object was last modified.

Moniker implementations should always use this method instead of using the **GetRunningObjectTable** API function. This makes it possible for future implementations of **IBindCtx** to modify binding behavior.

See Also

IMoniker, **IRunningObjectTable**

IBindCtx::RegisterObjectBound

Calls the **IUnknown::AddRef** method on the specified object to ensure that the object remains active until the bind context is released. The method stores a pointer to the object in the bind context's internal list of pointers.

```
HRESULT RegisterObjectBound(
  IUnknown *punk        //Pointer to the object being registered
 );
```

Parameter

punk
[in] Points to the object that is being registered as bound.

Return Values

S_OK
The object was successfully registered.

E_OUTOFMEMORY
Indicates insufficient memory.

Comments

Notes to Callers

If you are writing your own moniker class (that is, implementing the **IMoniker** interface), you should call this method whenever your implementation activates an object. This happens most often in the course of binding a moniker, but it can also happen while retrieving a moniker's display name, parsing a display name into a moniker, or retrieving the time an object was last modified.

Calling the **IBindCtx::RegisterObjectBound** creates an additional reference to the passed-in object (via **IUnknown::AddRef**); you are still required to release your own copy of the pointer. Note that calling this method twice for the same object creates two references to that object; you can release a reference to an object using the **IBindCtx::RevokeObjectBound** method. All references held by the bind context are released when the bind context itself is released.

Registering an object with a bind context keeps the object active until the bind context is released. Reusing a bind context in a subsequent binding operation (either for another piece of the same composite moniker, or for a different moniker) can make the subsequent binding operation more efficient because it doesn't have to reload that object. This performance improvement is present only if the subsequent binding operation requires some of the same objects as the original one. This performance improvement should be balanced against the costs of keeping objects activated unnecessarily.

Note that **IBindCtx** does not provide a method to retrieve a pointer to an object registered using **IBindCtx::RegisterObjectBound**. Assuming the object has registered itself with the Running Object Table, moniker implementations can use the Running Object Table to retrieve a pointer to the object.

See Also

IBindCtx::ReleaseBoundObjects, **IBindCtx::RevokeObjectBound**, **IRunningObjectTable::GetObject**

IBindCtx::RegisterObjectParam

Stores a pointer to the specified object under the specified key in the bind context's string-keyed table of pointers. The method calls **IUnknown::AddRef** on the stored pointer.

```
HRESULT RegisterObjectParam(
   LPOLESTR pszKey,         //Points to the key to be used
   IUnknown *punk           //Points to the object to be associated with the key
  );
```

Parameters

pszKey
: [in] Points to a zero-terminated string containing the key under which the object is being registered. For Win32 applications, the **LPOLESTR** type indicates a wide character string (two bytes per character); otherwise, the string has one byte per character. Key string comparsion is case sensitive.

punk
: [in] Points to the object that is to be registered.

Return Values

S_OK
: The pointer was successfully registered under the specified string.

E_OUTOFMEMORY
: Indicates insufficient memory.

Comments

A bind context maintains a table of interface pointers, each associated with a string key. This enables communication between a moniker implementation and the caller that initiated the binding operation. One party can store an interface pointer under a string known to both parties so that the other party can later retrieve it from the bind context.

Binding operations subsequent to the use of this method can use the **IBindCtx::GetObjectParam** method to retrieve the stored pointer.

Notes to Callers

You can call **IBindCtx::RegisterObjectParam** if you are either a moniker implementor or a sophisticated moniker client.

If you're a moniker implementor and an error occurs during moniker binding, you can use one of the keys predefined by OLE to inform the caller of the cause of an error. The following list describes the keys associated with various error conditions:

MK_E_EXCEEDEDDEADLINE
: If a binding operation exceeds its deadline because a given object is not running, then you should register the object's moniker using the first unused key from the list: "ExceededDeadline", "ExceededDeadline1", "ExceededDeadline2", etc. If the caller later finds the moniker in the Running Object Table, the caller can retry the binding operation.

MK_E_CONNECTMANUALLY
: The "ConnectManually" key indicates a moniker whose binding requires assistance from the end user. The caller can retry the binding operation after showing the moniker's display name to request that the end user manually connect to the object. The most common reasons for returning this value are that a password is needed or that a floppy needs to be mounted.

E_CLASSNOTFOUND
: The "ClassNotFound" key indicates a moniker whose class could not be found (i.e., the server for the object identified by this moniker could not be located). If this key is used in an OLE compound-document situation, the caller can use **IMoniker::BindToStorage** to bind to the object, and then try

to carry out a Treat As... or Convert To... operation to associate the object with a different server. If this is successful, the caller can retry the binding operation.

If you're a moniker client with detailed knowledge of the implementation of the moniker you're using, you can also call this method to pass private information to that implementation.

You can define new strings as keys for storing pointers. By convention, you should use key names that begin with the string form of the CLSID of the moniker class (see the **StringFromCLSID** API function).

If the *pszKey* parameter matches the name of an existing key in the bind context's table, the new object replaces the existing object in the table.

When you register an object using this method, the object is not released until one of the following occurs:

- It is replaced in the table by another object with the same key.
- It is removed from the table by a call to **IBindCtx::RevokeObjectParam**.
- The bind context is released. All registered objects are released when the bind context is released.

See Also

IBindCtx::GetObjectParam, **IBindCtx::RevokeObjectParam**, **IBindCtx::EnumObjectParam**

IBindCtx::ReleaseBoundObjects

Releases all pointers to all objects that were previously registered by calls to **IBindCtx::RegisterObjectBound**.

HRESULT ReleaseBoundObjects(*void***);**

Return Value

S_OK
    The objects were released successfully.

Comments

You rarely call this method directly. This method is called the system's **IBindCtx** implementation when a bind context is released. If a bind context is not released, all of the registered objects remain active.

If the same object has been registered more than once, this method calls the **IUnknown::Release** method on the object the number of times it was registered.

See Also

IBindCtx::RegisterObjectBound

IBindCtx::RevokeObjectBound

Calls the **IUnknown::Release** method on the specified object and removes the pointer to that object from the bind context's internal list of pointers. This undoes a previous call to **IBindCtx::RegisterObjectBound** for the same object.

```
HRESULT RevokeObjectBound(
    IUnknown *punk          //Pointer to the object whose registration is being revoked
  );
```

Parameter

punk
[in] Points to the object to be released.

Return Values

S_OK
The object was released successfully.

MK_E_NOTBOUND
Indicates that *punk* was not previously registered with a call to **IBindCtx::RegisterObjectBound**.

Comments

You rarely call this method. This method is included for completeness.

IBindCtx::RevokeObjectParam

Removes the specified key and its associated pointer from the bind context's string-keyed table of objects. The key must have previously been inserted into the table using the **IBindCtx::RegisterObjectParam** method.

```
HRESULT RevokeObjectParam(
  LPOLESTR pszKey        //Points to the key to be revoked
 );
```

Parameter

pszKey

[in] Points to a zero-terminated string containing the key to remove. For Win32 applications, the **LPOLESTR** type indicates a wide character string (two bytes per character); otherwise, the string has one byte per character. Key string comparsion is case sensitive.

Return Values

S_OK

The specified key was successfully removed from the table.

S_FALSE

Indicates that no object has been registered with the specified key.

Comments

A bind context maintains a table of interface pointers, each associated with a string key. This enables communication between a moniker implementation and the caller that initiated the binding operation. One party can store an interface pointer under a string known to both parties so that the other party can later retrieve it from the bind context.

This method is used to remove an entry from the table. If the specified key is found, the bind context releases its reference to the object.

See Also

IBindCtx::RegisterObjectParam

IBindCtx::SetBindOptions

Specifies new values for the binding parameters stored in the bind context. Subsequent binding operations can call **IBindCtx::GetBindOptions** to retrieve the parameters.

```
HRESULT SetBindOptions(
  BIND_OPTS *pbindopts        //Pointer to a BIND_OPTS structure
 );
```

Parameter

pbindopts

[in] Points to a **BIND_OPTS** structure containing the binding parameters. For more information on the **BIND_OPTS** structure, see the "Data Structures" section.

Return Values

S_OK

The parameters were stored successfully.

E_OUTOFMEMORY

Indicates insufficient memory.

Comments

A bind context contains a block of parameters, stored in a **BIND_OPTS** structure, that are common to most **IMoniker** operations and that do not change as the operation moves from piece to piece of a composite moniker.

Notes to Callers

This method can be called by moniker clients, that is, those who use monikers to acquire interface pointers to objects.

When you first create a bind context using the **CreateBindCtx** API function, the fields of the **BIND_OPTS** structure are initialized to the following values:

```
cbStruct = sizeof(BINDOPTS);
grfFlags = 0;
grfMode = STGM_READWRITE;
dwTickCountDeadline = 0;
```

You can use the **IBindCtx::SetBindOptions** method to modify these values before using the bind context, if you want values other than the defaults. See **BIND_OPTS** for more information.

See Also

IBindCtx::GetBindOptions

IClassFactory

The **IClassFactory** interface manages the creation of new objects. This interface is supported by class objects associated with CLSIDs.

When to Implement

You must implement this interface for every object definition so objects of that type can be created.

When to Use

Call the methods of this interface when creating new objects through a class object. First, call the **CoGetClassObject** API function to get an **IClassFactory** interface pointer to the class object, and then call the **CreateInstance** method to create an uninitialized object.

It is, however, not always necessary to go through this process to create an object. To create a single uninitialized object, you can, instead, just call **CoCreateInstance**. OLE also provides numerous helper functions (with names of the form **OleCreate***Xxx*) to create compound document objects.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IClassFactory Methods	**Description**
CreateInstance	Creates an uninitialized object.
LockServer	Locks object application open in memory.

See Also

CoGetClassObject, **CoCreateInstance** , **OleCreate**

IClassFactory::CreateInstance

Creates an uninitialized object.

```
HRESULT CreateInstance(
   IUnknown * pUnkOuter,         //Object is or isn't part of an aggregate
   REFIID riid,                  //Interface identifier
   void ** ppvObject             //On return, pointer to interface pointer location
  );
```

Parameters

pUnkOuter

[in] Indicates whether the object is being created as part of an aggregate. If so, you should use *pUnkOuter* to supply a pointer to the controlling unknown of the aggregate that will use the newly created object. If not, *pUnkOuter* should be NULL.

riid

[in]Specifies the IID of the interface to be used to communicate with the newly created object. If *pUnkOuter* is NULL, this parameter is frequently the IID of the initializing interface; if *pUnkOuter* is non-NULL, *riid* must be **IID_IUnknown**.

ppvObject

[out]On return, points to the location of the requested interface pointer. If the object does not support the interface specified in *riid*, *ppvObj* is set to NULL.

Return Values

S_OK

The specified object was created.

CLASS_E_NOAGGREGATION

The *pUnkOuter* parameter was non-NULL and the object does not support aggregation.

E_NOINTERFACE

The object that *ppvObj* points to does not support the interface identified by *riid*.

E_UNEXPECTED

An unexpected error occurred.

E_OUTOFMEMORY

Out of memory.

E_INVALIDARG

One or more arguments are invalid.

Comments

The **IClassFactory** interface is always on a class object. The **CreateInstance** method creates an uninitialized object, which is defined in the code associated with the class object.

The *pUnkOuter* parameter indicates whether the object is being created as part of an aggregate. Object definitions are not required to support aggregation – they must be specifically designed and implemented to support it.

The *riid* parameter specifies the IID (interface ID) of the interface through which you communicate with the new object. If *pUnkOuter* is non-NULL (indicating aggregation), the value of the *riid* parameter must be **IID_IUnknown**. If the object is not part of an aggregate, *riid* often specifies the interface though which the object will be initialized.

For OLE embeddings, the initialization interface is **IPersistStorage**, but in other situations, other interfaces are used. To initialize the object, there must be a subsequent call to an appropriate method in the initializing interface. Common initialization functions include **IPersistStorage::InitNew** (for new, blank embeddable components), **IPersistStorage::Load** (for reloaded embeddable components),

IPersistStream::Load, or **IPersistFile::Load**.

In general, if an application supports one object definition, and the class object is registered for single use, only one object can be created. The application must not create other objects, and a request to do so should return an error from **IClassFactory::CreateInstance**. The same is true for applications that support multiple classes, each with a class object registered for single use; a **CreateInstance** for one class followed by a **CreateInstance** for any of the classes should return an error.

To avoid returning an error, applications that support multiple object definitions with single-use class objects can revoke the registered class object of the first definition by calling **CoRevokeClassObject** when a request for instantiating a second is received. For example, suppose there are two classes, A and B. When **IClassFactory::CreateInstance** is called for class A, revoke the class object for B. When B is created, revoke the class object for A. This solution complicates shutdown because one of the class objects might have already been revoked (and cannot be revoked twice).

IClassFactory::LockServer

Called by the client of a class object to keep an open object application in memory, allowing instances to be created more quickly.

```
HRESULT LockServer(
   BOOL fLock          //Increments or decrements the lock count
  );
```

Parameter

fLock
If **TRUE**, increments the lock count; if **FALSE**, decrements the lock count.

Return Values

S_OK
The specified object was either locked (*fLock* = TRUE) or unlocked from memory (*fLock* = FALSE).

E_FAIL
Indicates an unspecified error.

E_OUTOFMEMORY
Out of memory.

E_UNEXPECTED
An unexpected error occurred.

Comments

IClassFactory::LockServer controls whether an object application is kept in memory. Keeping the application alive in memory allows instances to be created more quickly.

Notes to Callers

Most clients do not need to call this function. It is provided only for the benefit of sophisticated clients that require special performance in the creation of certain kinds of objects.

Notes to Implementors

If the lock count is zero, there are no more objects in use, and the application is not under user control, then the server can be closed. One way to implement **IClassFactory::LockServer** is to call **CoLockObjectExternal**.

The process that locks the object application is responsible for unlocking it. Once the class object is released, there is no mechanism that guarantees the caller connection to the same object definition later (as in the case where a class object is registered as single-use). It is important to count all calls, not just the last one, to **IClassFactory::LockServer**, because calls must be balanced before attempting to release the pointer to the class object or an error results. For every call to **LockServer** with *fLock* set to TRUE, there must be a call to **LockServer** with *fLock* set to FALSE. When the lock count and the class object reference count are both zero, the class object can be freed.

See Also

CoLockObjectExternal

IDataAdviseHolder

The **IDataAdviseHolder** interface creates and manages connections between an advise sink and a data object so that the advise sink can receive notification of changes in the data object. The **IDataAdviseHolder** interface also sends the notifications to the advise sinks it currently knows about when changes occur.

The object or container wishing to receive notification of changes in your data object creates an advise sink object with the **IAdviseSink** interface. The object or container application then calls your **IDataObject::DAdvise** method.

Typically, in your **IDataObject** implementation, you delegate handling of the advise sinks to the OLE-provided **IDataAdviseHolder** interface. This interface has methods to keep track of any **IDataObject::DAdvise** calls that your data object has received and to send notification back to the advise sinks as necessary when data has changed.

When to implement

Typically, you use the OLE-provided implementation of **IDataAdviseHolder** to simplify your implementation of the **DAdvise**, **DUnadvise**, and **EnumDAdvise** methods in the **IDataObject** interface.

When to use

Your implementation of **IDataObject** can call the methods in **IDataAdviseHolder**. The first time you receive a call to **IDataObject::DAdvise**, you call the API function **CreateDataAdviseHolder** to create an instance of the OLE-provided implementation of **IDataAdviseHolder**. Then, you delegate your **IDataObject** implementations of the **DAdvise**, **DUnadvise**, and **EnumDAdvise** methods to the corresponding methods in the OLE-provided **IDataAdviseHolder**.

When the data of interest to an advise sink actually changes, you call **IDataAdviseHolder::SendOnDataChange** from your data object to carry out the necessary notifications.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IDataAdviseHolder Methods	Description
Advise	Creates a connection between an advise sink and a data object so the advise sink can receive notification of change in the data object.
Unadvise	Destroys a notification connection previously set up with the **Advise** method.
EnumAdvise	Returns an object that can be used to enumerate the current advisory connections.
SendOnDataChange	Sends a change notification back to each advise sink that is currently being managed.

See Also

IDataObject, **IAdviseSink**

IDataAdviseHolder::Advise

Creates a connection between an advise sink and a data object for receiving notifications.

```
HRESULT Advise(
  IDataObject * pDataObject,      //Specifies the data object for which notifications are requested
  FORMATETC * pFormatetc,         //Description of data that is of interest to the advise sink
  DWORD advf,                     //Flags that specify how the notification takes place
  IAdviseSink * pAdvSink,         //The advise sink requesting notification
  DWORD * pdwConnection           //Identifies the connection for future calls to the Unadvise method
 );
```

Parameters

pDataObject

[in]Points to the data object for which notifications are requested. If data in this object changes, a notification is sent to the advise sinks that have requested notification.

pFormatetc

[in]Points to the specified format, medium, and target device that is of interest to the advise sink requesting notification. For example, one sink may want to know only when the bitmap representation of the data in the the data object changes. Another sink may be interested in only the metafile format of the same object. Each advise sink is notified when the data of interest changes. This data is passed back to the advise sink when notification occurs.

advf

[in]Contains a group of flags for controlling the advisory connection. Valid values are from the enumeration **ADVF**. However, only some of the possible **ADVF** values are relevant for this method. The following table briefly describes the relevant values; a more detailed description can be found in the description of the **ADVF** enumeration.

ADVF Value	Description
ADVF_NODATA	Asks that no data be sent along with the notification.
ADVF_ONLYONCE	Causes the advisory connection to be destroyed after the first notification is sent. An implicit call to **IDataAdviseHolder::Unadvise** is made on behalf of the caller to remove the connection.
ADVF_PRIMEFIRST	Causes an initial notification to be sent regardless of whether or not data has changed from its current state.
ADVF_DATAONSTOP	When specified with ADVF_NODATA, this flag causes a last notification with the data included to be sent before the data object is destroyed. When ADVF_NODATA is not specified, this flag has no effect.

pAdvSink

[in]Points to the advisory sink that receives the change notification.

pdwConnection

[in]Points to the location of a returned token that identifies this connection and can later be used to delete the advisory connection. It deletes the connection by passing it to

IDataAdviseHolder::Unadvise. If this value is zero, the connection was not established.

Return Values

S_OK
The advisory connection was created.

E_INVALIDARG
One or more arguments are invalid.

Comments

Through the connection established through this method, the advisory sink can receive future notifications in a call to **IAdviseSink::OnDataChange**.

An object issues a call to **IDataObject::DAdvise** to request notification on changes to the format, medium, or target device of interest. This data is specified in the *pFormatetc* parameter. The **DAdvise** method in turn calls **IDataAdviseHolder::Advise** and delegates the task of setting up and tracking a connection to the advise holder. When the format, medium, or target device in question changes, the data object calls **IDataAdviseHolder::SendOnDataChange** to actually send the necessary notifications.

The established connection can be deleted by passing the value in *pdwConnection* to **IDataAdviseHolder::Unadvise**.

See also

ADVF, **CreateDataAdviseHolder**, **FORMATETC**, **IDataAdviseHolder::Unadvise**, **IDataObject::DAdvise**

IDataAdviseHolder::EnumAdvise

Returns an object that can be used to enumerate the current advisory connections.

```
HRESULT EnumAdvise(
   IEnumSTATDATA ** ppenumAdvise         //Location for IEnumSTATDATA
  );
```

Parameter

ppenumAdvise
[out]Points to the location where the new enumerator object should be returned. If this value is NULL, there are no connections to advise sinks at this time.

Return Values

S_OK
The enumerator object is successfully instantiated or there are no connections.

E_OUTOFMEMORY
The enumerator object could not be instantiated due to lack of memory.

Comments

The enumerator object returned by this method implements the **IEnumSTATDATA** interface, one of the standard enumerator interfaces that contain the **Next**, **Reset**, **Clone**, and **Skip** methods. **IEnumSTATDATA** enumerates the data stored in an array of **STATDATA** structures. **IDataAdviseHolder::EnumAdvise** is typically called from within **IDataObject::DEnumAdvise**.

Adding more advisory connections while the enumerator object is active has an undefined effect on the enumeration that results from this method.

See Also

IEnum*XXXX*, **IEnumSTATDATA**, **IDataObject::EnumDAdvise**

IDataAdviseHolder::SendOnDataChange

Sends notifications to each advise sink by calling the **IAdviseSink::OnDataChange** method for each advise sink currently being handled in this instance of **IDataAdviseHolder**.

```
HRESULT SendOnDataChange(
  IDataObject * pDataObject,     //Points to the data object that has changed
  DWORD dwReserved,              //Reserved
  DWORD advf                     //Advise flag that specifies how to send the notification
 );
```

Parameters

pDataObject
[in]Points to the data object where the data has just changed. This data object is used in the subsequent calls to **IAdviseSink::OnDataChange**.

dwReserved
[in]Reserved for future use; must be zero.

advf
[in]Contains the advise flag values that specify how the call to **IAdviseSink::OnDataChange** is made. These flag values are from the enumeration **ADVF**. Typically, the value for *advf* is NULL. The only exception occurs when the data object is shutting down and must send a final notification that includes the actual data to sinks that have specified ADVF_DATAONSTOP and ADVF_NODATA in their call to **IDataObject::DAdvise**. In this case, *advf* contains ADVF_DATAONSTOP.

Return Values

S_OK
The call to **IAdviseSink::OnDataChange** was made.

E_OUTOFMEMORY
Ran out of memory and the notifications could not be sent.

Comments

The data object calls this method when it detects a change that would be of interest to an advise sink that has previously requested notification.

Most notifications include the actual data with them. The only exception is if the ADVF_NODATA flag was previously specified when the connection was initially set up in the **IDataAdviseHolder::Advise** method.

Before calling the **IAdviseSink::OnDataChange** method for each advise sink, this method obtains the actual data by calling the **IDataObject::GetData** method for the data object specified in the *pDataObject* parameter.

See Also

ADVF, **IAdviseSink::OnDataChange**

IDataAdviseHolder::Unadvise

Removes a connection between a data object and an advisory sink that was set up through a previous call to the **IDataAdviseHolder::Advise** method. **IDataAdviseHolder::Unadvise** is typically called in **IDataObject::DUnadvise**.

```
HRESULT Unadvise(
  DWORD dwConnection        //Specifies the connection to remove
 );
```

Parameter

dwConnection
Specifies the connection to remove. This value was returned by **IDataAdviseHolder::Advise** when the connection was originally established.

Return Values

S_OK
The specified connection was successfully deleted.

OLE_E_NOCONNECTION
The specified *dwConnection* is not a valid connection.

See Also

IDataAdviseHolder::Advise, **IDataObject::DUnadvise**

IDataObject

The **IDataObject** interface provides data transfer capabilities and notification of changes in data. The format of the transferred data is specified along with the medium through which OLE transfers the data. Optionally, the data can be rendered for a specific target device. In addition to retrieving and storing data, the **IDataObject** interface contains methods to enumerate available formats and to set up an advisory sink for handling change notifications.

A data object is an object that contains an **IDataObject** implementation. Data objects may support all or some of the functions available through the **IDataObject** interface. For example, some data objects do not allow callers to send them data. Other data objects do not support advisory connections and change notifications. However, for those data objects that do support change notifications, OLE provides an object called a data advise holder. Since a data object can have multiple connections, each with their own set of attributes, managing these connections and sending the appropriate notifications gets complex. The data advise holder exists to simplify this task.

When to Implement

You implement the **IDataObject** interface if you are developing a container or server application that is capable of transferring data. For example, if your application allows its data to be pasted or dropped into another application, you must implement the **IDataObject** interface.

OLE provides implementations for its default object handler and its cache.

When to Use

Any application that can receive data uses the methods in the **IDataObject** interface.

When you call the methods in the **IDataObject** interface, you specify a format, a medium, and, optionally, a target device for which the data should be rendered. If you want to be notified when the data changes, you can set up an advisory connection through which notifications can be sent.

Methods in VTable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IDataObject Methods	**Description**
GetData	Renders the data described in a **FORMATETC** structure and transfers it through the **STGMEDIUM** structure.
GetDataHere	Renders the data described in a **FORMATETC** structure and transfers it through the **STGMEDIUM** structure allocated by the caller.
QueryGetData	Determines whether the data object is capable of rendering the data described in the **FORMATETC** structure.
GetCanonicalFormatEtc	Provides a potentially different but logically equivalent **FORMATETC** structure.
SetData	Provides the source data object with

	data described by a **FORMATETC** structure and an **STGMEDIUM** structure.
EnumFormatEtc	Returns an object to enumerate the **FORMATETC** supported by the data object.
DAdvise	Creates a connection between a data object and an advise sink so the advise sink can receive notifications of changes in the data object.
DUnadvise	Destroys a notification previously set up with **DAdvise**.
EnumDAdvise	Returns an object to enumerate the current advisory connections.

IDataObject::DAdvise

Creates a connection between a data object and an advise sink so the advise sink can receive notifications of changes in the data object.

HRESULT DAdvise(
  FORMATETC * *pFormatetc*,   //Description of data that is of interest to the advise sink
  DWORD *advf*,   //Flags that specify how the notification takes place
  IAdviseSink * *pAdvSink*,   //The advise sink that is requesting notification
  DWORD * *pdwConnection*   //Identifies the connection for future calls to the Unadvise method
 );

Parameters

pFormatetc

[in]Points to the particular format, target device, aspect, and medium that will be used for future notifications. For example, one sink may want to know only when the bitmap representation of the data in the the data object changes. Another sink may be interested in only the metafile format of the same object. Each advise sink is notified when the data of interest changes. This data is passed back to the advise sink when notification occurs.

advf

[in]Contains a group of flags for controlling the advisory connection. Valid values are from the enumeration **ADVF**. However, only some of the possible **ADVF** values are relevant for this method. The following table briefly describes the relevant values. You can find a more detailed description in **ADVF**.

ADVF Value	**Description**
ADVF_NODATA	Asks the data object to avoid sending data with the notifications. Typically data is sent. This flag is a way to override the default behavior. When ADVF_NODATA is used, the TYMED member of the **STGMEDIUM** structure that is passed to OnDataChange will usually contain TYMED_NULL. The caller can then retrieve the data with a subsequent **IDataObject::GetData** call.
ADVF_ONLYONCE	Causes the advisory connection to be destroyed after the first change notification is sent. An implicit call to **IDataObject::DUnadvise** is made on behalf of the caller to remove the connection.
ADVF_PRIMEFIRST	Asks for an additional initial notification. The combination of ADVF_ONLYONCE and ADVF_PRIMEFIRST provides, in effect, an asynchronous **IDataObject::GetData** call.
ADVF_DATAONSTOP	When specified with ADVF_NODATA, this flag causes a last notification with the data included to to be sent before the

data object is destroyed.

If used without ADVF_NODATA, **IDataObject::DAdvise** can be implemented in one of the following ways:

- the ADVF_DATAONSTOP can be ignored.
- the object can behave as if ADVF_NODATA was specified.
- a change notification is sent only in the shutdown case. Data changes prior to shutdown do not cause a notification to be sent.

pAdvSink
[in]Points to the advisory sink that will receive the change notification.

pdwConnection
[in]Points to the location of a returned token that identifies this connection. You can use this token later to delete the advisory connection (by passing it to **IDataObject::DUnadvise**). If this value is zero, the connection was not established.

Return Values

S_OK
The advisory connection was created.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

E_OUTOFMEMORY
The connection was not created because the system ran out of memory.

E_NOTIMPL
This method is not implemented for the data object.

DV_E_LINDEX
Invalid value for *lindex*; currently, only -1 is supported.

DV_E_FORMATETC
Invalid value for *pFormatetc*.

OLE_E_ADVISENOTSUPPORTED
The data object does not support change notification.

Comments

The **IDataObject::DAdvise** method creates a connection between the data object and the caller. The caller provides an advisory sink through which the notifications can be sent when the object's data changes. The advise sink issues the call to **IDataObject::DAdvise**. Advisory connections are associated with a particular description of the data. Data clients specify the format, aspect, medium, and/or target device of interest in the **FORMATETC** structure passed in. If the data object does not support one or more of the requested attributes or the sending of notifications at all. It can refuse the connection by returning OLE_E_ADVISENOTSUPPORTED.

Objects used for data transfer typically do not support advisory notifications and return OLE_E_ADVISENOTSUPPORTED from **IDataObject::DAdvise**.

Containers of linked objects can set up advisory connections directly with the bound link source or indirectly via the standard OLE link object that manages the connection. Connections set up with the bound link source are not automatically deleted. The container must explicitly call **IDataObject::DUnAdvise** on the bound link source to delete it. Connections set up through the OLE link object are destroyed when the link object is deleted.

The OLE default link object creates a "wildcard advise" with the link source so OLE can maintain the time of last change. This advise is specifically used to note the time that anything changed. OLE ignores all data formats that may have changed, noting only the time of last change. To allow wildcard advises, set the **FORMATETC** members as follows before calling **IDataObject::DAdvise**:

```
cf == 0;
ptd == NULL;
dwAspect ==-1;
lindex == -1
tymed == -1;
```

The advise flags should also include ADVF_NODATA. Wildcard advises from OLE should always be accepted by applications.

For those data objects that support notifications, a data advise holder object can be used to simplify the implementation of **DAdvise** and the other notification methods in **IDataObject** (**DUnadvise** and **EnumAdvise**). The data advise holder is an object provided by OLE that manages the registration and sending of notifications. With the first invocation of **DAdvise**, the helper function **CreateDataAdviseHolder** can be called to create the data advise holder and provide the data object with a pointer to it. The data object then delegates to the **IDataAdviseHolder::Advise** method in the data advise holder to create the requested connection.

See also

ADVF, **FORMATETC**, **CreateDataAdviseHolder**, **IAdviseSink::OnDataChange**, **IDataObject::DUnAdvise**

IDataObject::DUnadvise

Destroys a previously set up notification

```
HRESULT DUnadvise(
   DWORD dwConnection        //Specifies the connection to remove
  );
```

Parameter

dwConnection
[in]Specifies the connection to remove. This value was returned by **IDataObject::DAdvise** when the connection was originally established.

Return Values

S_OK
The specified connection was successfully deleted.

OLE_E_NOCONNECTION
The specified *dwConnection* is not a valid connection.

OLE_E_ADVISENOTSUPPORTED
This **IDataObject** implementation does not support notification.

Comments

This methods destroys a notification created with a call to the **IDataObject::DAdvise** method.

If the advisory connection being deleted was initially set up using **IDataAdviseHolder::Advise** instead of **IDataObject::DAdvise**, you must call **IDataAdviseHolder::Unadvise** to delete it.

See Also

IDataObject::DAdvise

IDataObject::EnumDAdvise

Returns an object to enumerate the current advisory connections.

```
HRESULT EnumDAdvise(
  IEnumSTATDATA ** ppenumAdvise      //Location for IEnumSTATDATA
 );
```

Parameter

ppenumAdvise
[out]Points to the location where the new enumerator object should be returned. If the returned value is NULL, there are no connections to advise sinks at this time.

Return Values

S_OK
The enumerator object is successfully instantiated or there are no connections.

E_OUTOFMEMORY
The enumerator object could not be instantiated due to lack of memory.

OLE_E_ADVISENOTSUPPORTED
Advisory notifications are not supported by this object.

Comments

The enumerator object returned by this method implements the **IEnumSTATDATA** interface, one of the standard enumerator interfaces that contain the **Next**, **Reset**, **Clone**, and **Skip** methods. **IEnumSTATDATA** enumerates the data stored in an array of **STATDATA** structures.

While this enumeration is in progress, the effect of adding more advisory connections on the subsequent enumeration is undefined.

See Also

IEnumSTATDATA

IDataObject::EnumFormatEtc

Returns an object to enumerate the **FORMATETC** structures that can be used with **IDataObject::GetData** or **IDataObject::SetData**. You use this method when you want to enumerate the **FORMATETC** structures for a data object.

```
HRESULT EnumFormatEtc(
  DWORD dwDirection,                   //Specifies a value from the enumeration DATADIR
  IEnumFORMATETC ** ppenumFormatetc    //Location for IEnumFORMATETC
 );
```

Parameters

dwDirection
[in]Specifies a value from the enumeration **DATADIR**.

```
typedef enum tagDATADIR
{
    DATADIR_GET = 1,
    DATADIR_SET = 2,
} DATADIR;
```

The value DATADIR_GET enumerates the formats that can be passed in **IDataObject::GetData**. The value DATADIR_SET enumerates those formats that can be passed in **IDataObject::SetData**.

ppenumFormatetc
[out]Points to the location where the new enumerator object should be returned.

Return Values

S_OK
Enumerator object was successfully returned.

E_INVALIDARG
One or more arguments are invalid.

E_OUTOFMEMORY
Enumerator object was not returned because the system ran out of memory.

E_NOTIMPL
The direction specified by *dwDirection* is not supported.

OLE_S_USEREG
Requests that OLE enumerate the formats from the registry.

Comments

This method creates and returns an enumerator object that can be used to determine all of the ways the data object can describe data in a **FORMATETC** structure. The enumerator object implements **IEnumFORMATETC**, one of the standard enumerator interfaces. The caller is responsible for calling **IEnumFormatEtc::Release** when it is finished with the enumeration.

If a **FORMATETC** structure is enumerated by **IDataObject::EnumFormatEtc**, there is no guarantee that the format is supported because the formats can change over time. Accordingly, applications should treat the enumeration as a hint of the format types that can be passed.

IDataObject::EnumFormatEtc is called when one of the following actions occurs:

- An application calls **OleSetClipboard**. OLE must determine what data to place on the Clipboard and whether it is necessary to put OLE 1 compatibility formats on the Clipboard.
- Data is being pasted from the Clipboard or dropped. An application uses the first acceptable format.
- The Paste Special dialog box is displayed. The target application builds the list of formats from the **FORMATETC** entries.

Formats can be registered statically in the registration database or dynamically during object initialization. If an object has an unchanging list of formats and these formats are registered in the registration database, it can ask OLE to perform the enumeration using the registration database by calling the helper function **OleRegEnumFormatEtc** or by returning OLE_S_USEREG.

OLE_S_USEREG instructs the default object handler to call **OleRegEnumFormatEtc**. Object applications that are implemented as DLL object applications cannot return OLE_S_USEREG. They must call **OleRegEnumFormatEtc** directly.

Private formats can be enumerated for OLE 1 objects, if they are registered with the RequestDataFormats or SetDataFormats keys in the registration database. Also, private formats can be enumerated for OLE 2 objects, if they are registered with the GetDataFormats or SetDataFormats keys.

For OLE 1 objects whose servers do not have RequestDataFormats or SetDataFormats information registered in the registration database, a call to **EnumFormatEtc** with DATADIR_GET only enumerates the Native and Metafile formats, regardless of whether they support these formats or others. Calling **EnumFormatEtc** with DATADIR_SET on such objects only enumerates Native, regardless of whether the object supports being set with other formats.

The **FORMATETC** structure returned by the enumeration usually indicates a NULL target device (*ptd*). This is appropriate because, unlike the other members of **FORMATETC**, the target device does not participate in the object's decision as to whether it can accept or provide the data in either a **SetData** or **GetData** call.

The TYMED member of **FORMATETC** often indicates that more than one kind of storage medium is acceptable. You should always mask and test for it by using a Boolean OR operator.

See Also

FORMATETC, **OleRegEnumFormatEtc**, **IEnumFormatEtc**, **IDataObject::SetData**, **IDataObject::GetData**

IDataObject::GetCanonicalFormatEtc

Provides a potentially different but logically equivalent **FORMATETC** structure. You use this method when you have different **FORMATETC** structures that may return the same data.

```
HRESULT GetCanonicalFormatEtc(
   FORMATETC * pFormatetcIn,        //Points to the FORMATETC structure that the caller would like to use
   FORMATETC * pFormatetcOut        //Points to the canonical equivalent FORMATETC structure
  );
```

Parameters

pFormatetcIn
: [in]Points to the format, media, and target device that the caller would like to use in a subsequent call to retrieve data, for example, **IDataObject::GetData**. The TYMED member is not significant and should be ignored.

pFormatetcOut
: [out]Points to a **FORMATETC** structure that is canonically equivalent to *pFormatetcIn*. The caller allocates this structure and this method fills in the data. In a subsequent call to retrieve data, for example, **IDataObject::GetData**, the caller uses the *pFormatetcOut* value. This value is NULL if the method returns DATA_S_SAMEFORMATETC. The TYMED member is not significant and should be ignored.

Return Values

S_OK
: The returned **FORMATETC** structure is different from the one that was passed.

E_INVALIDARG
: One or more arguments are invalid.

E_UNEXPECTED
: An unexpected error occurred.

E_OUTOFMEMORY
: Structure was not created because the system ran out of memory.

DATA_S_SAMEFORMATETC
: The **FORMATETC** structures are the same and NULL is returned in *pFormatetcOut*.

DV_E_LINDEX
: Invalid value for *lindex*; currently, only -1 is supported.

DV_E_FORMATETC
: Invalid value for *pFormatetc*.

OLE_E_NOTRUNNING
: Object application is not running.

Comments

A data object can return exactly the same data even though you have specified different **FORMATETC** structures in subsequent requests to retrieve data. For example, the same data can be returned for different target devices if the data is not sensitive to the target devices.

The **IDataObject::GetCanonicalFormatEtc** method provides the canonical equivalent of a specified **FORMATETC** structure. By calling this function prior to retrieving the data object, you can use the canonical equivalent **FORMATETC** in your subsequent call to **IDataObject::GetData**. You can then eliminate unnecessary calls to retrieve data if two different **FORMATETC** structures have the same canonical equivalent.

If the data object never provides device-specific renderings, the implementation of **IDataObject::GetCanonicalFormatEtc** simply copies the input **FORMATETC** to the output

FORMATETC, stores a NULL in the *ptd* field of the output **FORMATETC**, and returns DATA_S_SAMEFORMATETC.

See Also

IDataObject::GetData, **FORMATETC**

IDataObject::GetData

Renders the data described in the specified **FORMATETC** structure and transfers it through the specified **STGMEDIUM** structure. The caller then assumes responsibility for releasing the **STGMEDIUM** structure. You use this method when you want to transfer data from a source object to a target.

```
HRESULT GetData(
  FORMATETC * pFormatetc,     //Points to the data and formats to be transferred
  STGMEDIUM * pmedium         //Points to the storage medium in which the returned data is transferred
 );
```

Parameters

pFormatetc
[in]Points to the format, media, and target device to use when passing the data. If you specify more than one medium by using the Boolean OR operator on the value specified in **TYMED**, **IDataObject::GetData** chooses the medium that best represents the data, does the allocation, and decides what is responsible for the release. The decision is indicated through the value returned in the **IUnknown** returned as part of *pmedium*.

pmedium
[out]Points to the storage medium containing the returned data. The medium is allocated and filled in by **IDataObject::GetData**. **STGMEDIUM** includes a value for the IUnknown to call for releasing the medium.

Return Values

S_OK
Data was successfully retrieved and placed in the storage medium provided.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

E_OUTOFMEMORY
The system did not retrieve data because it ran out of memory.

DV_E_LINDEX
Invalid value for *lindex*; currently, only -1 is supported.

DV_E_FORMATETC
Invalid value for *pFormatetc*.

DV_E_TYMED
Invalid *tymed* value.

DV_E_DVASPECT
Invalid *dwAspect* value.

OLE_E_NOTRUNNING
Object application is not running.

STG_E_MEDIUMFULL
An error occurred when allocating the medium.

Comments

The **IDataObject::GetData** method must check all fields in the **FORMATETC** structure. It is important that **IDataObject::GetData** renders the requested aspect and, if possible, uses the requested medium. If the data object cannot comply with the information specified in the **FORMATETC**, DV_E_FORMATETC is returned. If an attempt to allocate the medium fails, STG_E_MEDIUMFULL is returned. It is important to fill in all of the fields in the **STGMEDIUM**.

Although the caller can specify more than one medium for returning the data, only one medium can actually be returned. The **IDataObject::GetData** method chooses which medium to return. If this initial transfer fails, this method can try one of the other media specified before returning an error.

Data transferred across a stream extends from position zero of the stream pointer through to the position immediately before the current stream pointer (that is, the stream pointer position upon exit).

See Also

IDataObject::GetDataHere, **IDataObject::SetData**, **FORMATETC**, **STGMEDIUM**

IDataObject::GetDataHere

Renders the data described in a **FORMATETC** structure and transfers it through the **STGMEDIUM** structure allocated by the caller. You use this method when you want to transfer data from a source object to a target.

```
HRESULT GetDataHere(
    FORMATETC * pFormatetc,      //Points to the data and formats to be transferred
    STGMEDIUM * pmedium          //Points to the storage medium in which the returned data is transferred
  );
```

Parameters

pFormatetc

[in]Points to the format, media, and target device to use when passing the data. Only one medium can be specified in **TYMED**.

The only values valid for TYMED are TYMED_STORAGE, TYMED_STREAM, TYMED_HGLOBAL, or TYMED_FILE.

pmedium

[out]Points to the storage medium containing the the returned data. The medium is allocated by the caller and filled in by **IDataObject::GetDataHere**. Since the caller allocates the medium, the caller is responsible for freeing the medium. The caller will always get a return value of NULL for the **IUnknown** specified for releasing the medium.

Return Values

S_OK
Data was successfully retrieved and placed in the storage medium provided.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

E_OUTOFMEMORY
The system didn't retrieve data because it ran out of memory.

DV_E_LINDEX
Invalid value for *lindex*; currently, only -1 is supported.

DV_E_FORMATETC
Invalid value for *pFormatetc*.

DV_E_TYMED
Invalid *tymed* value.

DV_E_DVASPECT
Invalid *dwAspect* value.

OLE_E_NOTRUNNING
Object application is not running.

STG_E_MEDIUMFULL
The medium provided by the caller is not large enough to contain the data.

Comments

The **IDataObject::GetDataHere** method is similar to **IDataObject::GetData** except that the caller allocates the medium specified in *pmedium* and is responsible for freeing it. This method copies the data into the actual medium provided by the caller. For example, if the medium is TYMED_HGLOBAL, this method cannot resize the medium or allocate a new hGlobal.

When the transfer medium is a stream, OLE makes assumptions about where the data is being returned and the position of the stream's seek pointer. In a **GetData** call, the data returned is from stream position zero through just before the current seek pointer of the stream (that is, the position on exit). For **GetDataHere**, the data returned is from the stream position on entry through just before the position on exit. Note that some media are not appropriate when you are using **GetDataHere**, including GDI types such as metafiles. The **GetDataHere** method cannot put data into a caller-provided metafile.

See Also

IDataObject::GetData, **FORMATETC**, **STGMEDIUM**

IDataObject::QueryGetData

Determines whether the data object is capable of rendering the data described in the **FORMATETC** structure. Objects attempting a paste or drop operation can call this method before beginning the process. In some cases, the subsequent paste or drop operation may still fail.

```
HRESULT QueryGetData(
  FORMATETC * pFormatetc        //Points to the data and formats to be used for the query
 );
```

Parameter

pFormatetc
[in]Points to the format, media, and target device to use for the query.

Return Values

S_OK
Indicates a subsequent call to **IDataObject::GetData** would probably be successful.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

E_OUTOFMEMORY
The process didn't execute because the system ran out of memory.

DV_E_LINDEX
Invalid value for *lindex*; currently, only -1 is supported.

DV_E_FORMATETC
Invalid value for *pFormatetc*.

DV_E_TYMED
Invalid *tymed* value.

DV_E_DVASPECT
Invalid *dwAspect* value.

OLE_E_NOTRUNNING
Object application is not running.

See Also

IDataObject::GetData, **FORMATETC**

IDataObject::SetData

Transfers data to the source data object.

```
HRESULT SetData(
   FORMATETC * pFormatetc,       //Points to the data and formats used to interpret the data
   STGMEDIUM * pmedium,          //Points to the storage medium in which the data is transferred
   BOOL fRelease                 //Indicates what owns the storage medium after the call is completed
  );
```

Parameters

pFormatetc
[in]Points to the format used by the data object when interpreting the data contained in the storage medium.

pmedium
[in]Points to the storage medium.

fRelease
[in]Indicates what has ownership of the storage medium after the call is complete. If *fRelease* is TRUE, **IDataObject::SetData** takes ownership, freeing the medium after it has been used by calling **ReleaseStgMedium**. If *fRelease* is FALSE, the caller retains ownership and the callee uses the storage medium for the duration of the call only.

Return Values

S_OK
Data was successfully transferred.

E_FAIL
The data was not transferred.

E_NOTIMPL
This method is not implemented for the data object.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

E_OUTOFMEMORY
The data wasn't transferred because the system ran out of memory.

DV_E_LINDEX
Invalid value for *lindex*; currently, only -1 is supported.

DV_E_FORMATETC
Invalid value for *pFormatetc*.

DV_E_TYMED
Invalid *tymed* value.

DV_E_DVASPECT
Invalid *dwAspect* value.

OLE_E_NOTRUNNING
Object application is not running.

Comments

The callee does not take ownership of the data until it has successfully used it and no error code is returned.

The type of medium (TYMED) in the *pformatetc* and *pmedium* parameters must be equal. For example, one cannot be an hGlobal and the other a stream.

See Also

IDataObject::GetData, **FORMATETC**, **STGMEDIUM**

IDropSource

The **IDropSource** interface is one of the interfaces you implement to provide drag-and-drop operations in your application. It contains methods used in any application used as a data source in a drag-and-drop operation. The data source application in a drag-and-drop operation is responsible for:

- Determining the data being dragged based on the user's selection.
- Initiating the drag-and-drop operation based on the user's mouse actions.
- Generating some of the visual feedback during the drag-and-drop operation, such as setting the cursor and highlighting the data selected for the drag-and-drop operation.
- Canceling or completing the drag-and-drop operation based on the user's mouse actions.
- Performing any action on the original data caused by the drop operation, such as deleting the data on a drag move.

The **IDropSource** interface contains the methods for generating visual feedback to the end user and for canceling or completing the drag-and-drop operation. The **DoDragDrop** function, **RegisterDragDrop** function, and **RevokeDragDrop** function are also used in drag-and-drop operations.

When to Implement

Implement the **IDropSource** interface if you are developing a container or server application that can act as a data source for a drag-and-drop operation. The **IDropSource** interface is only required during the drag-and-drop operation.

If you implement the **IDropSource** interface, you must also implement the **IDataObject** interface to represent the data being transferred.

The **IDataObject** interface can be the same as the data object offered to the clipboard. Once you have implemented clipboard operations in your application, you can add drag-and-drop operations with only a little extra work.

When to Use

Your **IDropSource** methods are called by the **DoDragDrop** function. Your application calls the **DoDragDrop** function when it detects that the user has initiated a drag-and-drop operation. Then, **DoDragDrop** calls your **IDropSource** methods during the drag-and-drop operation.

For example, **DoDragDrop** calls **IDropSource::GiveFeedback** when you need to change the cursor shape or when you need to provide some other visual feedback. **DoDragDrop** calls **IDropSource::QueryContinueDrag** when there is a change in the mouse button state to determine if the drag-and-drop operation canceled or completed.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IDropSource Methods	**Description**
QueryContinueDrag	Determines whether a drag-and-drop operation should continue.
GiveFeedback	Gives visual feedback to an end user during a drag-and-drop operation.

See Also

DoDragDrop, **IDataObject**, **IDropTarget**

IDropSource::GiveFeedback

Enables a source application to give visual feedback to its end user during a drag-and-drop operation by providing the DoDragDrop function with an enumeration value specifying the visual effect.

```
HRESULT GiveFeedback(
    DWORD dwEffect          //Effect of a drop operation
  );
```

Parameter

dwEffect
[in] Specifies the DROPEFFECT value returned by the most recent call to IDropTarget::DragEnter, IDropTarget::DragOver, or IDropTarget::DragLeave. For a list of values, see the **DROPEFFECT** enumeration.

Return Values

S_OK
The function completed its task successfully using the cursor set by the source application.

DRAGDROP_S_USEDEFAULTCURSORS
The function completed successfully using the OLE-provided, default cursor.

E_INVALIDARG
One or more arguments are invalid.

E_OUTOFMEMORY
Out of memory.

E_UNEXPECTED
An unexpected error occurred.

Comments

When your application detects that the user has started a drag-and-drop operation, you call the DoDragDrop function. **DoDragDrop** enters a loop, calling IDropTarget::DragEnter when the mouse first enters a drop target window, IDropTarget::DragOver when the mouse changes its position within the target window, and IDropTarget::DragLeave when the mouse leaves the target window.

For every call to either **IDropTarget::DragEnter** or **IDropTarget::DragOver**, **DoDragDrop** calls **IDropSource::GiveFeedback**, passing it the DROPEFFECT value returned from the drop target call. **IDropSource::GiveFeedback** is responsible for changing the cursor shape or for changing the highlighted source based on the value of the *dwEffect* parameter.

DoDragDrop calls **IDropTarget::DragLeave** when the mouse has left the target window. Then, **DoDragDrop** calls **IDropSource::GiveFeedback** and passes the DROPEFFECT_NONE value in the *dwEffect* parameter.

The *dwEffect* parameter can include DROPEFFECT_SCROLL, indicating the source should put up the drag-scrolling variation of the appropriate pointer.

OLE defines a recommended set of cursor shapes that your application should use. See the User Interface Guidelines for more information.

Notes to Implementors

This function is called frequently during the DoDragDrop loop so you can gain performance advantages if you optimize your implementation as much as possible.

See Also

DoDragDrop, IDropTarget

IDropSource::QueryContinueDrag

Determines whether a drag-and-drop operation should be continued, canceled, or completed. You do not call this method directly. The OLE **DoDragDrop** function calls this method during a drag-and-drop operation.

```
HRESULT QueryContinueDrag(
  BOOL fEscapePressed,        //Status of escape key since previous call
  DWORD grfKeyState           //Current state of keyboard modifier keys
 );
```

Parameters

fEscapePressed

[in] Specifies whether the Esc key has been pressed since the previous call to **IDropSource::QueryContinueDrag** or to **DoDragDrop** if this is the first call to **QueryContinueDrag**. A TRUE value indicates the end user has pressed the escape key; a FALSE value indicates it has not been pressed.

grfKeyState

[in] Identifies the current state of the keyboard modifier keys on the keyboard. Valid values can be a combination of any of the flags MK_CONTROL, MK_SHIFT, MK_ALT, MK_BUTTON, MK_LBUTTON, MK_MBUTTON, and MK_RBUTTON.

Return Values

S_OK

The drag operation should continue. This result occurs if no errors are detected, the mouse button starting the drag-and-rop operation has not been released, and the Esc key has not been detected.

DRAGDROP_S_DROP

The drop operation should occur completing the drag operation. This result occurs if *grfKeyState* indicates that the key that started the drag-and-drop operation has been released.

DRAGDROP_S_CANCEL

The drag operation should be canceled with no drop operation occurring. This result occurs if *fEscapePressed* is true indicating the Esc key has been pressed.

E_OUTOFMEMORY

Out of memory.

E_UNEXPECTED

An unexpected error occurred.

Comments

The **DoDragDrop** function calls **IDropSource::QueryContinueDrag** whenever it detects a change in the keyboard or mouse button state during a drag-and-drop operation. **IDropSource::QueryContinueDrag** determines whether the drag-and-drop operation should be continued, canceled, or completed based on the contents of the parameters *grfKeyState* and *fEscapePressed*.

See Also

DoDragDrop

IDropTarget

The **IDropTarget** interface is one of the interfaces you implement to provide drag-and-drop operations in your application. It contains methods used in any application that can be a target for data during a drag-and-drop operation. A drop-target application is responsible for:

- Determining the effect of the drop on the target application.
- Incorporating any valid dropped data when the drop occurs.
- Communicating target feedback to the source so the source application can provide appropriate visual feedback such as setting the cursor.
- Implementing drag scrolling.
- Registering and revoking its application windows as drop targets.

The **IDropTarget** interface contains methods that provide all these responsibilities except registering and revoking the application window as a drop target.

When to Implement

Implement the **IDropTarget** interface if you are developing a container or server application that can act as a target for a drag-and-drop operation. The **IDropTarget** interface is associated with your application windows and is instantiated whenever the window is registered as a target for dropped data. You must call the **RegisterDragDrop** function and the **RevokeDragDrop** function to register and remove your application's windows as drop targets.

When to Use

The **DoDragDrop** function calls your **IDropTarget** methods within a loop during the drag-and-drop operation.

For example, **DoDragDrop** calls **IDropTarget::DragEnter** when it detects the mouse has moved over a window that is registered as a drag target. Once the mouse has entered a drag-target window, **DoDragDrop** calls **IDropTarget::DragOver** as the mouse moves through the window and calls **IDropTarget::DragLeave** if the mouse leaves the target window or if the user cancels or completes the drag-and-drop operation. **DoDragDrop** calls **IDropTarget::Drop** when the drop finally occurs if it occurs at all.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IDropTarget Methods	**Description**
DragEnter	Determines whether a drop can be accepted and its effect if it is accepted.
DragOver	Provides target feedback to the user through the **DoDragDrop** function.
DragLeave	Causes the drop target to suspend its feedback actions.
Drop	Drops the data into the target window.

See Also

DoDragDrop, **IDropSource**, **RegisterDragDrop**, **RevokeDragDrop**

IDropTarget::DragEnter

Indicates whether or not a drop can be accepted and the drop's effect if accepted.

```
HRESULT DragEnter(
  IDataObject * pDataObject,    //IDataObject interface for the source data
  DWORD grfKeyState,            //Current state of keyboard modifier keys
  POINTL pt,                    //Current cursor coordinates
  DWORD * pdwEffect             //Effect of the drag-and-drop operation
 );
```

Parameters

pDataObject

[in] Points to the data object being transferred in the drag-and-drop operation. The data is represented in the **IDataObject** interface. If the drop occurs, this data object will be incorporated into the target.

grfKeyState

[in] Identifies the current state of the keyboard modifier keys on the keyboard. Valid values can be a combination of any of the flags MK_CONTROL, MK_SHIFT, MK_ALT, MK_BUTTON, MK_LBUTTON, MK_MBUTTON, and MK_RBUTTON.

pt

[in] Points to the current cursor coordinates in the coordinate space of the drop target window.

pdwEffect

[in, out] Specifies the current effect flag. Valid values are from the **DROPEFFECT** enumeration.

Return Values

S_OK

The function completed its task successfully.

E_OUTOFMEMORY

Out of memory.

E_INVALIDARG

One or more arguments are invalid.

E_UNEXPECTED

An unexpected error occurred.

Comments

The **DoDragDrop** function calls this method the first time the user drags the mouse into the registered window of a drop target.

In the **IDropTarget::DragEnter** method, you must determine the effect of dropping the data on the target by examining the formats and medium specified for the data object along with the state of the modifier keys. The following modifier keys affect the result of the drop:

Key Combination	User-Visible Feedback	Drop Effect
CTRL + SHIFT	=	DROPEFFECT_LINK
CTRL	+	DROPEFFECT_COPY
No keys or SHIFT	None	DROPEFFECT_MOVE

You communicate the effect of the drop back to the source through **DoDragDrop** in the *pdwEffect* parameter. The **DoDragDrop** function then calls **IDropSource::GiveFeedback** so that the source application can display the appropriate visual feedback to the user.

On entry to **IDropTarget::DragEnter**, the *pdwEffect* parameter is set to the effects given to the **DoDragDrop** function. The **IDropTarget::DragEnter** method must choose one of these effects or disable the drop.

Upon return, *pdwEffect* is set to either DROPEFFECT_COPY to copy the dragged data to the target, DROPEFFECT_LINK to create a link to the source data, or DROPEFFECT_MOVE to allow the dragged data to be permanently moved from the source application to the target.

You may also wish to provide appropriate visual feedback in the target window.

See Also

DoDragDrop, **IDropSource**, **IDropTarget**, **RegisterDragDrop**, **RevokeDragDrop**

IDropTarget::DragLeave

Removes target feedback and releases the data object.

HRESULT DragLeave(*void***);**

Return Values

S_OK
The function completed its task successfully.

E_OUTOFMEMORY
Out of memory.

Comments

The **DoDragDrop** function calls this method in either of the following cases:

- When the user drags the cursor out of a given target window.
- When the user cancels the current drag-and-drop operation.

In the **IDropTarget::DragLeave** method, you must remove any target feedback that is currently displayed. You must also release any references you hold to the data transfer object.

See Also

DoDragDrop, **IDropSource**, **IDropTarget**, **RegisterDragDrop**, **RevokeDragDrop**

IDropTarget::DragOver

Provides target feedback to the user and communicates the drop's effect to the **DoDragDrop** function so the effect of the drop can be communicated through the **DoDragDrop** function back to the source.

```
HRESULT DragOver(
  DWORD grfKeyState,       //Current state of keyboard modifier keys
  POINTL pt,               //Current cursor coordinates
  DWORD * pdwEffect        //Effect of the drag-and-drop operation
 );
```

Parameters

grfKeyState
[in] Identifies the current state of the keyboard modifier keys on the keyboard. Valid values can be a combination of any of the flags MK_CONTROL, MK_SHIFT, MK_ALT, MK_BUTTON, MK_LBUTTON, MK_MBUTTON, and MK_RBUTTON.

pt
[in] Points to the current cursor coordinates in the coordinate space of the drop target window.

pdwEffect
[in, out] Specifies the current effect flag. Valid values are from the enumeration DROPEFFECT.

Return Values

S_OK
The function completed its task successfully.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

Comments

The **DoDragDrop** function calls this method each time the user moves the mouse across a given target window. **DoDragDrop** exits the loop if the drag-and-drop operation is canceled, if the user drags the mouse out of the target window, or if the drop is completed.

In **IDropTarget::DragOver**, you must provide features similar to **IDropTarget::DragEnter**. You must determine the effect of dropping the data on the target by examining the formats and medium specified for the data object along with the state of the modifier keys. The mouse position may also play a role in determining the effect of a drop. The following modifier keys affect the result of the drop:

Key Combination	User-Visible Feedback	Drop Effect
CTRL + SHIFT	=	DROPEFFECT_LINK
CTRL	+	DROPEFFECT_COPY
No keys or SHIFT	None	DROPEFFECT_MOVE

You communicate the effect of the drop back to the source through **DoDragDrop** in *pdwEffect*. The **DoDragDrop** function then calls **IDropSource::GiveFeedback** so the source application can display the appropriate visual feedback to the user.

On entry to **IDropTarget::DragOver**, the *pdwEffect* parameter is set to the effects given to the **DoDragDrop** function. The **IDropTarget::DragOver** method must choose one of these effects or disable the drop.

Upon return, *pdwEffect* is set to either DROPEFFECT_COPY to copy the dragged data to the target, DROPEFFECT_LINK to create a link to the source data, or DROPEFFECT_MOVE to allow the dragged data to be permanently moved from the source application to the target.

You may also wish to provide appropriate visual feedback in the target window. There may be some target feedback already displayed from a previous call to **IDropTarget::DragOver** or from the initial **IDropTarget::DragEnter**. If this feedback is no longer appropriate, you should remove it.

For efficiency reasons, a data object is not passed in **IDropTarget::DragOver**. The data object passed in the most recent call to **IDropTarget::DragEnter** is available and can be used.

Notes to Implementors

This function is called frequently during the **DoDragDrop** loop so it makes sense to optimize your implementation of the **DragOver** method as much as possible.

See Also

DoDragDrop, **IDropSource**, **IDropTarget**, **RegisterDragDrop**, **RevokeDragDrop**

IDropTarget::Drop

Incorporates the source data into the target window, removes target feedback, and releases the data object.

```
HRESULT Drop(
  IDataObject * pDataObject,     //IDataObject interface for the source data
  DWORD grfKeyState,             //Current state of keyboard modifier keys
  POINTL pt,                     //Current cursor coordinates
  DWORD * pdwEffect              //Effect of the drag-and-drop operation
 );
```

Parameters

pDataObject
[in] Points to the data object being transferred in the drag-and-drop operation. The data is represented in the **IDataObject** interface.

grfKeyState
[in] Identifies the current state of the keyboard modifier keys on the keyboard. Valid values can be a combination of any of the flags MK_CONTROL, MK_SHIFT, MK_ALT, MK_BUTTON, MK_LBUTTON, MK_MBUTTON, and MK_RBUTTON.

pt
[in] Points to the current cursor coordinates in the coordinate space of the drop target window.

pdwEffect
[in, out] Specifies the current effect flag. Valid values are from the enumeration **DROPEFFECT**.

Return Values

S_OK
The function completed its tasks successfully.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

Comments

The **DoDragDrop** function calls this method when the user completes the drag-and-drop operation.

In **IDropTarget::Drop**, you must incorporate the data object into the target. You can use the formats available in **IDataObject** along with the current state of the modifier keys to determine how the data is to be incorporated. This can be, for example, linking the data or embedding it.

In addition to incorporating the data, you must also clean up as you do in the **IDropTarget::DragLeave** method:

- Remove any target feedback that is currently displayed.
- Release any references to the data object.

You also pass the effect of this operation back to the source application through **DoDragDrop, **so the source application can clean up after the drag-and-drop operation is complete:

- Remove any source feedback that is currently displaying.
- Make any necessary changes to the data, for example, removing the data if the operation was a move.

See Also

DoDragDrop, IDropSource, IDropTarget, RegisterDragDrop, RevokeDragDrop

IEnum*XXXX*

To allow you to count the number of items of a given type an object maintains, OLE provides a set of enumeration interfaces, one for each type of item.

To use these interfaces, the client asks an object that maintains a collection of items to create an enumerator object. The interface on the enumeration object is one of the enumeration interfaces, all of which have a name of the form **IEnum***Item_name*. Enumeration interfaces are specific to the type of item. All have the same set of methods, and are used in the same way. For example, by repeatedly calling the **Next** method, the client gets successive pointers to each item in the collection.

The following table lists the set of enumeration interfaces that OLE defines, and the items enumerated.

Enumeration Interface Name	**Item Enumerated**
IEnumFORMATETC	An array of **FORMATETC** structures
IEnumMoniker	The components of a moniker, or the monikers in a table.
IEnumOLEVERB	The different verbs available for an object, in order of ascending verb number.
IEnumSTATDATA	An array of **STATDATA** structures which contain advisory connection information for a data object.
IEnumSTATSTG	An array of **STATSTG** structures,which contain statistical information about a storage, stream, or LockBytes object
IEnumString	Strings
IEnumUnknown	Enumerates **IUnknown** interface pointers
IEnumVARIANT	A collection of variants. It allows clients to enumerate heterogeneous collections of objects and intrinsic types when the clients cannot or do not know the specific type(s) of elements in the collection

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IEnum*XXXX* Methods	**Description**
Next	Retrieves a specified number of items in the enumeration sequence.
Skip	Skips over a specified number of items in the enumeration sequence.
Reset	Resets the enumeration sequence to the beginning.
Clone	Creates another enumerator that

contains the same enumeration state
as the current one.

IEnum*XXXX*::Clone

Creates another enumerator that contains the same enumeration state as the current one. Using this function, a client can record a particular point in the enumeration sequence, and then return to that point at a later time. The new enumerator has the same interface as the original one.

HRESULT Clone(
IEnum<ELT_T> ** *ppenum*
);

Parameter

ppenum
[out] On exit, contains the duplicate enumerator. If the function was unsuccessful, this parameter's value is undefined.

Return Values

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
Value of *ppenum* is invalid.

E_UNEXPECTED
An unexpected error occurred.

IEnum*XXXX*::Next

Retrieves the next *celt* items in the enumeration sequence. If there are fewer than the requested number of elements left in the sequence, it retrieves the remaining elements. The number of elements actually retrieved is returned through *pceltFetched* (unless the caller passed in **NULL** for that parameter).

HRESULT Next(
  ULONG *celt***,**
  ELT_T[ ] *rgelt***,**
  ULONG * *pceltFetched*
 );

Parameters

celt
  [in]The number of elements being requested.

rgelt
  [out]Receives an array of size *celt* (or larger) of the elements to be returned. The type of this parameter depends on the item being enumerated.

pceltFetched
  [in, out]On return, contains the number of elements actually returned in *rgelt*. Caller can pass in **NULL** if *celt* is one.

Return Value

S_OK if the number of elements returned is *celt*; S_FALSE otherwise.

IEnum*XXXX*::Reset

Resets the enumeration sequence to the beginning.

HRESULT Reset(*void***);**

Return Value

S_OK

Comments

There is no guarantee that the same set of objects will be enumerated after the reset, because it depends on the collection being enumerated. It can be too expensive for some collections (such as files in a directory) to guarantee this condition.

IEnum*XXXX*::Skip

Skips over the next specified number of elements in the enumeration sequence.

HRESULT Skip(
  ULONG *celt*
 );

Parameter

celt
  [in]The number of elements that are to be skipped.

Return Value

S_OK if the number of elements skipped is *celt*; otherwise S_FALSE.

IEnumFORMATETC

The **IEnumFORMATETC** interface is used to enumerate an array of **FORMATETC** structures. **IEnumFORMATETC** shares the same methods as all enumerator interfaces: **Next**, **Skip**, **Reset**, and **Clone**.

IEnumFORMATETC is implemented by all data objects to support calls to **IDataObject::EnumFormatEtc**. If the data object supports a different set of **FORMATETC** information depending on whether a call is made to **SetData** or **GetData**, the implementation of **IEnumFORMATETC** must be able to operate on both.

The order of formats enumerated through the **IEnumFORMATETC** object should be the same as the order that the formats would be in when placed on the clipboard. Typically, this order starts with private data formats and ends with presentation formats such as CF_METAFILEPICT.

The prototypes of the member functions are as follows:

HRESULT Next(**ULONG** *celt*, **FORMATETC *** *rgelt*, **ULONG *** *pceltFetched)*

HRESULT Skip(**ULONG** *celt*)

HRESULT Reset(*void*)

HRESULT Clone(**IEnumFORMATETC **** *ppenum*)

See Also

OleRegEnumFormatEtc, **FORMATETC**, **IEnum*XXXX***

IEnumMoniker

The **IEnumMoniker** interface is used to enumerate the components of a moniker or to enumerate the monikers in a table of monikers.

When to Implement

You need to implement **IEnumMoniker** if you are writing a new type of moniker and your monikers have an internal structure that can be enumerated. Your implementation of **IMoniker::Enum** must return an enumerator that implements **IEnumMoniker** and can enumerate your moniker's components. If your moniker has no structure that can be enumerated, your **IMoniker::Enum** method can simply return a NULL pointer.

When to Use

Use the **IEnumMoniker** interface if you need to enumerate the components of a composite moniker, or to enumerate the monikers in a table.

OLE defines two interfaces that return an **IEnumMoniker** interface pointer:

- The **IMoniker::Enum** method returns an **IEnumMoniker** implementation that can enumerate forwards or backwards through the components of the moniker. For a description of how two of the system-supplied types of monikers enumerate their components, see **IMoniker - File Moniker Implementation** and **IMoniker - Generic Composite Moniker Implementation**.
- The **IRunningObjectTable::EnumRunning** method returns an **IEnumMoniker** implementation that can enumerate the monikers registered in a Running Object Table.

The prototypes of the member functions are as follows:

HRESULT Next(**ULONG** *celt*, **IMoniker** ***** *rgelt*, **ULONG *** *pceltFetched)*

HRESULT Skip(**ULONG** *celt*)

HRESULT Reset(*void*)

HRESULT Clone(**IEnumMoniker **** *ppenum*)

See Also

IEnum*XXXX*, IMoniker::Enum, IRunningObjectTable::EnumRunning

IEnumOLEVERB

The **IEnumOLEVERB** interface enumerates the different verbs available for an object in order of ascending verb number. An enumerator that implements the **IEnumOLEVERB** interface is returned by the **IOleObject::EnumVerbs** member function.

When to Implement

You typically do not have to implement this interface. The OLE default handler provides an implementation which returns the entries in the registration database. Because calls to **IOleObject::EnumVerb** are always routed through the default handler; an OLE application can let the default handler do the work by implementing **IOleObject::EnumVerb** as a stub that simply returns **OLE_S_USEREG**. This informs the default handler that it should create the enumerator for you.

When to Use

Call this interface if you need to list the verbs than an OLE object supports.

The prototypes of the member functions are as follows:

HRESULT Next(**ULONG** *celt*, **OLEVERB** ***** *rgelt*, **ULONG *** *pceltFetched)*

HRESULT Skip(**ULONG** *celt*)

HRESULT Reset(*void*)

HRESULT Clone(**IEnumOLEVERB **** *ppenum*)

See Also

OLEVERB, **IEnum*XXXX***

IEnumSTATDATA

The **IEnumSTATDATA** interface is used to enumerate through an array of **STATDATA** structures which contain advisory connection information for a data object. **IEnumSTATDATA** shares the same methods as all enumerator interfaces: **Next**, **Skip**, **Reset**, and **Clone**.

IEnumSTATDATA is implemented to enumerate advisory connections. Containers usually call methods that return a pointer to **IEnumSTATDATA** so the container can instruct an object to release each of its advisory connections prior to closing down. Calls to **IDataObject::EnumDAdvise**, **IDataAdviseHolder::EnumAdvise**, and **IOleCache::EnumCache** methods all return a pointer to **IEnumSTATDATA**.

The prototypes of the member functions are as follows:

HRESULT Next(**ULONG** *celt*, **STATDATA *** *rgelt*, **ULONG *** *pceltFetched)*

HRESULT Skip(**ULONG** *celt*)

HRESULT Reset(*void*)

HRESULT Clone(**IEnumSTATDATA **** *ppenum*)

See Also

STATDATA, **IEnum*XXXX***, **IOleCache::EnumCache**, **IDataObject::EnumDAdvise**, **IDataAdviseHolder::EnumAdvise**, **IOleObject::EnumAdvise**

IEnumSTATSTG

The **IEnumSTATSTG** interface is used to enumerate through an array of **STATSTG** structures which contain statistical information about an open storage, stream, or byte array object. **IEnumSTATSTG** shares the same methods as all enumerator interfaces: **Next**, **Skip**, **Reset**, and **Clone**.

IEnumSTATSTG is implemented to enumerate the elements of a storage object. Containers usually call methods that return a pointer to **IEnumSTATSTG** so the container can manage its storage object and the elements within it. Calls to the **IStorage::EnumElements** method returns a pointer to **IEnumSTATDATA**. The caller allocates an array of **STATSTG** structures and the **IEnumSTATSTG** methods fill in each structure with the statistics about one of the nested elements in the storage object. If present, the *lpszName* member of the **STATSTG** structure requires additional memory allocations through the **IMalloc** interface, and the caller is responsible for freeing this memory, if it is allocated, by calling the **IMalloc::Free** method. If the *lpszName* member is NULL, no memory is allocated, and, therefore, no memory needs to be freed.

The prototypes of the member functions are as follows:

HRESULT Next(**ULONG** *celt*, **STATSTG *** *rgelt*, **ULONG *** *pceltFetched)*

HRESULT Skip(**ULONG** *celt*)

HRESULT Reset(*void*)

HRESULT Clone(**IEnumSTATSTG **** *ppenum*)

See Also

CoGetMalloc, **IEnum*XXXX***, **IStorage::EnumElements**, **STATSTG**

IEnumString

IEnumString is defined to enumerate strings. LPWSTR is the type that indicates a pointer to a zero-terminated string of wide, i.e., Unicode, characters.

The prototypes of the member functions are as follows:

HRESULT Next(**ULONG** *celt*, **LPWSTR *** *rgelt*, **ULONG *** *pceltFetched)*

HRESULT Skip(**ULONG** *celt*)

HRESULT Reset(*void*)

HRESULT Clone(**IEnumString **** *ppenum*)

IEnumUnknown

This enumerator enumerates objects with interface **IUnknown**. This enumerator is used by the **IOleContainer** interface to enumerate the objects in a compound document.

The prototypes of the member functions are as follows:

HRESULT Next(**ULONG** *celt*, **IUnknown **** *reelt*, **ULONG *** *pceltFetched*)

HRESULT Skip(**ULONG** *celt*)

HRESULT Reset(*void*)

HRESULT Clone(**IEnumUnknown *** *ppenum*)

See Also

IOleContainer

IExternalConnection

The **IExternalConnection** interface enables an embedded object to keep track of external locks on it, thereby enabling the safe and orderly shutdown of the object following silent updates. An object that supports links either to itself or to some portion of itself (a range of cells in a spreadsheet, for example) should implement this interface to prevent possible loss of data during shutdown.

Such data loss can occur when an object happens to have unsaved changes at a time when its stub manager's count of strong external references has reached zero. This situation would arise, for example, at the end of a silent update, when the final link client breaks its connection to the object. With the severing of this connection, the stub manager's count of strong external references would reach zero, causing it to release its pointers to the object and initiate shutdown of the object. When the object calls **IOleClientSite::SaveObject,** its container's return call to **IPersistStorage::Save** would fail because the stub manager would no longer have a pointer to the object. Any unsaved changes to the object would be lost.

If the object manages its own count of external locks, rather than relying on the stub manager to do so, it can save its data before the stub manager has a chance to release its pointers. An object can obtain a count of external connections by implementing the **IExternalConnection** interface. The stub manager calls this interface whenever a new strong external reference is added or deleted. The object combines this count with its own tally of strong internal references to maintain an accurate total of all locks.

When to Implement

All embeddable compound-document objects that support links to themselves or portions of themselves should implement **IExternalConnection** to prevent possible data loss during shutdown. In addition, an in-place container should call **OleLockRunning** to hold a strong lock on its embedded objects.

When to Use

An object's stub manager should call **IExternalConnection** whenever an external connection is added or released.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IExternalConnection Methods	Description
AddConnection	Increments count of external locks.
ReleaseConnection	Decrements count of external locks.

IExternalConnection::AddConnection

Increments an object's count of its strong external connections (links).

```
HRESULT AddConnection(
   DWORD exconn,            //Type of external connection
   DWORD dwreserved         //Used by OLE to pass connection information
  );
```

Parameters

exconn
[out] Specifies the type of external connection to the object. The only type of external connection currently supported by this interface is strong, which means that the object must remain alive as long as this external connection exists. Strong external connections are represented by the value EXTCONN_STRONG = 0x0001, which is defined in the enumeration **EXTCONN**.

dwreserved
Passes information about the connection. This parameter is reserved for use by OLE. Its value can be zero, but not necessarily. Therefore, implementations of **AddConnection** should not contain blocks of code whose execution depends on whether a zero value is returned.

Return Value

DWORD value
The number of reference counts on the object; used for debugging purposes only.

Comments

An object created by a EXE object application relies on its stub manager to call **IExternalConnection::AddConnection** whenever a link client activates and therefore creates an external lock on the object. When the link client breaks the connection, the stub manager calls **IExternalConnection::ReleaseConnection** to release the lock.

Since DLL object applications exist in the same process space as their objects, they do not utilize RPC (remote procedure calls) and therefore do not have stub managers to keep track of external connections. Therefore, DLL object applications that support external links to their objects must implement **IExternalConnection** so link clients can directly call the interface to inform them when connections are added or released.

The following is a typical implementation for the **AddConnection** method:

```
DWORD XX::AddConnection(DWORD extconn, DWORD dwReserved)
   {
      return extconn&EXTCONN_STRONG ? ++m_cStrong : 0;
   }
```

See Also

IExternalConnection::ReleaseConnection, **IRunnableObject::LockRunning**, **OleLockRunning**

IExternalConnection::ReleaseConnection

Decrements an object's count of its strong external connections (links).

```
HRESULT ReleaseConnection(
   DWORD extconn,                 //Type of external connection
   DWORD dwreserved,              //Used by OLE to pass connection information
   BOOL fLastReleaseCloses        //Indicates whether connection is last one or not
  );
```

Parameters

extconn
[out] Specifies the type of external connection to the object. The only type of external connection currently supported by this interface is strong, which means that the object must remain alive as long as this external connection exists. Strong external connections are represented by the value EXTCONN_STRONG = 0x0001, which is defined in the enumeration **EXTCONN**.

dwreserved
Passes information about the connection. This parameter is reserved for use by OLE. Its value can be zero, but not necessarily. Therefore, implementations of **ReleaseConnection** should not contain blocks of code whose execution depends on whether a zero value is returned.

fLastReleaseCloses
TRUE specifies that if the connection being released is the last external lock on the object, the object should close. FALSE specifies that the object should remain open until closed by the user or another process.

Return Value

DWORD value
The number of connections to the object; used for debugging purposes only.

Comments

If *fLastReleaseCloses* equals TRUE, calling **ReleaseConnection** causes the object to shut itself down. Calling this method is the only way in which a DLL object, running in the same process space as the container application, will know when to close following a silent update.

See Also

IExternalConnection::AddConnection

ILockBytes

The **ILockBytes** interface provides a byte array object used by the OLE-provided implementation of compound files. Its methods manipulate the byte array object that provides the physical storage for the root storage object of a compound file. The byte array for an **ILockBytes** implementation is backed by some physical storage, such as a disk file, global memory, or a database; and the **ILockBytes** implementation hides the details of accessing the physical storage.

When to Implement

OLE provides an implementation of the **ILockBytes** interface for its compound files. The OLE-provided **ILockBytes** interface presents a byte array stored in a physical disk file.

Most applications will never implement the **ILockBytes** interface. Instead, applications will typically call the **StgCreateDocfile** function to create a root storage object (in this case, an **IStorage** instance backed by a specified disk file); and this function internally creates an instance of the OLE-provided, file-based **ILockBytes** implementation.

OLE also provides a memory-based **ILockBytes** implementation (see **CreateILockBytesOnHGlobal**).

It is possible for an application to provide its own **ILockBytes** implementation. For example, a database application might want to implement an **ILockBytes** backed by the storage of its relational tables. However, it is strongly recommended that you use the OLE-provided implementations. For a discussion of the advantages of using the OLE implementations rather than creating your own, see the **StgCreateDocfileOnILockBytes** API function.

If you choose to implement your own **ILockBytes** interface, you should consider providing custom marshaling by implementing the **IMarshal** interface as part of your byte array object. The reason for this is that when the OLE-provided implementations of **IStorage** and **IStream** are marshaled to another process, their **ILockBytes** interface pointers are also marshaled to the other process. The default marshaling mechanism creates a proxy **ILockBytes** that transmits method calls back to the original LockBytes object. Custom marshaling can improve efficiency by creating a remote LockBytes object that can access the byte array directly.

When to Use

The **ILockBytes** methods are called by the OLE-provided implementations of **IStorage** and **IStream**, so application programmers do not need to call **ILockBytes** methods. If you write your own **ILockBytes** implementation, you can use the **StgCreateDocfileOnILockBytes** function to create an **IStorage** backed by your **ILockBytes**.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

ILockBytes Methods	**Description**
ReadAt	Reads a specified number of bytes starting at a specified offset from the beginning of the byte array.
WriteAt	Writes a specified number of bytes to a specified location in the byte array.
Flush	Ensures that any internal buffers maintained by the **ILockBytes**

	implementation are written out to the backing storage.
SetSize	Changes the size of the byte array.
LockRegion	Restricts access to a specified range of bytes in the byte array.
UnlockRegion	Removes the access restriction on a range of bytes previously restricted with **ILockBytes::LockRegion**.
Stat	Retrieves a **STATSTG** structure for this byte array object.

ILockBytes::Flush

Ensures that any internal buffers maintained by the **ILockBytes** implementation are written out to the backing storage.

HRESULT Flush(*void***);**

Return Values

S_OK
The flush operation was successful.

STG_E_ACCESSDENIED
The caller does not have permission to access the byte array.

STG_E_MEDIUMFULL
The flush operation is not completed because there is no space left on the storage device.

E_FAIL
General failure writing data.

STG_E_TOOMANYFILESOPEN
Under certain circumstances, **Flush** does a dump-and-close to flush. This can lead to STG_E_TOOMANYFILESOPEN if there are no file handles available.

STG_E_INVALIDHANDLE
An invalid floppy change has been made.

Comments

This method flushes internal buffers to the underlying storage device.

The OLE-provided implementation of compound files uses this method during a transacted commit operation to provide a two-phase commit process that protects against loss of data.

See Also

IStorage::Commit

ILockBytes::LockRegion

Restricts access to a specified range of bytes in the byte array.

```
HRESULT LockRegion(
   ULARGE_INTEGER libOffset,      //Specifies the byte offset for the beginning of the range
   ULARGE_INTEGER cb,             //Specifies the length of the range in bytes
   DWORD dwLockType               //Specifies the type of restriction on accessing the specified
                                  range
  );
```

Parameters

libOffset
[in]Specifies the byte offset for the beginning of the range.

cb
[in]Specifies, in bytes, the length of the range to be restricted.

dwLockType
[in]Specifies the type of restrictions being requested on accessing the range. This parameter uses one of the values from the **LOCKTYPE** enumeration.

Return Values

S_OK
The specified range of bytes was locked.

STG_E_INVALIDFUNCTION
Locking is not supported at all or the specific type of lock requested is not supported.

STG_E_LOCKVIOLATION
Requested lock is supported, but cannot be granted because of an existing lock.

Comments

This method restricts access to the specified range of bytes. Once a region is locked, attempts by others to perform the restricted access must fail with the STG_E_ACCESSDENIED error.

The byte range can extend past the current end of the current byte array. Locking beyond the end of an array is useful as a method of communication between different instances of the byte array object without changing data that is actually part of the byte array. For example, an implementation of compound files could rely on locking past the current end of the array as a means of access control, using specific locked regions to indicate permissions currently granted.

Three types of locking can be supported: locking to exclude other writers, locking to exclude other readers or writers, and locking that allows only one requestor to obtain a lock on the given range, which is usually an alias for one of the other two lock types. A given byte array might support either of the first two types, or both. The lock type is specified by *dwLockType*, using a value from the **LOCKTYPE** enumeration.

To determine the lock types supported by a particular **ILockBytes** implementation, you can examine the *grfLocksSupported* member of the **STATSTG** structure returned by a call to **ILockBytes::Stat**.

Any region locked with **ILockBytes::LockRegion** must later be explicitly unlocked by calling **ILockBytes::UnlockRegion** with exactly the same values for the *libOffset*, *cb*, and *dwLockTyp*e parameters. The region must be unlocked before the stream is released. Two adjacent regions cannot be locked separately and then unlocked with a single unlock call.

Notes to Callers

Since the type of locking supported is optional and can vary in different implementations of **ILockBytes**, you must provide code to deal with the STG_E_INVALIDFUNCTION error.

Notes to Implementors

Support for this method depends on how the storage objects built on top of the **ILockBytes** implementation is used. If you know that, at any given time, only one storage object can be opened on the storage device that underlies the byte array, then your **ILockBytes** implementation does not need to support locking. However, if multiple simultaneous openings of a storage object are possible, then region locking is needed to coordinate them.

A **LockRegion** implementation can choose to support all, some, or none of the lock types. For unsupported lock types, the implementation should return STG_E_INVALIDFUNCTION.

See Also

ILockBytes::Stat, **ILockBytes::UnlockRegion**, **IStream::LockRegion**, **LOCKTYPE**

ILockBytes::ReadAt

Reads a specified number of bytes starting at a specified offset from the beginning of the byte array object.

```
HRESULT ReadAt(
   ULARGE_INTEGER ulOffset,      //Specifies the starting point for reading data
   void *pv,                     //Points to the buffer into which the data is read
   ULONG cb,                     //Specifies the number of bytes to read
   ULONG *pcbRead                //Pointer to location that contains actual number of bytes read
  );
```

Parameters

ulOffset
[in]Specifies the starting point from the beginning of the byte array for reading data.

pv
[in]Points to the buffer into which the byte array is read.

cb
[in]Specifies the number of bytes of data to attempt to read from the byte array.

pcbRead
[out]Pointer to a location where this method writes the actual number of bytes read from the byte array. You can set this pointer to NULL to indicate that you are not interested in this value. In this case, this method does not provide the actual number of bytes read.

Return Values

S_OK
Indicates that the specified number of bytes were read, or the maximum number of bytes were read up to the end of the byte array.

E_FAIL
Data could not be read from the byte array.

STG_E_ACCESSDENIED
The caller does not have permission to access the byte array.

Comments

This method reads bytes from the byte array object. It reports the number of bytes that were actually read. This value may be less than the number of bytes requested if an error occurs or if the end of the byte array is reached during the read.

It is not an error to read less than the specified number of bytes if the operation encounters the end of the byte array. Note that this is the same end of file behavior as found in MS-DOS FAT file system files.

See Also

ILockBytes::WriteAt

ILockBytes::SetSize

Changes the size of the byte array.

```
HRESULT SetSize(
  ULARGE_INTEGER cb      //Specifies the new size of the byte array in bytes
 );
```

Parameter

cb
[in]Specifies the new size of the byte array as a number of bytes.

Return Values

S_OK
The size of the byte array was successfully changed.

STG_E_ACCESSDENIED
The caller does not have permission to access the byte array.

STG_E_MEDIUMFULL
The byte array size is not changed because there is no space left on the storage device.

Comments

This method changes the size of the byte array. If the *cb* parameter is larger than the current byte array, the byte array is extended to the indicated size by filling the intervening space with bytes of undefined value. This operation is similar to the **ILockBytes::WriteAt** method if the seek pointer is past the current end-of-stream.

If the *cb* parameter is smaller than the current byte array, then the byte array is truncated to the indicated size.

Notes to Callers

Callers cannot rely on STG_E_MEDIUMFULL being returned at the appropriate time because of cache buffering in the operating system or network. However, callers must be able to deal with this return code because some **ILockBytes** implementations might support it.

See Also

ILockBytes::ReadAt, **ILockBytes::WriteAt**

ILockBytes::Stat

Retrieves a **STATSTG** structure containing information for this byte array object.

```
HRESULT Stat(
   STATSTG *pstatstg,       //Location for STATSTG structure
   DWORD grfStatFlag        //Values taken from the STATFLAG enumeration
  );
```

Parameters

pstatstg
[out]Points to a **STATSTG** structure where this method places information about this byte array object. The pointer is NULL if an error occurs.

grfStatFlag
[in]Specifies that this method not return some of the fields in the **STATSTG** structure, thus saving a memory allocation operation. Values are taken from the **STATFLAG** enumeration.

Return Values

S_OK
The **STATSTG** structure was successfully returned at the specified location.

E_OUTOFMEMORY
The **STATSTG** structure was not returned due to a lack of memory for the name field in the structure.

STG_E_ACCESSDENIED
The **STATSTG** structure was not returned because the caller did not have access to the byte array.

STG_E_INSUFFICIENTMEMORY
The **STATSTG** structure was not returned due to a lack of memory.

STG_E_INVALIDFLAG
The value for the *grfStateFlag* parameter is not valid.

STG_E_INVALIDPOINTER
The value for the *pStatStg* parameter is not valid.

Comments

The OLE-provided **IStorage::Stat** implementation calls the **ILockBytes::Stat** method to retrieve information about a root storage object. If there is no reasonable name for the byte array, then the OLE-provided **ILockBytes::Stat** method returns NULL in the *pwcsName* field of the **STATSTG** structure.

See Also

STATFLAG, **STATSTG**

ILockBytes::UnlockRegion

Removes the access restriction on a previously locked range of bytes.

```
HRESULT UnlockRegion(
   ULARGE_INTEGER libOffset,       //Specifies the byte offset for the beginning of the range
   ULARGE_INTEGER cb,              //Specifies the length of the range in bytes
   DWORD dwLockType                //Specifies the access restriction previously placed on the range
  );
```

Parameters

libOffset
[in]Specifies the byte offset for the beginning of the range.

cb
[in]Specifies, in bytes, the length of the range that is restricted.

dwLockType
[in]Specifies the type of access restrictions previously placed on the range. This parameter uses a value from the **LOCKTYPE** enumeration.

Return Values

S_OK
The byte range was unlocked.

STG_E_INVALIDFUNCTION
Locking is not supported at all or the specific type of lock requested is not supported.

STG_E_LOCKVIOLATION
The requested unlock cannot be granted.

Comments

This method unlocks a region previously locked with the **ILockBytes::LockRegion** method. Locked regions must later be explicitly unlocked by calling **ILockBytes::UnlockRegion** with exactly the same values for the *libOffset*, *cb*, and *dwLockType* parameters. Two adjacent regions cannot be locked separately and then unlocked with a single unlock call.

See Also

ILockBytes::LockRegion, **LOCKTYPE**

ILockBytes::WriteAt

Writes the specified number of bytes starting at a specified offset from the beginning of the byte array.

```
HRESULT WriteAt(
   ULARGE_INTEGER ulOffset,      //Specifies the starting point for writing data
   void const *pv,               //Points to the buffer containing the data to be written
   ULONG cb,                     //Specifies the number of bytes to write
   ULONG *pcbWritten             //Pointer to location that contains actual number of bytes written
  );
```

Parameters

ulOffset
[in]Specifies the starting point from the beginning of the byte array for the data to be written.

pv
[in]Points to the buffer containing the data to be written.

cb
[in]Specifies the number of bytes of data to attempt to write into the byte array.

pcbRead
[out]Pointer to a location where this method specifies the actual number of bytes written to the byte array. You can set this pointer to NULL to indicate that you are not interested in this value. In this case, this method does not provide the actual number of bytes written.

Return Values

S_OK
Indicates that the specified number of bytes were written.

E_FAIL
A general failure occurred during the write.

STG_E_ACCESSDENIED
The caller does not have sufficient permissions for writing this byte array.

STG_E_MEDIUMFULL
The write operation was not completed because there is no space left on the storage device. The actual number of bytes written is still returned in *pcbWritten*.

Comments

This method writes the specified data at the specified location in the byte array. The number of bytes actually written is always returned in *pcbWritten*, even if an error is returned. If the byte count is zero bytes, the write operation has no effect.

If *ulOffset* is past the end of the byte array and *cb* is greater than zero, **ILockBytes::WriteAt** increases the size of the byte array. The fill bytes written to the byte array are not initialized to any particular value.

See Also

ILockBytes::ReadAt, **ILockBytes::SetSize**

IMalloc

The OLE implementation of **IMalloc** allocates, frees, and manages memory.

The OLE libraries and object handlers call the methods of the **IMalloc** implementation to manage memory. Those implementing object handlers should call **CoGetMalloc** to get a pointer to the **IMalloc** implementation to handle task memory management needs.

The **IMalloc** methods **Alloc**, **Free**, and **Realloc** are similar to the C library functions **malloc**, **free**, and **realloc**. For debugging, refer to the functions **CoRegisterMallocSpy** and **CoRevokeMallocSpy**.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IMalloc Methods	**Description**
Alloc	Allocates a block of memory.
Free	Frees a previously allocated block of memory.
Realloc	Changes the size of a previously allocated block of memory.
GetSize	Returns the size in bytes of a previously allocated block of memory
DidAlloc	Determines if this instance of IMalloc was used to allocate the specified block of memory.
HeapMinimize	Minimizes the heap by releasing unused memory to the operating system.

See Also

CoGetMalloc, **CoRegisterMallocSpy**, **CoRevokeMallocSpy**

IMalloc::Alloc

Allocates a block of memory.

```
void * Alloc(
   ULONG cb        //The size of the requested memory block in bytes
  );
```

Parameter

cb
[in] Specifies the size (in bytes) of the memory block to be allocated.

Return Values

Pointer to the allocated memory block
Memory block allocated successfully.

NULL
Insufficient memory available.

Comments

The **IMalloc::Alloc** method allocates a memory block in essentially the same way that the C Library **malloc** function does.

The initial contents of the returned memory block are undefined - there is no guarantee that the block has been initialized, so you should initialize it in your code. The allocated block may be larger than *cb* bytes because of the space required for alignment and for maintenance information.

If *cb* is zero, **IMalloc::Alloc** allocates a zero-length item and returns a valid pointer to that item. If there is insufficient memory available, **IMalloc::Alloc** returns NULL.

Note Applications should always check the return value from this method, even when requesting small amounts of memory, because there is no guarantee the memory will be allocated.

See Also

IMalloc::Free, **IMalloc::Realloc**, **CoTaskMemAlloc**

IMalloc::DidAlloc

Determines if this allocator was used to allocate the specified block of memory.

```
int DidAlloc(
   void *pv        //The pointer to the memory block
  );
```

Parameter

pv
   Specifies the pointer to the memory block; can be a NULL pointer, in which case, -1 is returned.

Return Values

1
   The memory block was allocated by this **IMalloc** instance.

0
   The memory block was *not* allocated by this **IMalloc** instance.

-1
   DidAlloc is unable to determine whether or not it allocated the memory block.

Comments

Calling **IMalloc::DidAlloc** is useful if a application is using multiple allocations, and needs to know whether a previously allocated block of memory was allocated by a particular allocation.

See Also

IMalloc::Alloc, **IMalloc::HeapMinimize**, **IMalloc::Realloc**

IMalloc::Free

Frees a previously allocated block of memory.

```
void Free(
    void * pv        //Pointer to the memory block to be freed
  );
```

Parameter

pv
  [in] Points to the memory block to be freed.

Comments

IMalloc:Free frees a block of memory previously allocated through a call to **IMalloc::Alloc** or **IMalloc::Realloc**. The number of bytes freed equals the number of bytes that were allocated. After the call, the memory block pointed to by *pv* is invalid and can no longer be used.

Note The *pv* parameter can be NULL. If so, this method has no effect.

See Also

IMalloc::Alloc, **IMalloc::Realloc**, **CoTaskMemFree**

IMalloc::GetSize

Returns the size (in bytes) of a previously allocated memory block previously allocated with **IMalloc::Alloc** or **IMalloc::Realloc**.

```
ULONG GetSize(
    void *pv        //Points to the memory block fo which the size is requested
  );
```

Parameter

pv
    [in] Points to the memory block for which the size is requested.

Return Value

Either the size of the allocated memory block in bytes or, if *pv* is a NULL pointer, -1.

Comments

Returns the size (in bytes) of a memory block previously allocated with **IMalloc::Alloc** or **IMalloc::Realloc**.The size returned is the actual size of the allocation; it may be greater than the size requested when the allocation was made.

See Also

IMalloc::Alloc, **IMalloc::Realloc**

IMalloc::HeapMinimize

Minimizes the heap as much as possible by releasing unused memory to the operating system, coalescing adjacent free blocks and committing free pages.

void HeapMinimize();

Comments

Calling **IMalloc::HeapMinimize** is useful when an application has been running for some time and the heap may be fragmented.

See Also

IMalloc::Alloc, **IMalloc::Free**, **IMalloc::Realloc**

IMalloc::Realloc

Changes the size of a previously allocated memory block.

void *Realloc(
  void ***pv*,**   //The memory block to be reallocated
  ULONG *cb*   //The size of the memory block in bytes
 );

Parameters

pv
  [in] Points to the memory block to be reallocated. It can be a NULL pointer, as discussed in the following Comments section.

cb
  [in] Specifies the size of the memory block (in bytes) to be reallocated. It can be zero, as discussed in the following remarks.

Return Values

Reallocated memory block
  Memory block successfully reallocated.

NULL
  Insufficient memory or *cb* is zero and *pv* is not NULL.

Comments

The initial contents of the returned memory block are undefined - there is no guarantee that the block has been initialized, so you should initialize it in your code. The allocated block may be larger than *cb* bytes because of the space required for alignment and for maintenance information.

The *pv* argument points to the beginning of the memory block. If *pv* is NULL, **IMalloc::Realloc** allocates a new memory block in the same way as **IMalloc::Alloc**. If *pv* is not NULL, it should be a pointer returned by a prior call to **IMalloc::Alloc**.

The *cb* argument specifies the size (in bytes) of the new block. The contents of the block are unchanged up to the shorter of the new and old sizes, although the new block can be in a different location. Because the new block can be in a different memory location, the pointer returned by **IMalloc::Realloc** is not guaranteed to be the pointer passed through the *pv* argument. If *pv* is not NULL and *cb* is zero, then the memory pointed to by *pv* is freed.

IMalloc::Realloc returns a void pointer to the reallocated (and possibly moved) memory block. The return value is NULL if the size is zero and the buffer argument is not NULL, or if there is not enough memory available to expand the block to the given size. In the first case, the original block is freed; in the second, the original block is unchanged.

The storage space pointed to by the return value is guaranteed to be suitably aligned for storage of any type of object. To get a pointer to a type other than **void**, use a type cast on the return value.

See Also

IMalloc::Alloc, **IMalloc::Free**

IMallocSpy

The **IMallocSpy** interface is a debugging interface that allows application developers to monitor (spy on) memory allocation, detect memory leaks and simulate memory failure.

Caution The IMallocSpy interface is intended to be used **only** to debug application code under development. Do not ship this interface to retail customers of your application, because it causes severe performance degradation and could conflict with user-installed software to produce unpredictable results.

When to Implement

Implement this interface to debug memory allocation during application development.

When to Use

IMallocSpy methods are called only by OLE; applications do not call them. An application provides an implementation of **IMallocSpy** (see the OLE SDK for an example); OLE uses this implementation. When an implementation of **IMallocSpy** is registered with **CoRegisterMallocSpy**, OLE calls the pair of **IMallocSpy** methods around the corresponding **IMalloc** method. The call to the pre-method through the return from the corresponding post-method is guaranteed to be thread-safe.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IMallocSpy Methods	**Description**
PreAlloc	Called before invoking **IMalloc::Alloc**.
PostAlloc	Called after invoking **IMalloc::Alloc**.
PreFree	Called before invoking **IMalloc::Free**.
PostFree	Called after invoking **IMalloc::Free**.
PreRealloc	Called before invoking **IMalloc::Realloc**.
PostRealloc	Called after invoking **IMalloc::Realloc**.
PreGetSize	Called before invoking **IMalloc::GetSize**.
PostGetSize	Called after invoking **IMalloc::GetSize**.
PreDidAlloc	Called before invoking **IMalloc::DidAlloc**.
PostDidAlloc	Called after invoking **IMalloc::DidAlloc**.
PreHeapMinimize	Called before invoking **IMalloc::DidAlloc**.
PostHeapMinimize	Called after invoking **IMalloc::HeapMinimize**.

See Also

IMalloc, **CoGetMalloc**, **CoRegisterMallocSpy**

IMallocSpy::PreAlloc

Called just prior to invoking **IMalloc::Alloc**.

```
ULONG PreAlloc(
   ULONG cbRequest        //The byte count passed to IMalloc::Alloc
  );
```

Parameter

cbRequest
The number of bytes in the allocation request the caller is passing to **IMalloc::Alloc**.

Return Value

The byte count actually passed to **IMalloc::Alloc**, which should be greater than or equal to the value of *cbRequest*.

Comments

The **IMallocSpy::PreAlloc** implementation may extend and/or modify the allocation to store debug-specific information with the allocation.

PreAlloc can force memory allocation failure by returning 0, allowing testing to ensure that the application handles allocation failure gracefully in all cases. In this case **PostAlloc** is not called and **Alloc** returns NULL. Forcing allocation failure is effective only if *cbRequest* is not equal to 0. If **PreAlloc** is forcing failure by returning NULL, **PostAlloc** is not called. However, if **IMalloc::Alloc** encounters a real memory failure and returns NULL, **PostAlloc** is called.

The call to **PreAlloc** through the return from **PostAlloc** is guaranteed to be thread safe.

See Also

IMalloc::Alloc, **IMallocSpy::PostAlloc**, **CoRegisterMallocSpy**, **CoRevokeMallocSpy**

IMallocSpy::PostAlloc

Called just after invoking **IMalloc::Alloc**.

```
void * PostAlloc(
   void * pActual      //Points to the allocation actually done by IMalloc::Alloc
  );
```

Parameter

pActual
Points to the allocation actually done by **IMalloc::Alloc**.

Return Value

The pointer returned to the **IMalloc::Alloc** caller. If debug information is written at the front of the caller's allocation, this should be a forward offset from *pActual*. The value is the same as *pActual* if debug information is appended or if no debug information is attached.

See Also

IMalloc::Alloc, **IMallocSpy::PreAlloc**, **CoRegisterMallocSpy**, **CoRevokeMallocSpy**

IMallocSpy::PreDidAlloc

Called just prior to invoking **IMalloc::DidAlloc**.

```
void * PreDidAlloc(
   void * pRequest,       //The pointer the caller is passing to IMalloc::DidAlloc
   BOOL fSpyed            //Indicates whether pRequest was allocated while this spy was active
  );
```

Parameters

pRequest
The pointer the caller is passing to **IMalloc::GetSize**.

fSpyed
TRUE if the allocation was done while this spy was active.

Comments

This method is included for completeness; it is not anticipated that developers will implement significant functionality in this method.

Return Value

The pointer whose allocation status is actually determined. This pointer is passed to **PostDidAlloc** as the *fActual* parameter.

See Also

IMalloc::DidAlloc, **IMallocSpy::PostDidAlloc**, **CoRegisterMallocSpy**, **CoRevokeMallocSpy**

IMallocSpy::PostDidAlloc

Called just prior to invoking **IMalloc::DidAlloc**.

```
int PostDidAlloc(
    void * pRequest       //The original pointer passed to IMalloc::DidAlloc
    BOOL fSpyed           //Indicates whether the allocation was done while this spy was active
    int fActual           //Indicates whether pRequest was actual value used in IMalloc call
  );
```

Parameters

pRequest
The pointer specified in the original call to **IMalloc::DidAlloc**.

fSpyed
TRUE if the allocation was done while this spy was active.

fActual
The actual value returned by **IMalloc::DidAlloc**.

Return Value

The value returned to the caller of **IMalloc::DidAlloc**.

Comments

This method is included for completeness; it is not anticipated that developers will implement significant functionality in this method.

For convenience, *pRequest*, the original pointer specified in the the call to **IMalloc::DidAlloc** is passed to **PostDidAlloc**, along with *fActual*, which is the actual value returned by **IMalloc::DidAlloc** called with the value returned by **IMallocSpy::PreDidAlloc**.

See Also

IMalloc::DidAlloc, **IMallocSpy::PreDidAlloc**, **CoRegisterMallocSpy**, **CoRevokeMallocSpy**

IMallocSpy::PreFree

Called just before invoking **IMalloc::Free**.

```
void * PreFree(
   void * pRequest,       //The pointer is passing to IMalloc::Free
   BOOL fSpyed            //TRUE if this memory was allocated while the spy was active
 );
```

Parameters

pRequest
The pointer to the block of memory that the caller is passing to **IMalloc::Free**.

fSpyed
The value is TRUE if the *pRequest* parameter of **IMallocSpy::PreFree** was allocated while the spy was installed. This value is also passed to **IMallocSpy::PostFree**.

Return Value

The actual pointer to pass to **IMalloc::Free**.

Comments

If the **PreAlloc** method modified the original allocation, **PreFree** needs to compute the actual pointer to be passed to **IMalloc::Free**. For example, if the **PreAlloc**/**PostAlloc** pair attached a header used to store debug information to the beginning of the user's allocation, then **PreFree** would return a pointer to this header so the true allocation block would be freed.

See Also

IMalloc::Free, **IMallocSpy::PostFree**, **CoRegisterMallocSpy**, **CoRevokeMallocSpy**

IMallocSpy::PostFree

Called just after invoking **IMalloc::Free**.

```
void PostFree(
    BOOL fSpyed        //Indicates whether the memory block to be freed was allocated while the spy is active
  );
```

Parameter

fSpyed

Boolean that is TRUE if the memory block to be freed was allocated while the current spy was active, otherwise FALSE.

See Also

IMalloc::Free, **IMallocSpy::PreFree**, **CoRegisterMallocSpy**, **CoRevokeMallocSpy**

IMallocSpy::PreGetSize

Called just prior to invoking **IMalloc::GetSize**.

```
void * PreGetSize(
    void * pRequest,        //Pointer the caller is passing to IMalloc::GetSize
    BOOL fSpyed             //TRUE if allocation done while this spy was active
  );
```

Parameters

pRequest
The pointer the caller is passing to **IMalloc::GetSize**

fSpyed
TRUE if the allocation was done while the spy was active.

Return Value

Pointer to the actual allocation for which the size is to be determined.

Comments

The **PreGetSize** method receives as its *pRequest* parameter the pointer the caller is passing to **IMalloc::GetSize**. It must then return a pointer to the actual allocation, which is then passed to **PostGetSize** as the *pRequest* parameter.

The size determined by **IMalloc::GetSize** is then passed to **IMallocSpy::PostGetSize** as *cbActual*.

Note The size determined by **IMalloc::GetSize** is the value returned by the Win32 API **HeapSize**. On Windows NT, this is the size originally requested. On Windows95, memory allocations are done on eight-byte boundaries. For example, a memory allocation request of 27 bytes on Windows NT would return an allocation of 32 bytes and **GetSize** would return 27. On Windows95, the same request would return an allocation of 28 bytes and **GetSize** would return 28. Implementors of **IMallocSpy::PostGetSize** cannot assume, for example, that if *cbActual* is sizeof(debug_header), that the value is the actual size of the user's allocation.

See Also

IMalloc::GetSize, **IMallocSpy::PostGetSize**, **CoRegisterMallocSpy**, **CoRevokeMallocSpy**

IMallocSpy::PostGetSize

Called just after invoking **IMalloc::GetSize**.

```
ULONG PostGetSize(
  ULONG cbActual,      //Actual size of the allocation
  BOOL fSpyed          //Indicates whether the allocation was done while a spy was active
 );
```

Parameters

cbActual
The actual number of bytes in the allocation, as returned by **IMalloc::GetSize**.

fSpyed
TRUE if the allocation was done while a spy was active.

Return Values

The size of the allocated memory block in bytes, which is returned to the caller of **IMalloc::GetSize**.

Comments

The size determined by **IMalloc::GetSize** is the value returned by the Win32 API **HeapSize**. On Windows NT, this is the size originally requested. On Windows95, memory allocations are done on eight-byte boundaries. For example, a memory allocation request of 27 bytes on Windows NT would return an allocation of 32 bytes and **GetSize** would return 27. On Windows95, the same request would return an allocation of 28 bytes and **GetSize** would return 28. Implementors of **IMallocSpy::PostGetSize** cannot assume, for example, that if *cbActual* is sizeof(debug_header), that the value is the actual size of the user's allocation.

See Also

IMalloc::GetSize, **IMallocSpy::PreGetSize**, **CoRegisterMallocSpy**, **CoRevokeMallocSpy**

IMallocSpy::PreHeapMinimize

Called just prior to invoking **IMalloc::HeapMinimize**.

void PreHeapMinimize(*void***);**

Comments

This method is included for completeness; it is not anticipated that developers will implement significant functionality in this method.

See Also

IMalloc::HeapMinimize, **IMallocSpy::PostHeapMinimize**, **CoRegisterMallocSpy**, **CoRevokeMallocSpy**

IMallocSpy::PostHeapMinimize

Called just after invoking **IMalloc::HeapMinimize**.

void PostHeapMinimize(*void***);**

Comments

This method is included for completeness; it is not anticipated that developers will implement significant functionality in this method.

See Also

IMalloc::HeapMinimize, **IMallocSpy::PreHeapMinimize**, **CoRegisterMallocSpy**, **CoRevokeMallocSpy**

IMallocSpy::PreRealloc

Called just before invoking **IMalloc::Alloc**.

```
ULONG PreRealloc(
  void * pRequest,              //The pointer the caller is passing to IMalloc::Realloc
  ULONG cbRequest,              //The byte count the caller is passing to IMalloc::Realloc
  void ** ppNewRequest,         //Points to the actual pointer to be reallocated
  BOOL fSpyed                   //Whether the original allocation was "spyed"
 );
```

Parameters

pRequest
The pointer the caller is passing to **IMalloc::Realloc**.

cbRequest
The byte count the caller is passing to **IMalloc::Realloc**.

ppNewRequest
[out] Points to the actual pointer to be reallocated. This may be different from the pointer in *pRequest* if the implementation of **IMallocSpy::PreRealloc** extends or modifies the reallocation. This is an out pointer and should always be stored by **PreRealloc**.

fSpyed
TRUE if the original allocation was done while the spy was active.

Return Value

The actual byte count to be passed to **IMalloc::Realloc**.

Comments

The **IMallocSpy::PreRealloc** implementation may extend and/or modify the allocation to store debug-specific information with the allocation. Thus, the *ppNewRequest* parameter may differ from *pRequest*, which the caller is passing to **IMalloc::Realloc**.

PreRealloc can force memory allocation failure by returning 0, allowing testing to ensure that the application handles allocation failure gracefully in all cases. In this case **PostRealloc** is not called and **Realloc** returns NULL. Forcing allocation failure is effective only if *cbRequest* is not equal to 0. Note that if **PreRealloc** is forcing failure by returning NULL, **PostRealloc** is not called. However, if **IMalloc::Realloc** encounters a real memory failure and returns NULL, **PostRealloc** is called.

See Also

IMalloc::Realloc, **IMallocSpy::PostRealloc**, **CoRegisterMallocSpy**, **CoRevokeMallocSpy**

IMallocSpy::PostRealloc

Called after invoking **IMalloc::Realloc**.

```
void * PostRealloc(
    void * pActual,       //The pointer returned by IMalloc::Realloc
    BOOL fSpyed           //Indicates whether the original allocation was "spyed"
  );
```

Parameters

pActual
: Points to the memory block reallocated by **IMalloc::Realloc**.

fSpyed
: If TRUE, the original memory allocation was done while the spy was active.

Return Values

The pointer to be returned to the **IMalloc::Realloc** caller. If debug information is written at the front of the caller's allocation, it should be a forward offset from *pActual*. The value should be the same as *pActual* if debug information is appended or if no debug information is attached.

See Also

IMalloc::Realloc, **IMallocSpy::PreRealloc**, **CoRegisterMallocSpy**, **CoRevokeMallocSpy**

IMarshal

The **IMarshal** interface allows an object to perform custom marshaling of interface pointers.

"Marshaling" is the process of packaging function parameters into data packets in order to send them to a different process or machine. "Unmarshaling" is the process of recovering those parameters when they are received. When a remote function call passes an interface pointer as a function parameter, that pointer is marshaled in such a way that it becomes possible for a client in one process to use that interface pointer to call methods on an object in another process.

OLE marshals an interface pointer by creating a proxy object in the client's process. A client holds an interface pointer to the proxy object and calls methods using that pointer; the proxy marshals the method parameters and sends them to the original process, thus forwarding the method call to the original object. The **IMarshal** interface allows an object to control what kind of proxies are created.

When to Implement

You almost never need to implement this interface. OLE provides a default implementation of **IMarshal** that uses OLE-generated proxy objects; this is suitable for almost all situations. You implement the **IMarshal** interface only if you want to perform custom marshaling; this allows you to write your own proxy objects.

OLE uses your implementation of **IMarshal** in the following manner: When it's necessary to create a remote interface pointer to your object (that is, when a pointer to your object is passed as a parameter in a remote function call), OLE queries your object for the **IMarshal** interface. If your object implements it, OLE uses your **IMarshal** implementation to create the proxy object. If your object does not implement **IMarshal**, OLE uses its standard implementation.

Some situations in which you might want to perform custom marshaling are as follows:

- The objects you are writing keep their state in shared memory. In this case, both the original process and the client process uses proxies that refer to the shared memory. This type of custom marshaling is possible only if the client process is on the same machine as the original process. The compound file implementations of **IStorage** and **IStream** are examples of this type of custom marshaling.
- The objects you are writing are immutable, that is, their state does not change after creation. Instead of forwarding method calls to the original objects, you can simply create copies of those objects in the client process. This technique avoids the cost of context switches (switching from one process to another). Monikers are an example of immutable objects; if you are implementing your own moniker class, you should perform custom marshaling.

Note that these situations do not require you to create a separate class of proxy objects; the same objects are used by both the original process and the client process. In other situations, you may need to write a separate class of proxy objects (with its own CLSID) that forward calls to the original objects, in addition to implementing **IMarshal** on the original objects.

When to Use

You typically do not need to call this interface. OLE is the primary user of this interface.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IMarshal Methods	Description

GetUnmarshalClass	Returns CLSID of unmarshaling code.
GetMarshalSizeMax	Returns size of buffer needed during marshaling.
MarshalInterface	Marshals an interface pointer.
UnmarshalInterface	Unmarshals an interface pointer.
ReleaseMarshalData	Destroys a marshaled data packet.
DisconnectObject	Severs all connections.

See Also

IStdMarshalInfo

IMarshal::DisconnectObject

Severs remote connections (that is, connections from other processes) being maintained on behalf of any interface pointers on the current object.

```
HRESULT IMarshal::DisconnectObject(
  DWORD dwReserved      //Reserved for future use
 );
```

Parameter

dwReserved
[in] Reserved for future use; must be set to zero by the caller. To ensure compatibility with future use, the callee *must not* check for zero.

Return Values

S_OK
The object was disconnected successfully.

E_FAIL
Indicates an unspecified error.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
One or more arguments were invalid.

E_UNEXPECTED
An unexpected error occured.

Comments

This method is called from the originating side (that is, the side doing the marshaling).

This method is called by the **CoDisconnectObject** API function if the object being disconnected implements the **IMarshal** interface. This is analogous to the way the **CoMarshalInterface** API function defers to the **IMarshal::MarshalInterface** method if the object supports **IMarshal**.

Notes to Implementors

If you are doing custom marshaling of immutable objects, your implementation doesn't need to do anything because there are no remote connections.

See Also

CoDisconnectObject

IMarshal::GetMarshalSizeMax

Returns an upper bound on the amount of data that would be written to the marshaling stream in a call to **IMarshal::MarshalInterface**.

```
HRESULT IMarshal::GetMarshalSizeMax(
    REFIID riid,                  //Identifies interface to be marshaled
    void *pv,                     //Points to interface pointer to be marshaled
    DWORD dwDestContext,          //Identifies destination process
    void * pvDestContext,         //Reserved for future use
    DWORD mshlflags,              //Specifies reason for marshaling
    ULONG * pSize                 //Receives upper bound
  );
```

Parameters

riid
[in]Specifies the IID of the interface pointer to be marshaled.

pv
[in]Points to the interface pointer to be marshaled; can be NULL.

dwDestContext
[in] Specifies the destination context, that is, the process in which the unmarshaling will be done. Different marshaling can be done depending on whether the unmarshaling will happen on the local workstation or on a workstation on the network; it's possible for an object to do custom marshaling in one case but not another. The legal values for *dwDestContext* are taken from the enumeration **MSHCTX**. For information on the **MSHCTX** enumeration, see the "Data Structures" section.

pvDestContext
[in] Reserved for use with future **MSHCTX** values.

mshlflags
[in] Specifies why marshaling is taking place. The legal values for *mshlflags* are taken from the enumeration **MSHLFLAGS**. For information on the **MSHLFLAGS** enumeration, see the "Data Structures" section.

pSize
[out]Receives the upper bound on the amount of data that would be written to the marshaling stream.

Return Values

S_OK
Maximum size was returned successfully.

E_FAIL
Indicates that the implementation failed.

E_NOINTERFACE
The specified interface was not supported.

Comments

This method is called from the originating side (that is, the side doing the marshaling). This method is used to preallocate the stream buffer that will be passed to **IMarshal::MarshalInterface**.

Notes to Implementors

The value your implementation returns must be an upper bound; a subsequent call to your implementation of **IMarshal::MarshalInterface** must use no more than the number of bytes returned by this method. The upper bound you return can be used by the caller to preallocate a stream buffer for using during marshaling.

See Also

IMarshal::MarshalInterface

IMarshal::GetUnmarshalClass

Returns the CLSID that should be used to create an uninitialized proxy on the receiving side (that is, the side doing the unmarshaling).

```
HRESULT IMarshal::GetUnmarshalClass(
    REFIID riid,                    //Identifies the interface being marshaled
    void * pv,                      //Points to the interface pointer being marshaled
    DWORD dwDestContext,            //Identifies the destination process
    void * pvDestContext,           //Reserved for future use
    DWORD mshlflags,                //Specifies reason for marshaling
    CLSID * pCid                    //Receives CLSID of proxy
  );
```

Parameters

riid
[in]Specifies the IID of the interface pointer to be marshaled.

pv
[in]Points to the interface pointer to be marshaled; can be NULL.

dwDestContext
[in] Specifies the destination context, that is, the process in which the unmarshaling will be done. Different marshaling can be done depending on whether the unmarshaling will happen on the local workstation or on a workstation on the network; it's possible for an object to do custom marshaling in one case but not another. The legal values for *dwDestContext* are taken from the enumeration **MSHCTX**. For information on the **MSHCTX** enumeration, see the "Data Structures" section.

pvDestContext
[in] Reserved for use with future **MSHCTX** values.

mshlflags
[in] Specifies why marshaling is taking place. Values for *mshlflags* are taken from the enumeration **MSHLFLAGS**. For information on the **MSHLFLAGS** enumeration, see the "Data Structures" section.

pCid
[out] Receives the CLSID to be used to create a proxy on the receiving side (that is, the side doing the unmarshaling).

Return Value

Returns S_OK if successful; otherwise, S_FALSE.

Comments

This method is called from the originating side (that is, the side doing the marshaling).

Notes to Implementors

The CLSID returned by this method identifies the class of the proxy object. This CLSID is typically written to a stream on the originating side and then read from the stream on the receiving side. Note that if you are creating a copy of the original object in the client process (that is, you are implementing custom marshaling for immutable objects), the CLSID returned is the same one as for the original object.

Your implementation can delegate some destination contexts to the standard marshaling implementation, available by calling the **CoGetStandardMarshal** API function. You should always delegate if the *dwDestContext* parameter contains a value that your implementation does not understand; this allows new destination contexts to be defined in the future.

You may want your implementation to use the pointer passed through the *pv* parameter, and, based on information retrieved from that pointer, determine the appropriate CLSID to return. If the *pv* parameter is NULL and your implementation needs that interface pointer, it can call **IUnknown::QueryInterface** on the current object to get it. The *pv* parameter exists merely to improve efficiency.

IMarshal::MarshalInterface

Marshals an interface pointer into the stream, that is, writes the appropriate data into the stream so that your **IMarshal::UnmarshalInterface** method can use that data to initialize a proxy object in another process.

```
HRESULT IMarshal::MarshalInterface(
    IStream *pStm,              //Stream used for marshaling
    REFIID riid,                //Identifies interface being marshaled
    void *pv,                   //Points to interface pointer being marshaled
    DWORD dwDestContext,        //Identifies destination process
    void *pvDestContext,        //Reserved for future use
    DWORD mshlflags             //Specifies reason for marshaling
  );
```

Parameters

pStm
[in]Points to the stream to be used during marshaling.

riid
[in]Specifies the IID of the interface pointer to be marshaled.

pv
[in]Points to the interface pointer to be marshaled; can be NULL.

dwDestContext
[in] Specifies the destination context, that is, the process in which the unmarshaling will be done. Different marshaling can be done depending on whether the unmarshaling will happen on the local workstation or on a workstation on the network; it's possible for an object to do custom marshaling in one case but not another. The legal values for *dwDestContext* are taken from the enumeration **MSHCTX**. For information on the **MSHCTX** enumeration, see the "Data Structures" section.

pvDestContext
[in] Reserved for use with future **MSHCTX** values.

mshlflags
[in]Specifies why marshaling is taking place. The legal values for *mshlflags* are taken from the **MSHLFLAGS** enumeration. For information on the **MSHLFLAGS** enumeration, see the "Data Structures" section.

Return Values

S_OK
The interface pointer was marshaled successfully.

E_FAIL
Indicates that the implementation failed.

E_NOINTERFACE
The specified interface was not supported.

STG_E_MEDIUMFULL
The stream was full.

Comments

This method is called from the originating side (that is, the side doing the marshaling). This method sends information to the receiving side by writing that information into the stream; on the receiving side, the information will be read from the stream and used to initialize a proxy object.

Notes to Implementors

Your implementation of **IMarshal::MarshalInterface** must write to the stream whatever data is needed to initialize the proxy on the receiving side. Your implementation can delegate some destination contexts to the standard marshaling implementation, which is available by calling the **CoGetStandardMarshal** API function. You should always delegate if the *dwDestContext* parameter contains a value that your implementation does not understand; this allows new destination contexts to be defined in the future.

Your implementation cannot assume the stream is large enough to hold all the data; it must still take into account the possibility that the stream may return STG_E_MEDIUMFULL errors. Just before exiting, your implementation should position the seek pointer in the stream immediately after the last byte of data written.

Note that the data marshaled by **IMarshal::MarshalInterface** can be unmarshaled zero or more times; this may happen if the marshaled data is stored in a global table, from which it may be unmarshaled many times by different client processes. Marshalers must be able to handle the result of multiple unmarshalings. For example, suppose the unmarshaling process creates a proxy that forwards calls to a stub interface in the server application; this stub interface must be able to deal with zero or more proxies being created from the same initialization data.

If the *pv* parameter is NULL and your implementation needs that interface pointer, it can call **IUnknown::QueryInterface** on the current object to get it. The *pv* parameter exists merely to improve efficiency.

See Also

CoGetStandardMarshal, **IMarshal::GetMarshalSizeMax**, **IMarshal::GetUnmarshalClass**, **IMarshal::UnmarshalInterface**

IMarshal::ReleaseMarshalData

Destroys a marshaled data packet.

```
HRESULT IMarshal::ReleaseMarshalData(
   IStream * pStm        //Stream used for unmarshaling
  );
```

Parameter

pStm
[in] Points to a stream that contains the data packet which is to be destroyed.

Return Values

S_OK
The data packet was released successfully.

E_FAIL
Indicates an unspecified error.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
One or more arguments were invalid.

E_UNEXPECTED
An unexpected error occured.

IStream errors
See the **IStream** interface for information on possible stream access errors.

Comments

This method is typically called from the receiving side (that is, the side doing the unmarshaling), sometimes via the **CoReleaseMarshalData** or **CoUnmarshalInterface** API functions. However, if the data packet does not arrive on the receiving side or if the appropriate proxy object cannot be successfully created there, this method is called from the originating side (that is, the side doing the marshaling).

As an analogy, the data packet can be thought of as a reference to the original object, just as if it were another interface pointer being held on the object. Like a real interface pointer, that data packet must be released at some point. The use of **IMarshal::ReleaseMarshalData** to release data packets is analogous to the use of **IUnknown::Release** to release interface pointers.

Notes to Implementors

If your implementation stores state information about marshaled data packets, use this method to release the state information associated with the data packet found in *pstm*. Your implementation should also position the seek pointer in the stream past the last byte of data.

See Also

CoUnMarshalInterface, **CoReleaseMarshalData**

IMarshal::UnmarshalInterface

Initializes a newly created proxy and returns an interface pointer to that proxy.

```
HRESULT IMarshal::UnmarshalInterface(
   IStream * pStm,        //Stream used for unmarshaling
   REFIID riid,           //IID of interface pointer desired
   void ** ppv            //Receives the interface pointer
  );
```

Parameters

pStm
[in]Points to the stream from which the interface pointer is to be unmarshaled.

riid
[in]Specifies the IID of the interface pointer desired.

ppv
[out]Receives the interface pointer.

Return Values

S_OK
The interface pointer was unmarshaled successfully.

E_FAIL
Indicates that the implementation failed.

E_NOINTERFACE
The specified interface was not supported.

Comments

This method is called from the receiving side (that is; the side doing the unmarshaling). This method is called on the proxy object itself; that is; the object whose CLSID was returned by **IMarshal::GetUnmarshalClass**.

Notes to Implementors

Your implementation should read the data written to the stream by your implementation of **IMarshal::MarshalInterface** and use that data to initialize the proxy object.

To return the appropriate interface pointer; your implementation can simply call **IUnknown::QueryInterface** on itself, passing the *riid* and *ppv* parameters. However; your implementation is free to create a different object and return a pointer to that object; if you find it necessary.

Just before exiting; your implementation should position the seek pointer in the stream immediately after the last byte of data read. This should be done even if the method exits with an error.

See Also

IMarshal::GetUnmarshalClass, **IMarshal::MarshalInterface**

IMessageFilter

This **IMessageFilter** interface allows applications to handle incoming or outgoing messages selectively while waiting for responses from synchronous calls. The ability to filter messages is often useful when a dialog box is being displayed or a lengthy operation is in progress. Performance can be improved when applications handle some messages and defer others.

When to Implement

Implement **IMessageFilter** in place of OLE's default message handling when there is a need to selectively handle incoming or outgoing messages, while at the same time waiting for responses from synchronous calls.

Although not as significant an issue as with 16-bit applications, you may still need to implement **IMessageFilter** as a way of solving deadlocks. **IMessageFilter** is still needed to find out if an application is blocking, so that you can task-switch to that application and give the user an opportunity to deal with the blocking application. For example, if you have Word talking to Excel, with Excel running in the background in formula mode, Excel won't check all calls, thereby blocking further action. **IMessageFilter** would put up a dialog indicating which task is blocking and provide the user with an opportunity to deal with the deadlock.

For applications that do not implement **IMessageFilter**, incoming and outgoing calls to OLE are handled automatically by OLE's default message handling procedures. OLE provides default message handling for each of the conditions that would be handled by an **IMessageFilter** method. For incoming calls, OLE dispatches all calls, regardless of their logical thread ID. For incoming Windows messages or Macintosh events, the default behavior is as though **IMessageFilter::MessagePending** had returned PENDINGMSG_WAITDEFPROCESS. This means that task-switching and window-activation messages are dispatched, as well as WM_TIMER messages. Other input messages are discarded and OLE continues to wait for the reply.

When to Use

You don't call this interface directly. It's provided by the application and called by the system.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns a pointer to a specified interface.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IMessageFilter Methods	**Description**
HandleIncomingCall	Provides a single entry point for incoming calls.
RetryRejectedCall	Provides application with opportunity to display dialog offering retry or cancel options or to switch tasks.
MessagePending	Indicates a Windows message has arrived while OLE is waiting to respond to a remote call.

See Also

In the *WIN32 Programmer's Reference*: "Messages and Message Queues"

IMessageFilter::HandleInComingCall

Provides an application with a single entry point for all incoming calls.

```
DWORD HandleInComingCall(
   DWORD dwCallType,                    //Type of incoming call
   HTASK threadIDCaller,                //Task handle calling this task
   DWORD dwTickCount,                   //Elapsed tick count
   LPINTERFACEINFO lpInterfaceInfo      //Points to INTERFACEINFO structure
  );
```

Parameters

dwCallType
   [in] Indicates the kind of incoming call that has been received; valid values are from the enumeration **CALLTYPE**. See the chapter on enumerations for details.

threadIDCaller
   [in] Specifies the handle of the task is calling this task.

dwTickCount
   [in] Specifies the elapsed tick count since the outgoing call was made if *dwCallType* is not CALLTYPE_TOPLEVEL. If *dwCallType* is CALLTYPE_TOPLEVEL, *dwTickCount* should be ignored.

lpInterfaceInfo
   Points to an **INTERFACEINFO** structure that contains information about the incoming call. Can also be NULL.

Return Values

SERVERCALL_ISHANDLED
   The application may be able to process the call.

SERVERCALL_REJECTED
   The application cannot handle the call due to an unforeseen problem, such as network unavailability.

SERVERCALL_RETRYLATER
   The application cannot handle the call at this time. For example, an application might return this value when it is in a user-controlled modal state.

Comments

If implemented, **IMessageFilter::HandleInComingCall** is called when an incoming OLE message is received.

Depending on the application's current state, the call can either be accepted and processed or rejected (permanently or temporarily). If SERVERCALL_ISHANDLED is returned, the application may be able to process the call. Success may depend on the interface for which the call is destined. If the call cannot be processed, OLE returns RPC_E_CALL_REJECTED.

Input-synchronized and asynchronous calls are dispatched even if the application returns SERVERCALL_REJECTED or SERVERCALL_RETRYLATER.

IMessageFilter::HandleInComingCall should not be used to hold off updates to objects during operations such as band printing. For that purpose, use **IViewObject::Freeze**.

You can also use **IMessageFilter::HandleInComingCall** to set up the application's state so the call can be processed in the future.

See Also

IViewObject::Freeze, **CALLTYPE**, **INTERFACEINFO**

IMessageFilter::MessagePending

Called if a Windows message or Macintosh event appears in the queue while OLE is waiting to reply to a remote call.

```
DWORD Create(
  HTASK threadIDCallee,     //Called applications task handle
  DWORD dwTickCount,        //Elapsed tick count
  DWORD dwPendingType       //Call type
 );
```

Parameters

threadIDCallee
[in] Specifies the task handle of the called application that has not yet responded.

dwTickCount
[in] Specifies the number of ticks since the call was made. It is calculated from the Windows **GetTickCount** function, or the Macintosh function, **TickCount**.

dwPendingType
[in] Indicates the type of call made during which a message or event was received. Valid values are from the enumeration **PENDINGTYPE** (where PENDINGTYPE_TOPLEVEL means the outgoing call was not nested within a call from another application and PENDINTGYPE_NESTED means the outgoing call was nested within a call from another application):

Return Values

PENDINGMSG_CANCELCALL
Cancel the outgoing call. This should be returned only under extreme conditions. Canceling a call that has not replied or been rejected can create orphan transactions and lose resources. OLE fails the original call and return RPC_E_CALL_CANCELLED.

PENDINGMSG_WAITNOPROCESS
Continue waiting for the reply and do not dispatch the message unless it is a task-switching or window-activation message. A subsequent message will trigger another call to **IMessageFilter::MessagePending**. Leaving messages or events in the queue enables them to be processed normally, if the outgoing call is completed.

PENDINGMSG_WAITDEFPROCESS
Because of the increased resources available in 32-bit systems, you are unlikely to get this return value. It now indicates the same state as PENDINGMSG_WAITNOPROCESS.

Keyboard and mouse messages are no longer dispatched, as was done with PENDINGMSG_WAITDEFPROCESS. However there are some cases where mouse and keyboard messages could cause the system to deadlock, and in these cases, mouse and keyboard messages are discarded. WM_PAINT messages are dispatched. Task-switching and activation messages are handled as before.

Comments

IMessageFilter::MessagePending is called by OLE after an application has made an OLE method call and, for example, while waiting for the reply, the user selects a menu command or double-clicks an object. Before OLE makes the **IMessageFilter::MessagePending** call, it calculates the elapsed time since the original OLE method call was made. OLE delivers the elapsed time in the *dwTickCount* parameter. In the meantime, OLE does not remove the message from the queue.

Windows messages or Macintosh events that appear in the caller's queue should remain in the queue until sufficient time has passed to ensure that the messages are probably not the result of typing ahead, but are, instead, an attempt to get attention. A two- or three-second delay is recommended. If that amount of time has passed and the call has not been completed, the messages should be flushed from the queue, and a dialog box should be displayed offering the user the choice of retrying (continue waiting) or switching to the task identified by the *threadIDCallee* parameter. This ensures that:

- if calls are completed in a reasonable amount of time, type ahead will be treated correctly.
- if the callee does not respond, type ahead is not misinterpreted and the user is able to act to solve the problem. For example, OLE 1 servers can queue up requests without responding when they are in modal dialogs.

Handling input while waiting for an outgoing call to finish can introduce complications. The application should determine whether to process the message without interrupting the call, continue waiting, or cancel the operation.

When there is no response to the original OLE call, the application can cancel the call and restore the OLE object to a consistent state by calling **IStorage::Revert** on its storage. The object can be released when the container can shut down. However, canceling a call can create orphaned operations and resource leaks. Canceling should be used *only* as a last resort. It is strongly recommended that applications not allow such calls to be canceled.

Caution Returning PENDINGMSG_WAITNOPROCESS can cause the message queue to fill.

See Also

IStorage::Revert

GetTickCount in Win32

IMessageFilter::RetryRejectedCall

Gives the application an opportunity to display a dialog box so the user can retry or cancel the call, or switch to the task identified by *threadIDCallee.*

```
DWORD RetryRejectedCall(
  HTASK threadIDCallee,      //Server task handle
  DWORD dwTickCount,         //Elapsed tick count
  DWORD dwRejectType         //Returned rejection message
 );
```

Parameters

threadIDCallee
 [in] Specifies the handle of the server task that rejected the call.

dwTickCount
 [in] Specifies the number of elapsed ticks since the call was made.

dwRejectType
 [in] Specifies either SERVERCALL_REJECTED or SERVERCALL_RETRYLATER, as returned by the object application.

Return Values

-1
 The call should be canceled. OLE then returns RPC_E_CALL_REJECTED from the original method call.

Value >= 0 and <100
 The call is to be retried immediately.

Value >= 100
 OLE will wait for this many milliseconds and then retry the call.

Comments

OLE calls **RetryRejectedCall** immediately after receiving SERVERCALL_RETRYLATER or SERVERCALL_REJECTED from **IMessageFilter::HandleInComingCall**.

If an application task rejects a call, the application is probably in a state where it cannot handle such calls, possibly only temporarily. When this occurs, OLE calls **IMessageFilter::RetryRejectedCall**.

Applications should silently retry calls that have returned with SERVERCALL_RETRYLATER and show a dialog box only after a reasonable amount of time has passed, say about five seconds. The callee may momentarily be in a state where calls can be handled. The option to wait and retry is provided for special kinds of calling applications, such as background tasks executing macros or scripts, so that they can retry the calls in a nonintrusive way.

If, after a dialog box is displayed, the user chooses to cancel, the call will appear to fail with RPC_E_CALL_REJECTED.

IMoniker

The **IMoniker** interface is used for naming COM objects. An object that implements **IMoniker** is called a "moniker," which is a name that uniquely identifies a COM object.

You can think of a moniker as a generalization of a pathname; in the same way that a pathname identifies a file in the file system, a moniker identifies a COM object. For example, suppose you have an object representing a range of cells in a spreadsheet which is itself embedded in a text document. The moniker for this object would contain information like "c:\work\report.doc\embedobj1\A1:E7." Monikers are more widely applicable than pathnames; whereas pathnames can only identify files, monikers can identify any kind of object, whether it's a file, an embedding, a selection within a document, or something else.

Monikers have a number of useful properties:

- Monikers can be saved to a persistent storage. When a moniker is loaded back into memory, it still identifies the same object it did before.
- Monikers support an operation called "binding," which is the process of locating the object named by the moniker, activating it (loading it in memory) if it isn't already, and returning an interface pointer to it.

As a result of these characteristics, you can treat monikers as a persistent version of a pointer to an object. Suppose your object holds an **ICellRange** pointer to the cell-range object mentioned in the paragraph above. You'd like to save your object and the cell-range object to disk, reload them later, and still have your object hold a pointer to the cell-range object. It's no use saving the **ICellRange** pointer to a disk file, because you can't guarantee that the cell-range object will reside at the same address in memory the next time it's loaded. However, you *can* save a moniker in a file. Then you can re-read the moniker into memory and bind the moniker, which reloads the cell-range object into memory and returns an **ICellRange** pointer to it. You've thus reestablished a connection between two objects after writing them to persistent storage.

Monikers also form the basis for linking in OLE. A linked object contains a moniker that identifies its source. When the user activates the linked object to edit it, the moniker is bound; this loads the link source into memory.

When to Implement

You implement **IMoniker** only if you are writing your own moniker class; this is necessary only if you need to identify objects that cannot be identified using the system-supplied moniker classes.

The system supplies the following moniker classes:

- **File monikers** – based on a path in the file system. File monikers can be used to identify objects that are saved in their own files.
- **Item monikers** – based on a string that identifies an object in a container. Item monikers can be used to identify objects smaller than a file, including embedded objects in a compound document, or a pseudo-object (like a range of cells in a spreadsheet).
- **Generic composite monikers** – consists of two or more monikers of arbitrary type that have been composed together. Generic composite monikers allow monikers of different classes to be used in combination.
- **Anti-monikers** – the inverse of file, item, or pointer monikers. Anti-monikers are used primarily for constructing relative monikers, which are analogous to relative pathnames (such as "..\backup\report.old"), and which specifies a location of an object *relative* to the location of another object).
- **Pointer monikers** – a non-persistent moniker that wraps an interface pointer to an object loaded in memory. Whereas most monikers identify objects that can be saved to persistent storage, pointer monikers identify objects that cannot; they allow such objects to participate in a moniker binding

operation.

These system-supplied moniker classes are sufficient for most situations. Before considering writing your own moniker class, you should make sure that your requirements cannot be satisified by these system-supplied moniker classes. See the description for each moniker class for information on how each one implements the **IMoniker** interface.

If you decide you need to write your own implementation of **IMoniker**, you must also implement the **IROTData** interface on your moniker class. This interface allows your monikers to be registered with the Running Object Table (ROT).

When to Use

You can use monikers whenever you need to create persistent connections between objects. You can also use monikers if you need a naming mechanism more general than pathnames; for example, if you need to designate embedded objects as well as files.

You can use monikers in two ways:

- As a moniker provider; that is, handing out monikers that identify your objects to make them accessible to other parties. When acting as a moniker provider, you need to understand the differences between the various system-supplied moniker classes in order to know which one(s) are appropriate for identifying your objects. You use certain API functions to create monikers, and you must implement other interfaces to allow the monikers you hand out to be bound.
- As a moniker client; that is, using monikers to get interface pointers to objects managed by a moniker provider. When acting as a moniker client, you typically don't need to know the class of the moniker you're using; you simply call methods using an **IMoniker** interface pointer.

The most common example of moniker providers are applications that act as link sources; that is, server applications that support linking and container applications that support linking to the embedded objects in their documents. See the descriptions for the moniker classes and for the moniker creation API functions for information on when to use each class.

The most common example of moniker clients are applications that act as link containers; that is, container applications that allow their documents to contain linked objects. However, link containers are a special case in that they make only infrequent calls to **IMoniker** methods. Generally, they manipulate linked objects through the **IOleLink** interface; the default handler implements this interface and calls the appropriate **IMoniker** methods as needed.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IPersist Methods	**Description**
GetClassID	Returns the object's CLSID.

IPersistStream Methods	**Description**
IsDirty	Checks whether object has been modified.
Load	Loads the object from a stream.
Save	Saves the object to a stream.
GetSizeMax	Returns the buffer size needed to save the object.

IMoniker Methods	Description
BindToObject	Binds to the object named by the moniker.
BindToStorage	Binds to the object's storage.
Reduce	Reduces the moniker to simplest form.
ComposeWith	Composes with another moniker.
Enum	Enumerates component monikers.
IsEqual	Compares with another moniker.
Hash	Returns a hash value.
IsRunning	Checks whether object is running.
GetTimeOfLastChange	Returns time the object was last changed.
Inverse	Returns the inverse of the moniker.
CommonPrefixWith	Finds the prefix that the moniker has in common with another moniker.
RelativePathTo	Constructs a relative moniker between the moniker and another.
GetDisplayName	Returns the display name.
ParseDisplayName	Converts a display name into a moniker.
IsSystemMoniker	Checks whether moniker is one of the system-supplied types.

See Also

BindMoniker, CreateBindCtx, CreateGenericComposite, CreateFileMoniker, CreateItemMoniker, CreateAntiMoniker, CreatePointerMoniker, IOleLink, IPersistStream, IROTData

IMoniker::BindToObject

"Binds" the moniker; that is, returns a pointer to the object identified by the moniker. The binding process involves finding the object, getting it into the running state if it's not already, and acquiring an interface pointer to it.

```
HRESULT BindToObject(
  IBindCtx *pbc,              //Bind context to be used
  IMoniker *pmkToLeft,        //Moniker that precedes this one in the composite
  REFIID riidResult,          //IID of interface pointer requested
  void **ppvResult            //Receives an interface pointer to the object
 );
```

Parameters

pbc
: [in] Points to the bind context to be used for this binding operation. The bind context caches objects bound during the binding process, contains parameters that apply to all operations using the bind context, and provides the means by which the moniker implementation should retrieve information about its environment. For more information, see **IBindCtx**.

pmkToLeft
: [in] Points to the moniker to the left of this moniker, if this moniker is part of a composite. This parameter is primarily used by moniker implementors to enable cooperation between the various components of a composite moniker; moniker clients can usually pass NULL.

riidResult
: [in] Specifies the IID of the interface pointer requested.

ppvResult
: [out] Receives a pointer to the object identified by the moniker. If an error occurs, the implementation sets **ppvResult* to NULL. If **ppvResult* is non-NULL, the implementation must call **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK
: The binding operation was successful.

MK_E_NOOBJECT
: Indicates that the object identifed by this moniker, or some object identified by the composite moniker of which this moniker is a part, could not be found.

MK_E_EXCEEDEDDEADLINE
: Indicates that the binding operation could not be completed within the time limit specified by the bind context's **BIND_OPTS** structure.

MK_E_CONNECTMANUALLY
: Indicates a moniker whose binding requires assistance from the end user. You can retry the binding operation after showing the moniker's display name to request that the end user manually connect to the object. The most common reasons for returning this value are that a password is needed or that a floppy needs to be mounted. The caller should call **IBindCtx::GetObjectParam** with the key "ConnectManually" to retrieve the moniker that caused the error, get the display name, display a dialog box asking the user for a password, and so on.

MK_E_INTERMEDIATEINTERFACENOTSUPPORTED
: An intermediate object was found but it did not support an interface required to complete the binding operation. For example, an item moniker returns this value if its container does not support the **IOleItemContainer** interface.

E_UNEXPECTED
: Indicates an unexpected error.

E_OUTOFMEMORY
Indicates insufficient memory.

STG_E_ACCESSDENIED
Unable to access the storage object.

IOleItemContainer::GetObject errors
Binding to a moniker containing an item moniker can return any of the errors associated with this function.

Comments

The **IMoniker::BindToObject** method implements the primary function of a moniker: returning an interface pointer to the object identified by the moniker.

Notes to Callers

If you are using a moniker as persistent connection between two objects, you activate the connection by calling **IMoniker::BindToObject**.

You typically call **IMoniker::BindToObject** with the following steps:

1. Create a bind context by calling the **CreateBindCtx** API function.
2. Call the **IMoniker::BindToObject** method on the moniker, retrieving an interface pointer.
3. Release the bind context.
4. Use the interface pointer.
5. Release the interface pointer.

Here's a code fragment that illustrates these steps:

```
// pMnk is an IMoniker * that points to a previously-acquired moniker
ICellRange *pCellRange;
IBindCtx *pbc;

CreateBindCtx( 0, &pbc );
pMnk->BindToObject( pbc, NULL, IID_ICellRange, &pCellRange );
pbc->Release();
// pCellRange now points to the object; safe to use pCellRange
pCellRange->Release();
```

You can also use the **BindMoniker** API function, which encapsulates the first three steps described above.

Note that link containers are a special case in that they typically don't need to call **IMoniker::BindToObject** directly. When a user activates a linked object, the link container can typically calls **IOleObject::DoVerb**. The link handler's implementation of **IOleObject::DoVerb** calls **IMoniker::BindToObject** on the moniker stored in the linked object (if it cannot handle the verb).

Notes to Implementors

What your implementation does depends on whether you expect your moniker to have a prefix, that is, whether you expect the *pmkToLeft* parameter to be NULL or not. For example, an item moniker, which identifies an object within a container, expects that *pmkToLeft* identifies the container. An item moniker consequently uses *pmkToLeft* to request services from that container. If you expect your moniker to have a prefix, you should use the *pmkToLeft* parameter (for instance, calling **IMoniker::BindToObject** on it) to request services from the object it identifies.

If you expect your moniker to have no prefix, your **IMoniker::BindToObject** implementation should first check the Running Object Table (ROT) to see if the object is already running. To acquire a pointer to the ROT, your implementation should call **IBindCtx::GetRunningObjectTable** on the *pbc* parameter.

You can then call the **IRunningObjectTable::GetObject** method to see if the current moniker has been registered in the ROT. If so, you can immediately use **IUnknown::QueryInterface** to return the desired interface pointer.

When your **IMoniker::BindToObject** implementation binds to some object, it should call **IBindCtx::RegisterObjectBound** on the *pbc* parameter to store a reference to the bound object in the bind context. This ensures that the bound object remains running until the bind context is released, which can avoid the expense of having a subsequent binding operation load it again later.

If the bind context's **BIND_OPTS** structure specifies the **BINDFLAGS_JUSTTESTEXISTENCE** flag, your implementation has the option of returning NULL in **ppvResult* (although you can also ignore the flag and perform the complete binding operation).

See Also

BindMoniker, **IMoniker::BindToStorage**

IMoniker::BindToStorage

Retrieves an interface pointer to the storage of the object identified by the moniker. Unlike the **IMoniker::BindToObject** method, this method does not run the object identified by the moniker.

```
HRESULT BindToStorage(
   IBindCtx *pbc,            //Bind context to be used
   IMoniker *pmkToLeft,      //Moniker to the left of this one in the composite
   REFIID riid,              //IID of interface pointer requested
   void **ppvObj             //Receives interface pointer to storage
  );
```

Parameters

pbc
[in] Points to the bind context to be used during this binding operation. The bind context caches objects bound during the binding process, contains parameters that apply to all operations using the bind context, and provides the means by which the moniker implementation should retrieve information about its environment. For more information, see **IBindCtx**.

pmkToLeft
[in] Points to the moniker to the left of this moniker, if this moniker is part of a composite. This parameter is primarily used by moniker implementors to enable cooperation between the various components of a composite moniker; moniker clients can usually pass NULL.

riid
[in] Indicates the requested interface pointer to be returned in *ppvObj*. Common interfaces requested include **IStorage**, **IStream**, and **ILockBytes**.

ppvObj
[out] Receives a pointer to the interface identified by *riid*. If an error occurs, the implementation sets **ppvObj* to NULL. If **ppvObj* is non-NULL, the implementation must call **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK
The binding operation was successful.

MK_E_NOSTORAGE
Indicates that the object identified by this moniker does not have its own storage.

MK_E_EXCEEDEDDEADLINE
Indicates that the operation could not be completed within the time limit specified by the bind context's **BIND_OPTS** structure.

MK_E_CONNECTMANUALLY
Indicates that the operation was unable to connect to the storage, possibly because a network device could not be connected to. For more information, see **IMoniker::BindToObject**.

MK_E_INTERMEDIATEINTERFACENOTSUPPORTED
An intermediate object was found but it did not support an interface required for an operation. For example, an item moniker returns this value if its container does not support the **IOleItemContainer** interface.

E_OUTOFMEMORY
Indicates insufficient memory.

STG_E_ACCESSDENIED
Unable to access the storage object.

IOleItemContainer::GetObject errors
Binding to a moniker containing an item moniker can return any of the errors associated with this function.

Comments

There is an important difference between the **IMoniker::BindToObject** and **IMoniker::BindToStorage** methods. If, for example, you have a moniker that identifies a spreadsheet object; calling **IMoniker::BindToObject** provides access to the spreadsheet object itself, while calling **IMoniker::BindToStorage** provides access to the storage object in which the spreadsheet resides.

Notes to Callers

The most common reason for calling the **IMoniker::BindToStorage** method is if you are implementing your own moniker class. You would call this method if your implementation of **IMoniker::BindToObject** needs information from the object identified by their *pmkToLeft* parameter, but it doesn't need to activate that object; all it needs is access to the persistent storage of that object. For example, if your monikers are used to identify objects that can be activated without activating their containers, you may find this method useful. (Note that none of the system-supplied moniker classes use this method in their binding operations.)

You could also call this method if you're a moniker client and you know how to read the storage of the object identified by the moniker you're holding.

Notes to Implementors

Your implementation should locate the persistent storage for the object identified by the current moniker and return the desired interface pointer. Some types of monikers represent pseudo-objects; that is, objects that don't have their own piece of persistent storage. Such objects comprise some portion of the internal state of its container; e.g. a range of cells in a spreadsheet. If your moniker class is this type of moniker, your implementation of **IMoniker::BindToStorage** should return the error MK_E_NOSTORAGE.

If the bind context's **BIND_OPTS** structure specifies the **BINDFLAGS_JUSTTESTEXISTENCE** flag, your implementation has the option of returning NULL in **ppvObj* (although it can also ignore the flag and perform the complete binding operation).

See Also

IMoniker::BindToObject

IMoniker::CommonPrefixWith

Creates a new moniker based on the common prefix that this moniker shares with the specified moniker.

```
HRESULT CommonPrefixWith(
  IMoniker *pmkOther,          //Moniker to be used for comparison
  IMoniker **ppmkPrefix        //Receives the prefix
 );
```

Parameters

pmkOther
[in] Points to the moniker to be compared with this one for a common prefix.

ppmkPrefix
[out] Receives a pointer to the moniker that is the common prefix of the this moniker and *pmkOther*. If an error occurs or if there is no common prefix, the implementation sets **ppmkPrefix* to NULL. If **ppmkPrefix* is non-NULL, the implementation must call **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK
A common prefix exists that is neither this moniker nor *pmkOther*.

MK_S_NOPREFIX
No common prefix exists.

MK_S_HIM
The entire *pmkOther* moniker is a prefix of this moniker.

MK_S_US
The two monikers are identical.

MK_S_ME
This moniker is a prefix of the *pmkOther* moniker.

MK_E_NOTBINDABLE
This method was called on a relative moniker. It is not meaningful to take the common prefix on a relative moniker.

E_OUTOFMEMORY
Indicates insufficient memory.

Comments

If, for example, you have one moniker that represents the path "c:\projects\secret\art\pict1.bmp" and another moniker that represents the path "c:\projects\secret\docs\chap1.txt." In that case, the common prefix of these two monikers would be a moniker representing the path "c:\projects\secret."

Notes to Callers

The **IMoniker::CommonPrefixWith** method is primarily called by the implementation of the **IMoniker::RelativePathTo** method. Moniker clients rarely need to call this method.

Note that it is not meaningful to take the common prefix on relative monikers. Consequently, you should call this method only if *pmkOther* and this moniker are both absolute monikers (where an absolute moniker is either a file moniker or a generic composite whose left-most component is a file moniker, and where the file moniker represents an absolute path). Do not call this method on relative monikers.

Notes to Implementors

Your implementation should first determine whether *pmkOther* is a moniker of a class that you recognize and for which you can provide special handling (for example, if it is of the same class as this moniker). If so, your implementation should determine the common prefix of the two monikers. Otherwise, it should pass both monikers in a call to the **MonikerCommonPrefixWith** API function, which correctly handles the generic case.

See Also

IMoniker::RelativePathTo, **MonikerCommonPrefixWith**

IMoniker::ComposeWith

Combines the current moniker with another moniker, creating a new, composite moniker.

```
HRESULT ComposeWith(
  IMoniker *pmkRight,           //Moniker to be composed onto this one
  BOOL fOnlyIfNotGeneric,       //Indicates if generic composition permissible
  IMoniker **ppmkComposite      //Receives the composite
 );
```

Parameters

pmkRight
[in] Points to the moniker to compose onto the end of this moniker.

fOnlyIfNotGeneric
[in] Indicates whether the method must perform a non-generic composition. If TRUE, the caller requires a non-generic composition, so the operation should proceed only if *pmkRight* is a moniker class that this moniker can compose with in some way other than forming a generic composite. If FALSE, the method can create a generic composite if necessary.

ppmkComposite
[out] Receives a pointer to the resulting composite moniker. If an error occurs or if the monikers compose to nothing (e.g., composing an anti-moniker with an item moniker or a file moniker), the implementation sets **ppmkComposite* to NULL. If **ppmkComposite* is non-NULL, the implementation must call **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK
Indicates that the monikers were successfully combined.

MK_E_NEEDGENERIC
Indicates that *fOnlyIfNotGeneric* was TRUE, but the monikers could not be composed together without creating a generic composite moniker.

E_OUTOFMEMORY
Indicates insufficient memory.

E_UNEXPECTED
Indicates an unexpected error.

Comments

The process of joining two monikers together is called composition. Sometimes two monikers of specific classes can be combined in a special manner; for example, a file moniker representing an incomplete path and another file moniker representing a relative path can be combined to form a single file moniker representing the complete path. This is an example of "non-generic" composition. "Generic" composition, on the other hand, can connect any two monikers, no matter what their classes. Because a non-generic composition depends on the class of the monikers involved, it can be performed only by an particular class's implementation of the **IMoniker::ComposeWith** method. You can define new types of non-generic compositions if you write a new moniker class. By contrast, generic compositions are performed by the **CreateGenericComposite** API function.

Composition of monikers is an associative operation. That is, if A, B, and C are monikers, then

Comp(Comp(A, B), C)

is always equal to

Comp(A, Comp(B, C))

where Comp() represents the composition operation.

Notes to Callers

If you want to combine two monikers, you should call the **IMoniker::ComposeWith** method rather than calling **CreateGenericComposite** API function; this gives the first moniker a chance to perform a non-generic composition.

A common situation in which you would combine two monikers is if you are a moniker provider using item monikers to identify your objects; for example, if your application is a server that supports linking to portions of a document, or if it's a container that supports linking to embedded objects within its documents. In such a situation, you would do the following:

1. Create an item moniker identifying an object.
2. Get a moniker that identifies the object's container.
3. Call **IMoniker::ComposeWith** on the moniker identifying the container, passing the item moniker as the *pmkRight* parameter.

Most callers of **IMoniker::ComposeWith** should set the *fOnlyIfNotGeneric* parameter to FALSE.

Notes to Implementors

As described above, there are two ways to compose *pmkRight* to your moniker: by performing a generic composition, or by performing a non-generic composition. The latter means seeing if you recognize the *pmkRight* parameter's class and if so, using the contents of *pmkRight* to perform a more intelligent composition.

When writing your moniker class, you must decide if there are any kinds of monikers to which you want to give special treatment (as described above in the example of the file monikers); these don't have to be monikers of your own class. You then implement **IMoniker::ComposeWith** to examine *pmkRight* to check whether it is such a special moniker; possible ways of doing this are to ask *pmkRight* for its class or call **IUnknown::QueryInterface** on it. The most common case of a special moniker is the inverse for your moniker class (that is, whatever you return from your implementation of **IMoniker::Inverse**).

If the *pmkRight* parameter is a moniker of a class that you give special treatment, your implementation of **IMoniker::ComposeWith** should do whatever is appropriate for that special case. If *pmkRight* completely negates the receiver so that the resulting composite is empty, you should pass back **NULL** in *ppmkComposite* and return the status code S_OK.

If the *pmkRight* parameter is not of a class to which you give special treatment, examine *fOnlyIfNotGeneric* to determine what to do next. If *fOnlyIfNotGeneric* is TRUE, pass back NULL through *ppmkComposite* and return the status code MK_E_NEEDGENERIC. If *fOnlyIfNotGeneric* is FALSE, call the **CreateGenericComposite** API to perform the composition generically.

See Also

CreateGenericComposite, **IMoniker::Inverse**

IMoniker::Enum

Returns a pointer to an enumerator that can enumerate the components of the moniker.

```
HRESULT Enum(
  BOOL fForward,                        //Specifies direction of enumeration
  IEnumMoniker **ppenumMoniker          //Receives the enumerator
 );
```

Parameters

fForward

[in] Specifies the enumeration order. TRUE enumerates the monikers from left to right; FALSE enumerates from right to left.

ppenumMoniker

[out] Receives a pointer to an **IEnumMoniker** enumerator for this moniker. If an error occurs or if the moniker has no enumerable components, the implementation sets **ppenumMoniker* to NULL. If **ppenumMoniker* is non-NULL, the implementation must call **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK

Indicates success. This value is returned even if the moniker does not provide an enumerator (that is, if **ppenumMoniker* equals NULL).

E_OUTOFMEMORY

Indicates insufficient memory.

E_UNEXPECTED

Indicates an unexpected error.

Comments

As examples of enumeration, the **IMoniker::Enum** method for a generic composite moniker creates an enumerator that returns the individual monikers that make up the composite, and the **IMoniker::Enum** method for a file moniker creates an enumerator that returns monikers representing each of the components in the pathname.

Notes to Callers

You would call this method if you want to examine the individual components that make up a composite moniker.

Notes to Implementors

If your monikers have no discernible internal structure, your implementation of this method can simply return S_OK and set **ppenumMoniker* to NULL.

See Also

IEnum*XXXX*

IMoniker::GetDisplayName

Returns the display name – that is, the user-readable representation – of this moniker.

```
HRESULT GetDisplayName(
  IBindCtx *pbc,                  //Bind context to be used
  IMoniker *pmkToLeft,            //Moniker to the left in the composite
  LPOLESTR *ppszDisplayName       //Receives the display name
 );
```

Parameters

pbc
[in] Points to the bind context to be used in this operation. The bind context caches objects bound during the binding process, contains parameters that apply to all operations using the bind context, and provides the means by which the moniker implementation should retrieve information about its environment. For more information, see **IBindCtx**.

pmkToLeft
[in] Points to the moniker to the left of this moniker, if this moniker is part of a composite. This parameter is primarily used by moniker implementors to enable cooperation between the various components of a composite moniker; moniker clients can usually pass NULL.

ppszDisplayName
[out] Receives a pointer to a zero-terminated string containing the display name of this moniker. If an error occurs, **ppszDisplayName* is set to NULL; otherwise, the implementation must use **IMalloc::Alloc** to allocate the string returned in *ppszDisplayName*, and the caller is responsible for calling **IMalloc::Free** to free it. Both caller and callee use the allocator returned by **CoGetMalloc**(MEMCTX_TASK, ...). For Win32 applications, the **LPOLESTR** type indicates a wide character string (two bytes per character); otherwise, the string has one byte per character.

Return Values

S_OK
Indicates the display name was successfully returned.

MK_E_EXCEEDEDDEADLINE
Indicates that the binding operation could not be completed within the time limit specified by the bind context's **BIND_OPTS** structure.

E_OUTOFMEMORY
Indicates insufficient memory.

E_NOTIMPL
There is no display name.

Comments

As examples of display names, a file moniker's display name is the path the moniker represents, and an item moniker's display name is the string contained in the moniker. However, a display name is not necessarily a complete representation of a moniker's internal state; it is simply a form readable by users. As a result, it is possible (though rare) for two different monikers to have the same display name. There is also no guarantee that the display name of a moniker can be parsed back into that moniker when calling the **MkParseDisplayName** API function with it, though failure to do so is also rare.

Notes to Callers

It's possible that retrieving a moniker's display name may be an expensive operation. For efficiency, you may want to cache the results of the first successful call to **IMoniker::GetDisplayName**, rather than making repeated calls.

Notes to Implementors

If you are writing a moniker class in which the display name does not change, simply cache the display name and return that when requested. If the display name can change over time, getting the current display name might mean that the moniker has to access the object's storage or bind to the object, either of which can be expensive operations. If this is the case, your implementation of **IMoniker::GetDisplayName** should return MK_E_EXCEEDEDDEADLINE if the name cannot be retrieved by the time specified in the bind context's **BIND_OPTS** structure.

A moniker that is intended to be part of a generic composite moniker should include any preceding delimiter (such as '\') as part of its display name. For example, the display name returned by an item moniker includes the delimiter specified when it was created with the **CreateItemMoniker** API function. The display name for a file moniker does not include a delimiter because file monikers always expect to be the leftmost component of a composite.

See Also

IMoniker::ParseDisplayName, **MkParseDisplayName**

IMoniker::GetTimeOfLastChange

Returns a number representing the time the object identified by this moniker was last changed. To be precise, the time returned is the earliest one OLE can identify after which no change has occurred, so this time may be later than the actual time of the last change to the object.

```
HRESULT GetTimeOfLastChange(
   IBindCtx *pbc,              //Bind context to be used
   IMoniker *pmkToLeft,        //Moniker to the left in the composite
   FILETIME *pFileTime         //Receives the time of last change
  );
```

Parameters

pbc
[in] Points to the bind context to be used in this binding operation. The bind context caches objects bound during the binding process, contains parameters that apply to all operations using the bind context, and provides the means by which the moniker implementation should retrieve information about its environment. For more information, see **IBindCtx**.

pmkToLeft
[in] Points to the moniker to the left of this moniker, if this moniker is part of a composite. This parameter is primarily used by moniker implementors to enable cooperation between the various components of a composite moniker; moniker clients can usually pass NULL.

pFileTime
[out] Points to the **FILETIME** structure receiving the time of last change. A value of {0xFFFFFFFF,0x7FFFFFFF} indicates an error (for example, exceeded time limit, information not available).

Return Values

S_OK
Indicates the method returned a time successfully.

MK_E_EXCEEDEDDEADLINE
Indicates that the binding operation could not be completed within the time limit specified by the bind context's **BIND_OPTS** structure.

MK_E_CONNECTMANUALLY
Indicates that the operation was unable to connect to the storage for this object, possibly because a network device could not be connected to. For more information, see **IMoniker::BindToObject**.

MK_E_UNAVAILABLE
The time of the change is unavailable, and will not be available no matter what deadline is used.

E_UNEXPECTED
Indicates an unexpected error.

Comments

Notes to Callers

If you're caching information returned by the object identified by the moniker, you may want to ensure that your information is up-to-date. To do so, you would call **IMoniker::GetTimeOfLastChange** and compare the time returned with the time you last retrieved information from the object.

For the monikers stored within linked objects, **IMoniker::GetTimeOfLastChange** is primarily called by the default handler's implementation of **IOleObject::IsUpToDate**. Container applications call **IOleObject::IsUpToDate** to determine if a linked object (or an embedded object containing linked objects) is up-to-date without actually binding to the object. This enables an application to quickly determine which linked objects require updating when the end user opens a document. The application can then bind only those linked objects that need updating (after prompting the end user to determine whether they should be updated), instead of binding every linked object in the document.

Notes to Implementors

It is important to perform this operation quickly because, for linked objects, this method is called when a user first opens a compound document. Consequently, your **IMoniker::GetTimeOfLastChange** implementation should not bind to any objects. In addition, your implementation should check the deadline parameter in the bind context and return MK_E_EXCEEDEDDEADLINE if the operation cannot be completed by the specified time.

There are a number of strategies you can attempt in your implementations:

- For many types of monikers, the *pmkToLeft* parameter identifies the container of the object identified by this moniker. If this is true of your moniker class, you can simply call **IMoniker::GetTimeOfLastChange** on the *pmkToLeft* parameter, since an object cannot have changed at a date later than its container.
- You can get a pointer to the Running Object Table (ROT) by calling **IBindCtx::GetRunningObjectTable** on the *pbc* parameter, and then calling **IRunningObjectTable::GetTimeOfLastChange**, since the ROT generally records the time of last change.
- You can get the storage associated with this moniker (or the *pmkToLeft* moniker) and return the storage's last modification time.

See Also

IBindCtx::GetRunningObjectTable, **IRunningObjectTable::GetTimeOfLastChange**

IMoniker::Hash

Calculates a 32-bit integer using the internal state of the moniker.

```
HRESULT Hash(
   DWORD *pdwHash       //Receives the hash value
  );
```

Parameter

pdwHash
[out] Receives the hash value.

Return Value

S_OK
Successfully received a 32-bit integer hash value.

Comments

Notes to Callers

You can use the value returned by this method to maintain a hash table of monikers. The hash value determines a hash bucket in the table. To search such a table for a specified moniker, calculate its hash value and then compare it to the monikers in that hash bucket using **IMoniker::IsEqual**.

Notes to Implementors

The hash value must be constant for the lifetime of the moniker. Two monikers that compare as equal using **IMoniker::IsEqual** must hash to the same value.

Marshaling and then unmarshaling a moniker should have no effect on its hash value. Consequently, your implementation of **IMoniker::Hash** should rely only on the internal state of the moniker, not on its memory address.

See Also

IMoniker::IsEqual

IMoniker::Inverse

Returns a moniker that, when composed to the right of this moniker or one of similar structure, will destroy it (that is, will compose to nothing).

```
HRESULT Inverse(
    IMoniker **ppmk        //Receives the inverse of the moniker
  );
```

Parameter

ppmk

[out] Receives a pointer to the inverse moniker. If an error occurs, the implementation sets **ppmk* to NULL. If **ppmk* is non-NULL, the implementation must call **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK

The inverse moniker has been returned successfully.

MK_E_NOINVERSE

Indicates that the moniker class does not have an inverse.

E_OUTOFMEMORY

Indicates insufficient memory.

Comments

The inverse of a moniker is analogous to the ".." directory in MS-DOS file systems; the ".." directory acts as the inverse to any other directory name, because appending ".." to a directory name results in an empty path. In the same way, the inverse of a moniker typically is also the inverse of all monikers in the same class. However, it is not necessarily the inverse of a moniker of a different class.

The inverse of a composite moniker is a composite consisting of the inverses of the components of the original moniker, arranged in reverse order. For example, if the inverse of A is Inv(A) and the composite of A, B, and C is Comp(A, B, C), then

Inv(Comp(A, B, C))

is equal to

Comp(Inv(C), Inv(B), Inv(A)).

Not all monikers have inverses. Most monikers that are themselves inverses do not have inverses; for example, anti-monikers do not have inverses. Monikers that have no inverse cannot have relative monikers formed from inside the objects they identify to other objects outside.

Notes to Callers

If you're a moniker client, you typically do not know the class of the moniker you're using. If you need to get the inverse of a moniker, you should always call **IMoniker::Inverse** rather than the **CreateAntiMoniker** API function, because you cannot be certain that the moniker you're using considers an anti-moniker to be its inverse.

The **IMoniker::Inverse** method is also called by the implementation of the **IMoniker::RelativePathTo** method, to assist in constructing a relative moniker.

Notes to Implementors

If your monikers have no internal structure, your implementation of **IMoniker::Inverse** can return an anti-moniker using the **CreateAntiMoniker** API function. Whatever you return from **IMoniker::Inverse** is what you must check for in your implementation of **IMoniker::ComposeWith**.

See Also

CreateAntiMoniker, IMoniker::ComposeWith, IMoniker::RelativePathTo

IMoniker::IsEqual

Compares the moniker with a specified moniker and indicates whether they are identical.

HRESULT IsEqual(
  IMoniker ***pmkOtherMoniker**   //Moniker to be used for comparison
 );

Parameter

pmkOtherMoniker
  [in] Points to the moniker to be used for comparison.

Return Values

S_OK
  The two monikers are identical.

S_FALSE
  The two monikers are not identical.

Comments

Previous implementations of the Running Object Table (ROT) called this method. The current implementation of the ROT uses the **IROTData** interface instead.

Notes to Callers

Call this method to determine if two monikers are identical or not. Note that the reduced form of a moniker is considered different from the unreduced form. You should call the **IMoniker::Reduce** method before calling **IMoniker::IsEqual**, because a reduced moniker is in its most specific form. **IMoniker::IsEqual** may return S_FALSE on two monikers before they are reduced, and S_OK after they are reduced.

Notes to Implementors

Your implementation should not reduce the current moniker before performing the comparison. It is the caller's responsibility to call **IMoniker::Reduce** in order to compare reduced monikers.

Note that two monikers that compare as equal must hash to the same value using **IMoniker::Hash**.

See Also

IMoniker::Reduce, **IMoniker::Hash**, **IROTData**

IMoniker::IsRunning

Determines whether the object identified by this moniker is currently loaded and running.

```
HRESULT IsRunning(
  IBindCtx *pbc,                //Bind context to be used
  IMoniker *pmkToLeft,          //Moniker to the left in the composite
  IMoniker *pmkNewlyRunning     //Moniker of a newly running object
 );
```

Parameters

pbc
: [in] Points to the bind context to be used in this binding operation. The bind context caches objects bound during the binding process, contains parameters that apply to all operations using the bind context, and provides the means by which the moniker implementation should retrieve information about its environment. For more information, see **IBindCtx**.

pmkToLeft
: [in] Points to the moniker to the left of this moniker if this moniker is part of a composite. This parameter is primarily used by moniker implementors to enable cooperation between the various components of a composite moniker; moniker clients can usually pass NULL.

pmkNewlyRunning
: [in] Points to the moniker most recently added to the Running Object Table (ROT). This can be NULL. If non-NULL, the implementation can return the results of calling **IMoniker::IsEqual** on the *pmkNewlyRunning* parameter, passing the current moniker. This parameter is intended to enable **IMoniker::IsRunning** implementations that are more efficient than just searching the ROT, but the implementation can choose to ignore *pmkNewlyRunning* without causing any harm.

Return Values

S_OK
: The moniker is running.

S_FALSE
: The moniker is not running.

E_UNEXPECTED
: Indicates an unexpected error.

Comments

Notes to Callers

If speed is important when you're requesting services from the object identified by the moniker, you may want those services *only* if the object is already running (because loading an object into the running state may be time-consuming). In such a situation, you'd call **IMoniker::IsRunning** to determine if the object is running.

For the monikers stored within linked objects, **IMoniker::IsRunning** is primarily called by the default handler's implementation of **IOleLink::BindIfRunning**.

Notes to Implementors

To get a pointer to the Running Object Table (ROT), your implementation should call **IBindCtx::GetRunningObjectTable** on the *pbc* parameter. Your implementation can then call **IRunningObjectTable::IsRunning** to determine whether the object identified by the moniker is running. Note that the object identified by the moniker must have registered itself with the ROT when it first began running.

See Also

IOleLink::BindIfRunning, **IBindCtx::GetRunningObjectTable**, **IRunningObjectTable::IsRunning**

IMoniker::IsSystemMoniker

Indicates whether this moniker is one of the system-supplied moniker classes.

```
HRESULT IsSystemMoniker(
  DWORD *pdwMksys       //Receives value from MKSYS enumeration
 );
```

Parameter

pdwMksys

[out] Receives an integer that is one of the values from the **MKSYS** enumeration. This parameter cannot be NULL. For information on the **MKSYS** enumeration, see the "Data Structures" section.

Return Values

S_OK

The moniker is a system moniker.

S_FALSE

The moniker is not a system moniker.

Comments

Notes to Callers

New values of the **MKSYS** enumeration may be defined in the future; therefore you should explicitly test for each value you are interested in.

Notes to Implementors

Your implementation of this method must return **MKSYS_NONE**. You cannot use this function to identify your own monikers (for example, in your implementation if **IMoniker::ComposeWith**). Instead, you should use your moniker's implementation of **IPersist::GetClassID** or use **IUnknown::QueryInterface** to test for your own private interface.

See Also

IPersist::GetClassID, **MKSYS**

IMoniker::ParseDisplayName

Reads as many characters of the specified display name as it understands and builds a moniker corresponding to the portion read; this procedure is known as "parsing" the display name.

```
HRESULT ParseDisplayName(
  IBindCtx *pbc,                //Bind context to be used
  IMoniker *pmkToLeft,          //Moniker to the left in the composite
  LPOLESTR pszDisplayName,      //Display name
  ULONG *pchEaten,              //Receives number of characters consumed
  IMoniker **ppmkOut            //Receives moniker built from display name
 );
```

Parameters

pbc
[in] Points to the bind context to be used in this binding operation. The bind context caches objects bound during the binding process, contains parameters that apply to all operations using the bind context, and provides the means by which the moniker implementation should retrieve information about its environment. For more information, see **IBindCtx**.

pmkToLeft
[in] Points to the moniker that has been built out of the display name up to this point.

pszDisplayName
[in] Points to a zero-terminated string containing the remaining display name to be parsed. For Win32 applications, the **LPOLESTR** type indicates a wide character string (two bytes per character); otherwise, the string has one byte per character.

pchEaten
[out] Receives the number of characters in *pszDisplayName* that were consumed in this step.

ppmkOut
[out] Receives a pointer to the moniker that was built from *pszDisplayName*. If an error occurs, the implementation sets **ppmkOut* to NULL. If **ppmkOut* is non-NULL, the implementation must call **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK
The parsing operation was completed successfully.

MK_E_SYNTAX
Indicates an error in the syntax of the input components (*pmkToLeft*, this moniker, and *pszDisplayName*). For example, a file moniker returns this error if *pmkToLeft* is non-NULL, and an item moniker returns it if *pmkToLeft* is NULL.

E_OUTOFMEMORY
Indicates insufficient memory.

E_UNEXPECTED
Indicates an unexpected error.

IMoniker::BindToObject errors
Parsing display names may cause binding. Thus, any error associated with this function may be returned.

Comments

Notes to Callers

Moniker clients do not typically call **IMoniker::ParseDisplayName** directly. Instead, they call the **MkParseDisplayName** API function when they want to convert a display name into a moniker (for example, in implementing the Links dialog box for a container application, or for implementing a macro language that supports references to objects outside the document). That API function first parses the initial portion of the display name itself. It then calls **IMoniker::ParseDisplayName** on the moniker it has just created, passing the remainder of the display name and getting a new moniker in return; this step is repeated until the entire display name has been parsed.

Notes to Implementors

Your implementation may be able to perform this parsing by itself if your moniker class is designed to designate only certain kinds of objects. Otherwise, you must get an **IParseDisplayName** interface pointer for the object identified by the moniker-so-far (i.e., the composition of *pmkToLeft* and this moniker) and then return the results of calling **IParseDisplayName::ParseDisplayName**.

There are different strategies for getting an **IParseDisplayName** pointer:

- You can try to get the object's CLSID (for example, by calling **IPersist::GetClassID** on the object), and then call the **CoGetClassObject** API function, requesting the **IParseDisplayName** interface on the class factory associated with that CLSID.
- You can try to bind to the object itself to get an **IParseDisplayName** pointer.
- You can try binding to the object identified by *pmkToLeft* to get an **IOleItemContainer** pointer, and then call **IOleItemContainer::GetObject** to get an **IParseDisplayName** pointer for the item.

Any objects that are bound should be registered with the bind context (see **IBindCtx::RegisterObjectBound**) to ensure that they remain running for the duration of the parsing operation.

See Also

IParseDisplayName, **MkParseDisplayName**

IMoniker::Reduce

Returns a reduced moniker; that is, another moniker that refers to the same object as this moniker but can be bound with equal or greater efficiency.

```
HRESULT Reduce(
  IBindCtx *pbc,               //Bind context to be used
  DWORD dwReduceHowFar,        //How much reduction should be done
  IMoniker **ppmkToLeft,       //Moniker to the left in the composite
  IMoniker **ppmkReduced       //Receives the reduced moniker
 );
```

Parameters

pbc

[in] Points to the bind context to be used in this binding operation. The bind context caches objects bound during the binding process, contains parameters that apply to all operations using the bind context, and provides the means by which the moniker implementation should retrieve information about its environment. For more information, see **IBindCtx**.

dwReduceHowFar

[in] Specifies how far this moniker should be reduced. This parameter must be one of the values from the **MKRREDUCE** enumeration. For more information on the **MKRREDUCE** enumeration, see the "Data Structures" section.

ppmkToLeft

[in, out] On entry, **ppmkToLeft* points to the moniker to the left of this moniker, if this moniker is part of a composite. This parameter is primarily used by moniker implementors to enable cooperation between the various components of a composite moniker; moniker clients can usually pass NULL.

On exit, **ppmkToLeft* is usually set to NULL, indicating no change in the original moniker to the left. In rare situations **ppmkToLeft* is set to non-NULL, indicating that the previous moniker to the left should be disregarded and the moniker returned through **ppmkToLeft* is the replacement. In such a situation, the implementation must call **IUnknown::Release** on the passed-in pointer and call **IUnknown::AddRef** on the returned moniker; the caller must release it later. If an error occurs, the implementation can either leave the parameter unchanged or set it to NULL.

ppmkReduced

[out] Receives a pointer to the reduced form of this moniker, which can be NULL if an error occurs or if this moniker is reduced to nothing. If this moniker cannot be reduced, **ppmkReduced* is simply set to this moniker and the return value is MK_S_REDUCED_TO_SELF. If *ppmkReduced* is non-NULL, the implementation must call **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**. (This is true even if **ppmkReduced* is set to this moniker.)

Return Values

S_OK

Indicates that this moniker was reduced.

MK_S_REDUCED_TO_SELF

Indicates that this moniker could not be reduced any further, in which case, **ppmkReduced* is this moniker.

MK_E_EXCEEDEDDEADLINE

Indicates that the operation could not be completed within the time limit specified by the bind context's **BIND_OPTS** structure.

E_UNEXPECTED

Indicates an unexpected error.

E_OUTOFMEMORY

Indicates insufficient memory.

Comments

The **IMoniker::Reduce** method is intended for the following uses:

- It enables the construction of user-defined macros or aliases as new kinds of moniker classes. When reduced, the moniker to which the macro evaluates is returned.
- It enables the construction of a kind of moniker that tracks data as it moves about. When reduced, the moniker of the data in its current location is returned.
- On file systems that support an ID-based method of accessing files which is independent of file names; a file moniker could be reduced to a moniker which contains one of these IDs.

The intent of the **MKRREDUCE** flags passed in the *dwReduceHowFar* parameter is to provide the ability to programmatically reduce a moniker to a form whose display name is recognizable to the user. For example, paths in the file system, bookmarks in word-processing documents, and range names in spreadsheets are all recognizable to users. In contrast, a macro or an alias encapsulated in a moniker are not recognizable to users.

Notes to Callers

The scenarios described above are not currently implemented by the system-supplied moniker classes.

You should call **IMoniker::Reduce** before comparing two monikers using the **IMoniker::IsEqual** method, because a reduced moniker is in its most specific form. **IMoniker::IsEqual** may return S_FALSE on two monikers before they are reduced and return S_OK after they are reduced.

Notes to Implementors

If the current moniker can be reduced, your implementation must not reduce the moniker in-place. Instead, it must return a new moniker that represents the reduced state of the current one. This way, the caller still has the option of using the non-reduced moniker (for example, enumerating its components). Your implementation should reduce the moniker *at least* as far as is requested.

See Also

IMoniker::IsEqual, **MKRREDUCE**

IMoniker::RelativePathTo

Returns a moniker that, when composed onto the end of this moniker (or one with a similar structure), yields the specified moniker.

```
HRESULT RelativePathTo(
  IMoniker *pmkOther,          //Moniker to which a relative path should be taken
  IMoniker **ppmkRelPath       //Receives the relative moniker
 );
```

Parameters

pmkOther
[in] Points to the moniker to which a relative path should be taken.

ppmkRelPath
[out] Receives a pointer to the relative moniker. If an error occurs, the implementation sets **ppmkRelPath* to NULL. If **ppmkRelPath* is non-NULL, the implementation must call **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK
A meaningful relative path has been returned.

MK_S_HIM
Indicates that there is no common prefix shared by the two monikers and that the moniker returned in **ppmkRelPath* is *pmkOther*.

MK_E_NOTBINDABLE
Indicates that this moniker is a relative moniker, such as an item moniker. This moniker must be composed with the moniker of its container before a relative path can be determined.

E_OUTOFMEMORY
Indicates insufficient memory.

E_UNEXPECTED
Indicates an unexpected error.

Comments

A relative moniker is analogous to a relative path (such as "..\backup"). For example, suppose you have one moniker that represents the path "c:\projects\secret\art\pict1.bmp" and another moniker that represents the path "c:\projects\secret\docs\chap1.txt." Calling **IMoniker::RelativePathTo** on the first moniker, passing the second one as the *pmkOther* parameter, would create a relative moniker representing the path "..\docs\chap1.txt."

Notes to Callers

Moniker clients typically do not need to call **IMoniker::RelativePathTo**. This method is primarily called by the default handler for linked objects. Linked objects contain both an absolute and a relative moniker to identify the link source (this enables link tracking if the user moves a directory tree containing both the container and source files). The default handler calls this method to create a relative moniker from the container document to the link source (that is, it calls **IMoniker::RelativePathTo** on the moniker identifying the container document, passing the moniker identifying the link source as the *pmkOther* parameter).

If you do call **IMoniker::RelativePathTo**, call it only on absolute monikers; for example, a file moniker or a composite moniker whose leftmost-component is a file moniker, where the file moniker represents an absolute path. Do not call this method on relative monikers.

Notes to Implementors

Your implementation of **IMoniker::RelativePathTo** should first determine whether *pmkOther* is a moniker of a class that you recognize and for which you can provide special handling (for example, if it is of the same class as this moniker). If so, your implementation should determine the relative path. Otherwise, it should pass both monikers in a call to the **MonikerRelativePathTo** API function, which correctly handles the generic case.

The first step in determining a relative path is determining the common prefix of this moniker and *pmkOther*. The next step is to break this moniker and *pmkOther* into two parts each, say (P, myTail) and (P, otherTail) respectively, where P is the common prefix. The correct relative path is then the inverse of myTail composed with otherTail:

Comp(Inv(myTail), otherTail)

Where Comp() represents the composition operation and Inv() represents the inverse operation.

Note that for certain types of monikers, you cannot use your **IMoniker::Inverse** method to construct the inverse of myTail. For example, a file moniker returns an anti-moniker as an inverse, while its **IMoniker::RelativePathTo** method must use one or more file monikers that each represent the path ".." to construct the inverse of myTail.

See Also

IMoniker::Inverse, **IMoniker::CommonPrefixWith**, **MonikerRelativePathTo**

IMoniker - Anti-Moniker Implementation

Anti-monikers are the inverse of the system's implementations of file, item, and pointer monikers. That is, an anti-moniker composed to the right of a file moniker, item moniker, or pointer moniker composes to nothing.

When To Use

If you're a moniker client, you typically do not need to use anti-monikers. When you need the inverse of a moniker, you should call the **IMoniker::Inverse** method. For example, if you need an inverse to remove the last piece of a composite moniker, use **IMoniker::Enum** to enumerate the pieces of the moniker and call **IMoniker::Inverse** on the rightmost piece. You shouldn't use an anti-moniker for this purpose because you can't be sure that the rightmost piece of a composite considers an anti-moniker to be its inverse.

The only situation in which you should explicitly use an anti-moniker is if you are writing a new moniker class and if you have no special requirements for constructing inverses to your monikers. In that situation, you can return anti-monikers from your implementation of **IMoniker::Inverse**. In your implementation of **IMoniker::ComposeWith**, you should then annihilate one of your monikers for every anti-moniker you encounter.

See Also

CreateAntiMoniker, **IMoniker**

Comments

IMoniker::BindToObject
This method is not implemented (that is, it returns E_NOTIMPL).

IMoniker::BindToStorage
This method is not implemented (that is, it returns E_NOTIMPL).

IMoniker::Reduce
This method returns MK_S_REDUCED_TO_SELF and passes back the same moniker.

IMoniker::ComposeWith
If *fOnlyIfNotGeneric* is TRUE, this method sets **ppmkComposite* to NULL moniker and returns MK_E_NEEDGENERIC; otherwise, the method returns the result of combining the two monikers into a generic composite. Note that composing a file, item, or pointer moniker to the right of an anti-moniker produces a generic composite rather than composing to nothing, as would be the case if the order of composition were reversed.

IMoniker::Enum
This method returns S_OK and sets **ppenumMoniker* to NULL.

IMoniker::IsEqual
This method returns S_OK if both are anti-monikers; otherwise, it returns S_FALSE.

IMoniker::Hash
This method calculates a hash value for the moniker.

IMoniker::IsRunning
This method checks the ROT to see if the object is running.

IMoniker::GetTimeOfLastChange
This method is not implemented (that is, it returns E_NOTIMPL).

IMoniker::Inverse
This method returns MK_E_NOINVERSE and sets **ppmk* to NULL.

IMoniker::CommonPrefixWith
If the other moniker is also an anti-moniker, the method returns MK_S_US and sets **ppmkPrefix* to this moniker. Otherwise, the method calls the **MonikerCommonPrefixWith** API function. This API

function correctly handles the case where the other moniker is a generic composite.

IMoniker::RelativePathTo

This method returns MK_S_HIM and sets **ppmkRelPath* to the other moniker.

IMoniker::GetDisplayName

For each anti-moniker contained in this moniker, this method return one instance of "\.."

IMoniker::ParseDisplayName

This method is not implemented (that is, it returns E_NOTIMPL).

IMoniker::IsSystemMoniker

This method returns S_OK and indicates MKSYS_ANTIMONIKER.

IMoniker - File Moniker Implementation

File monikers are monikers that represent pathnames in the file system; a file moniker can identify any object that is saved in its own file. To identify objects contained within a file, you can compose monikers of other classes (for example, item monikers) to the right of a file moniker. However, the moniker to the left of a file moniker within a composite must be either another file moniker or an anti-moniker (it is illegal, for example, for an item moniker to appear to the left of a file moniker in a composite).

Note that an anti-moniker is the inverse of an entire file moniker, not the inverse of a component of the pathname that the moniker represents; that is, when you compose an anti-moniker to the right of a file moniker, the entire file moniker is removed. If you want to remove just the rightmost component of the pathname represented by a file moniker, you must create a separate file moniker based on the ".." pathname and then compose that to the end of the file moniker.

When to Use

If you're a moniker client (that is, you're using a moniker to get an interface pointer to an object), you typically don't need to know the class of the moniker you're using; you simply call methods using an **IMoniker** interface pointer.

If you're a moniker provider (that is, you're handing out monikers that identify your objects to make them accessible to moniker clients), you must use file monikers if the objects you're identifying are stored in files. If each object resides in its own file, file monikers are the only type you need. If the objects you're identifying are smaller than a file, you need to use another type of moniker (for example, item monikers) in addition to file monikers.

To use file monikers, you must use the **CreateFileMoniker** API function to create the monikers. In order to allow your objects to be loaded when a file moniker is bound, your objects must implement the **IPersistFile** interface.

The most common example of moniker providers are OLE server applications that support linking. If your OLE server application supports linking only to file-based documents in their entirety, file monikers are the only type of moniker you need. If your OLE server application supports linking to objects smaller than a document (such as sections of a document or embedded objects), you must use item monikers as well as file monikers.

See Also

CreateFileMoniker, **IMoniker**, **IPersistFile**

Comments

IMoniker::BindToObject

This method requires that *pmkToLeft* be NULL. The method looks for the moniker in the ROT, and if found, queries the retrieved object for the requested interface pointer. If the moniker is not found in the ROT, the method loads the object from the file system and retrieves the requested interface pointer.

IMoniker::BindToStorage

This method opens the file specified by the pathname represented by the moniker and returns an **IStorage** pointer to that file. The method supports binding to **IStorage** interface only; if **IStream** or **ILockBytes** is requested in *riid*, the method returns E_UNSPEC, and if other interfaces are requested, this method returns E_NOINTERFACE. **IStream** and **ILockBytes** will be supported in future releases.

IMoniker::Reduce

This method returns MK_S_REDUCED_TO_SELF and passes back the same moniker.

IMoniker::ComposeWith

If *pmkRight* is an anti-moniker, the returned moniker is NULL. If *pmkRight* is a composite whose leftmost component is an anti-moniker, the returned moniker is the composite with the leftmost anti-moniker removed. If *pmkRight* is a file moniker, this method collapses the two monikers into a single file moniker, if possible. If not possible (e.g., if both file monikers represent absolute paths, as in *d:\work* and *e:\reports*), then the returned moniker is NULL and the return value is MK_E_SYNTAX. If *pmkRight* is neither an anti-moniker nor a file moniker, then the method checks the *fOnlyIfNotGeneric* parameter; if it is FALSE, the method combines the two monikers into a generic composite; if it is TRUE, the method sets **ppmkComposite* to NULL and returns MK_E_NEEDGENERIC.

IMoniker::Enum

This method returns S_OK and sets **ppenumMoniker* to NULL. In a future release this method will pass back an enumerator that enumerates the components of the path on which the moniker is based.

IMoniker::IsEqual

This method returns S_OK if **pmkOther* is a file moniker and the paths for both monikers are identical (using a case-insensitive comparison). Otherwise, the method returns S_FALSE.

IMoniker::Hash

This method calculates a hash value for the moniker.

IMoniker::IsRunning

If *pmkNewlyRunning* is non-NULL, this method returns TRUE if that moniker is equal to this moniker. Otherwise, the method asks the ROT whether this moniker is running. The method ignores *pmkToLeft*.

IMoniker::GetTimeOfLastChange

If this moniker is in the ROT, this method returns the last change time registered there; otherwise, it returns the last write time for the file. If the file cannot be found, this method returns MK_E_NOOBJECT.

IMoniker::Inverse

This method returns an anti-moniker (i.e., the results of calling **CreateAntiMoniker**).

IMoniker::CommonPrefixWith

If both monikers are file monikers, this method returns a file moniker that is based on the common components at the beginning of two file monikers. Components of a file moniker can be:

- A machine name of the form \\*server**share*. A machine name is treated as a single component, so two monikers representing the paths "\\myserver\public\work" and "\\myserver\private\games" do not have "\\myserver" as a common prefix.
- A drive designation (for example, "C:").
- A directory or filename.

If the other moniker is not a file moniker, this method passes both monikers in a call to the **MonikerCommonPrefixWith** API function. This API function correctly handles the case where the other moniker is a generic composite.

This method returns MK_E_NOPREFIX if there is no common prefix.

IMoniker::RelativePathTo

This method computes a moniker which when composed to the right of this moniker yields the other moniker. For example, if the path of this moniker is "C:\work\docs\report.doc" and if the other moniker is "C:\work\art\picture.bmp," then the path of the computed moniker would be "..\..\art\picture.bmp."

IMoniker::GetDisplayName

This method returns the pathname that the moniker represents. If this method is called by a 16-bit application, the method checks to see whether the specified file exists and, if so, returns a short name for that file because 16-bit applications are not equipped to handle long filenames.

IMoniker::ParseDisplayName

This method tries to acquire an **IParseDisplayName** pointer, first by binding to the class factory for the object identified by the moniker, and then by binding to the object itself. If either of these binding operations is successful, the file moniker passes the unparsed portion of the display name to the

IParseDisplayName::ParseDisplayName method.

This method returns MK_E_SYNTAX if *pmkToLeft* is non-NULL.

IMoniker::IsSystemMoniker

This method returns S_OK, and passes back MKSYS_FILEMONIKER.

IMoniker - Generic Composite Moniker Implementation

A generic composite moniker is a composite moniker whose components have no special knowledge of each other.

Composition is the process of joining two monikers together. Sometimes two monikers of specific classes can be combined in a special manner; for example, a file moniker representing an incomplete path and another file moniker representing a relative path can be combined to form a single file moniker representing the complete path. This is an example of "non-generic" composition. "Generic" composition, on the other hand, can connect any two monikers, no matter what their classes. Because a non-generic composition depends on the class of the monikers involved, it can be performed only by a particular class's implementation of the **IMoniker::ComposeWith** method. You can define new types of non-generic compositions if you write a new moniker class. By contrast, generic compositions are performed by the **CreateGenericComposite** API function.

When to Use

If you're a moniker client (that is, you're using a moniker to get an interface pointer to an object), you typically don't need to know the class of the moniker you're using, or whether it is a generic composite or a non-generic composite; you simply call methods using an **IMoniker** interface pointer.

If you're a moniker provider (that is, you're handing out monikers that identify your objects to make them accessible to moniker clients), you may have to compose two monikers together. (For example, if you are using an item moniker to identify an object, you must compose it with the moniker identifying the object's container before you hand it out.) You use the **IMoniker::ComposeWith** method to do this, calling the method on the first moniker and passing the second moniker as a parameter; this method may produce either a generic or a non-generic composite.

The only time you should explicitly create a generic composite moniker is if you are writing your own moniker class. In your implementation of **IMoniker::ComposeWith**, you should attempt to perform a non-generic composition whenever possible; if you cannot perform a non-generic composition and generic composition is acceptable, you can call the **CreateGenericComposite** API function to create a generic composite moniker.

See Also

CreateGenericComposite, **IMoniker**

Comments

IMoniker::BindToObject

If *pmkToLeft* is NULL, this method looks for the moniker in the ROT, and if found, queries the retrieved object for the requested interface pointer. If *pmkToLeft* is not NULL, the method recursively calls **IMoniker::BindToObject** on the rightmost component of the composite, passing the rest of the composite as the *pmkToLeft* parameter for that call.

IMoniker::BindToStorage

This method recursively calls **BindToStorage** on the rightmost component of the composite, passing the rest of the composite as the *pmkToLeft* parameter for that call.

IMoniker::Reduce

This method recursively calls **Reduce** for each of its component monikers. If any of the components reduces itself, the method returns S_OK and passes back a composite of the reduced components. If no reduction occurred, the method passes back the same moniker and returns MK_S_REDUCED_TO_SELF.

IMoniker::ComposeWith

If *fOnlyIfNotGeneric* is TRUE, this method sets **pmkComposite* to NULL and returns MK_E_NEEDGENERIC; otherwise, the method returns the result of combining the two monikers by calling the **CreateGenericComposite** API function.

IMoniker::Enum

If successful, this method returns S_OK and passes back an enumerator that enumerates the component monikers that make up the composite; otherwise, the method returns E_OUTOFMEMORY.

IMoniker::IsEqual

This method returns S_OK if the components of both monikers are equal when compared in the left-to-right order.

IMoniker::Hash

This method calculates a hash value for the moniker.

IMoniker::IsRunning

If *pmkToLeft* is non-NULL, this method composes *pmkToLeft* with this moniker and calls **IsRunning** on the result.

If *pmkToLeft* is NULL, this method returns TRUE if *pmkNewlyRunning* is non-NULL and is equal to this moniker.

If *pmkToLeft* and *pmkNewlyRunning* are both NULL, this method checks the ROT to see whether the moniker is running. If so, the method returns S_OK; otherwise, it recursively calls **IMoniker::IsRunning** on the rightmost component of the composite, passing the remainder of the composite as the *pmkToLeft* parameter for that call. This handles the case where the moniker identifies a pseudo-object that is not registered as running; see the Item Moniker implementation of **IMoniker::IsRunning**.

IMoniker::GetTimeOfLastChange

This method creates a composite of *pmkToLeft* (if non-NULL) and this moniker and uses the ROT to retrieve the time of last change. If the object is not in the ROT, the method recursively call **IMoniker::GetTimeOfLastChange** on the rightmost component of the composite, passing the remainder of the composite as the *pmkToLeft* parameter for that call.

IMoniker::Inverse

This method returns a composite moniker that consists of the inverses of each of the components of the original composite, stored in reverse order. For example, if the inverse of *A* is $A_{(-1)}$, then the inverse of the composite $A\ _{(^{\circ})}\ B\ _{(^{\circ})}\ C$ is $C_{(-1)}\ _{(^{\circ})}\ B_{(-1)}\ _{(^{\circ})}\ A_{(-1)}$.

IMoniker::CommonPrefixWith

If the other moniker is a composite, this method compares the components of each composite from left to right. The returned common prefix moniker might also be a composite moniker, depending on how many of the leftmost components were common to both monikers. If the other moniker is not a composite, the method simply compares it to the leftmost component of this moniker.

If the monikers are equal, the method returns MK_S_US and sets **ppmkPrefix* to this moniker. If the other moniker is a prefix of this moniker, the method returns MK_S_HIM and sets **ppmkPrefix* to the other moniker. If this moniker is a prefix of the other, this method returns MK_S_ME and sets **ppmkPrefix* to this moniker.

If there is no common prefix, this method returns MK_E_NOPREFIX and sets **ppmkPrefix* to NULL.

IMoniker::RelativePathTo

This method finds the common prefix of the two monikers and creates two monikers that consist of the remainder when the common prefix is removed. Then it creates the inverse for the remainder of this moniker and composes the remainder of the other moniker on the right of it.

IMoniker::GetDisplayName

This method returns the concatenation of the display names returned by each component moniker of the composite.

IMoniker::ParseDisplayName

This method recursively calls **IMoniker::ParseDisplayName** on the rightmost component of the composite, passing everything else as the *pmkToLeft* parameter for that call.

IMoniker::IsSystemMoniker

This method returns S_OK and indicates MKSYS_GENERICCOMPOSITE.

IMoniker - Item Moniker Implementation

Item monikers are used to identify objects within containers, such as a portion of a document, an embedded object within a compound document, or a range of cells within a spreadsheet. Item monikers are often used in combination with file monikers; a file moniker is used to identify the container while an item moniker is used to identify the item within the container.

An item moniker contains a text string; this string is used by the container object to distinguish the contained item from the others. The container object must implement the **IOleItemContainer** interface; this interface enables the item moniker code to acquire a pointer to an object, given only the string that identifies the object.

When to Use

If you're a moniker client (that is, you're using a moniker to get an interface pointer to an object), you typically don't need to know the class of the moniker you're using; you simply call methods using an **IMoniker** interface pointer.

If you're a moniker provider (that is, you're handing out monikers that identify your objects to make them accessible to moniker clients), you must use item monikers if the objects you're identifying are contained within another object and can be individually identified using a string. You'll also need to use another type of moniker (for example, file monikers) in order to identify the container object.

To use item monikers, you must use the **CreateItemMoniker** API function to create the monikers. In order to allow your objects to be loaded when an item moniker is bound, the container of your objects must implement the **IOleItemContainer** interface.

The most common example of moniker providers are OLE applications that support linking. If your OLE application supports linking to objects smaller than a file-based documents, you need to use item monikers. For a server application that allows linking to a selection within a document, you use the item monikers to identify those objects. For a container application that allows linking to embedded objects, you use the item monikers to identify the embedded objects.

See Also

CreateItemMoniker, **IMoniker**, **IOleItemContainer**

Comments

IMoniker::BindToObject
If *pmkToLeft* is NULL, this method returns E_INVALIDARG. Otherwise, the method calls **IMoniker::BindToObject** on the *pmkToLeft* parameter, requesting an **IOleItemContainer** interface pointer. The method then calls **IOleItemContainer::GetObject**, passing the string contained within the moniker, and returns the requested interface pointer.

IMoniker::BindToStorage
If *pmkToLeft* is NULL, this method returns E_INVALIDARG. Otherwise, the method calls **IMoniker::BindToObject** on the *pmkToLeft* parameter, requesting an **IOleItemContainer** interface pointer. The method then calls **IOleItemContainer::GetObjectStorage** for the requested interface.

IMoniker::Reduce
This method returns MK_S_REDUCED_TO_SELF and passes back the same moniker.

IMoniker::ComposeWith
If *pmkRight* is an anti-moniker, the returned moniker is NULL; if *pmkRight* is a composite whose leftmost component is an anti-moniker, the returned moniker is the composite after the leftmost anti-moniker is removed. If *pmkRight* is not an anti-moniker, the method combines the two monikers into a generic composite if *fOnlyIfNotGeneric* is FALSE; if *fOnlyIfNotGeneric* is TRUE, the method returns a NULL moniker and a return value of MK_E_NEEDGENERIC.

IMoniker::Enum

This method returns S_OK and sets **ppenumMoniker* to NULL.

IMoniker::IsEqual

This method returns S_OK if both monikers are item monikers and their display names are identical (using a case-insensitive comparison); otherwise, the method returns S_FALSE.

IMoniker::Hash

This method calculates a hash value for the moniker.

IMoniker::IsRunning

If *pmkToLeft* is NULL, this method returns TRUE if *pmkNewlyRunning* is non-NULL and is equal to this moniker. Otherwise, the method checks the ROT to see whether this moniker is running.

If *pmkToLeft* is non-NULL, the method calls **IMoniker::BindToObject** on the *pmkToLeft* parameter, requesting an **IOleItemContainer** interface pointer. The method then calls **IOleItemContainer::IsRunning**, passing the string contained within this moniker.

IMoniker::GetTimeOfLastChange

If *pmkToLeft* is NULL, this method returns MK_E_NOTBINDABLE. Otherwise, the method creates a composite of *pmkToLeft* and this moniker and uses the ROT to access the time of last change. If the object is not in the ROT, the method calls **IMoniker::GetTimeOfLastChange** on the *pmkToLeft* parameter.

IMoniker::Inverse

This method returns an anti-moniker (i.e., the results of calling **CreateAntiMoniker**).

IMoniker::CommonPrefixWith

If other moniker is an item moniker that is equal to this moniker, this method sets **ppmkPrefix* to this moniker and returns MK_S_US; otherwise, the method calls the **MonikerCommonPrefixWith** API function. This API function correctly handles the case where the other moniker is a generic composite.

IMoniker::RelativePathTo

This method returns MK_E_NOTBINDABLE and sets **ppmkRelPath* to NULL.

IMoniker::GetDisplayName

This method returns the concatenation of the delimiter and the item name that were specified when the item moniker was created.

IMoniker::ParseDisplayName

If *pmkToLeft* is NULL, this method returns MK_E_SYNTAX. Otherwise, the method calls **IMoniker::BindToObject** on the *pmkToLeft* parameter, requesting an **IOleItemContainer** interface pointer. The method then calls **IOleItemContainer::GetObject**, requesting an **IParseDisplayName** interface pointer to the object identified by the moniker, and passes the display name to **IParseDisplayName::ParseDisplayName**.

IMoniker::IsSystemMoniker

This method returns S_OK and indicates MKSYS_ITEMMONIKER.

IMoniker - Pointer Moniker Implementation

A pointer moniker essentially wraps an interface pointer so that it looks like a moniker and can be passed to those interfaces that require monikers. Binding a pointer moniker is done by calling the pointer's **QueryInterface** method.

Instances of pointer monikers refuse to be serialized, that is, **IPersistStream::Save** will return an error. These monikers can, however, be marshaled to a different process in an RPC call; internally, the system marshals and unmarshals the pointer using the standard paradigm for marshaling interface pointers.

When to Use

Pointer monikers are rarely needed. Use pointer monikers only if you need monikers to identify objects that have no persistent representation. Pointer monikers allow such objects to participate in a moniker-binding operation.

See Also

IMoniker, **CreatePointerMoniker**

Comments

IMoniker::BindToObject
This method queries the wrapped pointer for the requested interface.

IMoniker::BindToStorage
This method queries the wrapped pointer for the requested interface.

IMoniker::Reduce
This method returns MK_S_REDUCED_TO_SELF and passes back the same moniker.

IMoniker::ComposeWith
If *pmkRight* is an anti-moniker, the returned moniker is NULL; if *pmkRight* is a composite whose leftmost component is an anti-moniker, the returned moniker is the composite after the leftmost anti-moniker is removed. If *fOnlyIfNotGeneric* is FALSE, the returned moniker is a generic composite of the two monikers; otherwise, the method sets **ppmkComposite* to NULL and returns MK_E_NEEDGENERIC.

IMoniker::Enum
This method is not implemented (that is, it returns E_NOTIMPL).

IMoniker::IsEqual
This method returns S_OK only if both are pointer monikers and the interface pointers that they wrap are identical.

IMoniker::Hash
This method calculates a hash value for the moniker.

IMoniker::IsRunning
This method always returns S_OK, because the object identified by a pointer moniker must always be running.

IMoniker::GetTimeOfLastChange
This method is not implemented (that is, it returns E_NOTIMPL).

IMoniker::Inverse
This method returns an anti-moniker (i.e., the results of calling **CreateAntiMoniker**).

IMoniker::CommonPrefixWith
If the two monikers are equal, this method returns MK_S_US and sets **ppmkPrefix* to this moniker. Otherwise, the method returns MK_E_NOPREFIX and sets **ppmkPrefix* to NULL.

IMoniker::RelativePathTo
This method is not implemented (that is, it returns E_NOTIMPL).

IMoniker::GetDisplayName

This method is not implemented (that is, it returns E_NOTIMPL).

IMoniker::ParseDisplayName

This method queries the wrapped pointer for the **IParseDisplayName** interface and passes the display name to **IParseDisplayName::ParseDisplayName**.

IMoniker::IsSystemMoniker

This method returns S_OK and indicates MKSYS_POINTERMONIKER.

IOleAdviseHolder

The **IOleAdviseHolder** interface enables object applications to delegate the management of compound document registration requests and notifications to a special object called the OLE advise holder. The OLE advise holder implements the **IOleAdviseHolder** interface, which provides methods for establishing and deleting advisory connections and for sending compound document notifications to advise sinks.

When an object application uses the OLE advise holder, it passes notifications directly to the **IOleAdviseHolder** interface. The advise holder keeps track of which advise sinks are interested in which notifications and passes along the notifications as appropriate.

When to Implement

OLE provides a default implementation of the OLE advise holder as a convenience to programmers. Use of the default implementation is optional because object applications can be written to manage notifications themselves. By using the default code, however, you save yourself the work of re-creating the advise holder's functionality from scratch as part of your implementation of **IOleObject::Advise**. Few applications will require notification capabilities beyond those which the default advise holder provides.

Applications instantiate an OLE advise holder by calling the OLE API function **CreateOleAdviseHolder**. (OLE also provides a data advise holder to manage data notifications. Applications create a data advise holder by calling the OLE API function **CreateDataAdviseHolder**.)

When to Use

Containers and other objects that wish to receive compound document notifications must implement the **IAdviseSink** interface to receive those notifications, and call the **IOleAdviseHolder** interface to establish an advisory connection and inform the object of what specific notifications it wishes to receive.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IOleAdviseHolder Methods	Description
Advise	Establishes advisory connection with sink.
Unadvise	Deletes advisory connection with sink.
EnumAdvise	Lists current advisory connections.
SendOnRename	Advises that name of object has changed.
SendOnSave	Advises that object has been saved.
SendOnClose	Advises that object has been closed.

IOleAdviseHolder::Advise

Establishes an advisory connection between an OLE object and the calling object's advise sink. Through that sink, the calling object can receive notification when the OLE object is renamed, saved, or closed.

```
HRESULT Advise(
   IAdviseSink * pAdvise,         //Pointer to advise sink
   DWORD * pdwConnection          //Pointer to a token
  );
```

Parameters

pAdvise
[in] Points to the advisory sink that should be informed of changes.

pdwConnection
[out] Points to a token that can be passed to the **IOleAdviseHolder::Unadvise** method to delete the advisory connection. The calling object is responsible for calling both **IUnknown::AddRef** and **IUnknown::Release** on this pointer.

Return Values

S_OK
Advisory connections set up successfully.

E_INVALIDARG
pAdvise is NULL.

Comments

Containers, object handlers, and link objects all create advise sinks to receive compound-document notifications (**OnSave**, **OnRename**, **and OnClose**) for objects of interest. They also call **IOleObject::Advise** to establish advisory connections with those objects. Objects whose implementations of **IOleObject::Advise** include a call to **CreateOleAdviseHolder** delegate these calls to **IOleAdviseHolder::Advise**.

If the attempt to establish an advisory connection is successful, the object receiving the call returns a nonzero value through *pdwConnection*. If the attempt fails, the object returns a zero. To delete an advisory connection, the object with the advise sink passes this nonzero token back to the object by calling **IOleAdviseHolder::Unadvise**.

See Also

IOleAdviseHolder::UnAdvise, **IOleAdviseHolder::EnumAdvise**, **IOleObject::Advise**

IOleAdviseHolder::EnumAdvise

Enumerates the advisory connections currently established for an object.

```
HRESULT EnumAdvise(
   IEnumSTATDATA ** ppenumAdvise          //A pointer to a pointer to the new enumerator
  );
```

Parameter

ppenumAdvise

[out] Points to where the new enumerator should be returned. NULL is a legal return value indicating that there are presently no advisory connections on the object. If an error is returned; this parameter must be set to NULL. Each time an OLE advise holder receives a call to **IOleAdviseHolder::EnumAdvise**; it must increase the reference count on the pointer it returns. It is the caller's responsibility to call **IUnknown::Release** when it is done with the pointer.

Return Values

S_OK

Enumerator returned successfully.

E_FAIL

Enumerator could not be returned.

E_NOTIMPL

EnumAdvise is not implemented.

Comments

While an enumeration is in progress; the effect of registering or revoking advisory connections on what is to be enumerated is undefined. The returned enumerator is of type **IEnumSTATDATA**. It enumerates items of type **STATDATA**; which are defined as follows:

```
typedef struct tagSTATDATA {
     FORMATETC          Formatetc;
     DWORD              grfAdvf;
     IAdviseSink *      pAdvise;
     DWORD              dwConnection;
     }STATDATA;
```

See Also

IOleAdviseHolder::Advise, **IOleAdviseHolder::UnAdvise**, **IOleObject::EnumAdvise**

IOleAdviseHolder::SendOnClose

Sends **OnClose** notifications to all advisory sinks currently registered with the advise holder.

HRESULT SendOnClose();

Return Value

S_OK
    Advise sinks were sent OnClose notifications.

See Also

IAdviseSink::OnClose

IOleAdviseHolder::SendOnRename

Sends **IAdviseSink::OnRename** notifications to all advisory sinks currently registered with the advise holder.

```
HRESULT SendOnRename(
    IMoniker *pmk          //Pointer to a IMoniker interface
  );
```

Parameter

pmk
[in] Points to the new full moniker of the object.

Return Value

S_OK
Advise sinks were sent **IAdviseSink::OnRename** notifications.

See Also

IAdviseSink::OnRename

IOleAdviseHolder::SendOnSave

Sends **IAdviseSink::OnSave** notifications to all advisory sinks currently registered with the advise holder.

HRESULT SendOnSave();

Return Value

S_OK
  Advise sinks were sent **IAdviseSink::OnSave** notifications.

See Also

IAdviseSink::OnSave

IOleAdviseHolder::Unadvise

Deletes a previously established advisory connection.

```
HRESULT Unadvise(
   DWORD dwConnection        //A nonzero value
  );
```

Parameter

dwConnection
[in] Contains a nonzero value previously returned by **IOleAdviseHolder::Advise** in *pdwConnection*.

Return Values

S_OK
Advisory connection deleted successfully.

OLE_E_NOCONNECTION
The *dwConnection* parameter does not represent a valid advisory connection.

Comments

Normally, containers call this method at shutdown or when an object is deleted. In certain cases, containers could choose to call this method on objects that are running but not currently visible, as a way of reducing the overhead of maintaining multiple advisory connections.

See Also

IOleAdviseHolder::Advise, **IOleAdviseHolder::EnumAdvise**, **IOleObject::Unadvise**

IOleCache

The **IOleCache** interface provides control of the presentation data that gets cached inside of an object. Cached presentation data is available to the container of the object even when server application is not running or is unavailable.

When to Implement

The **IOleCache** interface can be implemented by an object handler and an in-process server. However, you typically use or aggregate the OLE-provided implementation.

When to Use

The OLE-provided implementation creates a class node for each aspect and format of the data that it stores. In addition to the presentation data, the class node holds the sink pointer for the advisory connection and manages the sending of view notifications to the advise sink.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IOleCache Methods	Description
Cache	Adds a presentation to the data or view cache.
Uncache	Removes a presentation previously added with **IOleCache::Cache**.
EnumCache	Returns an object to enumerate the current cache connections.
InitCache	Fills the cache with all the presentation data from the data object.
SetData	Fills the cache with specified format of presentation data.

IOleCache::Cache

Specifies the format and other data to be cached inside an embedded object.

```
HRESULT Cache(
   FORMATETC * pFormatetc,     //Points to the data and formats to be cached
   DWORD advf,                 //Flags that control the caching
   DWORD * pdwConnection       //Identifies the connection for future calls to uncache
  );
```

Parameters

pFormatetc

[in]Points to the format and other data to be cached. View caching is specified by passing a zero clipboard format in *pFormatetc*.

advf

[in]Contains a group of flags that control the caching. Valid values can be derived by using an OR operation on the values in the **ADVF** enumeration. However, only some of the possible **ADVF** values are relevant for this method. The following table briefly describes the relevant values and how they control caching. See the **ADVF** enumeration for a more detailed description.

ADVF Value	Description
ADVF_NODATA	Asks the data object to avoid sending data with the notifications. This flag is a way to override the default behavior of sending data with the notification. Typically, this flag is used when the iconic aspect of an object is cached. The cache can then be updated explicitly by calling **IOleCache::SetData**, **IDataObject::SetData**, or **IOleCache2::UpdateCache**.
ADVF_ONLYONCE	Causes the advisory connection to be destroyed after the first notification is sent.
ADVF_PRIMEFIRST	Causes an initial notification to be sent regardless of whether data has changed from its current state.
ADVFCACHE_NOHANDLER	Synonym for ADVFCACHE_FORCEBUILTIN.
ADVFCACHE_FORCEBUILTIN	For cache connections, this flag caches data that requires only code shipped with OLE or the underlying operating system to be present in order to produce it with **IDataObject::GetData** or **IViewObject::Draw**. By specifying this value, the container can ensure that the data can be retrieved even when the object or handler code is not available. This value is used by DLL object applications and object handlers that perform the drawing of their objects.

	ADVFCACHE_FORCEBUILTIN instructs OLE to cache presentation data to ensure that there is a presentation in the cache.
ADVFCACHE_ONSAVE	Updates the cached representation only when the object containing the cache is saved. The cache is also updated when the OLE object changes from the running state back to the loaded state (because a subsequent save operation would require running the object again).

pdwConnection
[out]Points to the location of a returned token that identifies this connection and can later be used to turn caching off (by passing it to **IOleCache::Uncache**). If this value is zero, the connection was not established. The OLE-provided implementation uses nonzero numbers for connection identifiers.

Return Values

S_OK
Requested data or view successfully cached.

E_INVALIDARG
One or more arguments are invalid.

E_OUTOFMEMORY
Ran out of memory.

E_UNEXPECTED
An unexpected error occurred.

CACHE_S_FORMATETC_NOTSUPPORTED
Indicates the cache was created, but the object application does not support the specified format. Cache creation succeeds even if the format is not supported, allowing the caller to fill the cache. If, however, the caller does not need to keep the cache, call **IOleCache::UnCache**.

CACHE_S_SAMECACHE
Indicates a cache already exists for the **FORMATETC** passed to **IOleCache::Cache**. In this case, the new advise flags are assigned to the cache, and the previously assigned connection identifier is returned.

DV_E_LINDEX
Invalid value for *lindex*; currently only -1 is supported.

DV_E_TYMED
The value is not valid for *pFormatetc->tymed*.

DV_E_DVASPECT
The value is not valid for *pFormatetc->dwAspect*.

DV_E_CLIPFORMAT
The value is not valid for *pFormatetc->cfFormat*.

CO_E_NOTINITIALIZED
The cache's storage is not initialized.

DV_E_DVTARGETDEVICE
The value is not valid for *pFormatetc->ptd*.

OLE_E_STATIC
The cache is for static object and it already has a cache node.

Comments

IOleCache::Cache can specify either data caching by passing a valid data format in *pFormatetc* or view (presentation) caching by passing a zero data format in *pFormatetc* as follows:

```
pFormatetc->cfFormat == 0
```

With view caching, the cache object itself decides on the format to cache.

A custom object handler can choose not to store data in a given format. Instead, it can synthesize it on demand when requested.

The *advf* value of ADVFCACHE_FORCEBUILTIN ensures that presentation data can be retrieved after the container document has been moved where the object application or object handler is not available.

When an **IOleCache::Cache** call is made to cache either CF_DIB or CF_BITMAP, the OLE cache implementation will implicitly cache the other format. Thus, the enumerator (**IOleCache::EnumCache**) will have two entries (CF_DIB and CF_BITMAP) for the one cached format. Both of these entries have the same connection ID, so one call to **IOleCache::Uncache** will remove both. CF_DIB and CFBITMAP may also be explicitly cached separately (by two calls to **IOleCache::Cache**).

See Also

ADVF, **IOleCache::Uncache**

IOleCache::EnumCache

Returns an object that can be used to enumerate the current cache connections.

```
HRESULT EnumCache(
  IEnumSTATDATA ** ppenumSTATDATA          //Location for IEnumSTATDATA
 );
```

Parameter

ppenumSTATDATA
[out]Points to the location where the new enumerator object should be returned. If this value is NULL, there are currently no cache connections at this time.

Return Values

S_OK
The enumerator object is successfully instantiated or there are no cache connections. Check the *ppenumSTATDATA* parameter to determine which result occurred. If the *ppenumSTATDATA* parameter is NULL, then there are no cache connections at this time.

E_OUTOFMEMORY
The cache enumerator could not be instantiated due to lack of memory.

Comments

The enumerator object returned by this method implements the **IEnumSTATDATA** interface, one of the standard enumerator interfaces that contain the **Next**, **Reset**, **Clone**, and **Skip** methods. **IEnumSTATDATA** enumerates the data stored in an array of **STATDATA** structures containing information about current cache connections.

See Also

IEnum*XXXX*, **IEnumSTATDATA**, **IOleCache::Cache**

IOleCache::InitCache

Fills the cache as needed using the data provided by the specified data object.

```
HRESULT InitCache(
  IDataObject * pDataObject        //Points to the data object from which the cache is initialized
 );
```

Parameter

pDataObject
[in]Points to the data object from which the cache is to be initialized.

Return Values

S_OK
The cache was filled using the data provided.

E_INVALIDARG
The value for *pDataObject* is not valid.

E_OUTOFMEMORY
The cache could not be initialized due to lack of memory.

OLE_E_NOTRUNNING
The cache is not running.

CACHE_E_NOCACHE_UPDATED
None of the caches were updated.

CACHE_S_SOMECACHES_NOTUPDATED
Only some of the existing caches were updated.

Comments

IOleCache::InitCache is usually used when creating an object from a drag-and-drop operation or from a clipboard paste operation. It fills the cache as needed with presentation data from all the data formats provided by the data object provided on the clipboard or in the drag-and-drop operation. Helper functions like **OleCreateFromData** or **OleCreateLinkFromData** call this method when needed. If a container does not use these helper functions to create compound document objects, it can use **IOleCache::Cache** to set up the cache entries which are then filled by **IOleCache::InitCache**.

See Also

IOleCache::Cache

IOleCache::SetData

Initializes the cache with data in a specified format and on a specified medium.

```
HRESULT SetData(
   FORMATETC * pFormatetc,     //Points to the format of the presentation data to be placed in the cache
   STGMEDIUM * pmedium,        //Points to the storage medium containing the data to be placed in the
                               cache
   BOOL fRelease               //Indicates ownership of the storage medium after this method is
                               completed
  );
```

Parameters

pFormatetc
[in] Points to the format of the presentation data being placed in the cache.

pmedium
[in] Points to the storage medium that contains the presentation data.

fRelease
[in] Indicates ownership of the storage medium after completion of the method. If *fRelease* is TRUE, the cache takes ownership, freeing the medium when it is finished using it. When *fRelease* is FALSE, the caller retains ownership and is responsible for freeing the medium. The cache can only use the storage medium for the duration of the call.

Return Values

S_OK
The cache was filled.

E_OUTOFMEMORY
The cache could not be filled due to a lack of memory.

DV_E_LINDEX
The value is not valid for *pFormatetc->lindex.* Currently, only -1 is supported.

DV_E_FORMATETC
The **FORMATETC** structure is invalid.

DV_E_TYMED
The value is not valid for *pFormatetc->tymed*.

DV_E_DVASPECT
The value is not valid for *pFormatetc->dwAspect*.

OLE_E_BLANK
Uninitialized object.

DV_E_TARGETDEVICE
The object is static and *pFormatetc->ptd* is non-NULL.

STG_E_MEDIUMFULL
The storage medium is full.

Comments

IOleCache::SetData is usually called when an object is created from the Clipboard or through a drag-and-drop operation and Embed Source data is used to create the object.

IOleCache::SetData and **IOleCache::InitCache** are very similar. There are two main differences. The first difference is that while **IOleCache::InitCache** initializes the cache with the any presentation format provided by the data object, **IOleCache::SetData** initializes it with a single format. Second, the **IOleCache::SetData** method ignores the ADVF_NODATA flag while **IOleCache::InitCache** obeys this flag.

A container can use this method to maintain a single aspect of an object, such as the icon aspect of the object.

See Also

IDataObject::SetData, **IOleCache::Cache**

IOleCache::Uncache

Removes a cache connection created in a prior call to **IOleCache::Cache**.

```
HRESULT Uncache(
   DWORD dwConnection        //Specifies the cache connection to remove
  );
```

Parameter

dwConnection

[in]Specifies the cache connection to remove. This non-zero value was returned by **IOleCache::Cache** when the cache was originally established.

Return Values

S_OK

The cache connection was deleted.

OLE_E_NOCONNECTION

No cache connection exists for *dwConnection*.

Comments

The **IOleCache::Uncache** method removes a cache connection that was created in a prior call to **IOleCache::Cache**. It uses the *dwConnection* parameter that was returned by the prior call to **IOleCache::Cache**.

See Also

IOleCache::Cache

IOleCache2

The **IOleCache2** interface allows object clients to selectively update each cache that was created with **IOleCache::Cache**.

When to Implement

The **IOleCache** interface can be implemented by an object handler and an in-process server. However, you typically use or aggregate the OLE-provided implemention.

When to Use

The OLE-provided implementation is an extension to the **IOleCache** interface. The **IOleCache2** interface is used by container applications, object handlers, or in process servers to update one or more of the caches that were created with the **IOleCache::Cache** method. This extended interface was added so client applications can exercise precise control over updates to the caches being maintained.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IOleCache2 Methods	Description
UpdateCache	Updates the specified cache(s).
DiscardCache	Discards cache(s) found in memory.

IOleCache2::DiscardCache

Discards the caches in memory.

```
HRESULT DiscardCache(
   DWORD dwDiscardOptions      //Save data or discard it
  );
```

Parameter

dwDiscardOptions

Indicates whether data is to be saved prior to the discard. Valid values are from the enumeration **DISCARDCACHE**. Containers that have drawn a large object and need to free up memory may want to specify DISCARDCACHE_SAVEIFDIRTY so that the newest presentation is saved for the next time the object must be drawn.

Containers that have activated an embedded object, made some changes, and then called **IOleObject::Close(**OLECLOSE_NOSAVE) to roll back the changes can specify DISCARDCACHE_NOSAVE to ensure that the native and presentation data are not out of synchronization.

Return Values

S_OK
The cache(s) were discarded according to the value specified in *dwDiscardOptions*.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

OLE_E_NOSTORAGE
There is no storage available for saving the data in the cache.

STG_E_MEDIUMFULL
The storage medium is full.

Comments

The **IOleCache2::DiscardCache** method is commonly used to handle low memory conditions by freeing memory currently being used by presentation caches.

Once discarded, the cache will satisfy subsequent **IDataObject::GetData** calls by reverting to disk-based data.

See Also

DISCARDCACHE, **IOleCache**, **IOleCacheControl**

IOleCache2::UpdateCache

Updates specified cache(s). It updates the cache according to the value of a parameter. This method is used when the application needs precise control over caching.

```
HRESULT UpdateCache(
   IDataObject * pDataObject,          //Points to the data object from which the cache is updated
   DWORD grfUpdt,                      //Specifies the type of cache to update
   LPVOID pReserved                    //Reserved, must be NULL
  );
```

Parameters

pDataObject

[in]Points to the data object from which the cache is updated. Object handlers and in process servers typically pass a non-NULL value. A container application usually passes NULL, and the source is obtained from the currently running object.

grfUpdt

Specifies the type of cache to update. The value is obtained by combining values from the following table:

Cache Control Values	Description
UPDFCACHE_NODATACACHE	Updates caches created by using ADVF_NODATA in the call to **IOleCache::Cache**.
UPDFCACHE_ONSAVECACHE	Updates caches created by using ADVFCACHE_ONSAVE in the call to **IOleCache::Cache**.
UPDFCACHE_ONSTOPCACHE	Updates caches created by using ADVFCACHE_ONSTOP in the call to **IOleCache::Cache**.
UPDFCACHE_NORMALCACHE	Dynamically updates the caches (as is normally done when the object sends out OnDataChange notices).
UPDFCACHE_IFBLANK	Updates the cache if blank, regardless of any other flag specified.
UPDFCACHE_ONLYIFBLANK	Updates only caches that are blank.
UPDFCACHE_ IFBLANKORONSAVECACHE	The equivalent of using an OR operation to combine UPDFCACHE_IFBLANK and UPDFCACHE_ONSAVECACHE.
UPDFCACHE_ALL	Updates all caches.
UPDFCACHE_ ALLBUTNODATACACHE	Updates all caches except those created with ADVF_NODATA in the call to **IOleCache::Cache**. Thus, you can control updates to the caches created with the ADVF_NODATA flag and only update these caches explicitly.

pReserved

Reserved for future use; must be NULL.

Return Values

S_OK
The cache(s) were updated according to the value specified in *grfUpdt*.

E_OUTOFMEMORY
The cache(s) were not updated due to a lack of memory.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

OLE_E_NOTRUNNING
The specified *pDataObject* is not running.

CACHE_E_NOCACHE_UPDATED
None of the caches were updated.

CACHE_S_SOMECACHES_NOTUPDATED
Some of the caches were updated.

See Also

IOleCache, **IOleCacheControl**

IOleCacheControl

The **IOleCacheControl** interface provides proper maintenance of caches. It maintains the caches by connecting the running object's **IDataObject** implementation to the cache allowing the cache to receive notifications from the running object.

When to Implement

The OLE-provided implementation is used by most handlers and in process servers.

When to Use

Object handlers and in process servers use this interface internally to connect the cache part of the handler to the **IDataObject** implementation of the running object. Container applications have no need for this interface; they use **IRunnableObject** or **OleRun** instead.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IOleCacheControl Methods	Description
OnRun	Notifies the cache when the data object is running so the cache object can establish advise sinks as needed.
OnStop	Notifies the cache to terminate any existing advise sinks.

IOleCacheControl::OnRun

Notifies the cache that the data source object has now entered its running state.

HRESULT OnRun(
  DATAOBJECT * *pDataObject* //Points to the object that is now running
 );

Parameter

pDataObject
 [in]Points to the object that is entering the running state.

Return Values

S_OK
 The cache was notified and *pDataObject* is valid.

E_OUTOFMEMORY
 The cache was not notified because the system ran out of memory.

E_INVALIDARG
 One or more invalid arguments.

E_UNEXPECTED
 An unexpected error occurred.

Comments

When **IOleCacheControl::OnRun** is called, the cache sets up advisory connections as necessary with the source data object so it can receive notifications. The advise sink created between the running object and the cache is destroyed when **IOleCacheControl::OnStop** is called.

Some object handlers or in process servers might use the cache in a passive manner and not call **IOleCacheControl::OnRun**. These applications will need to call **IOleCache2::UpdateCache**, **IOleCache::InitCache**, or **IOleCache::SetData** to fill the cache when necessary to ensure the cache gets updated.

IOleCacheControl::OnRun does not create a reference count on *pDataObject.* It is the responsibility of the caller of **OleRun** to ensure that the lifetime of *pDataObject* lasts until **OnStop** is called. As a result, the caller must be holding a pointer to *pDataObject*.

See Also

IOleCache2::UpdateCache, **IOleCacheControl::OnStop**

IOleCacheControl::OnStop

Notifies the cache it should terminate any existing connection previously given to it by using **IOleCacheControl::OnRun**. No indication is given as to whether a connection existed or not.

HRESULT OnStop( );

Return Values

S_OK
  The cache was notified and the advise sink was successfully removed.

E_OUTOFMEMORY
  The advise sink was not removed because the system ran out of memory.

E_UNEXPECTED
  An unexpected error occurred.

Comments

The data advisory sink between the running object and the cache is destroyed as part of calling **IOleCacheControl::OnStop**.

See Also

IOleCacheControl::OnRun

IOleClientSite

The **IOleClientSite** interface is the primary means by which an embedded object obtains information about the location and extent of its display site; its moniker; its user interface; and other resources provided by its container. An object application calls **IOleClientSite** to request services from the container. A container must provide one instance of **IOleClientSite** for every compound-document object it contains.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IOleClientSite Methods	Description
SaveObject	Saves embedded object.
GetMoniker	Requests object's moniker.
GetContainer	Requests pointer to object's container.
ShowObject	Asks container to display object.
OnShowWindow	Notifies container when object becomes visible or invisible
RequestNewObjectLayout	Asks container to resize display site.

IOleClientSite::GetContainer

Returns a pointer to the container's **IOleContainer** interface.

```
HRESULT GetContainer(
  LPOLECONTAINER FAR* ppContainer        //Points to where the IOleContainer pointer should be returned
 );
```

Parameter

ppContainer
[out] Points to where the object's **IOleContainer** interface pointer is to be returned. If an error is returned, this parameter must be set to NULL.

Return Values

S_OK
The pointer to the container's **IOleContainer** interface was successfully returned.

OLE_E_NOT_SUPPORTED
Client site is in OLE 1 container.

E_NOINTERFACE
The container does not implement the **IOleContainer** interface.

Comments

If a container supports links to its embedded objects, implementing **IOleClientSite::GetContainer** enables link clients to enumerate the container's objects and recursively traverse a containment hierarchy. This method is optional but recommended for all containers that expect to support links to their embedded objects.

Link clients can traverse a hierarchy of compound-document objects by recursively calling **IOleClientSite::GetContainer** to get a pointer to the link source's container; followed by **IOleContainer::QueryInterface** to get a pointer to the container's **IOleObject** interface and; finally, **IOleObject::GetClientSite** to get the container's client site in its container.

Simple containers that do not support links to their embedded objects probably do not need to implement this method. Instead, they can return E_NOINTERFACE and set *ppContainer* to NULL.

IOleClientSite::GetMoniker

Returns a moniker to an object's client site. An object can force the assignment of its own or its container's moniker by specifying a value for *dwAssign*.

```
HRESULT GetMoniker(
    DWORD dwAssign,               //Value specifying how moniker is assigned
    DWORD dwWhichMoniker,         //Value specifying which moniker is assigned
    IMoniker ** ppmk              //Pointer to storage of returned moniker
  );
```

Parameters

dwAssign
: [in] Specifies whether to get a moniker only if one has already exists, force assignment of a moniker, create a temporary moniker, or remove a moniker that has been assigned. In practice you will usually request that the container force assignment of the moniker. Values defining these choices are contained in the enumeration **OLEGETMONIKER**.

dwWhichMoniker
: [in] Specifies whether to return the container's moniker, the object's moniker relative to the container, or the object's full moniker. In practice, you will usually request the object's full moniker. Values defining these choices are contained in the enumeration **OLEWHICHMK**.

ppmk
: [out] Points to where to return the object moniker. If an error is returned, this parameter must be set to NULL. Each time a container receives a call to **IOleClientSite::GetMoniker**, it must increase the reference count on the pointer it returns. It is the caller's responsibility to call Release when it is done with the pointer.

Return Values

S_OK
: Requested moniker returned successfully.

E_FAIL
: An unspecified error occurred.

E_UNEXPECTED
: A serious, unexpected failure has occurred.

E_NOTIMPL
: This container cannot assign monikers to objects. This is the case with OLE 1 containers.

Comments

Containers implement **IOleClientSite::GetMoniker** as a way of passing out monikers for their embedded objects to clients wishing to link to those objects.

When a link is made to an embedded object or to a pseudo-object within it (a range of cells in a spreadsheet; for example), the object needs a moniker to construct the composite moniker indicating the source of the link. If the embedded object does not already have a moniker, it can call **IOleClientSite::GetMoniker** to request one.

Every container that expects to contain links to embeddings should support **IOleClientSite::GetMoniker** to give out OLEWHICHMK_CONTAINER, thus enabling link tracking when the link client and link source files move, but maintain the same relative position.

An object must not persistently store its full moniker or its container's moniker, because these can change while the object is not loaded. For example, either the container or the object could be renamed, in which event, storing the container's moniker or the object's full moniker would make it impossible to for a client to track a link to the object.

In some very specialized cases, an object may no longer need a moniker previously assigned to it and may wish to have it removed as an optimization. In such cases, the object can call **IOleClientSite::GetMoniker** with OLEGETMONIKER_UNASSIGN to have the moniker removed.

See Also

IOleObject::GetMoniker, **IOleObject::SetMoniker**

IOleClientSite::OnShowWindow

Notifies a container when an embedded object's window is about to become visible or invisible. This method does not apply to an object that is activated in place and therefore has no window separate from that of its container.

```
HRESULT OnShowWindow(
    BOOL fShow        //Value indicating if window is becoming visible
  );
```

Parameter

fShow
 [in] Indicates whether an object's window is open (TRUE) or closed (FALSE).

Return Value

S_OK
 Shading or hatching has been successfully added or removed

Comments

An embedded object calls **IOleClientSite::OnShowWindow** to keep its container informed when the object is open in a window. This window may or may not be currently visible to the end user. The container uses this information to shade the object's client site when the object is displayed in a window, and to remove the shading when the object is not. A shaded object, having received this notification, knows that it already has an open window and therefore can respond to being double-clicked by bringing this window quickly to the top, instead of launching its application in order to obtain a new one.

IOleClientSite::RequestNewObjectLayout

Asks container to allocate more or less space for displaying an embedded object.

HRESULT RequestNewObjectLayout();

Return Values

S_OK
Request for new layout succeeded.

E_NOTIMPL
Client site does not support requests for new layout.

Comments

Currently, there is no standard mechanism by which a container can negotiate *how much* room an object would like. When such a negotiation is defined, responding to this method will be optional for containers.

IOleClientSite::SaveObject

Saves the object associated with the client site. This function is synchronous; by the time it returns, the save will be completed.

HRESULT SaveObject();

Parameter

HRESULT **SaveObject(***void***)**

Return Values

S_OK
  The object was saved.

E_FAIL
  The object was not saved.

Comments

An embedded object calls **IOleClientSite::SaveObject** to ask its container to save it to persistent storage when an end user chooses the File Update or Exit commands. The call is synchronous, meaning that by the time it returns, the save operation will be completed.

Calls to **IOleClientSite::SaveObject** occur in most implementations of **IOleObject::Close**. Normally, when a container tells an object to close, the container passes a flag specifying whether the object should save itself before closing, prompt the user for instructions, or close without saving itself. If an object is instructed to save itself, either by its container or an end user, it calls **IOleClientSite::SaveObject** to ask the container application to save the object's contents before the object closes itself. If a container instructs an object not to save itself, the object should not call **SaveObject**.

See Also

IOleObject::Close

IOleClientSite::ShowObject

Tells the container to position the object so it is visible to the user. This method ensures that the container itself is visible and not minimized.

HRESULT ShowObject();

Return Values

S_OK
  Container has tried to make the object visible.

OLE_E_NOT_SUPPORTED
  Client site is in an OLE 1 container.

Comments

After a link client binds to a link source, it commonly calls **IOleObject::DoVerb** on the link source, usually requesting the source to perform some action requiring that it display itself to the user. As part of its implementation of **DoVerb**, the link source can call **IOleClientSite::ShowObject**, which forces the client to show the link source as best it can. If the link source's container is itself an embedded object, it will recursively invoke **IOleClientSite::ShowObject** on its own container.

Having called the **ShowObject** method, a link source has no guarantee of being appropriately displayed because its container may not be able to do so at the time of the call. The **ShowObject** method does not guarantee visibility, only that the container will do the best it can.

See Also

IOleObject::DoVerb

IOleContainer

The **IOleContainer** interface is used to enumerate objects in a compound document or lock a container in the running state. Container and object applications both implement this interface.

When to Implement

Applications that support links and links to embedded objects implement this interface to provide object enumeration, name parsing, and silent updates of link sources. Simple, nonlinking containers do not need to implement **IOleContainer** if it is useful mainly to support links to embedded objects.

When to Use

Call **IOleContainer** to enumerate the objects in a compound document or to lock a container so that silent updates of link sources can be carried out safely.

Many applications inherit the functions of **IOleContainer** by implementing **IOleItemContainer**, which is used to bind item monikers.

Methods in VTable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IParseDisplayName Method	**Description**
ParseDisplayName	Parses object's display name to form moniker.

IOleContainer Methods	**Description**
EnumObjects	Enumerates objects in a container.
LockContainer	Keeps container running until explicitly released.

See Also

IOleItemContainer, **IParseDisplayName**

IOleContainer::EnumObjects

Enumerates objects in the current container.

```
HRESULT EnumObjects(
   DWORD grfFlags,              //Value specifying what is to be enumerated
   IEnumUnknown **ppenum        //Ptr to storage of enumeration
  );
```

Parameters

grfFlags

[in] This value specifies which objects in a container are to be enumerated, as defined in the enumeration **OLECONTF**.

ppenum

[out] Points to an address where the enumerator, of type **IEnumUnknown**, should be returned. If an error is returned, this parameter must be set to NULL. Each time a container receives a call to **EnumObjects**, it must increase the reference count on the pointer the method returns. It is the caller's responsibility to call **IUnknown::Release** when it is done with the pointer.

Return Values

S_OK

Enumerator successfully returned.

E_FAIL

An unspecified error occurred.

E_NOTIMPL

Object enumeration not supported.

Comments

A container should implement **EnumObjects** to enable programmatic clients to find out what objects it holds. This method, however, is not called in standard linking scenarios.

See Also

IEnumUnknown, **IOleItemContainer**, **OLECONTF**

IOleContainer::LockContainer

Keeps an embedded object's container running.

```
HRESULT LockContainer(
    BOOL fLock        //Value indicating lock or unlock
  );
```

Parameter

fLock
[in] Specifies whether to lock (TRUE) or unlock (FALSE) a container.

Return Values

S_OK
Container was locked successfully.

E_FAIL
An unspecified error occurred.

E_OUTOFMEMORY
Container could not be locked due to lack of memory.

Comments

An embedded object calls **IOleContainer::LockContainer** to keep its container running when the object has link clients that require an update. If an end-user selects File Close from the container's menu, however, the container ignores all outstanding **LockContainer** locks and closes the document anyway.

Notes to Callers

When an embedded object changes from the loaded to the running state, it should call **IOleContainer::LockContainer** with the *fLock* parameter set to TRUE. When the embedded object shuts down (transitions from running to loaded), it should call **IOleContainer::LockContainer** with the *fLock* parameter set to FALSE.

Each call to **LockContainer** with *fLock* set to TRUE must be balanced by a call to **LockContainer** with *fLock* set to FALSE. Object applications typically need not call **LockContainer**; the default handler makes these calls automatically for object applications implemented as .EXEs as the object makes the transition to and from the running state. Object applications not using the default handler, such as DLL object applications, must make the calls directly.

An object should have no strong locks on it when it registers in the Running Object Table, but it should be locked as soon as the first external client connects to it. Therefore, following registration of the object in the Running Object Table, object handlers and DLL object applications, as part of their implementation of **IRunnableObject::Run**, should call **IOleContainer::LockContainer(**TRUE**)** to lock the object.

Notes to Implementors

The container must keep track of whether and how many calls to **LockContainer(**TRUE**)** have been made. To increment or decrement the reference count, **IOleContainer::LockContainer** calls **CoLockObjectExternal** with a flag set to match *fLock*.

See Also

CoLockObjectExternal, **IRunnableObject::Run**

IOleInPlaceActiveObject

The **IOleInPlaceActiveObject** interface provides a direct channel of communication between an in-place object and the associated application's outer-most frame window and the document window within the application that contains the embedded object. The communication involves the translation of messages, the state of the frame window (activated or deactivated), and the state of the document window (activated or deactivated). It also informs the object when its needs to resize its borders, and manages modeless dialog boxes.

When to Implement

This interface is implemented by object applications in order to provide support their objects while they are active in place.

When to Use

These methods are used by the in-place object's top-level container to manipulate objects while they are active.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns a pointer to a specified interface.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IOleWindow Methods	Description
GetWindow	Gets a window handle.
ContextSensitiveHelp	Controls enabling of context sensitive help.

IOleInPlaceActiveObject Methods	Description
TranslateAccelerator	Translates messages.
OnFrameWindowActivate	State of container's top-level frame.
OnDocWindowActivate	State of container document window.
ResizeBorder	Alert object of need to resize border space.
EnableModeless	Enable or disable modeless dialog boxes.

See Also

IOleWindow

IOleInPlaceActiveObject::EnableModeless

Enables or disables modeless dialog boxes when the container creates or destroys a modal dialog box.

```
HRESULT EnableModeless(
    BOOL fEnable          //Enable or disable modeless
  );
```

Parameter

fEnable
[in] Specifies TRUE to enable modeless dialog box windows; FALSE to disable them.

Return Value

S_OK
The method completed successfully.

Comments

Notes to Callers

IOleInPlaceActiveObject::EnableModeless is called by the top-level container to enable and disable modeless dialog boxes that the object displays. In order for the container to display a modal dialog box, it must first call the **IOleInPlaceActiveObject::EnableModeless** method, specifying FALSE to disable the object's modeless dialog box windows. When the container is through displaying its modal dialog box, it calls **IOleInPlaceActiveObject::EnableModeless**, specifying TRUE to reenable the object's modeless dialog boxes.

See Also

IOleInPlaceFrame::EnableModeless

IOleInPlaceActiveObject::OnDocWindowActivate

Notifies the active in-place object when the container's document window is activated or deactivated.

```
HRESULT OnDocWindowActivate(
    BOOL fActivate         //State of MDI child document window
  );
```

Parameter

fActivate
[in] Indicates the state of the MDI child document window. It is TRUE if the window is in the act of activating; FALSE if it is in the act of deactivating.

Return Value

S_OK
The method completed successfully.

Comments

Notes to Callers

Call **IOleInPlaceActiveObject::OnDocWindowActivate** when the MDI child document window is activated or deactivated and the object is currently the active object for the document.

Notes to Implementors

You should include code in this method that installs frame-level tools during object activation. These tools include the shared composite menu and/or optional toolbars and frame adornments. You should then take focus. When deactivating, the object should remove the frame-level tools. Note that if you do not call **IOleInPlaceUIWindow::SetBorderSpace** with *pborderwidths* set to NULL, you can avoid having to renegotiate border space.

Note While executing **IOleInPlaceActiveObject::OnDocWindowActivate**, do not make calls to the Windows **PeekMessage** or **GetMessage** functions, or a dialog box. Doing so may cause the system to deadlock. There are further restrictions on which OLE interface methods and functions can be called from within **OnDocWindowActivate**.

See Also

PeekMessage, **GetMessage** in Win32

IOleInPlaceActiveObject::OnFrameWindowActivate

Notifies the object when the container's top-level frame window is activated or deactivated.

```
HRESULT OnFrameWindowActivate(
   BOOL fActivate          //State of container's top-level window
  );
```

Parameter

fActivate
[in] Indicates the state of the container's top-level frame window. TRUE if the window is activating; FALSE if it is deactivating.

Return Value

S_OK
The method notified the object successfully.

Comments

Notes to Callers

The container must call **IOleInPlaceActiveObject::OnFrameWindowActivate** when the container's top-level frame window is either being activated or deactivated and the object is the current, active object for the frame.

Note While executing **IOleInPlaceActiveObject::OnFrameWindowActivate**, do not make calls to the Windows **PeekMessage** or **GetMessage** functions, or a dialog box. Doing so may cause the system to deadlock. There are further restrictions on which OLE interface methods and functions can be called from within **OnFrameWindowActivate**.

See Also

PeekMessage, **GetMessage** in Win32

IOleInPlaceActiveObject::ResizeBorder

Alerts the object that it needs to resize its border space.

```
HRESULT ResizeBorder(
  LPCRECT prcBorder,                //New outer rectangle for border space
  IOleInPlaceUIWindow pUIWindow,    //Frame or document window border change
  BOOL fFrameWindow                 //Frame window object is or isn't calling ResizeBorder
 );
```

Parameters

prcBorder
[in] Points to a **RECT** structure containing the new outer rectangle within which the object can request border space for its tools.

pUIWindow
[in] Points to the frame or document window object whose border has changed.

fFrameWindow
[in] Specifies TRUE if the frame window object is calling **ResizeBorder**; otherwise, FALSE.

Return Values

S_OK
The method alerted the object successfully.

E_INVALIDARG
One or more arguments are invalid.

E_OUTOFMEMORY
Out of memory.

E_UNEXPECTED
An unexpected error occurred.

Comments

Notes to Callers

IOleInPlaceActiveObject::ResizeBorder is called by the top-level container's document or frame window object when the border space allocated to the object should change. Because the active in-place object is not informed about which window has changed (the frame- or document-level window), **IOleInPlaceActiveObject::ResizeBorder** must be passed the pointer to the window's **IOleInPlaceUIWindow** interface.

Notes to Implementors

In most cases, resizing only requires that you grow, shrink, or scale your ojbect's frame adornments. However, for more complicated adornments, you may be required to renegotiate for border space with calls to **IOleInPlaceUIWindow::RequestBorderSpace** and **IOleInPlaceUIWindow::SetBorderSpace**.

Note While executing **IOleInPlaceActiveObject::ResizeBorder**, do not make calls to the Windows **PeekMessage** or **GetMessage** functions, or a dialog box. Doing so may cause the system to deadlock. There are further restrictions on which OLE interface mehtods and functions can be called from within **ResizeBorder**.

See Also

IOleInPlaceUIWindow::GetBorder

PeekMessage, GetMessage in Win32

IOleInPlaceActiveObject::TranslateAccelerator

Processes menu accelerator-key messages from the container's message queue. This method should only be used for objects created by a DLL object application.

```
HRESULT TranslateAccelerator(
    LPMSG lpmsg        //Points to message that may need translating
  );
```

Parameter

lpmsg
[in] Points to the message that might need to be translated.

Return Values

S_OK
The message was translated successfully.

S_FALSE
The message was not translated.

E_INVALIDARG
One or more arguments are invalid.

E_OUTOFMEMORY
Out of memory.

E_UNEXPECTED
An unexpected error occurred.

Comments

Notes to Callers

Active in-place objects must always be given the first chance at translating accelerator keystrokes. You can provide this opportunity by calling **IOleInPlaceActiveObject::TranslateAccelerator** from your container's message loop before doing any other translation. You should apply your own translation only when this method returns S_FALSE.

If you call **IOleInPlaceActiveObject::TranslateAccelerator** for an object that is not created by a DLL object application, the default object handler returns S_FALSE.

Notes to Implementors

An object created by an EXE object application gets keystrokes from its own message pump, so the container does not get those messages.

If you need to implement this method, you can do so by simply wrapping the call to the Window's **TranslateAccelerator** function.

See Also

OleTranslateAccelerator

TranslateAccelerator in Win32

IOleInPlaceFrame

The **IOleInPlaceFrame** interface controls the container's top-level frame window. This control involves allowing the container to insert its menu group into the composite menu, install the composite menu into the appropriate window frame, and remove the container's menu elements from the composite menu. It sets and displays status text relevant to the in-place object. It also enables or disables the frame's modeless dialog boxes, and translates accelerator keystrokes intended for the container's frame.

When to Implement

You will need to implement this interface if you are writing a container application that will be participating in in-place activation.

When to Use

This interface is used by object applications to control the display and placement of composite menus, keystroke accelerator translation, context-sensitive help mode, and modeless dialog boxes.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns a pointer to a specified interface.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IOleWindow Methods	Description
GetWindow	Gets a window handle.
ContextSensitiveHelp	Controls enabling of context-sensitive help.

IOleInPlaceUIWindow Methods	Description
GetBorder	Translates messages.
RequestBorderSpace	State of container's top-level frame.
SetBorderSpace	State of container document window.
SetActiveObject	Alert object of need to resize border space.

IOleInPlaceFrame Methods	Description
InsertMenus	Allows container to insert menus.
SetMenu	Adds a composite menu to window frame.
RemoveMenus	Removes a container's menu elements.
SetStatusText	Sets and displays status text about.
EnableModeless	Enables or disables modeless dialog.
TranslateAccelerator	Translates keystrokes.

See Also

IOleWindow, IOleInPlaceUIWindow

IOleInPlaceFrame::EnableModeless

Enables or disables a frame's modeless dialog boxes.

```
HRESULT EnableModeless(
    BOOL fEnable          //Enable or disable modeless dialog
  );
```

Parameter

fEnable
[in] Specifies whether the modeless dialog boxes are to be enabled by specifying TRUE or disabled by specifying FALSE.

Return Values

S_OK
The dialog box was either enabled or disabled successfully, depending on the value for *fEnable*.

E_UNEXPECTED
An unexpected error occurred.

Comments

Notes to Callers

The active in-place object calls **IOleInPlaceFrame::EnableModeless** to enable or disable modeless dialog boxes that the container might be displaying. To display a modal dialog box, the object first calls **IOleInPlaceFrame::EnableModeless** , specifying FALSE to disable the container's modeless dialog box windows. After completion, the object calls **IOleInPlaceFrame::EnableModeless** specifying TRUE to reenable them.

Notes to Implementors

You should track the value of **EnableModeless** and check it before displaying a dialog box.

See Also

IOleInPlaceActiveObject::EnableModeless

IOleInPlaceFrame::InsertMenus

Allows the container to insert its menu groups into the composite menu to be used during the in-place session.

```
HRESULT InsertMenus(
  HMENU hmenuShared,                          //Handle to empty menu
  LPOLEMENUGROUPWIDTHS lpMenuWidths           //OLEMENUGROUPWIDTHS array
 );
```

Parameters

hmenuShared
[in] Specifies a handle to an empty menu.

lpMenuWidths
[in, out] Points to an **OLEMENUGROUPWIDTHS** array of six LONG values. The container fills in elements 0, 2, and 4 to reflect the number of menu elements it provided in the File, View, and Window menu groups.

Return Values

S_OK
The menu groups were inserted successfully.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

Comments

Notes to Callers

This method is called by object applications when they are first being activated. They call it in order to insert their menus into the frame-level user interface.

The object application asks the container to add its menus to the menu specified in *hmenuShared* and to set the group counts in the **OLEMENUGROUPWIDTHS** array pointed to by *lpMenuWidths*. The object application then adds its own menus and counts. Objects can call **IOleInPlaceFrame::InsertMenus** as many times as necessary to build up the composite menus. The container should use the initial menu handle associated with the composite menu for all menu items in the drop-down menus.

IOleInPlaceFrame::RemoveMenus

Gives the container a chance to remove its menu elements from the in-place composite menu.

HRESULT RemoveMenus(
  HMENU *hmenuShared*   //Handle to in-place composite menu
 );

Parameter

hmenuShared
[in] Specifies a handle to the in-place composite menu that was constructed by calls to **IOleInPlaceFrame::InsertMenus** and the Windows **InsertMenu** function.

Return Values

S_OK
The method completed successfully.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

Comments

The object should always give the container a chance to remove its menu elements from the composite menu before deactivating the shared user interface.

Notes to Callers

Called by the object application while it is being UI-deactivated in order to remove its menus.

See Also

IOleInPlaceFrame::SetMenu

InsertMenu in Win32

IOleInPlaceFrame::SetMenu

Installs the composite menu in the window frame containing the object being activated in place.

```
HRESULT SetMenu(
   HMENU hmenuShared,          //Handle to composite menu
   HOLEMENU holemenu,          //Handle to menu descriptor
   HWND hwndActiveObject       //Handle to object's window
  );
```

Parameters

hmenuShared
[in] Specifies a handle to the composite menu constructed by calls to **IOleInPlaceFrame::InsertMenus** and the Windows **InsertMenu** function.

holemenu
[in] Specifies the handle to the menu descriptor returned by the **OleCreateMenuDescriptor** function.

hwndActiveObject
[in] Specifies a handle to a window owned by the object and to which menu messages, commands, and accelerators are to be sent.

Return Values

S_OK
The method completed successfully.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

Comments

Notes to Callers

The object calls **IOleInPlaceFrame::SetMenu** to ask the container to install the composite menu structure set up by calls to **IOleInPlaceFrame::InsertMenus**.

Notes to Implementors

An SDI container's implementation of this method should call the Windows **SetMenu** function. An MDI container should send a WM_MDISETMENU message, using *hmenuShared* as the menu to install. The container should call **OleSetMenuDescriptor** to install the OLE dispatching code.

When deactivating, the container *must* call **IOleInPlaceFrame::SetMenu** specifying NULL to remove the shared menu. This is done to help minimize window repaints. The container should also call **OleSetMenuDescriptor** specifying NULL to unhook the dispatching code. Finally, the object application calls **OleDestroyMenuDescriptor** to free the data structure.

Note While executing **IOleInPlaceFrame::SetMenu**, do not make calls to the Windows **PeekMessage** or **GetMessage** functions, or a dialog box. Doing so may cause the system to deadlock. There are further restrictions on which OLE interface methods and functions can be called from within **SetMenu**.

See Also

IOleInPlaceFrame::InsertMenus, **OleSetMenuDescriptor**, **OleDestroyMenuDescriptor**

PeekMessage, **GetMessage** in Win32

IOleInPlaceFrame::SetStatusText

Sets and displays status text about the in-place object in the container's frame window status line.

```
HRESULT SetStatusText(
  LPCOLESTR pszStatusText       //Points to message to display
 );
```

Parameter

pszStatusText
[in] Points to a null-terminated character string containing the message to display.

Return Values

S_OK
The text was displayed.

S_TRUNCATED
Some text was displayed but the message was too long and was truncated.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

E_FAIL
The text could not be displayed.

Comments

Notes to Callers

You should call **SetStatusText** when you need to ask the container to display object text in its frame's status line, if it has one. Because the container's frame window owns the status line, calling **IOleInPlaceFrame::SetStatusText** is the *only* way an object can display status information in the container's frame window. If the container refuses the object's request, the object application can, however, negotiate for border space to display its own status window.

Note When switching between menus owned by the container and the in-place active object, the status bar text is not reflected properly if the object does not call the container's **IOleInPlaceFrame::SetStatusText** method. For example, if, during an in-place session, the user were to select the File menu, the status bar would reflect the action that would occur if the user selected this menu. If the user then selects the Edit menu (which is owned by the in-place object), the status bar text would not change unless the **IOleInPlaceFrame::SetStatusText** happened to be called. This is because there is no way for the container to recognize that one of the object's menus has been made active because all the messages that the container would trap are now going to the object.

Notes to Implementors

To avoid potential problems, all objects being activated in place should process the WM_MENUSELECT message and call **IOleInPlaceFrame::SetStatusText** − even if the object does not usually provide status information (in which case the object can just pass a NULL string for the requested status text).

Note While executing **IOleInPlaceFrame::SetStatusText**, do not make calls to the Windows **PeekMessage** or **GetMessage** functions, or a dialog box. Doing so may cause the system to deadlock. There are further restrictions on which OLE interface methods and functions can be called from within **GerBorder**.

See Also

PeekMessage, **GetMessage** in Win32

IOleInPlaceFrame::TranslateAccelerator

Translates accelerator keystrokes intended for the container's frame while an object is active in place.

```
HRESULT TranslateAccelerator(
   LPMSG lpmsg,          //Points to MSG structure
   WORD wID              //Command Identifier value
  );
```

Parameters

lpmsg
[in] Points to an **MSG** structure containing the keystroke message.

wID
[in] Contains the command identifier value corresponding to the keystroke in the container-provided accelerator table. Containers should use this value instead of translating again.

Return Values

S_OK
The keystroke was used.

S_FALSE
The keystroke was not used.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

Comments

Notes to Callers

The **IOleInPlaceFrame::TranslateAccelerator** method is called indirectly by **OleTranslateAccelerator** when a keystroke accelerator intended for the container (frame) is received.

Notes to Implementors

The container application should perform its usual accelerator processing, or use *wID* directly, and then return, indicating whether the keystroke accelerator was processed. If the container is an MDI application and the Windows **TranslateAccelerator** call fails, the container can call the Windows **TranslateMDISysAccel** function, just as it does for its usual message processing.

In-place objects should be given first chance at translating accelerator messages. However, because objects implemented by DLL object applications do not have their own message pump, they receive their messages from the container's message queue. To ensure that the object has first chance at translating messages, a container should always call **IOleInPlaceActiveObject::TranslateAccelerator** before doing its own accelerator translation. Conversely, an executable object application should call **OleTranslateAccelerator** after calling **TranslateAccelerator**, calling **TranslateMessage** and **DispatchMessage** only if both translation functions fail.

Note You should define accelerator tables for containers so they will work properly with object applications that do their own accelerator keystroke translations. Tables should be defined as follows:

```
"char", wID, VIRTKEY, CONTROL
```

This is the most common way to describe keyboard accelerators. Failure to do so can result in keystrokes being lost or sent to the wrong object during an in-place session.

See Also

OleTranslateAccelerator, **IOleInPlaceActiveObject::TranslateAccelerator**

TranslateAccelerator, **TranslateMessage**, **DispatchMessage**, **TranslateMDISysAccel** in Win32

IOleInPlaceObject

The **IOleInPlaceObject** interface manages the activation and deactivation of in-place objects, and determines how much of the in-place object should be visible.

You can obtain a pointer to **IOleInPlaceObject** by calling **QueryInterface** on **IOleObject**.

When to Implement

You must implement this interface if you are writing an object application that will participate in in-place activation.

When to Use

Used by an object's immediate container to activate or deactivate the object.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns a pointer to a specified interface.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IOleWindow Methods	Description
GetWindow	Gets a window handle.
ContextSensitiveHelp	Controls enabling of context sensitive help.

IOleInPlaceObject Methods	Description
InPlaceDeactivate	Deactivate active in-place object.
UIDeactivate	Deactivate and remove UI of active object.
SetObjectRects	Portion of in-place object to be visible.
ReactivateAndUndo	Reactivate previously deactivated object.

See Also

IOleObject, IOleWindow

IOleInPlaceObject::InPlaceDeactivate

Deactivates an active in-place object and discards the object's undo state.

HRESULT InPlaceDeactivate();

Return Values

S_OK
The object was successfully deactivated.

E_UNEXPECTED
An unexpected error occurred.

Comments

Notes to Callers

This method is called by an active object's immediate container to deactivate the active object and discard its undo state.

Notes to Implementors

On return from **IOleInPlaceObject::InPlaceDeactivate**, the object discards its undo state. The object application should not shut down immediately after this call. Instead, it should wait for an explicit call to **IOleObject::Close** or for the object's reference count to reach zero.

Before deactivating, the object application should give the container a chance to put its user interface back on the frame window by calling **IOleInPlaceSite::OnUIDeactivate**.

If the in-place user interface is still visible during the call to **InPlaceDeactivate**, the object application should call its own **IOleInPlaceObject::UIDeactivate** method to hide the user interface. The in-place user interface can be optionally destroyed during calls to **IOleInPlaceObject::UIDeactivate** and **IOleInPlaceObject::InPlaceDeactivate**. But if the user interface has not already been destroyed when the container calls **IOleObject::Close**, then it *must* be destroyed during the call to **IOleObject::Close**.

During the call to **IOleObject::Close**, the object should check to see whether it is still active in place. If so, it should call **InPlaceDeactivate**.

See Also

IOleInPlaceSite::OnInPlaceDeactivate, **IOleInPlaceSite::OnUIDeactivate**, **IOleObject::Close**

IOleInPlaceObject::ReactivateAndUndo

Reactivates a previously deactivated object, undoing the last state of the object.

HRESULT ReactivateAndUndo();

Return Values

S_OK
The object was successfully reactivated.

E_NOTUNDOABLE
Called when the Undo state is not available.

E_INVALIDARG
One or more arguments are invalid.

E_OUTOFMEMORY
Out of memory.

E_UNEXPECTED
An unexpected error occurred.

Comments

If the user chooses the Undo command before the Undo state of the object is lost, the object's immediate container calls **IOleInPlaceObject::ReactivateAndUndo** to activate the user interface, carry out the Undo operation, and return the object to the active state.

IOleInPlaceObject::SetObjectRects

Indicates how much of the in-place object is visible.

```
HRESULT SetObjectRects(
   LPCRECT lprcPosRect,       //The position of the in-place object
   LPCRECT lprcClipRect       //The outer rectangle containing the in-place object's position rectangle
  );
```

Parameters

lprcPosRect
[in] Points to the rectangle containing the position of the in-place object using the client coordinates of its parent window.

lprcClipRect
[in] Points to the outer rectangle containing the in-place object's position rectangle (*PosRect*). This rectangle is relative to the client area of the object's parent window.

Return Values

S_OK
The operation successfully indicated the rectangle.

E_INVALIDARG
One or more arguments are invalid.

E_OUTOFMEMORY
Out of memory.

E_UNEXPECTED
An unexpected error occurred.

Comments

It is possible for *lprcClipRect* to change without *lprcPosRect* changing.

The size of an in-place object's rectangle is *always* calculated in pixels. This is different from other OLE object's visualizations, which are in HIMETRIC.

Note While executing **IOleInPlaceObject::SetObjectRects**, do not make calls to the Windows **PeekMessage** or **GetMessage** functions, or a dialog box. Doing so may cause the system to deadlock. There are further restrictions on which OLE interface mehtods and functions can be called form within **SetObjectRects**.

Notes to Callers

The container should call **IOleInPlaceObject::SetObjectRects** whenever the window position of the in-place object and/or the visible part of the in-place object changes.

Notes to Implementors

The object must size its in-place window to match the intersection of *lprcPosRect* and *lprcClipRect*. The object must also draw its contents into the object's in-place window so that proper clipping takes place.

The object should compare its width and height with those provided by its container (conveyed through *lprcPosRect*). If the comparison does not result in a match, the container is applying scaling to the object. The object must then decide whether it should continue the in-place editing in the scale/zoom mode or deactivate.

See Also

IOleInPlaceSite::OnPosRectChange

PeekMessage, **GetMessage** in Win32

IOleInPlaceObject::UIDeactivate

Deactivates and removes the user interface that supports in-place activation.

HRESULT UIDeactivate();

Return Values

S_OK
The in-place UI was deactivated and removed.

E_UNEXPECTED
An unexpected error occurred.

Notes to Callers

This method is called by the object's immediate container when, for example, the user has clicked in the client area outside the object.

If the container has called **IOleInPlaceObject::UIDeactivate**, it should later call **IOleInPlaceObject::InPlaceDeactivate** to properly clean up resources. The container can assume that stopping or releasing the object cleans up resources if necessary. The object must be prepared to do so if **IOleInPlaceObject::InPlaceDeactivate** has not been called. but either **IOleInPlaceObject::UIDeactivate** or **IOleObject::Close** has been called.

Notes to Implementors

Resources such as menus and windows can be either cleaned up or kept in a hidden state until your object is completely deactivated by calls to either **IOleInPlaceObject::InPlaceDeactivate** or **IOleObject::Close**. The object application must call **IOleInPlaceSite::OnUIDeactivate** before doing anything with the composite menus so that the container can first be detached from the frame window. On deactivating the in-place object's user interface, the object is left in a ready state so it can be quickly reactivated. The object stays in this state until the undo state of the document changes. The container should then call **IOleInPlaceObject::InPlaceDeactivate** to tell the object to discard its undo state.

See Also

IOleInPlaceObject::InPlaceDeactivate, **IOleInPlaceSite::OnUIDeactivate**, **IOleInPlaceObject::ReactivateAndUndo**, **IOleObject::Close**

IOleInPlaceSite

The **IOleInPlaceSite** interface manages interaction between the container and the object's in-place client site. Recall that the client site is the display site for embedded objects, and provides position and conceptual information about the object.

This interface provides methods that manage in-place objects. With **IOleInPlaceSite**, you can determine if an object can be activated and manage its activation and deactivation. You can notify the container when one of its objects is being activated and inform the container that a composite menu will replace the container's regular menu. It provides methods that make it possible for the in-place object to retrieve the window object hierarchy, and the position in the parent window where the object should place its in-place activation window. And finally, it manages how the container scrolls the object, its undo state, and notifies the object when its borders have changed.

When to Implement

You must implement this interface if you are writing a container application that will participate in in-place activation.

When to Use

Use this interface to allow your object to control in-place activation from within the container.

The **IOleInPlaceSite** interface pointer is obtained by calling **QueryInterface** on the object's **IOleClientSite** interface.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns a pointer to a specified interface.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IOleWindow Methods	Description
GetWindow	Gets a window handle.
ContextSensitiveHelp	Controls enabling of context-sensitive help.

IOleInPlaceSite Methods	Description
CanInPlaceActivate	Determines if the container can activate the object in place.
OnInPlaceActivate	Notifies the container that one of its objects is being activated in place.
OnUIActivate	Notifies the container that the object is about to be activated in place, and that the main menu will be replaced by a composite menu.
GetWindowContext	Enables an in-place object to retrieve window interfaces that form at the window object hierarchy, and the position in the parent window to locate the object's in-place activation window.
Scroll	Specifies the number of pixels by which the container is to scroll the

	object.
OnUIDeactivate	Notifies the container to reinstall its user interface and take focus.
OnInPlaceDeactivate	Notifies the container that the object is no longer active in place.
DiscardUndoState	Instructs the container to discard its undo state.
DeactivateAndUndo	Deactivate the object and revert to undo state.
OnPosRectChange	Object's extents have changed.

See Also

IOleWindow, IOleClientSite

IOleInPlaceSite::CanInPlaceActivate

Determines whether or not the container can activate the object in place.

HRESULT CanInPlaceActivate();

Return Values

S_OK
  The container allows in-place activation for this object.

S_FALSE
  The container does not allow in-place activation for this object.

E_INVALIDARG
  One or more arguments are invalid.

E_UNEXPECTED
  An unexpected error occurred.

Comments

Only objects being displayed as DVASPECT_CONTENT can be activated in place.

Notes to Callers

IOleInPlaceSite::CanInPlaceActivate is called by the client site's immediate child object when this object must activate in place. This function allows the container application to accept or refuse the activation request.

IOleInPlaceSite::DeactivateAndUndo

Causes the container to end the in-place session, deactivates the object, and revert to its own saved undo state.

HRESULT IOleInPlaceSite::DeactivateAndUndo();

Return Values

S_OK
The method completed successfully.

E_UNEXPECTED
An unexpected error occurred.

Comments

Notes to Callers

Called by the active object when the user invokes undo just after activating the object.

Notes to Implementors

Upon completion of this call, the container should call **IOleInPlaceObject::UIDeactivate** to remove the user interface for the object, activate itself, and undo.

IOleInPlaceSite::DiscardUndoState

Tells the container that the object.no longer has any undo state and that the container should not call IOleInPlaceObject::ReActivateAndUndo.

HRESULT IOleInPlaceSite::DiscardUndoState();

Return Values

S_OK
The method completed successfully.

E_UNEXPECTED
An unexpected error occurred.

Comments

If an object is activated in place and the object's associated object application maintains only one level of undo, there is no need to have more than one entry on the undo stack. That is, once a change has been made to the active object that invalidates its undo state saved by the container, there is no need to maintain this undo state in the container.

Notes to Callers

IOleInPlaceSite::DiscardUndoState is called by the active object while performing some action that would discard the undo state of the object. The in-place object calls this method to notify the container to discard the object's last saved undo state.

See Also

IOleInPlaceSite::DeactivateAndUndo

IOleInPlaceSite::GetWindowContext

Enables the in-place object to retrieve the window interfaces that form the window object hierarchy, and the position in the parent window where the object's in-place activation window should be placed.

```
HRESULT GetWindowContext(
  IOleInPlaceFrame **ppFrame,              //Location to return frame interface pointer
  IOleInPlaceUIWindow **ppDoc,             //Location to return document window pointer
  LPRECT lprcPosRect,                      //Points to position of in-place object
  LPRECT lprcClipRect,                     //Points to in-place object's position rectangle
  LPOLEINPLACEFRAMEINFO lpFrameInfo        //Points to OLEINPLACEFRAMEINFO structure
 );
```

Parameters

ppFrame
[out] Points to where the pointer to the frame interface is to be returned. If an error is returned, this parameter must be set to NULL.

ppDoc
[out] Points to where the pointer to the document window interface is to be returned. NULL is returned through the *ppDoc* pointer if the document window is the same as the frame window. In this case, the object can only use *ppFrame* or border negotiation. If an error is returned, this parameter must be set to NULL.

lprcPosRect
[out] Points to the rectangle containing the position of the in-place object in the client coordinates of its parent window. If an error is returned, this parameter must be set to NULL.

lprcClipRect
[out] Points to the outer rectangle containing the in-place object's position rectangle (*PosRect*). This rectangle is relative to the client area of the object's parent window. If an error is returned, this parameter must be set to NULL.

lpFrameInfo
[out] Points to an **OLEINPLACEFRAMEINFO** structure the container is to fill in with appropriate data. If an error is returned, this parameter must be set to NULL.

Return Values

S_OK
The method completed successfully.

E_INVALIDARG
One or more arguments are invalid (for example, an invalid pointer or parameter).

E_UNEXPECTED
An unexpected error occurred.

Comments

The **OLEINPLACEFRAMEINFO** structure provides data needed by OLE to dispatch keystroke accelerators to a container frame while an object is active in place.

When an object is activated, it calls **GetWindowContext** from its container. The container returns the handle to its in-place accelerator table through the **OLEINPLACEFRAMEINFO** structure. Before calling **GetWindowContext**, the object must provide the size of the **OLEINPLACEFRAMEINFO** structure by filling in the *cb* member, pointed to by *lpFrameInfo*.

IOleInPlaceSite::OnInPlaceActivate

Notifies the container that one of its objects is being activated in place.

HRESULT OnInPlaceActivate();

Return Values

S_OK
The container allows the in-place activation.

E_UNEXPECTED
An unexpected error occurred.

Comments

Notes to Callers

IOleInPlaceSite::OnInPlaceActivate is called by the active, embedded object when it is activated in-place for the first time. The container should note that the object is becoming active.

Notes to Implementors

A container that supports linking to embedded objects must properly manage the running of its in-place objects when they are UI inactive and running in the hidden state. To reactivate the in-place object quickly, a container should *not* call **IOleObject::Close** until the container's **IOleInPlaceSite::DeactivateAndUndo** method is called. To safeguard against the object being left in an unstable state if a linking client updates silently, the container should call **OleLockRunning** to lock the object in the running state. This prevents the hidden in-place object from shutting down before it can be saved in its container.

IOleInPlaceSite::OnInPlaceDeactivate

Notifies the container that the object is no longer active in place.

HRESULT OnInPlaceDeactivate();

Return Values

S_OK
  The method successfully notified the container.

E_UNEXPECTED
  An unexpected error occurred.

Comments

Notes to Callers

IOleInPlaceSite::OnInPlaceDeactivate is called by an in-place object when it is fully deactivated. This function notifies the container that the object has been deactivated, and it gives the container a chance to run code pertinent to the object's deactivation. In particular, **IOleInPlaceSite::OnInPlaceDeactivate** is called as a result of **IOleInPlaceObject::InPlaceDeactivate** being called. Calling **IOleInPlaceSite::OnInPlaceDeactivate** indicates that the object can no longer support Undo.

Notes to Implementors

If the container is holding pointers to the **IOleInPlaceObject** and **IOleInPlaceActiveObject** interface implementations, it should release them after the **IOleInPlaceSite::OnInPlaceDeactivate** call.

See Also

IOleInPlaceObject::InPlaceDeactivate

IOleInPlaceSite::OnPosRectChange

Indicates the object's extents have changed.

```
HRESULT OnPosRectChange(
   LPCRECT lprcPosRect        //Points to rectangle containing the position of in-place object
  );
```

Parameter

lprcPosRect
[in] Points to the rectangle containing the position of the in-place object in the client coordinates of its parent window.

Return Values

S_OK
The method completed successfully.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

Comments

Notes to Callers

The **IOleInPlaceSite::OnPosRectChange** method is called by the in-place object.

Notes to Implementors

When the in-place object calls **IOleInPlaceSite::OnPosRectChange**, the container must call **IOleInPlaceObject::SetObjectRects** to specify the new position of the in-place window and the *ClipRect*. Only then does the object resize its window.

In most cases, the object grows to the right and/or down. There could be cases where the object grows to the left and/or up, as conveyed through *lprcPosRect*. It is also possible to change the object's position without changing its size.

See Also

IOleInPlaceObject::SetObjectRects

IOleInPlaceSite::OnUIActivate

Notifies the container that the object is about to be activated in place and that the object is going to replace the container's main menu with an in-place composite menu.

HRESULT IOleInPlaceSite::OnUIActivate();

Return Values

S_OK
: The container allows the in-place activation.

E_UNEXPECTED
: An unexpected error occurred.

Comments

Notes to Callers

The in-place object calls **IOleInPlaceSite::OnUIActivate** just before activating its user interface.

Notes to Implementors

The container should remove any user interface associated with its own activation. If the container is itself an embedded object, it should remove its document-level user interface.

If there is already an object active in place in the same document, the container should call **IOleInPlaceObject::UIDeactivate** before calling **OnUIDeactivate**.

See Also

IOleInPlaceObject::UIDeactivate

IOleInPlaceSite::OnUIDeactivate

Notifies the container on deactivation that it should reinstall its user interface and take focus, and whether or not the object has an undoable state.

```
HRESULT OnUIDeactivate(
    BOOL fUndoable         //Specifies if object can undo changes
  );
```

Parameter

fUndoable
[in] Specifies whether the object can undo changes. It is TRUE if the object can undo, FALSE if it cannot.

Return Values

S_OK
The method completed successfully.

E_UNEXPECTED
An unexpected error occurred.

Comments

The object indicates whether it can undo changes through the *fUndoable* flag. If the object can undo changes, the container can (by the user invoking the Edit Undo command) call the **IOleInPlaceObject::ReactivateAndUndo** method to undo the changes.

Notes to Callers

IOleInPlaceSite::OnUIDeactivate is called by the site's immediate child object when it is deactivating to notify the container that it should reinstall its own user interface components and take focus. The container should wait for the call to **IOleInPlaceSite::OnUIDeactivate** to complete before fully cleaning up and destroying any composite submenus.

See Also

IOleInPlaceObject::ReactivateAndUndo

IOleInPlaceSite::Scroll

Tells the container to scroll the view of the object by a specified number of pixels.

```
HRESULT Scroll(
   SIZE scrollExtent         //Number of pixels
  );
```

Parameter

scrollExtent
[in] Contains the number of pixels by which to scroll in the *X* and *Y* directions.

Return Values

S_OK
The method successfully executed the view scroll instruction.

E_INVALIDARG
An invalid argument.

E_UNEXPECTED
An unexpected error occurred.

Comments

As a result of scrolling, the object's visible rectangle can change. If that happens, the container should give the new *ClipRect* to the object by calling **IOleInPlaceObject::SetObjectRects**. The intersection of the *ClipRect* and *PosRect* rectangles gives the new visible rectangle. See **IOleInPlaceSite::GetWindowContext** for a discussion of *ClipRect* and *PosRect*.

Notes to Callers

Called by an active, in-place object when it is asking the container to scroll.

See Also

IOleInPlaceObject::SetObjectRects

IOleInPlaceUIWindow

The **IOleInPlaceUIWindow** interface is implemented by container applications and used by object applications to negotiate border space on the document or frame window. The container provides a **RECT** structure in which the object can place toolbars and other similar controls, determines if tools can in fact be installed around the object's window frame, allocates space for the border, and establishes a communication channel between the object and each frame and document window.

The document window may not exist in all applications. When this is the case, **IOleInPlaceSite::GetWindowContext** returns NULL for **IOleInPlaceUIWindow**.

When to Implement

You must implement this interface if you are writing a container application that will participate in-place activation.

When to Use

Used by object applications to negotiate border space on the document or frame window when one of its objects is being activated or to renegotiate border space if the size of the object changes.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns a pointer to a specified interface.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IOleWindow Methods	Description
GetWindow	Gets a window handle.
ContextSensitiveHelp	Controls enabling of context sensitive help.

IOleInPlaceUIWindow Methods	Description
GetBorder	Specifies a RECT structure for toolbars and controls.
RequestBorderSpace	Determines if tools can be installed around object's window frame.
SetBorderSpace	Allocates space for the border.
SetActiveObject	Provides for direct communication between the object and each document and frame window.

See Also

IOleWindow

IOleInPlaceUIWindow::GetBorder

Returns a **RECT** structure in which the object can put toolbars and similar controls while active in place.

```
HRESULT GetBorder(
   LPRECT lprectBorder       //Points to RECT structure
  );
```

Parameter

lprectBorder
   [out] Points to a **RECT** structure where the outer rectangle is to be returned. The **RECT** structure's coordinates are relative to the window being represented by the interface.

Return Values

S_OK
   The rectangle was successfully returned.

E_NOTOOLSPACE
   The object cannot install toolbars in this window object.

E_INVALIDARG
   One or more arguments are invalid.

E_OUTOFMEMORY
   Out of memory.

E_UNEXPECTED
   An unexpected error occurred.

Comments

Notes to Callers

The **IOleInPlaceUIWindow::GetBorder** function, when called on a document or frame window object, returns the outer rectangle (relative to the window) where the object can put toolbars or similar controls.

If the object is to install these tools, it should negotiate space for the tools within this rectangle using **IOleInPlaceUIWindow::RequestBorderSpace** and then call **IOleInPlaceUIWindow::SetBorderSpace** to get this space allocated.

Note While executing **IOleInPlaceUIWindow::GetBorder**, do not make calls to the Windows **PeekMessage** or **GetMessage** functions, or a dialog box. Doing so may cause the system to deadlock. There are further restrictions on which OLE interface mehtods and functions can be called from within **GerBorder**.

See Also

IOleInPlaceUIWindow::RequestBorderSpace, **IOleInPlaceUIWindow::SetBorderSpace**

PeekMessage, **GetMessage** in Win32

IOleInPlaceUIWindow::RequestBorderSpace

Determines if there is available space for tools to be installed around the object's window frame while the object is active in place.

```
HRESULT RequestBorderSpace(
   LPCBORDERWIDTHS pborderwidths       //Points to a BORDERWIDTHS structure
  );
```

Parameter

pborderwidths
[in] Points to a **BORDERWIDTHS** structure containing the requested widths (in pixels) needed on each side of the window for the tools.

Return Values

S_OK
The requested space could be allocated to the object.

E_NOTOOLSPACE
The object cannot install toolbars in this window object because the implementation does not support toolbars, or there is insufficient space to install the toolbars.

E_INVALIDARG
One or more invalid arguments (for example, an invalid pointer or parameter).

E_UNEXPECTED
An unexpected error occurred.

Comments

Notes to Callers

The active in-place object calls **IOleInPlaceUIWindow::RequestBorderSpace** to ask if tools can be installed inside the window frame. These tools would be allocated between the rectangle returned by **IOleInPlaceUIWindow::GetBorder** and the **BORDERWIDTHS** structure specified in the argument to this call.

The space for the tools is not actually allocated to the object until it calls **IOleInPlaceUIWindow::SetBorderSpace**, allowing the object to negotiate for space (such as while dragging toolbars around), but deferring the moving of tools until the action is completed.

The object can install these tools by passing the width in pixels that is to be used on each side. For example, if the object required 10 pixels on the top, 0 pixels on the bottom, and 5 pixels on the left and right sides, it would pass the following **BORDERWIDTHS** structure to **IOleInPlaceUIWindow::RequestBorderSpace**:

```
lpbw->top    = 10
lpbw->bottom =  0
lpbw->lLeft  =  5
lpbw->right  =  5
```

Note While executing **IOleInPlaceUIWindow::RequestBorderSpace**, do not make calls to the Windows **PeekMessage** or **GetMessage** functions, or a dialog box. Doing so may cause the system to deadlock. There are further restrictions on which OLE interface methods and functions can be called from within **RequestBorderSpace**.

Notes to Implementors

If the amount of space an active object uses for its toolbars is irrelevant to the container, it can simply return NOERROR as shown in the following **IOleInPlaceUIWindow::RequestBorderSpace** example. Containers should not unduly restrict the display of tools by an active in-place object.

```
HRESULT InPlaceUIWindow_RequestBorderSpace(
    IOleInPlaceFrame *  lpThis,
    LPCBORDERWIDTHS     pborderwidths)
{
    /* Container allows the object to have as much border space as it
    ** wants.
    */
    return NOERROR;
}
```

See Also

IOleInPlaceUIWindow::GetBorder, **IOleInPlaceUIWindow::SetBorderSpace**

PeekMessage, **GetMessage** in Win32

IOleInPlaceUIWindow::SetActiveObject

Provides a direct channel of communication between the object and each of the frame and document windows.

```
HRESULT SetActiveObject(
  IOleInPlaceActiveObject *pActiveObject,     //Points to active in-place object
  LPCOLESTR pszObjName                        //Points to name describing the object
 );
```

Parameters

pActiveObject
: [in] Points to the active in-place object's **IOleInPlaceActiveObject** interface.

pszObjName
: [in] Points to a string containing a name that describes the object an embedding container can use in composing its window title. It can be NULL if the object does not require the container to change its window titles. The *Microsoft Windows User Interface Design Guide* recommends that containers ignore this parameter and always use their own name in the title bar.

Return Values

S_OK
: The method completed successfully.

E_INVALIDARG
: One or more arguments are invalid.

E_UNEXPECTED
: An unexpected error occurred.

Comments

Generally, an embedded object should pass NULL for the *pszObjName* parameter (see "Notes to Implementors" below). However, if you are working in conjunction with a container that does display the name of the in-place active object in its title bar, then you should compose a string in the following form:

<application name> - <object short-type name>

Notes to Callers

IOleInPlaceUIWindow::SetActiveObject is called by the object to establish a direct communication link between itself and the document and frame windows.

When deactivating, the object calls **IOleInPlaceUIWindow::SetActiveObject**, passing NULL for the *pActiveObject* and *pszObjName* parameters.

An object *must* call **IOleInPlaceUIWindow::SetActiveObject** before calling **IOleInPlaceFrame::SetMenu** to give the container the pointer to the active object. The container then uses this pointer in processing **IOleInPlaceFrame::SetMenu** and to pass to **OleSetMenuDescriptor**.

Notes to Implementors

The *Microsoft Windows User Interface Design Guide* recommends that an in-place container ignore the *pszObjName* parameter passed in this method. The guide says "The title bar is not affected by in-place activation. It always displays the top-level container's name."

See Also

IOleInPlaceFrame::SetMenu, **OleSetMenuDescriptor**

IOleInPlaceUIWindow::SetBorderSpace

Allocates space for the border requested in the call to **IOleInPlaceUIWindow::RequestBorderSpace**.

```
HRESULT SetBorderSpace(
   LPCBORDERWIDTHS pborderwidths        //Points to a BORDERWIDTHS structure
  );
```

Parameter

pborderwidths
[in] Points to a **BORDERWIDTHS** structure containing the requested width (in pixels) of the tools. It can be NULL, indicating the object does not need any space.

Return Values

S_OK
The requested space has been allocated to the object.

OLE_E_INVALIDRECT
The rectangle does not lie within the specifications returned by **IOleInPlaceUIWindow::GetBorder**.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

Comments

The object must call **IOleInPlaceUIWindow::SetBorderSpace**. It can do any of the following:

1. Use its own toolbars, requesting border space of a specific size; or,
2. Use no toolbars, but force the container to remove its toolbars by passing a valid **BORDERWIDTHS** structure containing nothing but zeros in the *pborderwidths* parameter; or,
3. Use no toolbars but allow the in-place container to leave its toolbars up by passing NULL as the *pborderwidths* parameter.

The **BORDERWIDTHS** structure used in this call would generally have been passed in a previous call to **IOleInPlaceUIWindow::RequestBorderSpace**, which must have returned S_OK.

If an object must renegotiate space on the border, it can call **SetBorderSpace** again with the new widths. If the call to **SetBorderSpace** fails, the object can do a full negotiation for border space with calls to **GetBorder**, **RequestBorderSpace**, and **SetBorderSpace**.

Note While executing **IOleInPlaceUIWindow::SetBorderSpace**, do not make calls to the Windows **PeekMessage** or **GetMessage** functions, or a dialog box. Doing so may cause the system to deadlock. There are further restrictions on which OLE interface methods and functions can be called from within **SetBorderSpace**.

See Also

IOleInPlaceUIWindow::GetBorder, **IOleInPlaceUIWindow::RequestBorderSpace**

PeekMessage, **GetMessage** in Win32

IOleItemContainer

The **IOleItemContainer** interface is used by item monikers when they are bound to the objects they identify.

When any container of objects uses item monikers to identify its objects, it must define a naming scheme for those objects. The container's **IOleItemContainer** implementation uses knowledge of that naming scheme to retrieve an object given a particular name. Item monikers use the container's **IOleItemContainer** implementation during binding.

When to Implement

You must implement **IOleItemContainer** if you're a moniker provider handing out item monikers. Being a moniker provider means handing out monikers that identify your objects to make them accessible to moniker clients. You must use item monikers if the objects you're identifying are contained within another object and can be individually identified using a string.

The most common example of moniker providers are OLE applications that support linking. If your OLE application supports linking to objects smaller than a file-based documents, you need to use item monikers. For a server application that allows linking to a portion of a document (such as selections within a document), you use the item monikers to identify those objects. For a container application that allows linking to embedded objects, you use the item monikers to identify the embedded objects.

You must define a naming scheme for identifying the objects within the container; for example, embedded objects in a document could be identified with names of the form "embedobj1," "embedobj2," and so forth, while ranges of cells in a spreadsheet could be identified with names of the form "A1:E7," "G5:M9," and so forth. (Ranges of cells in a spreadsheet are examples of as "pseudo-objects" because they do not have their own persistent storage, they simply represent a portion of the container's internal state.) You create an item moniker that represents an object's name using the **CreateItemMoniker** API function and hand it out to a moniker client. When an item moniker is bound, your implementation of **IOleItemContainer** must be able to take a name and retrieve the corresponding object.

When to Use

Applications typically do not call **IOleItemContainer** methods directly. The item moniker implementation of **IMoniker** is the primary caller of **IOleItemContainer** methods.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IParseDisplayName Method	Description
ParseDisplayName	Parses object's display name to form moniker.

IOleContainer Methods	Description
EnumObjects	Enumerates objects in a container.
LockContainer	Keeps container running until explicitly released.

IOleItemContainer Methods	Description
GetObject	Returns a pointer to a specified

	object.
GetObjectStorage	Returns a pointer to an object's storage.
IsRunning	Checks whether an object is running.

See Also

CreateItemMoniker, **IMoniker - Item Moniker Implementation**

IOleItemContainer::GetObject

Returns a pointer to the object identified by the specified name.

```
HRESULT GetObject(
   LPOLESTR pszItem,            //Name of the object requested
   DWORD dwSpeedNeeded,         //Speed requirements on binding
   IBindCtx *pbc,               //Bind context to be used
   REFIID riid,                 //IID of interface pointer desired
   void **ppvObject             //Receives interface pointer
  );
```

Parameters

pszItem
[in] Points to a zero-terminated string containing the container's name for the requested object. For Win32 applications, the **LPOLESTR** type indicates a wide character string (two bytes per character); otherwise, the string has one byte per character.

dwSpeedNeeded
[in] Indicates approximately how long the caller will wait to get the object. The legal values for *dwSpeedNeeded* are taken from the enumeration **BINDSPEED**. For information on the **BINDSPEED** enumeration, see the "Data Structures" section.

pbc
[in] Points to the bind context to be used in this binding operation. The bind context caches objects bound during the binding process, contains parameters that apply to all operations using the bind context, and provides the means by which the binding implementation should retrieve information about its environment. For more information, see **IBindCtx**.

riid
[in] Specifies the type of interface pointer requested.

ppvObject
[out] Receives a pointer to the object named by *pszItem*. If an error occurs, the implementation sets **ppvObject* to NULL. If **ppvObject* is non-NULL, the implementation must call **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK
The specified object was successfully returned.

MK_E_EXCEEDEDDEADLINE
The binding operation could not be completed within the time limit specified by the bind context's **BIND_OPTS** structure, or with the speed indicated by the *dwSpeedNeeded* parameter.

MK_E_NOOBJECT
The parameter *pszItem* does not identify a object in this container.

E_NOINTERFACE
The requested interface was not available.

E_OUTOFMEMORY
Insufficient memory.

Comments

The item moniker implementation of **IMoniker::BindToObject** calls this method, passing the name stored within the item moniker as the *pszItem* parameter.

Notes to Implementors

Your implementation of **IOleItemContainer::GetObject** should first determine whether *pszItem* is a valid name for one of the container's objects. If not, you should return MK_E_NOOBJECT.

If *pszItem* names an embedded or linked object, your implementation must check the value of the *dwSpeedNeeded* parameter. If the value is BINDSPEED_IMMEDIATE and the object is not yet loaded, you should return MK_E_EXCEEDEDDEADLINE. If the object is loaded, your implementation should determine whether the object is running (for example, by calling the **OleIsRunning** API function). If it is not running and the *dwSpeedNeeded* value is BINDSPEED_MODERATE, your implementation should return MK_E_EXCEEDEDDEADLINE. If the object is not running and *dwSpeedNeeded* is BINDSPEED_INDEFINITE, your implementation should call the **OleRun** API function to put the object in the running state. Then it can query the object for the requested interface. Note that it is important the object be running before you query for the interface.

If *pszItem* names a pseudo-object, your implementation can ignore the *dwSpeedNeeded* parameter because a pseudo-object is running whenever its container is running. In this case, your implementation can simply query for the requested interface.

If you want more specific information about the time limit than is given by *dwSpeedNeeded*, you can call **IBindCtx::GetBindOptions** on the *pbc* parameter to get the actual deadline parameter.

See Also

IMoniker::BindToObject, **IBindCtx::GetBindOptions**, **OleIsRunning**, **OleRun**

IOleItemContainer::GetObjectStorage

Returns a pointer to the storage for the object identified by the specified name.

```
HRESULT GetObjectStorage(
   LPOLESTR pszItem,          //Name of the object whose storage is requested
   IBindCtx *pbc,             //Bind context to be used
   REFIID riid,               //IID of interface pointer desired
   void **ppvStorage          //Receives interface pointer
  );
```

Parameters

pszItem
[in] Points to a zero-terminated string containing the compound document's name for the object whose storage is requested. For Win32 applications, the **LPOLESTR** type indicates a wide character string (two bytes per character); otherwise, the string has one byte per character.

pbc
[in] Points to the bind context to be used in this binding operation. The bind context caches objects bound during the binding process, contains parameters that apply to all operations using the bind context, and provides the means by which the binding implementation should retrieve information about its environment. For more information, see **IBindCtx**.

riid
[in] Identifies the type of interface pointer requested.

ppvStorage
[out] Receives a pointer to the storage of the object named by *pszItem*. If an error occurs, the implementation sets **ppvStorage* to NULL. If **ppvStorage* is non-NULL, the implementation must call **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK
The storage of the specified object was successfully returned.

MK_E_NOOBJECT
The parameter *pszItem* does not identify a object in this container.

MK_E_NOSTORAGE
The object does not have its own independent storage.

E_NOINTERFACE
The requested interface is not available.

E_OUTOFMEMORY
Insufficient memory.

Comments

The item moniker implementation of **IMoniker::GetObjectStorage** calls this method.

Notes to Implementors

If *pszItem* designates a pseudo-object, your implementation should return MK_E_NOSTORAGE, because pseudo-objects do not have their own independent storage. If *pszItem* designates an embedded object, or a portion of the document that has its own storage, your implementation should return the specified interface pointer on the appropriate storage object.

See Also

IOleItemContainer::GetObjectStorage

IOleItemContainer::IsRunning

Indicates whether the object identified by the specified name is running.

```
HRESULT IsRunning(
  LPOLESTR pszItem        //Name of object whose status is being requested
 );
```

Parameter

pszItem
[in] Points to a zero-terminated string containing the container's name for the object. For Win32 applications, the **LPOLESTR** type indicates a wide character string (two bytes per character); otherwise, the string has one byte per character.

Return Values

S_OK
The specified object is running.

S_FALSE
The object is not running.

MK_E_NOOBJECT
The parameter *pszItem* does not identify a object in this container.

Comments

The item moniker implementation of **IMoniker::IsRunning** calls this method.

Notes to Implementors

Your implementation of **IOleItemContainer::IsRunning** should first determine whether *pszItem* identifies one of the container's objects. If it does not, your implementation should return MK_E_NOOBJECT. If the object is not loaded, your implementation should return S_FALSE. If it is loaded, your implementation can call the **OleIsRunning** API function to determine whether it is running.

If *pszItem* names a pseudo-object, your implementation can simply return S_OK because a pseudo-object is running whenever its container is running.

See Also

IMoniker::IsRunning

IOleLink

The **IOleLink** interface is the means by which a linked object provides its container with functions pertaining to linking. The most important of these functions is binding to the link source, that is, activating the connection to the document that stores the linked object's native data. **IOleLink** also defines functions for managing information about the linked object, such as the location of the link source and the cached presentation data for the linked object.

A container application can distinguish between embedded objects and linked objects by querying for **IOleLink**; only linked objects implement **IOleLink**.

When to Implement

You do not have to implement this interface yourself; the system supplies an implementation of **IOleLink** that is suitable for all situations. This implementation is used automatically whenever you create or load a linked object.

When to Use

You must use **IOleLink** if you are writing a container application that allows its documents to contain linked objects. You primarily call **IOleLink** methods in order to implement the Links dialog box. If you use the **OleUIEditLinks** API function to display the Links dialog box, your calls to **IOleLink** methods take place in your implementation of the **IOleUILinkContainer** interface.

Some **IOleLink** methods don't have to be called directly. Instead, you call methods of **IOleObject**; the default linked object provides an implementation of **IOleObject** that often calls methods of **IOleLink**. For example, a container application typically activates a linked object by calling **IOleObject::DoVerb**, which in turn calls **IOleLink::BindToSource**.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IOleLink Methods	Description
SetUpdateOptions	Sets the update options.
GetUpdateOptions	Returns the update options.
SetSourceMoniker	Sets the moniker for the link source.
GetSourceMoniker	Returns the moniker for the link source.
SetSourceDisplayName	Sets the display name for the link source.
GetSourceDisplayName	Returns the display name for the link source.
BindToSource	Binds the moniker to the link source.
BindIfRunning	Binds the moniker if the source is running.
GetBoundSource	Returns a pointer to the link source if it's running.
UnbindSource	Break connection to the link source.
Update	Update the cached views of the link

source.

See Also

IOleObject, **IOleUILinkContainer**, **OleUIEditLinks**

IOleLink::BindIfRunning

Activates the connection between the linked object and the link source if the link source is already running.

HRESULT BindIfRunning(*void***);**

Return Values

S_OK
: The link source was bound.

S_FALSE
: The link source is not running.

CreateBindCtx, **IMoniker::IsRunning**, or **IOleLink::BindToSource** errors
: Binding the moniker might require calling these functions; therefore, errors generated by these functions may be returned.

Comments

You typically do not need to call **IOleLink::BindIfRunning**. This method is primarily called by the linked object.

Notes on Provided Implementation

The linked object's implementation of **IOleLink::BindIfRunning** checks the Running Object Table (ROT) to determine whether the link source is already running. It checks both the relative and absolute monikers. If the link source is running, **IOleLink::BindIfRunning** calls **IOleLink::BindToSource** to connect the linked object to the link source.

See Also

IOleLink::BindToSource

IOleLink::BindToSource

Activates the connection to the link source by binding the moniker stored within the linked object.

```
HRESULT BindToSource(
  DWORD bindflags,      //Flag in case CLSID of link source is different
  IBindCtx *pbc         //Bind context to be used
 );
```

Parameters

bindflags
: [in] Specifies how to proceed if the link source has a different CLSID from the last time it was bound. If this parameter is zero and the CLSIDs are different, the method fails and returns OLE_E_CLASSDIFF. If the OLELINKBIND_EVENIFCLASSDIFF value from the **OLELINKBIND** enumeration is specified and the CLSIDs are different, the method binds successfully and updates the CLSID stored in the linked object.

pbc
: [in] Points to the bind context to be used in this binding operation. This parameter can be NULL. The bind context caches objects bound during the binding process, contains parameters that apply to all operations using the bind context, and provides the means by which the binding implementation should retrieve information about its environment. For more information, see **IBindCtx**.

Return Values

S_OK
: Indicates that the link source is bound.

OLE_E_CLASSDIFF
: Indicates that the link source was not bound because its CLSID has changed. This error is returned only if the OLELINKBIND_EVENIFCLASSDIFF flag is not specified in the *bindflags* parameter.

MK_E_NOOBJECT
: Indicates that the link source could not be found or (if the link source's moniker is a composite) some intermediate object identified in the composite could not be found.

E_UNSPEC
: Indicates that the link's moniker is NULL.

CreateBindCtx errors
: Binding the moniker might require calling this function; therefore, this method may return errors generated by this function.

Comments

Notes to Callers

Typically, your container application does not need to call the **IOleLink::BindToSource** method directly. When it's necessary to activate the connection to the link source, your container typically calls **IOleObject::DoVerb**, **IOleObject::Update**, or **IOleLink::Update**. The linked object's implementation of these methods calls **IOleLink::BindToSource**. Your container can also call the **OleRun** API function, which – when called on a linked object – calls **IOleLink::BindToSource**.

In each of the examples listed above, in which **IOleLink::BindToSource** is called indirectly, the *bindflags* parameter is set to zero. Consequently, these calls can fail with the OLE_E_CLASSDIFF error if the class of the link source is different from what it was the last time the linked object was bound. This could happen, for example, if the original link source was an embedded Lotus spreadsheet that an end user had subsequently converted (using the Change Type dialog box) to an Excel spreadsheet.

If you want your container to bind even though the link source now has a different CLSID, you can call

IOleLink::BindToSource directly and specify OLELINKBIND_EVENIFCLASSDIFF for the *bindflags* parameter. This call binds to the link source and updates the link object's CLSID. Alternatively, your container can delete the existing link and use the **OleCreateLink** API function to create a new linked object.

Notes on Provided Implementation

The linked object caches the interface pointer to the link source acquired during binding.

The linked object's **IOleLink::BindToSource** implementation first tries to bind using a moniker consisting of the compound document's moniker composed with the link source's relative moniker. If successful, it updates the link's absolute moniker. Otherwise, it tries to bind using the absolute moniker, updating the relative moniker if successful.

If **IOleLink::BindToSource** binds to the link source, it calls the compound document's **IOleContainer::LockContainer** implementation to keep the containing compound document alive while the link source is running. **IOleLink::BindToSource** also calls the **IOleObject::Advise** and **IDataObject::DAdvise** implementations of the link source to set up advisory connections. The **IOleLink::UnbindSource** implementation unlocks the container and deletes the advisory connections.

See Also

IDataObject::DAdvise, **IOleContainer::LockContainer**, **IOleLink::Update**, **IOleLink::UnbindSource**, **IOleObject::Advise**, **IOleObject::DoVerb**, **IOleObject::Update**, **OleRun**

IOleLink::GetBoundSource

Returns an **IUnknown** pointer to the link source if the connection is currently active.

```
HRESULT GetBoundSource(
  IUnknown **ppunk        //Receives pointer to link source
 );
```

Parameter

ppunk
[out] Receives an **IUnknown** pointer to the link source. If an error occurs, the implementation sets **ppunk* to NULL. If **ppunk* is non-NULL, the implementation must call **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK
Indicates that a pointer was returned successfully.

E_FAIL
Indicates that the link source is not bound.

Comments

You typically do not need to call **IOleLink::GetBoundSource**.

IOleLink::GetSourceDisplayName

Retrieves the display name of the link source of the linked object.

```
HRESULT GetSourceDisplayName(
   LPOLESTR *ppszDisplayName        //Receives display name of link source
  );
```

Parameter

ppszDisplayName

[out] Receives a pointer to a zero-terminated string containing the display name of the link source. If an error occurs, **ppszDisplayName* is set to NULL; otherwise, the implementation must use **IMalloc::Alloc** to allocate the string returned in *ppszDisplayName*, and the caller is responsible for calling **IMalloc::Free** to free it. Both caller and callee use the allocator returned by **CoGetMalloc**(MEMCTX_TASK, ...). For Win32 applications, the **LPOLESTR** type indicates a wide character string (two bytes per character); otherwise, the string has one byte per character.

Return Values

S_OK
: The display name was successfully retrieved.

E_FAIL
: The link's moniker is NULL.

CreateBindCtx and **IMoniker::GetDisplayName** errors
: Retrieving the display name requires calling these functions; therefore, this method may return errors generated by these functions.

Comments

Notes to Callers

Your container application can call **IOleLink::GetSourceDisplayName** in order to display the current source of a link.

The current source of a link is displayed in the Links dialog box. If you use the **OleUIEditLinks** API function to display the Links dialog box, you must implement the **IOleUILinkContainer** interface. The dialog box calls your implementations of **IOleUILinkContainer::GetLinkSource** to get the string it should display. Your implementation of that method can call **IOleLink::GetSourceDisplayName**.

Notes on Provided Implementation

The linked object's implementation of **IOleLink::GetSourceDisplayName** calls **IOleLink::GetSourceMoniker** to get the link source moniker, and then calls **IMoniker::GetDisplayName** to get that moniker's display name. This operation is potentially expensive because it might require binding the moniker. All of the system-supplied monikers can return a display name without binding, but there is no guarantee that other moniker implementations can. Instead of making repeated calls to **IOleLink::GetSourceDisplayName**, your container application can cache the name and update it whenever the link source is bound.

See Also

IOleLink::SetSourceDisplayName, **IOleUILinkContainer**, **IMoniker::GetDisplayName**, **OleUIEditLinks**

IOleLink::GetSourceMoniker

Retrieves the moniker identifying the link source of a linked object.

```
HRESULT GetSourceMoniker(
   IMoniker **ppmk          //Receives moniker identifying link source
  );
```

Parameter

ppmk

[out] Receives a pointer to an absolute moniker that identifies the link source. May be NULL if an error occurs. If **ppmk* is non-NULL, the implementation must call **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK
The moniker was returned successfully.

MK_E_UNAVAILABLE
No moniker is available.

Comments

Notes to Callers

Your container application can call **IOleLink::GetSourceMoniker** to display the current source of a link in the Links dialog box. Note that this requires your container to use the **IMoniker::GetDisplayName** method to get the display name of the moniker. If you'd rather get the display name directly, your container can call **IOleLink::GetSourceDisplayName** instead of **IOleLink::GetSourceMoniker**.

If you use the **OleUIEditLinks** API function to display the Links dialog box, you must implement the **IOleUILinkContainer** interface. The dialog box calls your implementations of **IOleUILinkContainer::GetLinkSource** to get the string it should display. Your implementation of that method can call **IOleLink::GetSourceMoniker**.

Notes on Provided Implementation

The linked object stores both an absolute and a relative moniker for the link source. If the relative moniker is non-NULL and a moniker is available for the compound document, **IOleLink::GetSourceMoniker** returns the moniker created by composing the relative moniker onto the end of the compound document's moniker. Otherwise, it returns the absolute moniker or, if an error occurs, NULL.

The container specifies the absolute moniker when it calls one of the **OleCreateLink** functions to create a link. The application can call **IOleLink::SetSourceMoniker** or **IOleLink::SetSourceDisplayName** to change the absolute moniker. In addition, the linked object automatically updates the monikers whenever it successfully binds to the link source, or when it is bound to the link source and it receives a rename notification through the **IAdviseSink::OnRename** method.

See Also

IOleLink::SetSourceDisplayName, **IOleLink::SetSourceMoniker**

IOleLink::GetUpdateOptions

Retrieves a value indicating how often the linked object updates its cached data.

```
HRESULT GetUpdateOptions(
   DWORD *pdwUpdateOpt       //Receives update option
  );
```

Parameter

pdwUpdateOpt

[out] Receives the current value for the linked object's update option, indicating how often the linked object updates the cached data for the linked object. The legal values for *pdwUpdateOpt* are taken from the enumeration **OLEUPDATE**. For information on the **OLEUPDATE** enumeration, see the "Data Structures" section.

Return Value

S_OK

The update option was retrieved successfully.

Comments

Notes to Callers

Your container application should call **IOleLink::GetUpdateOptions** to display the current update option for a linked object.

A linked object's current update option is displayed in the Links dialog box. If you use the **OleUIEditLinks** API function to display the Links dialog box, you must implement the **IOleUILinkContainer** interface. The dialog box calls your implementation of **IOleUILinkContainer::GetLinkUpdateOptions** to determine which update option it should display. Your implementation of that method should call **IOleLink::GetUpdateOptions** to retrieve the current update option.

See Also

IOleLink::SetUpdateOptions, **IOleUILinkContainer**, **OleUIEditLinks**

IOleLink::SetSourceDisplayName

Specifies the new link source of a linked object using a display name.

```
HRESULT SetSourceDisplayName(
   LPCOLESTR pszStatusText        //Display name of new link source
  );
```

Parameter

pszStatusText
[in] The display name of the new link source. It may not be NULL.

Return Values

S_OK
The display name was set successfully.

MkParseDisplayName errors
Setting the display name requires calling this function; therefore, this method may return errors generated by this function.

Comments

Notes to Callers

Your container application can call **IOleLink::SetSourceDisplayName** when the end user changes the source of a link or breaks a link. Note that this requires the linked object to create a moniker out of the display name. If you'd rather parse the display name into a moniker yourself, your container can call **IOleLink::SetSourceMoniker** instead of **IOleLink::SetSourceDisplayName**.

If you use the **OleUIEditLinks** API function to display the Links dialog box, you must implement the **IOleUILinkContainer** interface. The dialog box calls your implementations of **IOleUILinkContainer::SetLinkSource** and **IOleUILinkContainer::CancelLink**. Your implementation of these methods can call **IOleLink::SetSourceDisplayName**.

If your container application is immediately going to bind to a newly specified link source, you should call **MkParseDisplayName** and **IOleLink::SetSourceMoniker** instead, and then call **IOleLink::BindToSource** using the bind context from the parsing operation. By reusing the bind context, you can avoid redundant loading of objects that might otherwise occur.

Notes on Provided Implementation

The contract for **IOleLink::SetSourceDisplayName** does not specify when the linked object will parse the display name into a moniker. The parsing can occur before **IOleLink::SetSourceDisplayName** returns, or the linked object can store the display name and parse it only when it needs to bind to the link source. Note that parsing the display name is potentially an expensive operation because it might require binding to the link source. The provided implementation of **IOleLink::SetSourceDisplayName** parses the display name and then releases the bind context used in the parse operation. This can result in running and then stopping the link source server.

If the linked object is bound to the current link source, the implementation of **IOleLink::SetSourceDisplayName** breaks the connection.

For more information on how the linked object stores and uses the moniker to the link source, see **IOleLink::SetSourceMoniker**.

See Also

IOleLink::SetSourceMoniker, **IOleUILinkContainer**, **MkParseDisplayName**, **OleUIEditLinks**

IOleLink::SetSourceMoniker

Specifies the new link source of a linked object using a moniker.

```
HRESULT SetSourceMoniker(
    IMoniker *pmk,          //Moniker identifying new link source
    REFCLSID rclsid         //CLSID of link source
  );
```

Parameters

pmk
[in] Points to a moniker that identifies the new link source of the linked object. A value of NULL breaks the link.

rclsid
[in] Specifies the CLSID of the link source that the linked object should use to access information about the linked object when it is not bound.

Return Value

S_OK
The moniker was set successfully.

Comments

Notes to Callers

Your container application can call **IOleLink::SetSourceMoniker** when the end user changes the source of a link or breaks a link. Note that this requires your container to use the **MkParseDisplayName** API function to create a moniker out of the display name that the end user enters. If you'd rather have the linked object perform the parsing, your container can call **IOleLink::SetSourceDisplayName** instead of **IOleLink::SetSourceMoniker**.

The end user changes the source of a link or breaks a link using the Links dialog box. If you use the **OleUIEditLinks** API function to display the Links dialog box, you must implement the **IOleUILinkContainer** interface. The dialog box calls your implementations of **IOleUILinkContainer::SetLinkSource** and **IOleUILinkContainer::CancelLink**; your implementation of these methods can call **IOleLink::SetSourceMoniker**.

If the linked object is currently bound to its link source, the linked object's implementation of **IOleLink::SetSourceMoniker** closes the link before changing the moniker.

Notes on Provided Implementation

The **IOleLink** contract does not specify how the linked object stores or uses the link source moniker. The provided implementation stores the absolute moniker specified when the link is created or when the moniker is changed; it then computes and stores a relative moniker. Future implementations might manage monikers differently to provide better moniker tracking. The absolute moniker provides the complete path to the link source. The linked object uses this absolute moniker and the moniker of the compound document to compute a relative moniker that identifies the link source relative to the compound document that contains the link.

```
pmkCompoundDoc->RelativePathTo(pmkAbsolute, ppmkRelative)
```

When binding to the link source, the linked object first tries to bind using the relative moniker. If that fails, it tries to bind the absolute moniker.

When the linked object successfully binds using either the relative or the absolute moniker, it automatically updates the other moniker. The linked object also updates both monikers when it is bound to the link source and it receives a rename notification through the **IAdviseSink::OnRename** method. A container application can also use the **IOleLink::SetSourceDisplayName** method to change a link's moniker.

The linked object's implementation of **IPersistStorage::Save** saves both the relative and the absolute moniker.

See Also

IOleLink::GetSourceMoniker, **IOleLink::SetSourceDisplayName**, **IOleUILinkContainer**, **OleUIEditLinks**

IOleLink::SetUpdateOptions

Specifies how often a linked object should update its cached data.

```
HRESULT SetUpdateOptions(
  DWORD dwUpdateOpt        //Update option
 );
```

Parameter

dwUpdateOpt
[in] Specifies how often a linked object should update its cached data. The legal values for *dwUpdateOpt* are taken from the enumeration **OLEUPDATE**. For information on the **OLEUPDATE** enumeration, see the "Data Structures" section.

Return Values

S_OK
The update option was successfully set.

E_INVALIDARG
The *dwUpdateOpt* value is invalid.

Comments

Notes to Callers

Your container application should call **IOleLink::SetUpdateOptions** when the end user changes the update option for a linked object.

The end user selects the update option for a linked object using the Links dialog box. If you use the **OleUIEditLinks** API function to display this dialog box, you must implement the **IOleUILinkContainer** interface. The dialog box calls your **IOleUILinkContainer::SetLinkUpdateOptions** method to specify the update option chosen by the end user. Your implementation of this method should call the **IOleLink::SetUpdateOptions** method to pass the selected option to the linked object.

Notes on Provided Implementation

The default update option is OLEUDPATE_ALWAYS. The linked object's implementation of **IPersistStorage::Save** saves the current update option.

If OLEUDPATE_ALWAYS is specified as the update option, the linked object updates the link's caches in the following situations:

- When the update option is changed from manual to automatic, if the link source is running.
- Whenever the linked object binds to the link source.
- Whenever the link source is running and the linked object's **IOleObject::Close**, **IPersistStorage::Save**, or **IAdviseSink::OnSave** implementations are called.

For both manual and automatic links, the linked object updates the cache whenever the container application calls **IOleObject::Update** or **IOleLink::Update**.

See Also

IOleObject::Update, **IOleLink::GetUpdateOptions**, **IOleLink::Update**, **IOleUILinkContainer**, **OleUIEditLinks**

IOleLink::UnbindSource

Deactivates the connection between a linked object and its link source.

HRESULT UnbindSource(*void***);**

Return Value

S_OK
    Indicates the connection was deactivated.

Comments

You typically do not call **IOleLink::UnbindSource** directly. When it's necessary to deactivate the connection to the link source, your container typically calls **IOleObject::Close** or **IUnknown::Release**; the linked object's implementation of these methods calls **IOleLink::UnbindSource**. The linked object's **IAdviseSink::OnClose** implementation also calls **IOleLink::UnbindSource**.

Notes on Provided Implementation

The linked object's implementation of **IOleLink::UnbindSource** does nothing if the link source is not currently bound. If the link source is bound, **IOleLink::UnbindSource** calls the link source's **IOleObject::Unadvise** and **IDataObject::DUnadvise** implementations to delete the advisory connections to the link source. The **IOleLink::UnbindSource** method also calls the compound document's **IOleClientSite::LockContainer** implementation to unlock the containing compound document. This undoes the lock on the container and the advisory connections that were established in **IOleLink::BindToSource**. **IOleLink::UnbindSource** releases all the linked object's interface pointers to the link source.

See Also

IAdviseSink::OnClose, **IDataObject::DUnadvise**, **IOleObject::Close**, **IOleObject::Unadvise**, **IOleLink::BindToSource**

IOleLink::Update

Updates the compound document's cached data for a linked object. This involves binding to the link source, if it is not already bound.

```
HRESULT Update(
   IBindCtx *pbc          //Bind context to be used
  );
```

Parameter

pbc
[in] Points to the bind context to be used in binding the link source. This parameter can be NULL. The bind context caches objects bound during the binding process, contains parameters that apply to all operations using the bind context, and provides the means by which the binding implementation should retrieve information about its environment. For more information, see **IBindCtx**.

Return Values

S_OK
All caches were updated successfully.

CACHE_E_NOCACHE_UPDATED
The bind operation worked but no caches were updated.

CACHE_S_SOMECACHES_NOTUPDATED
The bind operation worked but not all caches were updated.

OLE_E_CANT_BINDTOSOURCE
Unable to bind to the link source.

Comments

Notes to Callers

Your container application should call **IOleLink::Update** if the end user updates the cached data for a linked object.

The end user can update the cached data for a linked object by choosing the Update Now button in the Links dialog box. If you use the **OleUIEditLinks** API function to display the Links dialog box, you must implement the **IOleUILinkContainer** interface. The dialog box calls your implementations of **IOleUILinkContainer::UpdateLink** when the end user chooses the Update Now button. Your implementation of that method can call **IOleLink::Update**.

Your container application can also call **IOleObject::Update** to update a linked object, because that method – when called on a linked object – calls **IOleLink::Update**.

This method updates both automatic links and manual links. For manual links, calling **IOleLink::Update** or **IOleObject::Update** is the only way to update the caches. For more information on automatic and manual links, see **IOleLink::SetUpdateOptions**.

Notes on Provided Implementation

If *pbc* is non-NULL, the linked object's implementation of **IOleLink::Update** calls **IBindCtx::RegisterObjectBound** to register the bound link source. This ensures that the link source remains running until the bind context is released.

The current caches are left intact if the link source cannot be bound.

See Also

IBindCtx::RegisterObjectBound, **IOleLink::SetUpdateOptions**, **IOleObject::Update**,

IOleUILinkContainer, **OleUIEditLinks**

IOleObject

The **IOleObject** interface is the principal means by which an embedded object provides basic functionality to, and communicates with, its container.

When to Implement

An object application must implement this interface, along with at least IDataObject and IPersistStorage, for each type of embedded object that it supports. Although this interface contains 21 member functions, only three are both essential and nontrivial to implement: **DoVerb**, **SetHostNames**, **and Close**. Six of the member functions are optional: **SetExtent**, **InitFromData**, **GetClipboardData**, **SetColorScheme**, **SetMoniker**, and **GetMoniker**. The latter two member functions are useful mainly for enabling links to embedded objects.

When to Use

Call the methods of this interface to enable a container to communicate with an embedded object. A container must call DoVerb to activate an embedded object, SetHostNames to communicate the names of the container application and container document, and Close to move an object from a running to a loaded state. Calls to all other member functions are optional.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IOleObject Methods	Description
SetClientSite	Informs object of its client site in container.
GetClientSite	Retrieves object's client site.
SetHostNames	Communicates names of container application and container document.
Close	Moves object from running to loaded state.
SetMoniker	Informs object of its moniker.
GetMoniker	Retrieves object's moniker.
InitFromData	Initializes embedded object from selected data.
GetClipboardData	Retrieves a data transfer object from the Clipboard.
DoVerb	Invokes object to perform one of its enumerated actions ("verbs").
EnumVerbs	Enumerates actions ("verbs") for an object.
Update	Updates an object.
IsUpToDate	Checks if object is up to date.
GetUserClassID	Returns an object's class identifier.
GetUserType	Retrieves object's user-type name.
SetExtent	Sets extent of object's display area.

GetExtent	Retrieves extent of object's display area.
Advise	Establishes advisory connection with object.
Unadvise	Destroys advisory connection with object.
EnumAdvise	Enumerates object's advisory connections.
GetMiscStatus	Retrieves status of object.
SetColorScheme	Recommends color scheme to object application.

IOleObject::Advise

Establishes an advisory connection between a compound document object and the calling object's advise sink, through which the calling object receives notification when the compound document object is renamed, saved, or closed.

```
HRESULT Advise(
    IAdviseSink *pAdvSink,        //Pointer to advisory sink
    DWORD *pdwConnection          //Pointer to a token
  );
```

Parameters

pAdvSink
[in] Points to the advise sink of the calling object.

pdwConnection
[out] Points to a token that can be passed to **IOleObject::Unadvise** to delete the advisory connection.

Return Values

S_OK
Advisory connection is successfully established.

E_OUTOFMEMORY
Attempt to establish advisory connection fails, owing to insufficient memory.

Comments

The **Advise** method sets up an advisory connection between an object and its container, through which the object informs the container's advise sink of close, save, rename, and link-source change events in the object. A container calls this method, normally as part of initializing an object, to register its advisory sink with the object. In return, the object sends the container compound-document notifications by calling **IAdviseSink** or **IAdviseSink2**.

If container and object successfully establish an advisory connection, the object receiving the call returns a nonzero value through *pdwConnection* to the container. If the attempt to establish an advisory connection fails, the object returns zero. To delete an advisory connection, the container calls **IOleObject::Unadvise** and passes this nonzero token back to the object.

An object can delegate the job of managing and tracking advisory events to an OLE advise holder, to which you obtain a pointer by calling **CreateOleAdviseHolder**. The returned **IOleAdviseHolder** interface has three methods for sending advisory notifications, as well as **Advise**, **Unadvise**, and **EnumAdvise** methods that are identical to those for **IOleObject**. Calls to **IOleObject:Advise**, **Unadvise**, or **EnumAdvise** are delegated to corresponding methods in the advise holder.

To destroy the advise holder, simply call Release on the **IOleAdviseHolder** interface.

See Also

CreateOleAdviseHolder, **IOleObject::UnAdvise**, **IOleObject::EnumAdvise**, **IOleAdviseHolder::Advise**

IOleObject::Close

Changes an embedded object from the running to the loaded state. Disconnects a linked object from its link source.

```
HRESULT Close(
   DWORD dwSaveOption       //Indicates whether to save object before closing
  );
```

Parameter

dwSaveOption
[in] Indicates whether the object is to be saved as part of the transition to the loaded state. Valid values are taken from the enumeration **OLECLOSE**.

Note The OLE 2 user model recommends that object applications do not prompt users before saving linked or embedded objects, including those activated in place. This policy represents a change from the OLE 1 user model, in which object applications always prompt the user to decide whether to save changes.

Return Values

S_OK
The object closed successfully.

OLE_E_PROMPTSAVECANCELLED
The user was prompted to save but chose the Cancel button from the prompt message box.

Comments

Notes to Callers

A container application calls **IOleObject::Close** when it wants to move the object from a running to a loaded state. Following such a call, the object still appears in its container but is not open for editing. Calling **IOleObject::Close** on an object that is loaded but not running has no effect. Closing a linked object simply means disconnecting it.

Notes to Implementors

Upon receiving a call to **IOleObject::Close**, a running object should do the following:

- If the object has been changed since it was last opened for editing, it should request to be saved, or not, according to instructions specified in *dwSaveOption*. If the option is to save the object, then it should call its container's **IOleClientSite::SaveObject** interface.
- If the object has **IDataObject::DAdvise** connections with ADVF_DATAONSTOP flags, then it should send an OnDataChange notification. See **IDataObject::DAdvise** for details.
- If the object currently owns the Clipboard, it should empty it by calling **OleFlushClipboard**.
- If the object is currently visible, notify its container by calling **IOleClientSite::OnShowWindow** with the *fshow* argument set to FALSE.
- Send **IAdvise::OnClose** notifications to appropriate advise sinks.
- Finally, forcibly cut off all remoting clients by calling **CoDisconnectObject**.

If the object application is a local server (an EXE rather than a DLL), closing the object should also shut down the object application unless the latter is supporting other running objects or has another reason to remain in the running state. Such reasons might include the presence of **IClassFactory::LockServer** locks, end-user control of the application, or the existence of other open documents requiring access to the application.

Calling **IOleObject::Close** on a linked object disconnects it from, but does not shut down, its source application. A source application that is visible to the user when the object is closed remains visible and running after the disconnection and does not send an **OnClose** notification to the link container.

See Also

CoDisconnectObject, **IAdviseSink::OnClose**, **IClassFactory::LockServer**, **IDataObject::DAdvise**, **IOleClientSite::OnShowWindow**, **IOleClientSite::SaveObject**, **OLECLOSE**, **OleFlushClipboard**

IOleObject::DoVerb

Requests an object to perform an action in response to an end-user's action. The possible actions are enumerated for the object in **IOleObject::EnumVerbs**.

```
HRESULT DoVerb(
  LONG iVerb,                       //Value representing verb to be performed
  LPMSG lpmsg,                      //Ptr to Windows message
  IOleClientSite *pActiveSite,      //Ptr to active client site
  LONG lindex,                      //Reserved for future use
  HWND hwndParent,                  //Handle of window containing the object
  LPCRECT lprcPosRect               //Ptr to object's display rectangle
 );
```

Parameters

iVerb
[in] The number assigned to the verb in the **OLEVERB** structure returned by **IOleObject::EnumVerbs**.

lpmsg
[in] Points to the MSG structure describing the event (such as a double-click) that invoked the verb.

pActiveSite
[in] Points to the object's active client site, where the event occurred that invoked the verb.

lindex
[in] Reserved for future use; should be zero.

hwndParent
[in] Handle of the document window containing the object. This and *lprcPosRect* together make it possible to open a temporary window for an object, where *hwndParent* is the parent window in which the object's window is to be displayed, and *lprcPosRect* defines the area available for displaying the object window within that parent. A temporary window is useful, for example, to a multimedia object that opens itself for playback but not for editing.

lprcPosRect
[in] Points to the RECT structure containing the coordinates, in pixels, that define an object's bounding rectangle in *hwndParent*. This and *hwndParent* together enable opening multimedia objects for playback but not for editing.

Return Values

S_OK
Object successfully invoked specified verb.

OLE_E_NOT_INPLACEACTIVE
iVerb set to OLEIVERB_UIACTIVATE or OLEIVERB_INPLACEACTIVATE and object is not already visible.

OLE_E_CANT_BINDTOSOURCE
The object handler or link object cannot connect to the link source.

DV_E_LINDEX
Invalid *lindex*.

OLEOBJ_S_CANNOT_DOVERB_NOW
The verb is valid, but in the object's current state it cannot carry out the corresponding action.

OLEOBJ_S_INVALIDHWND
DoVerb was successful but *hwndParent* is invalid.

OLEOBJ_E_NOVERBS
The object does not support any verbs.

OLEOBJ_S_INVALIDVERB

Object does not recognize a positive verb number. Verb is treated as OLEIVERB_PRIMARY.

MK_E_CONNECT
Link source is across a network that is not connected to a drive on this machine.

OLE_E_CLASSDIFF
Class for source of link has undergone a conversion.

E_NOTIMPL
Object does not support in-place activation or does not recognize a negative verb number.

Comments

A "verb" is an action that an OLE object takes in response to a message from its container. An object's container, or a client linked to the object, normally calls **IOleObject::DoVerb** in response to some end-user action, such as double-clicking on the object. The various actions that are available for a given object are enumerated in an **OLEVERB** structure, which the container obtains by calling **IOleObject::EnumVerbs**. **IOleObject::DoVerb** matches the value of *iVerb* against the *iVerb* member of the structure to determine which verb to invoke.

Through **IOleObject::EnumVerbs**, an object, rather than its container, determines which verbs (i.e., actions) it supports. OLE 2 defines seven verbs that are available, but not necessarily useful, to all objects. In addition, each object can define additional verbs that are unique to it. The following table describes the verbs defined by OLE:

Verb	**Description**
OLEIVERB_PRIMARY (0L)	Specifies the action that occurs when an end-user double-clicks the object in its container. The object, not the container, determines this action. If the object supports in-place activation, the primary verb usually activates the object in place.
OLEIVERB_SHOW (-1)	Instructs an object to show itself for editing or viewing. Called to display newly inserted objects for initial editing and to show link sources. Usually an alias for some other object-defined verb.
OLEIVERB_OPEN (-2)	Instructs an object, including one that otherwise supports in-place activation, to open itself for editing in a window separate from that of its container. If the object does not support in-place activation, this verb has the same semantics as OLEIVERB_SHOW.
OLEIVERB_HIDE (-3)	Causes an object to remove its user interface from the view. Applies only to objects that are activated in-place.
OLEIVERB_UIACTIVATE (-4)	Activates an object in place, along with its full set of user-interface tools, including menus, toolbars, and its name in the title bar of the container window. If the object does not support in-place activation, it should return E_NOTIMPL.
OLEIVERB_INPLACEACTIVATE (-5)	Activates an object in place without

	displaying tools, such as menus and toolbars, that end-users need to change the behavior or appearance of the object. Single-clicking such an object causes it to negotiate the display of its user-interface tools with its container. If the container refuses, the object remains active but without its tools displayed.
OLEIVERB_DISCARDUNDOSTATE (-6)	Used to tell objects to discard any undo state that they may be maintaining without deactivating the object.

Note to Callers

Containers call **IOleObject::DoVerb** as part of initializing a newly created object. Before making the call, containers should first call **IOleObject::SetClientSite** to inform the object of its display location and **IOleObject::SetHostNames** to alert the object that it is an embedded object and to trigger appropriate changes to the user interface of the object application in preparation for opening an editing window.

Like the **OleActivate** function in OLE 1, **IOleObject::DoVerb** automatically runs the OLE server application. If an error occurs during verb execution, the object application is shut down.

If an end-user invokes a verb by some means other than selecting a command from a menu (say, by double-clicking or, more rarely, single-clicking an object), the object's container should pass a pointer (*lpmsg*) to a Windows **MSG** structure containing the appropriate message. For example, if the end-user invokes a verb by double-clicking the object, the container should pass a **MSG** structure containing WM_LBUTTONDBLCLK, WM_MBUTTONDBLCLK, or WM_RBUTTONDBLCLK. If the container passes no message, *lpmsg* should be set to NULL. The object should ignore the *hwnd* member of the passed **MSG** structure, but can use all the other **MSG** members.

If the object's embedding container calls **IOleObject::DoVerb**, the client-site pointer (*pClientSite*) passed to **DoVerb** is the same as that of the embedding site. If the embedded object is a link source, the pointer passed to **DoVerb** is that of the linking client's client site.

When **IOleObject::DoVerb** is invoked on an OLE link, it may return OLE_E_CLASSDIFF or MK_CONNECTMANUALLY. The link object returns the former error when the link source has been subjected to some sort of conversion while the link was passive. The link object returns the latter error when the link source is located on a network drive that is not currently connected to the caller's computer. The only way to connect a link under these conditions is to first call **QueryInterface**, ask for **IOleLink**, allocate a bind context, and run the link source by calling **IOleLink::BindToSource**.

Container applications that do not support general in-place activation can still use the *hwndParent* and *lprcPosRect* parameters to support in-place playback of multimedia files. Containers must pass valid *hwndParent* and *lprcPosRect* parameters to **IOleObject::DoVerb**.

Some code samples pass a *lindex* value of -1 instead of zero. The value -1 works but should be avoided in favor of zero. The *lindex* parameter is a reserved parameter, and for reasons of consistency Microsoft recommends assigning a zero value to all reserved parameters.

Notes to Implementors

In addition to the above verbs, an object can define in its **OLEVERB** structure additional verbs that are specific to itself. Positive numbers designate these object-specific verbs. An object should treat any unknown *positive* verb number as if it were the primary verb and return OLE_S_INVALIDVERB to the calling function. The object should ignore verbs with negative numbers that it does not recognize and return E_NOTIMPL.

If the verb being executed places the object in the running state, you should register the object in the Running Object Table (ROT) even if its server application doesn't support linking. Registration is important because the object at some point may serve as the source of a link in a container that supports links to embeddings. Registering the object with the ROT enables the link client to get a pointer to the object directly, instead of having to go through the object's container. To perform the registration, call **IOleClientSite::GetMoniker** to get the full moniker of the object, call the **GetRunningObjectTable** API function to get a pointer to the ROT, and then call **IRunningObjectTable::Register**.

Note When the object leaves the running state, remember to revoke the object's registration with the ROT by calling **IOleObject::Close**. If the object's container document is renamed while the object is running, you should revoke the object's registration and re-register it with the ROT, using its new name. The container should inform the object of its new moniker either by calling **IOleObject::SetMoniker** or by responding to the object's calling **IOleClientSite::GetMoniker**.

When showing a window as a result of **DoVerb**, it is very important for the object to explicitly call **SetForegroundWindow** on its editing window. This ensures that the object's window will be visible to the user even if another process originally obscured it. For more information about **SetForegroundWindow** and **SetActiveWindow**, see the *Win32 SDK*.

See Also

GetRunningObjectTable, **IOleClientSite::GetMoniker**, **IOleLink::BindToSource**, **IOleObject::Close**, **IOleObject::EnumVerbs**, **IOleObject::GetMoniker**, **IOleObject::SetMoniker**, **IRunningObjectTable::Register**, **OleRun**

SetForegroundWindow, **SetActiveWindow** in Win32

IOleObject::EnumAdvise

Enumerates the advisory connections registered for an object, so a container can know what to release prior to closing down.

```
HRESULT EnumAdvise(
  IEnumSTATDATA **ppenumAdvise        //Pointer to storage of a struct
 );
```

Parameter

ppenumAdvise
[out] Points to where the new enumerator should be returned. NULL is a legal return value, indicating that the object does not have any advisory connections. If an error is returned, this parameter must be set to NULL. Each time an object receives a call to **EnumAdvise**, it must increase the reference count on the pointer it returns. It is the caller's responsibility to call Release when it is done with the pointer.

Return Values

S_OK
Enumerator returned successfully.

E_FAIL
Unspecified error encountered.

E_NOTIMPL
EnumAdvise is not implemented.

Comments

The **EnumAdvise** method provides a way for containers to keep track of advisory connections registered for their objects. A container normally would call this function so that it can instruct an object to release each of its advisory connections prior to closing down.

IOleObject::EnumAdvise enumerates items of type **STATDATA**. Upon receiving the pointer, the container can then loop through **STATDATA** and call **IOleObject::Unadvise** for each enumerated connection.

The usual way to implement this function is to delegate the call to the **IOleAdviseHolder** interface. Only the *pAdvise* and *dwConnection* members of **STATDATA** are relevant for **IOleObject::EnumAdvise**.

See Also

IOleObject::Advise, **IOleObject::UnAdvise**

IOleObject::EnumVerbs

Exposes a pull-down menu listing the verbs available for an object in ascending order by verb number.

HRESULT EnumVerbs(
  IEnumOleVerb ***ppEnumOleVerb* //Ptr to storage of enumerator
 );

Parameter

ppEnumOleVerb
  [out] Points to where the new enumerator should be returned. If an error is returned, this parameter must be set to NULL. Each time an object receives a call to **EnumVerbs**, it must increase the reference count on the pointer the method returns. It is the caller's responsibility to call Release when it is done with the pointer.

Return Values

S_OK
  Verb(s) enumerated successfully.

OLE_S_USEREG
  Delegate to the default handler to use the entries in the registry to provide the enumeration.

OLEOBJ_E_NOVERBS
  Object does not support any verbs.

Comments

Notes to Callers

Containers call this method to expose a pull-down menu of the verbs available for their embedded objects. You may want your container to call **IOleObject::EnumVerbs** each and every time such a menu is selected in order to enable such objects as media clips, whose verbs may change while they are running, to update their menus. The default verb for a media clip, for example, changes from "Play" before it is activated to "Stop" once it is running.

Notes to Implementors

The default handler's implementation of **IOleObject::EnumVerbs** uses the registry to enumerate an object's verbs. If an object application is to use the default handler's implementation, it should return OLE_S_USEREG.

The enumeration returned is of type **IEnumOLEVERB**:

```
typedef Enum < OLEVERB > IEnumOLEVERB;
```

where **OLEVERB** is defined as:

```
typedef struct tagOLEVERB
{
     LONG    iVerb;
     LPOLESTR   lpszVerbName;
     DWORD   fuFlags;
     DWORD   grfAttribs;
} OLEVERB;
```

The following table describes the members of the **OLEVERB** structure:

OLEVERB Member	Description
iVerb	Verb number being enumerated. If the object supports OLEIVERB_OPEN, OLEIVERB_SHOW

and/or OLEIVERB_HIDE (or another predefined verb), these will be the first verbs enumerated, since they have the lowest verb numbers.

lpszVerbName

Name of the verb.

In Windows, this value, along with optional embedded ampersand characters to indicate accelerator keys, can be passed to the **AppendMenu** function.

On the Macintosh, the following metacharacters may be passed along with this value:

- ! marks the menu item with the subsequent character
- < sets the character style of the item
- (disables the item.

The metacharacters / and ^ are not permitted.

fuFlags

In Windows, a group of flags taken from the flag constants beginning with MF_ defined in **AppendMenu**. Containers should use these flags in building an object's verb menu. All Flags defined in **AppendMenu** are supported except for:

- MF_BITMAP
- MF_OWNERDRAW
- MF_POPUP

grfAttribs

In Windows, a group of flag bits taken from the enumeration **OLEVERBATTRIB**. The flag OLEVERBATTRIB_NEVERDIRTIES indicates that executing this verb will not cause the object to become dirty and is therefore in need of saving to persistent storage.

OLEVERBATTRIB_ONCONTAINERMENU indicates that this verb should be placed on the container's menu of object verbs when the object is selected. OLEIVERB_HIDE, OLEIVERB_SHOW, and OLEIVERB_OPEN never have this value set.

For more information on the Windows **AppendMenu** function, see the Microsoft *Win32 SDK*.

See Also

IOleObject::DoVerb, **OleRegEnumVerbs**

IOleObject::GetClientSite

Obtains a pointer to an embedded object's client site.

```
HRESULT GetClientSite(
  IOleClientSite **ppClientSite      //Storage of pointer to object's client site
 );
```

Parameter

ppClientSite

[out] Points to an address where the returned client-site pointer is stored. If an object does not yet know its client site, or an error has occurred, this parameter must be set to NULL. Each time an object receives a call to **GetClientSite**, it must increase the reference count on the pointer the method returns. It is the caller's responsibility to call Release when it is done with the pointer.

Return Value

S_OK

Client site pointer returned successfully.

Comments

Link clients most commonly call the **IOleObject::GetClientSite** method in conjunction with the **IOleClientSite::GetContainer** method to traverse a hierarchy of nested objects. A link client calls **IOleObject::GetClientSite** to get a pointer to the link source's client site. The client then calls **IOleClientSite::GetContainer** to get a pointer to the link source's container. Finally, the client calls **IOleContainer::QueryInterface** to get **IOleObject** and **IOleObject::GetClientSite** to get the container's client site within its container. By repeating this sequence of calls, the caller can eventually retrieve a pointer to the master container in which all the other objects are nested.

Notes to Callers

The returned client-site pointer will be NULL if an embedded object has not yet been informed of its client site. This will be the case with a newly loaded or created object when a container has passed a NULL client-site pointer to one of the object-creation helper functions but has not yet called **IOleObject::SetClientSite** as part of initializing the object.

See Also

IOleObject::SetClientSite

IOleObject::GetClipboardData

Retrieves a data object containing the current contents of the embedded object on which this method is called. Using the pointer to this data object, it is possible to create a new embedded object with the same data as the original.

```
HRESULT GetClipboardData(
  DWORD dwReserved,              //Not currently implemented
  IDataObject **ppDataObject     //Pointer to storage of interface pointer
 );
```

Parameters

dwReserved
[in] Reserved for future use; must be zero.

ppDataObject
[out] Points to an interface pointer to the data object. If an error is returned, this parameter must be set to NULL. Each time an object receives a call to **GetClipboardData**, it must increase the reference count on the pointer that the method returns. It is the caller's responsibility to call Release when it is done with the pointer.

Return Values

S_OK
The data transfer object is successfully returned.

E_NOTIMPL
GetClipboardData is not supported.

OLE_E_NOTRUNNING
The object is not running.

Comments

You can use the **GetClipboardData** method to convert a linked object to an embedded object, in which case the container application would call **IOleObject::GetClipboardData** and then pass the data received to **OleCreateFromData**. This method returns a pointer to a data object that is identical to what would have been passed to the Clipboard by a standard copy operation.

Notes to Callers

If you want a stable snapshot of the current contents of an embedded object, call **IOleObject::GetClipboardData**. Should the data change, you will need to call the function again for an updated snapshot. If you want the caller to be informed of changes that occur to the data, call **IDataObject::QueryInterface**, then call **IDataObject::DAdvise**.

Notes to Implementors

If you implement this function, you must return an **IDataObject** pointer for an object whose data will not change.

See Also

IDataObject, **IOleObject::InitFromData**, **IUnknown::QueryInterface**, **OleCreateFromData**

IOleObject::GetExtent

Retrieves a running object's current display size.

```
HRESULT GetExtent(
   DWORD dwDrawAspect,          //Value indicating object aspect
   SIZEL *psizel                //Pointer to storage of object size limit
  );
```

Parameters

dwDrawAspect
[in] Describes the aspect of the object whose limit is to be retrieved; the value is obtained from the enumeration **DVASPECT**. The most common value for this method is DVASPECT_CONTENT, which specifies a full rendering of the object within its container.

psizel
[out] Points to where the object's size is to be returned.

Return Values

S_OK
Extent information successfully returned.

E_INVALIDARG
Invalid value for *dwAspect.*

Comments

A container calls **IOleObject::GetExtent** on a running object to retrieve its current display size. If the container can accommodate that size, it will normally do so because the object, after all, knows what size it should be better than the container does. A container normally makes this call as part of initializing an object.

The display size returned by **IOleObject::GetExtent** may differ from the size last set by **IOleObject::SetExtent** because the latter method dictates the object's display space at the time the method is called but does not necessarily change the object's native size, as determined by its application.

Notes to Callers

Because a container can make this call only to a running object, the container must instead call **IViewObject2::GetExtent** if it wants to get the display size of a loaded object from its cache.

Notes to Implementors

Implementation consists of filling the *sizel* structure with an object's height and width.

See Also

IOleObject::SetExtent, **IViewObject2::GetExtent**

IOleObject::GetMiscStatus

Returns a value indicating the status of an object at creation and loading.

```
HRESULT GetMiscStatus(
   DWORD dwAspect,          //Value indicating object aspect
   DWORD *pdwStatus         //Pointer to storage of status information
  );
```

Parameters

dwAspect
[in] The aspect of an object about which status information is being requested. The value is obtained from the enumeration **DVASPECT** (see "FORMATETC Data Structure").

pdwStatus
[out] Points to where the status information is returned. May not be NULL.

Return Values

S_OK
Status information returned successfully.

OLE_S_USEREG
Delegate the retrieval of miscellaneous status information to the default handler's implementation of this method.

CO_E_CLASSNOTREG
There is no CLSID registered for the object.

CO_E_READREGDB
Error accessing the registry.

Comments

A container normally calls **IOleObject::GetMiscStatus** when it creates or loads an object in order to determine how to display the object and what types of behaviors it supports.

Objects store status information in the registry. If the object is not running, the default handler's implementation of **IOleObject::GetMiscStatus** retrieves this information from the registry. If the object is running, the default handler invokes **IOleObject::GetMiscStatus** on the object itself.

The information that is actually stored in the registry varies with individual objects. The status values to be returned are defined in the enumeration **OLEMISC**.

Notes to Implementors

Implementation normally consists of delegating the call to the default handler.

See Also

DVASPECT, **FORMATETC**, **OLEMISC**

IOleObject::GetMoniker

Returns a embedded object's moniker, which the caller can use to link to the object.

```
HRESULT GetMoniker(
  DWORD dwAssign,            //Specifies how moniker is assigned
  DWORD dwWhichMoniker,      //Specifies which moniker is assigned
  IMoniker **ppmk            //Storage of returned pointer
 );
```

Parameters

dwAssign

[in] Determines how the moniker is assigned to the object. Depending on the value of *dwAssign*, **IOleObject::GetMoniker** does one of the following:

- Obtains a moniker only if one has already been assigned,
- Forces assignment of a moniker, if necessary, in order to satisfy the call, or
- Obtains a temporary moniker.

Values for *dwAssign* are specified in the enumeration **OLEGETMONIKER**.

Note You cannot pass OLEGETMONIKER_UNASSIGN when calling **IOleObject::GetMoniker**. This value is valid only when calling **IOleClientSite::GetMoniker**.

dwWhichMoniker

[in] Specifies the form of the moniker being requested. Valid values are taken from the enumeration **OLEWHICHMK**.

ppmk

[out] Points to an address where to return the object's moniker. If an error is returned, this parameter must be set to NULL. Each time an object receives a call to GetMoniker, it must increase the reference count on the pointer the method returns. It is the caller's responsibility to call Release when it is done with the pointer.

Comments

The **IOleObject::GetMoniker** method returns an object's moniker. Like **IOleObject::SetMoniker**, this method is important only in the context of managing links to embedded objects and even in that case is optional. A potential link client that requires an object's moniker to bind to the object can call this method to obtain that moniker. The default implementation of **IOleObject::GetMoniker** calls the **IOleClientSite::GetMoniker**, returning E_UNEXPECTED if the object is not running or does not have a valid pointer to a client site.

See Also

CreateItemMoniker, **IOleClientSite::GetMoniker**, **IOleObject::SetMoniker**, **OLEGETMONIKER**, **OLEWHICHMK**

IOleObject::GetUserClassID

Returns an object's class identifier, the CLSID corresponding to the string identifying the object to an end user.

```
HRESULT GetUserClassID(
   CLSID *pClsid          //Pointer to the class identifier
  );
```

Parameter

pClsid

[out] Points to the class identifier (CLSID) to be returned. An object's CLSID is the binary equivalent of the user-type name returned by **IOleObject::GetUserType**.

Return Values

S_OK

CLSID returned successfully.

E_FAIL

An unspecified error occurred.

Comments

GetUserClassID returns the CLSID associated with the object in the registration database. Normally, this value is identical to the CLSID stored with the object, which is returned by **IPersist::GetClassID**. For linked objects, this is the CLSID of the last bound link source. If the object is running in an application different from the one in which it was created and for the purpose of being edited is emulating a class that the container application recognizes, the CLSID returned will be that of the class being emulated rather than that of the object's own class.

See Also

IOleObject::GetUserType, **IPersist::GetClassID**, **OleDoAutoConvert**, **OleGetAutoConvert**, **OleSetAutoConvert**, **GetConvertStg**, **SetConvertStg**

IOleObject::GetUserType

Retrieves the user-type name of an object for display in user-interface elements such as menus, list boxes, and dialog boxes.

```
HRESULT GetUserType(
   DWORD dwFormOfType,          //Specifies form of type name
   LPOLESTR *pszUserType        //Pointer to storage of string pointer
  );
```

Parameters

dwFormOfType
: [in] A value specifying the form of the user-type name to be presented to users. Valid values are obtained from the **USERCLASSTYPE** enumeration.

pszUserType
: [out] Points to where a pointer to the user-type string will be placed. The caller must free *lpszUserType* using the current **IMalloc** instance. If an error is returned, this parameter must be set to NULL.

Return Values

S_OK
: The object's user-type name is successfully returned.

OLE_S_USEREG
: Delegate to the default handler's implementation using the registry to provide the requested information.

Comments

Containers call **IOleObject::GetUserType** in order to represent embedded objects in list boxes, menus, and dialog boxes by their normal, user-recognizable names. Examples include "Word Document," "Excel Chart," and "Paintbrush Object." The information returned by **IOleObject::GetUserType** is the user-readable equivalent of the binary class identifier returned by **IOleObject::GetUserClassID**.

Notes to Callers

The default handler's implementation of **IOleObject::GetUserType** uses the object's class identifier (the *pClsid* parameter returned by **IOleObject::GetUserClassID**) and the *dwFormOfType* parameter together as a key into the registry. If an entry is found that matches the key exactly, then the user type specified by that entry is returned. If only the CLSID part of the key matches, then the lowest-numbered entry available (usually the full name) is used. If the CLSID is not found, or there are no user types registered for the class, the user type currently found in the object's storage is used.

Notes to Implementors

You can use the implementation provided by the default handler by returning OLE_S_USEREG as your application's implementation of this method. If the user type name is an empty string, the message "Unknown Object" is returned.

You can call the OLE helper function **OleRegGetUserType** to return the appropriate user type.

See Also

IOleObject::SetHostNames, **IOleObject::GetUserClassID**, **OleRegGetUserType**, **ReadFmtUserTypeStg**, **USERCLASSTYPE**

IOleObject::InitFromData

Initializes a newly created object with data from a specified data object, which can reside either in the same container or on the Clipboard.

```
HRESULT InitFromData(
   IDataObject *pDataObject,        //Pointer to data object
   BOOL fCreation,                  //Specifies how object is created
   DWORD dwReserved                 //Not used in OLE 2
  );
```

Parameters

pDataObject
[in] Points to the data object from which the initialization data is to be obtained. This parameter can be NULL, which indicates that the caller wants to know if it is worthwhile trying to send data; that is, whether the container is capable of initializing an object from data passed to it. The data object to be passed can be based on either the current selection within the container document or on data transferred to the container from an external source.

fCreation
[in] TRUE indicates the container is inserting a new object inside itself and initializing that object with data from the current selection; FALSE indicates a more general programmatic data transfer, most likely from a source other than the current selection.

dwReserved
[in] Reserved for future use; must be zero.

Return Values

S_OK
If *pDataObject* is not NULL, the object successfully attempted to initialize itself from the provided data; if *pDataObject* is NULL, the object is able to attempt a successful initialization .

S_FALSE
If *pDataObject* is not NULL, the object made no attempt to initialize itself; if *pDataObject* is NULL, the object cannot attempt to initialize itself from the data provided.

E_NOTIMPL
The object does not support **InitFromData**.

OLE_E_NOTRUNNING
The object is not running and therefore cannot perform the operation.

Comments

This method enables a container document to insert within itself a new object whose content is based on a current data selection within the container. For example, a spreadsheet document may want to create a graph object based on data in a selected range of cells.

Using this method, a container can also replace the contents of an embedded object with data transferred from another source. This provides a convenient way of updating an embedded object.

Notes to Callers

Following initialization, the container should call **IOleObject::GetMiscStatus** to check the value of the **OLEMISC_INSERTNOTREPLACE** bit. If the bit is on, the new object inserts itself following the selected data. If the bit is off, the new object replaces the selected data.

Notes to Implementors

A container specifies whether to base a new object on the current selection by passing either TRUE or FALSE to the *fCreation* parameter.

If *fCreation* is TRUE, the container is attempting to create a *new* instance of an object, initializing it with the selected data specified by the data object.

If *fCreation* is FALSE, the caller is attempting to replace the object's current contents with that pointed to by *pDataObject*. The usual constraints that apply to an object during a paste operation should be applied here. For example, if the type of the data provided is unacceptable, the object should fail to initialize and return S_FALSE.

If the object returns S_FALSE, it cannot initialize itself from the provided data.

See Also

IOleObject::GetMiscStatus, **IDataObject::SetData**

IOleObject::IsUpToDate

Checks recursively whether or not an object is up to date.

HRESULT IsUpToDate();

Return Values

S_OK
Object is up to date.

S_FALSE
Object is not up to date.

OLE_E_UNAVAILABLE
Status of object cannot be determined in a timely manner.

Comments

The **IsUpToDate** method provides a way for containers to check recursively whether or not all objects are up to date.That is, when the container calls this method on the first object, the object in turn calls it for all its own objects, and they in turn for all of theirs, until all objects have been checked.

Notes to Implementors

Because of the recursive nature of **IOleObject:IsUpToDate**, determining whether an object is out-of-date, particularly one containing one or more other objects, can be as time-consuming as simply updating the object in the first place. If you would rather avoid lengthy queries of this type, make sure that **IOleObject::IsUpToDate** returns OLE_E_UNAVAILABLE. In cases where the object to be queried is small and contains no objects itself, thereby making an efficient query possible, this method can return either S_OK or S_FALSE.

See Also

IOleObject::UpDate

IOleObject::SetClientSite

Informs an embedded object of its display location, called a "client site," within its container.

```
HRESULT SetClientSite(
  IOleClientSite *pClientSite        //Pointer to an embedded object's client site
 );
```

Parameter

pClientSite
[in] Points to the container application's client-site interface.

Return Values

S_OK
Client site successfully set.

E_UNEXPECTED
Object is not embedded in a container.

Comments

Within a compound document, each embedded object has its own client site – the place where it is displayed and through which it receives information about its storage, user interface, and other resources. **IOleObject::SetClientSite** is the only method enabling an embedded object to obtain a pointer to its client site.

Notes to Callers

A container can notify an object of its client site either at the time the object is created or, subsequently, when the object is initialized.

When creating or loading an object, a container may pass a client-site pointer (along with other arguments) to one of the following helper functions: **OleCreate**, **OleCreateFromFile**, **OleCreateFromData** or **OleLoad**. These helper functions load an object handler for the new object and call **IOleObject::SetClientSite** on the container's behalf before returning a pointer to the new object.

Passing a client-site pointer informs the object handler that the client site is ready to process requests. If the client site is unlikely to be ready immediately after the handler is loaded, you may want your container to pass a NULL client-site pointer to the helper function. The NULL pointer says that that no client site is available and thereby defers notifying the object handler of the client site until the object is initialized. In response, the helper function returns a pointer to the object, but upon receiving that pointer the container must call **SetClientSite** as part of initializing the new object.

Notes to Implementors

Implementation consists simply of incrementing the reference count on, and storing, the pointer to the client site.

See Also

IOleClientSite, **IOleObject::GetClientSite**, **OleCreate**, **OleCreateFromFile**, **OleCreateFromData**, **OleLoad**

IOleObject::SetColorScheme

Specifies the color palette that the object application should use when it edits the specified object.

```
HRESULT SetColorScheme(
   LOGPALETTE *pLogpal       //Pointer to a structure
  );
```

Parameter

pLogpal
[in] Points to a **LOGPALETTE** structure that specifies the recommended palette.

Return Values

S_OK
Color palette received successfully.

E_NOTIMPL
Object does not support setting palettes.

OLE_E_PALETTE
Invalid **LOGPALETTE** structure pointed to by *lpLogPal*.

OLE_E_NOTRUNNING
Object must be running to perform this operation.

Comments

The **IOleObject::SetColorScheme** method sends the container application's recommended color palette to the object application, which is not obliged to use it.

Notes to Implementors

Upon receiving the palette, the object application should:

1. Allocate and fill in its own **LOGPALETTE** structure with the colors specified in the container application's **LOGPALETTE** structure.
2. Call **CreatePalette** to create a palette from the resulting **LOGPALETTE** structure. This palette can be used to render objects and color menus as the user edits objects in the document.

The first palette entry in the **LOGPALETTE** structure specifies the foreground color recommended by the container. The second palette entry specifies the recommended background color. The first half of the remaining palette entries are fill colors and the second half are colors for the lines and text.

Container applications typically specify an even number of palette entries. If a container specifies an odd number of entries, the object application should assume that the first half of the total entries plus one designate fill colors, while the remainder designate line and text colors. For example, if there are five entries, the first three should be interpreted as fill colors and the last two as line and text colors.

IOleObject::SetExtent

Informs an object of how much display space its container has assigned it.

HRESULT SetExtent(
 DWORD *dwDrawAspect*,
 SIZEL **pSizel*
);

Parameters

dwDrawAspect
 [in] Describes which form, or "aspect," of an object is to be displayed. The object's container obtains this value from the enumeration **DVASPECT** (see "**FORMATETC** Data Structure"). The most common aspect is DVASPECT_CONTENT, which specifies a full rendering of the object within its container. An object can also be rendered as an icon, a thumbnail version for display in a browsing tool, or a print version, which displays the object as it would be rendered using the File Print command.

psizel
 [in] Specifies the size limit for the object.

Return Values

S_OK
 The object has resized successfully.

E_FAIL
 The object's size is fixed.

OLE_E_NOTRUNNING
 The object is not running.

Comments

A container calls **IOleObject::SetExtent** when it needs to dictate to an embedded object the size at which it will be displayed. Often, this call occurs in response to an end user resizing the object window. Upon receiving the call, the object, if possible, should recompose itself gracefully to fit the new window.

Whenever possible, a container seeks to display an object at its finest resolution, sometimes called the object's *native size*. All objects, however, have a default display size specified by their applications, and in the absence of other constraints, this is the size they will use to display themselves. Since an object knows its optimum display size better than does its container, the latter normally requests that size from a running object by calling **IOleObject::GetExtent**. Only in cases where the container cannot accommodate the value returned by the object does it override the object's preference by calling **IOleObject::SetExtent**.

Notes to Callers

You can call **SetExtent** on an object only when the object is running. If a container resizes an object while an object is not running, the container should keep track of the object's new size but defer calling **IOleObject::SetExtent** until a user activates the object. If the OLEMISC_RECOMPOSEONRESIZE bit is set on an object, its container should force the object to run before calling **OleObject::SetExtent**.

As noted above, a container may want to delegate responsibility for setting the size of an object's display site to the object itself, by calling **IOleObject::GetExtent**.

Notes to Implementors

You may want to implement this method so that your object rescales itself to match as closely as possible the maximum space available to it in its container.

If an object's size is fixed, that is, if it cannot be set by its container, **OleObject::SetExtent** should return E_FAIL. This is always the case with linked objects, whose sizes are set by their link sources, not by their containers.

See Also

IAdviseSink::OnViewChange, **IOleObject::GetExtent**, **IViewObject2::GetExtent**

IOleObject::SetHostNames

Provides an object with the name of its container application and the compound document in which it is embedded.

```
HRESULT SetHostNames(
    LPCOLESTR szContainerApp,        //Pointer to name of container application
    LPCOLESTR szContainerObj         //Pointer to name of container document
  );
```

Parameters

szContainerApp
[in] Points to the name of the container application in which the object is running.

szContainerObj
[in] Points to the name of the compound document that contains the object. If you do not wish to display the name of the compound document, you can set this parameter to NULL.

Return Value

S_OK
Window title information set successfully.

Comments

When a container initializes an embedded object, it calls this function to inform the object of the names of both the container application and container document. When the object is opened for editing, it displays these names in the title bar of its window.

Notes to Callers

Call **SetHostNames** only for embedded objects, because for linked objects, the link source supplies its own separate editing window and title bar information.

Notes to Implementors

An object's application of **SetHostNames** should include whatever modifications to its user interface may be appropriate to an object's embedded state. Such modifications typically will include adding and removing menu commands and altering the text displayed in the title bar of the editing window.

The complete window title for an embedded object in an SDI container application or an MDI application with a maximized child window should appear as follows:

<*object application name*> - <*object short type*> in <*container document*>

Otherwise, the title should be:

<*object application name*> - <*container document*>

The "object short type" refers to a form of an object's name short enough to be displayed in full in a list box.

Since these identifying strings are not stored as part of the persistent state of the object, **IOleObject::SetHostNames** must be called each time the object loads or runs.

See Also

IOleObject::GetUserType

IOleObject::SetMoniker

Notifies an object of its container's moniker, the object's own moniker relative to the container, or the object's full moniker.

```
HRESULT SetMoniker(
    DWORD dwWhichMoniker,        //Value specifying moniker being set
    IMoniker *pmk                //Pointer to moniker
  );
```

Parameters

dwWhichMoniker
[in] Specifies which moniker is passed in *pmk*. Values are from the enumeration **OLEWHICHMK**.

pmk
[in] Points to where to return the moniker.

Return Values

S_OK
Moniker successfully set.

E_FAIL
No client site for object.

Comments

A container that supports links to embedded objects must be able to inform an embedded object when its moniker has changed. Otherwise, subsequent attempts by link clients to bind to the object will fail.The **IOleObject::SetMoniker** method provides one way for a container to communicate this information.

The container can pass either its own moniker, an object's moniker relative to the container, or an object's full moniker. In practice, if a container passes anything other than an object's full moniker, each object calls the container back to request assignment of the full moniker, which the object requires to register itself in the running object table.

The moniker of an object relative to its container is stored by the object handler as part of the object's persistent state. The moniker of the object's container, however, must not be persistently stored inside the object because the container can be renamed at any time.

Notes to Callers

A container calls **IOleObject::SetMoniker** when the container has been renamed, and the container's embedded objects currently or can potentially serve as link sources. Containers call **SetMoniker** mainly in the context of linking because an embedded object is already is aware of its moniker. Even in the context of linking, calling this method is optional because objects can call **IOleClientSite::GetMoniker** to force assignment of a new moniker.

Note to Implementors

Upon receiving a call to **SetMoniker**, an object should register its full moniker in the running object table and send OnRename notification to all advise sinks that exist for the object.

See Also

CreateItemMoniker, **IAdviseSink::OnRename**, **IOleClientSite::GetMoniker**, **IOleObject::GetMoniker**

IOleObject::Unadvise

Deletes a previously established advisory connection.

```
HRESULT Unadvise(
  DWORD dwConnection        //A token
 );
```

Parameter

dwConnection
[in] Contains a token of nonzero value, which was previously returned from **IOleObject::Advise** through its *pdwConnection* parameter.

Return Values

S_OK
Advisory connection deleted successfully.

E_FAIL
Unspecified error encountered.

OLE_E_NOCONNECTION
dwConnection is not represent a valid advisory connection.

Comments

Normally, containers call **IOleObject::Unadvise** at shutdown or when an object is deleted. In certain cases, containers can call this method on objects that are running but not currently visible as a way of reducing the overhead of maintaining multiple advisory connections. The easiest way to implement this method is to delegate the call to **IOleAdviseHolder::Unadvise**.

See Also

IOleObject::Advise, **IOleObject::EnumAdvise**, **IOleAdviseHolder::Unadvise**

IOleObject::Update

Updates an object handler's or link object's data or view caches.

HRESULT Update();

Return Values

S_OK
All caches are up to date.

E_FAIL
An unspecified error occurred.

OLE_E_CANT_BINDTOSOURCE
Cannot run object to get updated data. The object is for some reason unavailable to the caller.

CACHE_E_NOCACHE_UPDATED
No caches were updated.

CACHE_S_SOMECACHES_NOTUPDATED
Some caches were not updated.

Comments

The **Update** method provides a way for containers to keep data updated in their linked and embedded objects. A link object can become out-of-date if the link source has been updated. An embedded object that contains links to other objects can also become out of date. An embedded object that does not contain links cannot become out of date because its data is not linked to another source.

Notes to Implementors

When a container calls a link object's **IOleObject::Update** method, the link object finds the link source and gets a new presentation from it. This process may also involve running one or more object applications, which could be time-consuming.

When a container calls an embedded object's **IOleObject::Update** method, it is requesting the object to update all link objects it may contain. In response, the object handler recursively calls **IOleObject::Update** for each of its own linked objects, running each one as needed.

See Also

IOleObject::IsUpToDate

IOleUILinkContainer

The **IOleUILinkContainer** interface is implemented by containers and used by OLE common dialogs. It supports these dialogs by providing the methods needed to manage a container's links.

The **IOleUILinkContainer** methods enumerate the links associated with a container, and specify how they should be updated, automatically or manually. They change the source of a link and obtain information associated with a link. They also open a link's source document, update links, and break a link to the source.

When to Implement

You must implement this interface if you are creating a container application that will use the Links, Change Source, or Update Links dialogs, as well as the Object Properties dialog, which uses this interface indirectly. The Links dialog calls back to the container application to perform OLE functions that manipulate the links within the container.

When to Use

OLE common dialogs use only this interface to manage the properties of a container's links. They can also use it to manage non-OLE (DDE and other container-specific) links.

Methods in Vtable Order

IUnknown Methods	Description
QueryInterface	Returns a pointer to a specified interface.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IOleUILinkContainer Methods	Description
GetNextLink	Enumerates the links in the container.
SetLinkUpdateOptions	Sets update options.
GetLinkUpdateOptions	Determines current update options for the link.
SetLinkSource	Changes the source of the link.
GetLinkSource	Returns Links dialog information about the link.
OpenLinkSource	Opens a link's source.
UpdateLink	Forces a link to connect to its source and update.
CancelLink	Disconnects selected links.

See Also

OleUIEditLinks, **OleUIChangeSource**, **OleUIUpdateLinks**, **OleUIObjectProperties**, **OLEUIEDITLINKS**

IOleUILinkContainer::CancelLink

Disconnects the selected links.

```
HRESULT CancelLink(
   DWORD dwLink          //Unique 32-bit link identifier
  );
```

Parameter

dwLink
A container-defined unique 32-bit identifier for a single link. Containers can use the pointer to the link's container site for this value.

Return Values

S_OK
Successfully disconnected the selected links.

E_FAIL
Unable to disconnect the selected links.

E_INVALIDARG
One or more invalid arguments.

E_OUTOFMEMORY
Insufficient memory.

E_ACCESSDENIED
Insufficient access permissions.

Comments

Notes To Callers

Call **CancelLink** when the user selects the Break Link button from the Links dialog. The link should be converted to a picture. The Links dialog will not be dismissed for OLE links.

Notes To Implementors

For OLE links, **OleCreateStaticFromData** can be used to create a static picture object using the **IDataObject** interface of the link as the source.

See Also

IDataObject, **OleCreateStaticFromData**

IOleUILinkContainer::GetLinkSource

Returns information about a link that can be displayed in the UI.

```
HRESULT GetLinkSource(
    DWORD dwLink,                          //Unique 32-bit link identifier
    LPTSTR FAR* lplpszDisplayName,         //Length of display name portion
    ULONG FAR* lplenFileName,              //Length of filename portion
    LPTSTR FAR* lplpszFullLinkType,        //Full link type string
    LPTSTR FAR* lplpszShortLinkType,       //Short link type string
    BOOL FAR* lpfSourceAvailable,          //Availability of link
    BOOL FAR* lpfIsSelected                //Indicates that link entry should be selected in listbox
  );
```

Parameters

dwLink
A container-defined unique 32-bit identifier for a single link. See **IOleUILinkContainer::GetNextLink**.

lplpszDisplayName
[out] Points to the allocated full link source display name string. The Links dialog will free this string.

lplenFileName
[out] Points to the length of the leading filename portion of the *lplpszDisplayName* string. If the link source is not stored in a file, then **lplenFileName* should be 0. For OLE links, call **IOleLink::GetSourceDisplayName**.

lplpszFullLinkType
[out] Points to the allocated full link type string that is displayed at the bottom of the Links dialog. The Links dialog will free this string. For OLE links, this should be the full User Type name. Use **IOleObject::GetUserType**, specifying USERCLASSTYPE_FULL for *dwFormOfType*.

lplpszShortLinkType
[out] Points to the allocated short link type string that is displayed in the listbox of the Links dialog. The Links dialog will free this string. For OLE links, this should be the short User Type name. Use **IOleObject::GetUserType**, specifying USERCLASSTYPE_SHORT for *dwFormOfType*.

lpfSourceAvailable
Returns FALSE if it is known that a link is unavailable since the link is to some known but unavailable document. Certain options, such as Update Now, are disabled (grayed in the UI) for such cases.

lpfIsSelected
Used to tell the Edit Links dialog that this link's entry should be selected in the dialog's multi-slelection listbox. **OleUIEditLinks** calls this method at least once for each item to be placed in the links list. If none of them return TRUE, then none of them will be selected when the dialog is first displayed. If all of them return TRUE, then all will be displayed. That is, it returns TRUE if this link is currently part of the selection in the underlying document, FALSE if not. Any links that are selected in the underlying document are selected in the dialog; this way, the user can select a set of links and use the dialog to update them or change their source(s) simultaneously.

Return Values

S_OK
Successfully returned link information.

E_FAIL
Unable to return link information.

E_INVALIDARG
One or more invalid arguments.

E_OUTOFMEMORY
Insufficient memory.

E_ACCESSDENIED
Insufficient access permissions.

Comments

Notes To Callers

Call this method during dialog initialization, after returning from the Change Source dialog.

See Also

IOleLink::GetSourceDisplayName, **IOleObject::GetUserType**, **USERCLASSTYPE**, **OleUIChangeSource**, **OLEUICHANGESOURCE**

IOleUILinkContainer::GetLinkUpdateOptions

Determine the current update options for a link.

```
HRESULT GetNextLink(
   DWORD dwLink,                          //Unique 32-bit link identifier
   DWORD FAR * lpdwUpdateOpt              //Address to return update option
  );
```

Parameters

dwLink
A container-defined unique 32-bit identifier for a single link. See **IOleUILinkContainer::GetNextLink**.

lpdwUpdateOpt
The address to return the current update options.

Return Values

S_OK
Successfully determined update options.

E_FAIL
Unable to determine update options.

E_INVALIDARG
One or more invalid arguments.

E_OUTOFMEMORY
Insufficient memory.

E_ACCESSDENIED
Insufficient access permissions.

Comment

Notes To Implementors

Containers can implement this method for OLE links simply by calling **IOleLink::SetUpdateOptions** on the link object.

See Also

IOleUILinkContainer::GetNextLink, **IOleUILinkContainer::SetLinkUpdateOptions**, **IOleLink::SetUpdateOptions**

IOleUILinkContainer::GetNextLink

Enumerates the links in a container.

```
DWORD GetNextLink(
   DWORD dwLink        //Unique 32-bit link-identifier, indicating the current link
  );
```

Parameter

dwLink
A container-defined unique 32-bit identifier for a single link. This value is only passed to other methods on this interface, so it can be any value that uniquely identifies a link to the container. Containers frequently use the pointer to the link's container site object for this value.

Return Values

Returns a container's link identifiers in sequence; NULL if it has returned the last link.

Notes to Callers

Call this method to enumerate the links in a container. If the value passed in *dwLink* is NULL, then the container should return the first link's 32-bit identifier. If *dwLink* identifies the last link in the container, then the container should return NULL.

See Also

IOleUILinkContainer::SetLinkUpdateOptions, **IOleUILinkContainer::GetLinkUpdateOptions**

IOleUILinkContainer::OpenLinkSource

Opens the link's source.

```
HRESULT OpenLinkSource(
   DWORD dwLink          //Unique 32-bit link identifier
  );
```

Parameter

dwLink
A container-defined unique 32-bit identifier for a single link. Containers can use the pointer to the link's container site for this value.

Return Values

S_OK
Successfully opened the link's source.

E_FAIL
Unable to open the links source.

E_INVALIDARG
One or more invalid arguments.

E_OUTOFMEMORY
Insufficient memory.

E_ACCESSDENIED
Insufficient access permissions.

Comments

Notes To Callers

The **OpenLinkSource** method is called when the Open Source button is selected from the Links dialog. For OLE links, call **IOleObject::DoVerb**, specifying OLEIVERB_SHOW for *iVerb*.

See Also

IOleObject::DoVerb, **OLEVERB**

IOleUILinkContainer::SetLinkSource

Changes the source of a link.

```
HRESULT SetLinkSource(
   DWORD dwLink,                 //Unique 32-bit link identifier
   LPTSTR lpszDisplayName,       //Source string to parse
   ULONG FAR* lenFileName,       //The length of the filename portion
   ULONG FAR* pchEaten,          //Number of characters successfully parsed
   BOOL fValidateSource          //Specifies whether or not moniker should be validated
  );
```

Parameters

dwLink
A container-defined unique 32-bit identifier for a single link. See **IOleUILinkContainer::GetNextLink** .

lpszDisplayName
Points to new source string to be parsed.

lenFileName
The length of the leading filename portion of the *lpszDisplayName* string. If the link source is not stored in a file, then *lenFileName* should be 0. For OLE links, call **IOleLink::GetSourceDisplayName**.

pchEaten
[out] Points to the number of characters successfully parsed in *lpszDisplayName*.

fValidateSource
TRUE if the moniker should be validated; for OLE links, **MkParseDisplayName** should be called. FALSE if the moniker should not be validated. If possible, the link should accept the unvalidated source, and mark itself as unavailable.

Return Values

S_OK
Successfully changed the links source.

E_FAIL
Unable to change the links source.

E_INVALIDARG
One or more invalid arguments.

E_OUTOFMEMORY
Insufficient memory.

E_ACCESSDENIED
Insufficient access permissions.

Comments

Notes To Callers

Call this method from the Change Source dialog, with *fValidateSource* initially set to TRUE. Change Source can be called directly or from the Links dialog. If this call to **SetLinkSource** returns an error (e.g., **MkParseDisplayName** failed because the source was unavailable), then you should display an Invalid Link Source message, and the user should be allowed to decide whether or not to fix the source. If the user chooses to fix the source, then the user should be returned to the Change Source dialog with the invalid portion of the input string highlighted. If the user chooses not to fix the source, then **SetLinkSource** should be called a second time with *fValidateSource* set to FALSE, and the user should be returned to the Links dialog with the link marked Unavailable.

See Also

MkParseDisplayName

IOleUILinkContainer::SetLinkUpdateOptions

Sets a link's update options to Automatic (OLEUPDATE_ALWAYS) or Manual (OLEUPDATE_ONCALL).

```
HRESULT SetLinkUpdate(
  DWORD dwLink,            //Unique 32-bit link identifier
  DWORD dwUpdateOpt        //Update options
 );
```

Parameters

dwLink
A container-defined unique 32-bit identifier for a single link. See **IOleUILinkContainer::GetNextLink**.

dwUpdateOpt
Specifies the update options, which can be Automatic (OLEUPDATE_ALWAYS) or Manual (OLEUPDATE_ONCALL).

Return Values

S_OK
Successfully set the links update options.

E_FAIL
Unable to set the links update options.

E_INVALIDARG
One or more invalid arguments.

E_OUTOFMEMORY
Insufficient memory.

E_ACCESSDENIED
Insufficient access permissions.

Comments

The user selects update options from the Links dialog.

Notes To Implementors

Containers can implement this method for OLE links by simply calling **IOleLink::SetUpdateOptions** on the link object.

See Also

IOleUILinkContainer::GetNextLink, **IOleUILinkContainer::GetLinkUpdateOptions**, **IOleLink::SetUpdateOptions**

IOleUILinkContainer::UpdateLink

Forces selected links to connect to their source and retrieve current information.

```
HRESULT UpdateLink(
  DWORD dwLink,            //Link identifier
  DWORD fErrorMessage,     //Determines whether or not caller should display error message
  DWORD fReserved          //Reserved for future use
 );
```

Parameters

dwLink
A container-defined unique 32-bit identifier for a single link. Containers can use the pointer to the link's container site for this value.

fErrorMessage
Determines whether or not the caller (implementor of **IOleUILinkContainer**) should show an error message upon failure to update a link. The Update Links dialog sets this to FALSE. The Object Properties and Links dialogs set it to TRUE.

fReserved
Reserved for future use. Currently, this parameter should always be set to FALSE.

Return Values

S_OK
Successfully updated linked objects.

E_FAIL
Unable to update linked objects.

E_INVALIDARG
One or more invalid arguments.

E_OUTOFMEMORY
Insufficient memory.

E_ACCESSDENIED
Insufficient access permissions.

Comments

Notes To Callers

Call this method with *fErrorMessage* set to TRUE in cases where the user expressly presses a button to have a link updated, that is, presses the Links' Update Now button. Call it with FALSE in cases where the container should never display an error message, that is, where a large set of operations are being performed and the error should be propagated back to the user later, as might occur with the Update links progress meter. Rather than providing one message for each failure, assuming there are failures, provide a single message for all failures at the end of the operation.

Notes To Implementors

For OLE links, call **IOleObject::Update**.

See Also

IOleObject::Update

IOleUILinkInfo

IOleUILinkInfo is an extension of the **IOleUILinkContainer** interface. It returns the time that an object was last updated, which is link information that **IOleUILinkContainer** does not provide.

When To Implement

You must implement this interface so your container can support the "Link" page of the Object Properties dialog. If you are writing a container that does not implement links, you do not need to implement this interface.

Methods in Vtable Order

IUnknown Methods	Description
QueryInterface	Returns a pointer to a specified interface.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IOleUILinkContainer Methods	Description
GetNextLink	Enumerates the links in the container.
SetLinkUpdateOptions	Sets update options.
GetLinkUpdateOptions	Determines current update options for the link.
SetLinkSource	Changes the source of the link.
GetLinkSource	Returns Links dialog information about link.
OpenLinkSource	Opens a link's source.
UpdateLink	Forces a link to connect to its source and update.
CancelLink	Breaks the link.

IOleUILinkInfo Methods	Description
GetLastUpdate	Determines the last time the object was updated, whether automatically or manually.

IOleUILinkInfo::GetLastUpdate

Indicates when the object was last updated.

```
HRESULT GetLastUpdate(
   DWORD dwLink,                        //Object's identifier
   FILETIME FAR * lpLastUpdate          //Time object was last updated
  );
```

Parameters

dwLink
A container-defined unique 32-bit identifier for a single link. Containers can use the pointer to the link's container site for this value.

lpLastUpdate
Points to the time that the object was last updated.

Return Values

S_OK
Successfully returned link information.

E_FAIL
Unable to return link information.

E_INVALIDARG
One or more invalid arguments.

E_OUTOFMEMORY
Insufficient memory.

E_ACCESSDENIED
Insufficient access permissions.

Comment

Notes To Implementors

If the time that the object was last updated is known, copy it to **lpLastUpdate*. If it is not known, then leave **lpLastUpdate* unchanged and Unknown will be displayed in the link page.

IOleUIObjInfo

This interface is implemented by containers and used by the container's Object Properties dialog and by the Convert dialog. It provides information used by the General and View pages of the Object Properties dialog, which display information about the object's size, location, type, and name. It also allows the object to be converted via the Convert dialog. The View page allows the object's icon to be modified from its original form, and its display aspect to be changed (iconic versus content). Optionally, you can have your implementation of this interface allow the scale of the object to be changed.

When To Implement

You must implement this interface so your container application can support the **OleUIObjectProperties** function and the dialog that it implements.

When To Use

Use this interface when you need to get and set information required by the Object Properties dialog, and to support the Convert dialog.

Methods in Vtable Order

IUnknown Methods	Description
QueryInterface	Returns a pointer to a specified interface.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IOleUILinkContainer Methods	Description
GetNextLink	Enumerates the links in the container.
SetLinkUpdateOptions	Sets update options.
GetLinkUpdateOptions	Determines current update options for the link.
SetLinkSource	Changes the source of the link.
GetLinkSource	Returns Links dialog information about link.
OpenLinkSource	Opens a link's source.
UpdateLink	Forces a link to connect to its source and update.
CancelLink	Breaks the link.

IOleUIObjInfo Methods	Description
GetObjectInfo	Gets general information about the object.
GetConvertInfo	Gets information that is used for the Convert dialog.
ConvertObject	Converts the object once the user selects a destination type.
GetViewInfo	Gets the current icon, aspect, and scale of the object.
SetViewInfo	Sets the current icon, aspect, and scale of the object.

See Also

OleUIObjectProperties

IOleUIObjInfo::ConvertObject

Converts the object to the type of the new CLSID.

```
HRESULT ConvertObject(
   DWORD dwObject,          //32-bit object identifier
   REFCLSID clsidNew        //CLSID to convert the object to
  );
```

Parameters

dwObject
A unique 32-bit identifier for the object.

clsidNew
The CLSID to convert the object to.

Return Values

S_OK
Successfully returned link information.

E_FAIL
Unable to return link information.

E_INVALIDARG
One or more invalid arguments.

E_OUTOFMEMORY
Insufficient memory.

E_ACCESSDENIED
Insufficient access permissions.

Comment

Notes To Implementors

Your implementation of **ConvertObject** needs to convert the object to the CLSID specified. The actions taken by the convert operation are similar to the actions taken after calling **OleUIConvert**.

See Also

OleUIConvert

IOleUIObjInfo::GetConvertInfo

Gets the conversion information associated with the specified object.

```
HRESULT GetConvertInfo(
   DWORD dwObject,                          //32-bit object identifier
   CLSID FAR * lpClassID,                   //CLSID of the object
   WORD FAR *lpwFormat,                     //CLIPFORMAT of the object's storage
   CLSID FAR * lpConvertDefaultClassID,     //Default class to convert to
   LPCLSID FAR * lplpClsidExclude,          //Excluded CLSIDs
   UINT FAR * lpcClsidExclude               //Number of CLSIDs in lplpClsidExclude
  );
```

Parameters

dwObject
A unique 32-bit identifier for the object.

lpClassID
[out] Points to the location to return the object's CLSID.

lpwFormat
[out] Points to the clipboard format of the object.

lpConvertDefaultClassID
[out] Points to the default class, selected from the UI, to convert the object to.

lplpClsidExclude
[out] Points to to an array of CLSIDs that should be excluded from the UI for this object. May be NULL, if *lplpClsidExclude* is 0 (zero).

lpcClsidExclude
[out] The number of CLSIDs in *lplpClsidExclude*. May be 0 (zero).

Return Values

S_OK
Successfully returned link information.

E_FAIL
Unable to return link information.

E_INVALIDARG
One or more invalid arguments.

E_OUTOFMEMORY
Insufficient memory.

E_ACCESSDENIED
Insufficient access permissions.

Comment

Notes To Implementors

You must fill in the CLSID of the object at a minimum. *lpwFormat* may be left at zero if the format of the storage is unknown.

IOleUIObjInfo::GetObjectInfo

Gets size, type, name and location information about an object.

```
HRESULT GetObjectInfo(
   DWORD dwObject,                    //32-bit object identifier
   DWORD FAR *lpdwObjSize,            //Points to object's size
   LPTSTR FAR *lplpszLabel,           //Points to object's label.
   LPTSTR FAR * lplpszType,           //Points to object's "long" type
   LPTSTR FAR * lplpszShortType,      //Points to object's "short" type
   LPTSTR FAR * lplpszLocation        //Points to the objects source
  );
```

Parameters

dwObject
A unique 32-bit identifier for the object.

lpdwObjSize
[out] The object's size, in bytes, on disk. This may be an approximate value.

lplpszLabel
[out] Points to the object's label. May be NULL, which indicates that the implementation should not fill this in.

lplpszType
[out] Points to the object's "long" type. May be NULL, which indicates that the implementation should not fill this in.

lplpszShortType
[out] Points to the object's "short" type. May be NULL, which indicates that the implementation should not fill this in.

lplpszLocation
[out] Points to the object's source. May be NULL, which indicates that the implementation should not fill this in.

Return Values

S_OK
Successfully returned object information.

E_FAIL
Unable to get object information.

E_INVALIDARG
One or more invalid arguments.

E_OUTOFMEMORY
Insufficient memory.

Comment

The strings and the object's size are displayed in the object properties "General" page.

Notes To Implementors

Your implementation of **GetObjectInfo** should place each of the object's attributes in the out parameters provided. Set *lpdwObjSize* to (DWORD)-1 when the size of the object is unknown. Allocate all strings (the rest of the params) with the OLE task allocator obtained via **CoGetMalloc**, as is standard for all OLE interfaces with [out] string parameters, or you can simply use **CoTaskMemAlloc**.

See Also

CoGetMalloc, **CoTaskMemAlloc**

IOleUIObjInfo::GetViewInfo

Gets the view information associated with the object.

```
HRESULT GetViewInfo(
    DWORD dwObject,                    //32-bit object identifier
    HGLOBAL FAR * phMetaPict,          //Object's current icon
    DWORD * pdvAspect,                 //Object's current aspect
    int * pnCurrentScale               //Object's current scale
  );
```

Parameters

dwObject
A 32-bit identifier for the object.

phMetaPict
The object's current icon. Could be NULL, indicating that the caller is not interested in the object's current presentation.

pdvAspect
The object's current aspect. Could be NULL, indicating that the caller is not interested in the object's current aspect, i.e., DVASPECT_ICONIC or DVASPECT_CONTENT.

pnCurrentScale
The object's current scale. Could be NULL, indicating that the caller is not interested in the current scaling factor applied to the object in the container's view.

Return Values

S_OK
Successfully returned link information.

E_FAIL
Unable to return link information.

E_INVALIDARG
One or more invalid arguments.

E_OUTOFMEMORY
Insufficient memory.

E_ACCESSDENIED
Insufficient access permissions.

Comment

Notes To Implementors

You must fill in the object's current icon, aspect, and scale.

See Also

OLEUIVIEWPROPS

IOleUIObjInfo::SetViewInfo

Sets the view information associated with the object.

```
HRESULT SetViewInfo(
   DWORD dwObject,                //32-bit object identifier
   HGLOBAL hMetaPict,             //New icon for the object
   DWORD dvAspect,                //New display aspect
   int nCurrentScale,             //New scale
   BOOL bRelativeToOrig           //Scale relative to origin
  );
```

Parameters

dwObject
A 32-bit identifier for the object.

hMetaPict
The new icon for the object.

dvAspect
The object's new display aspect or view.

nCurrentScale
The object's new scale.

bRelativeToOrig
The scale of the object, relative to the origin. This value is TRUE if the new scale should be relative to the original scale of the object. If FALSE, *nCurrentScale* applies to the object's current size.

Return Values

S_OK
Successfully returned link information.

E_FAIL
Unable to return link information.

E_INVALIDARG
One or more invalid arguments.

E_OUTOFMEMORY
Insufficient memory.

E_ACCESSDENIED
Insufficient access permissions.

Comment

Notes To Implementors

You should apply the new attributes (icon, aspect, and scale) to the object. If *bRelativeToOrig* is set to TRUE, *nCurrentScale* (in percentage units) applies to the original size of the object before it was scaled. If *bRelativeToOrig* is FALSE, *nCurrentScale* applies to the object's current size.

See Also

DVASPECT

IOleWindow

The **IOleWindow** interface methods allow an application to obtain the handle to the various windows that participate in in-place activation, and also to enter and exit context-sensitive help mode.

Five other in-place activation interfaces are derived from the **IOleWindow** interface. Containers and objects must implement and use these six interfaces in order to support in-place activation. The following table briefly summarizes each of these interfaces:

IOleWindow	The base interface. Implemented and used by containers and objects to window handles and manage context-sensitive help.
IOleInPlaceObject	Implemented by objects and used by an object's immediate container to activate and deactivate the object.
IOleInPlaceActiveObject	Implemented by objects and used by the top-level container to manipulate the object while it is active. Provides a direct channel of communication between an active object and its frame and document windows.
IOleInPlaceUIWindow	Implemented by containers and used by objects to manipulate the container's document window.
IOleInPlaceFrame	Implemented by containers and used by objects to control the container's frame window.
IOleInPlaceSite	Implemented by containers and used by objects to interact with the in-place client site.

These interfaces can be arranged in three hierarchical levels with various interfaces implemented at each level. Calls that install user-interface menus and frame adornments, activate and switch between windows, and dispatch menu and keystrokes take place between the top-level container and the active object. Calls that support activating, deactivating, scrolling, or clipping span the containment hierarchy, with each level performing the correct actions.

When to Implement

The inherited methods of this interface are implemented by all in-place objects and containers.

When to Use

Use this interface to obtain the window handle to the windows associated with in-place activation (frame, document, parent, and in-place object). It is also used to enter and exit context-sensitive help.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns a pointer to a specified interface.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IOleWindow Methods	Description

GetWindow	Gets a window handle.
ContextSensitiveHelp	Controls enabling of context sensitive help.

See Also

IOleInPlaceObject, **IOleInPlaceActiveObject**, **IOleInPlaceUIWindow**, **IOleInPlaceFrame**, **IOleInPlaceSite**, **OleCreateMenuDescriptor**, **OleDestroyMenuDescriptor**, **OleTranslateAccelerator**

IOleWindow::ContextSensitiveHelp

Determines whether context-sensitive help mode should be entered during an in-place activation session.

```
HRESULT ContextSensitiveHelp(
  BOOL fEnterMode        //Specifies whether or not to enter help mode
 );
```

Parameter

fEnterMode
[in] Specifies TRUE if help mode should be entered; FALSE if it should be exited.

Return Values

S_OK
The help mode was entered or exited successfully, depending on the value passed in *fEnterMode*.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
The argument is invalid.

E_UNEXPECTED
An unexpected error occurred.

Comments

Applications can invoke context-sensitive help when the user

- presses SHIFT+F1, then clicks a topic
- presses F1 when a menu item is selected.

When SHIFT+F1 is pressed, either the frame or active object can receive the keystrokes. If the container's frame receives the keystrokes, it calls its containing document's **IOleWindow::ContextSensitiveHelp** method with *fEnterMode* set to TRUE. This propagates the help state to all of its in-place objects so they can correctly handle the mouse click or WM_COMMAND.

If an active object receives the SHIFT+F1 keystrokes, it calls the container's **IOleInPlaceSite::ContextSensitiveHelp** method with *fEnterMode* TRUE, which then recursively calls each of its in-place sites until there are no more to be notified. The container then calls its document's or frame's **ContextSensitiveHelp** method with *fEnterMode* TRUE.

When in context-sensitive help mode, an object that receives the mouse click can either:

1. Ignore the click if it does not support context-sensitive help, or
2. Tell all the other objects to exit context-sensitive help mode with **ContextSensitiveHelp** set to FALSE and then provide help for that context.

An object in context-sensitive help mode that receives a WM_COMMAND should tell all the other in-place objects to exit context-sensitive help mode and then provide help for the command.

If a container application is to support context-sensitive help on menu items, it must either provide its own message filter so that it can intercept the F1 key or ask the OLE library to add a message filter by calling **OleSetMenuDescriptor**, passing valid, non-NULL values for the *lpFrame* and *lpActiveObj* parameters.

See Also

OleSetMenuDescriptor

IOleWindow::GetWindow

Returns the window handle to one of the windows participating in in-place activation (frame, document, parent, or in-place object window).

```
HRESULT GetWindow(
   HWND * phwind         //Where to return window handle
  );
```

Parameter

phwnd
[out] Points to where to return the window handle.

Return Values

S_OK
The window handle was successfully returned.

E_INVALIDARG
One or more arguments are invalid.

E_OUTOFMEMORY
Out of memory.

E_UNEXPECTED
An unexpected error occurred.

E_FAIL
There is no window handle currently attached to this object.

Comments

Five types of windows comprise the windows hierarchy. When a object is active in place, it has access to some or all of these windows:

Window	Description
Frame	The outermost main window where the container application's main menu resides.
Document	The window that displays the compound document containing the embedded object to the user.
Pane	The subwindow of the document window that contains the object's view. Applicable only for applications with split-pane windows.
Parent	The container window that contains that object's view. The object application installs its window as a child of this window.
In-place	The window containing the active in-place object. The object application creates this window and installs it as a child of its hatch window, which is a child of the container's parent window.

Each type of window has a different role in the in-place activation architecture. However, it is not necessary to employ a separate physical window for each type. Many container applications use the same window for their frame, document, pane, and parent windows.

IParseDisplayName

The **IParseDisplayName** interface parses a human-readable display name string to convert it into a moniker. Display name parsing is necessary when the end user inputs a string to identify a component, as in the following situations:

- A compound document application that supports linked components typically supports the Edit:Links... dialog. Through this dialog, the end user can enter a display name to specify a new link source for a specified linked component. The compound document needs to have this input string converted into a moniker.
- A script language such as the macro language of a spreadsheet can allow textual references to a component. The language's interpreter needs to have such a reference converted into a moniker in order to execute the macro.

When to Implement

Compound document applications that support links to embedded components or to pseudo-objects within their documents must provide an implementation of the **IOleItemContainer** interface, which is derived indirectly from **IParseDisplayName**. In effect, such a compound document is providing a namespace for identifying its internal components; and its **IOleItemContainer** implementation (which includes the **IParseDisplayName** implementation) is the interface through which another application can access this namespace. Alternatively, the compound document application can implement **IParseDisplayName** as part of its class object, which is accessible through the **CoGetClassObject** function.

When to Use

If you are implementing your own moniker class, you might need to use this interface from your implementation of **IMoniker::ParseDisplayName**. If you call the **MkParseDisplayName** function, you are indirectly using **IParseDisplayName**.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IParseDisplayName Method	Description
ParseDisplayName	

See Also

IMoniker::ParseDisplayName, **IOleItemContainer**, **MkParseDisplayName**

IParseDisplayName::ParseDisplayName

Parses the display name to extract a component of the string that it can convert into a moniker, using the maximum number of characters from the lefthand side of the string.

```
HRESULT ParseDisplayName(
  IBindCtx *pbc,                  //Pointer to bind context
  LPOLESTR pszDisplayName,        //Pointer to string containing display name
  ULONG *pchEaten,                //Length, in characters, of display name
  IMoniker **ppmkOut              //Pointer to moniker that results
 );
```

Parameters

pbc
[in] Points to the bind context to be used in this binding operation.

pszDisplayName
[in] Points to a zero-terminated string containing the display name to be parsed. For Win32 applications, the **LPOLESTR** type indicates a wide character string (two bytes per character); otherwise, the string has one byte per character.

pchEaten
[out] Receives the number of characters in the display name that correspond to the *ppmkOut* moniker.

ppmkOut
[out] Receives a pointer to the resulting moniker. If an error occurs, the implementation sets **ppmkOut* to NULL. If **ppmkOut* is non-NULL, the implementation must call *(*ppmkOut)*->**IUnknown::AddRef**; so it is the caller's responsibility to call *(*ppmkOut)*->**IUnknown::Release**.

Return Values

S_OK
Indicates success.

MK_E_SYNTAX
Indicates a syntax error in the display name.

MK_E_NOOBJECT
Indicates that the display name does not identify a component in this namespace.

E_UNEXPECTED
Indicates an unexpected error.

E_OUTOFMEMORY
Indicates insufficient memory.

See Also

MkParseDisplayName, **IMoniker::ParseDisplayName**

IPersist

The **IPersist** interface is the base interface for three other persistence-related interfaces: **IPersistStorage**, **IPersistStream**, and **IPersistFile**. These three interfaces are offered by objects that can serialize themselves to a storage, stream, or file. That is, they can save their state for later instantiations, and they can load their saved state, thus transitioning from a passive to a running state.

Typically, the persistence interfaces are implemented by an embedded or linked object. The container application or the default object handler calls the methods in one of the persistence interfaces to have the object save or load itself. Thus, the object that offers a persistence interface can change itself to an active or running state by request of its container application or its object handler.

The **IPersist** interface is the base from which the other persistence interfaces are derived. Its one method returns the class identifier (CLSID) of an object that implements one of the other persistence interfaces.

When to Implement

You must implement the **IPersist** interface whenever you implement any one of the other persistence interfaces: **IPersistStorage**, **IPersistStream**, or **IPersistFile**. Typically, embedded objects implement the **IPersistStorage** interface; linked objects implement the **IPersistFile** interface; and new moniker classes implement the **IPersistStream** interface.

When to Use

The **IPersist** interface is used primarily by the default object handler to get the class identifier of an embedded object. Typically, container applications would not call the **IPersist** interface unless they provide object handlers for specific classes of objects.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IPersist Method	**Description**
GetClassID	Returns the class identifier (CLSID) for the component object.

IPersist::GetClassID

Returns the class identifier (CLSID) the component object. The CLSID is a unique value that identifies the application that can manipulate the component object data.

```
HRESULT GetClassID(
  CLSID *pClassID        //Pointer to CLSID of component object
 );
```

Parameter

pClassID
[out]Points to the location where the CLSID is returned. The CLSID is a globally unique identifier (GUID) that uniquely represents an object class.

Return Values

S_OK
The CLSID was successfully returned.

E_FAIL
The CLSID could not be returned.

Comments

This method returns the class identifier (CLSID) for an object, used in later operations to load object-specific code into the caller's context.

Notes to Callers

Most container applications have no need to call the **IPersist::GetClassID** method directly. A container application might use this method when the container needs to change the CLSID and save the new CLSID for the component object. For example, if the container is treating a component as a different class and an editing operation occurs so that the component object can no longer be edited by its original application. In this case, the container needs to change the CLSID for the component. Typically, in this case, the container calls the **OleSave** helper function which performs all the necessary steps.

The exception would be a container that provides an object handler for certain component objects. In particular, a container application should not get a component object CLSID and then use it to retrieve class specific information from the registry. Instead, the container should use **IOleObject** and **IDataObject** interfaces to retrieve such class-specific information directly from the object.

Notes to Implementors

Typically, a component object simply returns a constant CLSID in its implementation of this method. This is not the case, however, if your component object supports "Treat As" operations in which it edits other classes of components as if they were its own class. If your object is running a component of another class, your implementation of **IPersist::GetClassID** must return the CLSID stored in the component storage.

The implementation of **IPersist::GetClassID** in the default object handler passes the call to the implementation in the component object if the component object is running. Otherwise, the default handler calls the **ReadClassStg** function to read the CLSID that is saved in the component object storage.

If you are writing a custom object handler for your component, you might want to simply delegate this method to the default handler implementation (see **OleCreateDefaultHandler**).

See Also

IDataObject::EnumFormatEtc, **IOleObject::EnumVerbs**, **IOleObject::GetMiscStatus**,

IOleObject::GetUserType, **OleCreateDefaultHandler**, **ReadClassStg**

IPersistFile

The **IPersistFile** interface provides methods for an object to load and save itself in a disk file, instead of saving itself to a storage object. Because the information needed to open a file varies greatly from one application to another, the object must be responsible for opening its disk file.

When to Implement

The **IPersistFile** interface is implemented by any object or container application that can be linked through a file moniker, including the following:

- Any object that supports links to its files or to *pseudo-objects* within its files
- A container application that supports links to component objects within its compound file

Typically, you implement the **IPersistFile** interface as part of an aggregate object that includes other interfaces that are appropriate for the type of moniker binding that is supported.

For example, in either of the cases mentioned above, the moniker for the linked object can be a composite moniker. In the first case, a composite moniker identifies the pseudo-object contained within the file. In the second case, a composite moniker identifies the embedded object contained within the compound file. In either case of composite monikers, you must implement the **IPersistFile** interface as part of same object that also implements the **IOleItemContainer** interface. Then, when the application for the linked object is run, OLE queries for the **IOleItemContainer** interface to locate the embedded object or the pseudo-object contained in the file.

As another example, if the moniker is a simple file moniker (i.e., the link is to the entire file), OLE queries for the interface that the initiator of the bind operation requested. Typically, this is one of the compound document interfaces, such as **IOleObject**, **IDataObject**, or **IPersistStorage**.

When to Use

You call methods in the **IPersistFile** interface to load or save a linked component object in a specified file.

When the application for the linked object is run, OLE calls the **IPersistFile::Load** method. Once the file is loaded, OLE calls **IPersistFile::QueryInterface** to get another interface pointer to the loaded object. The **IPersistFile** interface is typically part of an aggregate object that offers other interfaces.

Note that OLE calls only the **IPersistFile::Load** method. It does not call any of the other **IPersistFile** methods. Typically, applications do not call these methods either. The **IPersistFile::Load** method enables OLE to load a file on behalf of a container application when the container needs to run the application for the linked object. The other **IPersistFile** methods support saving a component to a file, but container applications do not tell a linked object to save itself. It is entirely up to the end user and the application for the linked object to decide when the component should be saved. This differs from the situation for an embedded component, in which the container application uses the **IPersistStorage** interface to provide the storage and to tell the component when to save itself.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IPersistFile Methods	**Description**
IsDirty	Checks an object for changes since it was last saved to its current file.

Load	Opens the specified file and initializes an object from the file contents.
Save	Saves the object into the specified file.
SaveCompleted	Notifies the object that it can revert from NoScribble mode to Normal mode.
GetCurFile	Gets the current name of the file associated with the component.

IPersistFile::GetCurFile

Returns either the absolute path to the object's current working file or, if there is no current working file, the object's default save prompt .

HRESULT GetCurFile(
  LPOLESTR **ppszFileName*   //Pointer to the path for the current file or the default save prompt
 );

Parameter

ppszFileName
 [out]Points to a zero-terminated string containing the path for the current file or the default save prompt. If an error occurs, **ppszFileName* is set to NULL.

Return Values

S_OK
 A valid absolute path was successfully returned.

S_FALSE
 The default save prompt was returned.

E_OUTOFMEMORY
 The operation failed due to insufficient memory.

E_FAIL
 The operation failed due to some reason other than insufficient memory.

Comments

This method returns the current filename or the default save prompt for the object.

This method allocates memory for the string returned in the *ppszFileName* parameter using the **IMalloc::Alloc** method. The caller is responsible for calling the **IMalloc::Free** method to free the string. Both the caller and this method use the allocator returned by **CoGetMalloc**(MEMCTX_TASK, ...).

The **LPOLESTR** type indicates a wide character string (two bytes per character); otherwise, the string has one byte per character.

The filename returned in *ppszFileName* is the name specified in the **IPersistFile::Load** method when the document was loaded; or in **IPersistFile::SaveCompleted** if the document was saved to a different file.

If the object does not have a current working file, it should return the default filename prompt that it would display in a "Save As" dialog. For example, the default save prompt for a word processor object could be:

*.txt

Notes to Callers

OLE does not call the **IPersistFile::GetCurFile** method. Typically, applications do not either.

If you want to perform a "Save" operation on the object, you can call this method before calling **IPersistFile:Save** to determine whether the object has an associated file. If this method returns S_OK, you can then call **IPersistFile::Save** with a NULL filename and a TRUE value for the *fRemember* parameter to tell the object to save itself to its current file. If this method returns S_FALSE, you can use the save prompt returned in the *ppszFileName* parameter to ask the end user to provide a filename. Then, you can call **IPersistFile::Save** with the filename that the user entered to perform a "Save As" operation.

See Also

IPersistFile::Load, **IPersistFile::Save**, **IPersistFile::SaveCompleted**

IPersistFile::IsDirty

Checks an object for changes since it was last saved to its current file.

HRESULT IsDirty(*void***);**

Return Values

S_OK
: The object has changed since it was last saved.

S_FALSE
: The object has not changed since the last save.

Comments

This method checks whether an object has changed since it was last saved so you can avoid losing information in objects that have not yet been saved. The dirty flag for an object is conditionally cleared in the **IPersistFile::Save** method.

Notes to Callers

OLE does not call **IPersistFile::IsDirty**. Typically, applications do not either.

You should treat any error return codes as an indication that the object has changed. In other words, unless this method explicitly returns S_FALSE, you must assume that the object needs to be saved.

Notes to Implementors

An object with no contained objects simply checks its dirty flag to return the appropriate result.

A container object with one or more contained objects must maintain an internal dirty flag that reflects any one contained object being dirty. To do this, the container object should register for data change notifications with each link or embedding (see **IDataObject::DAdvise**). Then, the container can maintain its internal dirty flag that it sets when it receives an **IAdviseSink::OnDataChange** notification. If the container does not register for data change notifications, the **IPersistFile::IsDirty** implementation must call **IPersistStorage::IsDirty** for each of its contained objects to determine whether they have changed.

The container object can clear its dirty flag whenever it is saved, as long as the file saved to is the current working file after the save. In other words, the dirty flag is cleared after a successful "Save" or "Save As" operation, but not after a "Save A Copy As . . ." operation.

See Also

IAdviseSink::OnDataChange, **IDataObject::DAdvise**, **IPersistStorage::IsDirty**

IPersistFile::Load

Opens the specified file and initializes an object from the file contents.

```
HRESULT Load(
  LPCOLESTR pszFileName,        //Pointer to absolute path of the file to open
  DWORD dwMode                  //Specifies the access mode from the STGM enumeration
 );
```

Parameters

pszFileName
[in]Points to a zero-terminated string containing the absolute path of the file to open.

dwMode
[in]Specifies some combination of the values from the **STGM** enumeration to indicate the access mode to use when opening the file. The **IPersistFile::Load** method can treat this value as a suggestion, adding more restrictive permissions if necessary. If *dwMode* is zero, the implementation should open the file using whatever default permissions are used when a user opens the file.

Return Values

S_OK
The object was successfully loaded.

E_NOMEMORY
The object could not be loaded due to a lack of memory.

E_FAIL
The object could not be loaded for some reason other than a lack of memory.

IPersistFile::Load STG_E_* error codes.

Comments

This method loads the object from the specified file. This method is for initialization only and does not show the object to the end user; it is not equivalent to the end user selecting the File Open command.

Notes to Callers

The **BindToObject** method in file monikers calls this method to load an object during a moniker binding operation, that is, when the linked object is run.

Typically, applications do not call this method directly.

Notes to Implementors

Because the information needed to open a file varies greatly from one application to another, the object that implements this method must also open the file specified by the *pszFileName* parameter. Note that this differs from the **IPersistStorage::Load** and **IPersistStream::Load** in which the caller opens the storage or stream and then passes an open storage or stream pointer to the loaded object.

If your application normally uses the OLE-provided compound files, then your **IPersistFile::Load** implementation can simply call the **StgOpenStorage** helper API function to open the storage object in the specified file. Then, you can proceed with the normal document initialization. Applications that do not use storage objects can perform their normal file opening procedures.

When the document has been loaded, your implementation should register the document object in the Running Object Table (see **IRunningObjectTable::Register**).

See Also

IRunningObjectTable::Register, **StgOpenStorage**

IPersistFile::Save

Saves a copy of the object into the specified file.

```
HRESULT Save(
    LPCOLESTR pszFileName,        //Pointer to absolute path of the file where the object is saved
    BOOL fRemember                //Specifies whether the file is to be the current working file or not
  );
```

Parameters

pszFileName
: [in]Points to a zero-terminated string containing the absolute path of the file to which the object should be saved. If *pszFileName* is NULL, the object should save its data to the current file, if there is one.

fRemember
: [in]Indicates whether the *pszFileName* parameter is to be used as the current working file. If TRUE, *pszFileName* becomes the current file and the object should clear its dirty flag after the save. If FALSE, this save operation is a "Save A Copy As ..." operation. In this case, the current file is unchanged and the object should not clear its dirty flag. If *pszFileName* is NULL, the implementation should ignore the *fRemember* flag.

Return Values

S_OK
: The object was successfully saved.

E_FAIL
: The file was not saved.

IPersistFile::Save STG_E_* errors.

Comments

This method saves the object to the specified file. There are three different save scenarios that can use this method:

Save
: The caller calls the **IPersistFile::GetCurFile** method first to determine whether the object has an associated filename. If it does, the caller specifies NULL for the *pszFileName* parameter in this method to indicate that the object should "Save" to its current file. The caller must call the **IPersistFile::SaveCompleted** method to indicate completion.

Save As
: The caller specifies TRUE in the *fRemember* parameter and a non-NULL value for the *pszFileName* parameter to indicate saving to a new file. The caller must call the **IPersistFile::SaveCompleted** method to indicate completion.

Save a Copy As
: The caller specifies FALSE in the *fRemember* parameter and a non-NULL value for the *pszFileName* parameter to save a copy to the specified file.

The implementor must detect which save scenario is being requested by the caller. If the *pszFileName* parameter is NULL, the Save scenario is being requested. If the *pszFileName* parameter is not NULL, use the value of the *fRemember* parameter to distinguish between a Save As and a Save a Copy As call.

In the "Save" or "Save As" scenarios, the **IPersistFile::Save** method clears the internal dirty flag after the save and sends **IAdviseSink::OnSave** notifications to any advisory connections (see also **IOleAdviseHolder::SendOnSave**). Also, in these scenarios, the object is in NoScribble mode until it receives an **IPersistFile::SaveCompleted** call. In NoScribble mode, the object must not write to the file.

In the "Save As" scenario, the implementation should also send **IAdviseSink::OnRename** notifications to any advisory connections (see also **IOleAdviseHolder::SendOnRename**).

In the "Save a Copy As" scenario, the implementation does not clear the internal dirty flag after the save.

Notes to Callers

OLE does not call **IPersistFile::Save**. Typically, applications do not either.

See Also

IOleAdviseHolder::SendOnRename, **IOleAdviseHolder::SendOnSave**, **IPersistFile::GetCurFile**, **IPersistFile::SaveCompleted**

IPersistFile::SaveCompleted

Notifies the object that it can write to its file. It does this by notifying the object that it can revert from NoScribble mode (in which it must not write to its file), to Normal mode (in which it can). The component enters NoScribble mode when it receives an **IPersistFile::Save** call.

```
HRESULT SaveCompleted(
  LPCOLESTR pszFileName        //Pointer to absolute path of the file where the object was saved
 );
```

Parameter

pszFileName
[in]Points to the absolute path of the file where the object was previously saved.

Return Value

S_OK
Returned in all cases.

Comments

This method is called when a call to **IPersistFile::Save** is completed, and the file that was saved is now the current working file (that is, the "Save" or "Save As" scenarios). In these cases, the object has been placed into NoScribble mode so it cannot write to its file. When the save is completed, the object can revert to Normal mode and it is free to write to its file.

Notes to Callers

OLE does not call the **IPersistFile::SaveCompleted** method. Typically, applications do not either.

See Also

IPersistFile::Save

IPersistStorage

The **IPersistStorage** interface enables a container application to pass a storage object to one of its contained objects and to load and save the storage object. This interface supports the structured storage scheme in which each component object has its own storage that is nested within the container's storage.

When to Implement

Any object that can be embedded in a container object must implement the **IPersistStorage** interface. This interface is one of the primary interfaces for a compound document object. It is part of the aggregate object that includes the **IOleObject** and **IDataObject** interfaces.

The OLE default handler for embedded objects provides an implementation of the **IPersistStorage** interface that is used when the object is in the loaded state. Similarly, the OLE default link handler provides an **IPersistStorage** implementation that manages storage for a linked object. These default handlers both interact with the OLE default cache implementation, which has its own **IPersistStorage** implementation.

If you are providing a custom embedding or link handler for your objects, the handler must include an implementation of **IPersistStorage**. You can delegate calls to the default handler so you can take advantage of the default cache implementation.

When to Use

When an OLE container creates a new object, loads an existing object from storage, or inserts a new object in a clipboard or a drag-and-drop operation, the container uses the **IPersistStorage** interface to initialize the object and put it in the loaded or running state. When a component is loaded or running, an OLE container uses the other **IPersistStorage** methods to instruct the component to perform various save operations or to instruct the component to release its storage.

Typically, applications use helper functions such as **OleLoad** or **OleCreate**, rather than calling the **IPersistStorage::Load** or **IPersistStorage::InitNew** methods directly. Similarly, applications typically call the **OleSave** helper function rather than calling **IPersistStorage::Save** directly.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IPersistStorage Methods	**Description**
IsDirty	Indicates whether the object has changed since it was last saved to its current storage.
InitNew	Initializes a new storage object.
Load	Initializes an object from its existing storage.
Save	Saves an object, and any nested objects that it contains, into the specified storage object. The object enters NoScribble mode.
SaveCompleted	Notifies the object that it can revert from NoScribble or HandsOff mode, in

	which in must not write to its storage object, to Normal mode in which it can.
HandsOffStorage	Instructs the object to release all storage objects that have been passed to it by its container and to enter HandsOffAfterSave or HandsOffFromNormal mode.

See Also

IDataObject, **IOleObject**, **OleCreate**, **OleLoad**, **OleSave**

IPersistStorage::HandsOffStorage

Instructs the object to release all storage objects that have been passed to it by its container and to enter HandsOff mode, in which the object cannot do anything and the only operation that works is a close operation.

HRESULT HandsOffStorage(*void***);**

Return Value

S_OK
    The object has successfully entered HandsOff mode.

Comments

This method causes an object to release any storage objects that it is holding and to enter the HandsOff mode until a subsequent **IPersistStorage::SaveCompleted** call. In HandsOff mode, the object cannot do anything and the only operation that works is a close operation.

A container application typically calls this method during a full save or low-memory full save operation to force the component object to release all pointers to its current storage. In these scenarios, the **HandsOffStorage** call comes after calling **OleSave** or **IPersistStorage::Save** for the component object; and it puts the component object in HandsOffAfterSave mode. Calling this method is necessary so the container application can delete the current file as part of a full save, or so it can call the **IRootStorage::SwitchToFile** method as part of a low-memory save.

A container application also calls this method when a component object is in Normal mode to put the component object in HandsOffFromNormal mode.

While the component object is in either HandsOff mode, most operations on the object will fail. Thus, the container should restore the component object to Normal mode as soon as possible. The container application does this by calling the **IPersistStorage::SaveCompleted** method, which passes a storage pointer back to the component object for the new storage object.

Notes to Implementors

This method must release all pointers to the component's current storage object, including pointers to any nested streams and storages. If the component object contains nested component objects, the container application must recursively call this method for any nested component objects that are loaded or running.

See Also

OleSave, **IPersistStorage::Save**, **IPersistStorage::SaveCompleted**, **IRootStorage::SwitchToFile**

IPersistStorage::InitNew

Initializes a new storage object.

```
HRESULT InitNew(
   IStorage *pStg         //Points to the new storage object
  );
```

Parameter

pStg
: [in]Points to the new storage object to initialize and use. The container creates a nested storage object in its storage object (see **IStorage::CreateStorage**). Then, the container calls the **WriteClassStg** function to initialize the new storage object with the component object class identifier (CLSID).

Return Values

S_OK
: The new storage object was successfully initialized.

CO_E_ALREADYINITIALIZED
: The component object has already been initialized by a previous call to either the **IPersistStorage::Load** method or the **IPersistStorage::InitNew** method.

E_OUTOFMEMORY
: The storage object was not initialized due to a lack of memory.

E_FAIL
: The storage object was not initialized due to some reason besides a lack of memory.

Comments

A container application can call this method when it needs to create a new object, for example, with an InsertObject command.

An object that uses the **IPersistStorage** interface must have access to a valid **IStorage** instance at all times while it is running. This includes the time just after the object has been created but before it has been made persistent. The object's container must provide the object with access to an **IStorage** instance during this time by calling **IPersistStorage::InitNew**. Depending on the container's state, a temporary file might need to be created for this purpose.

If the object wants to retain the **IStorage** instance, it must call **IUnknown::AddRef** to increment its reference count.

After the call to **IPersistStorage::InitNew**, the component object is in either the loaded or running state. For example, if the object class has an in-process server, the object will be in the running state. However, if the object uses the default handler, the container's call to **InitNew** only invokes the handler's implementation which does not run the component object. Later if the container runs the component object, the handler calls the **IPersistStorage::InitNew** method for the object.

Notes to Callers

Rather than calling **IPersistStorage::InitNew** directly, you typically call the **OleCreate** helper function which performs the following steps:

1. Calls the **CoCreateInstance** function to create an instance of the object class
2. Queries the new instance for the **IPersistStorage** interface
3. Calls the **IPersistStorage::InitNew** method to initialize the component object

The container application should cache the **PersistStorage** interface for the object for use in later operations on the object.

Notes to Implementors

An implementation of **IPersistStorage::InitNew** should initialize the component object to its default state. In particular, your implementation should perform the following steps to initialize the storage object:

1. Pre-open and cache the pointers to any streams or storages that the component will need to save itself to this storage.
2. Call **IPersistStorage::AddRef** and cache the passed in storage pointer.
3. Call the **WriteFmtUserTypeStg** function to write the native clipboard format and user type string for the component object to the storage object.
4. Set the dirty flag for the component object.

The first two steps are particularly important for ensuring that the component object can save itself in low memory situations. By pre-opening the necessary streams, the component object can guarantee that a save operation to this storage object will not fail due to insufficient memory.

Your implementation of this method should return the CO_E_ALREADYINITIALIZED error code if it receives a call to either the **IPersistStorage::InitNew** method or the **IPersistStorage::Load** method after it is already initialized.

See Also

IPersistStorage::Load, **OleCreate**, **WriteFmtUserTypeStg**

IPersistStorage::IsDirty

Indicates whether the object has changed since it was last saved to its current storage.

HRESULT IsDirty(*void***);**

Return Values

S_OK
The object has changed since it was last saved.

S_FALSE
The object has not changed since the last save.

Comments

This method checks whether an object has changed since it was last saved so you can avoid losing information in the object that has not yet been saved. The dirty flag for an object is conditionally cleared in the **IPersistFile::Save** method.

For example, you could optimize a File:Save operation by calling the **IPersistStorage::IsDirty** method for each component object and then calling the **IPersistStorage::Save** method only for those component objects that are dirty.

Notes to Callers

You should treat any error return codes as an indication that the object has changed. In other words, unless this method explicitly returns S_FALSE, you must assume that the object needs to be saved.

Notes to Implementors

A container object with one or more contained objects must maintain an internal dirty flag that reflects any one contained object being dirty. To do this, the container object should register for data change notifications with each link or embedding (see **IDataObject::DAdvise**). Then, the container can maintain its internal dirty flag that it sets when it receives an **IAdviseSink::OnDataChange** notification. If the container does not register for data change notifications, the **IPersistFile::IsDirty** implementation must call **IPersistStorage::IsDirty** for each of its contained objects to determine whether they have changed.

A component object can clear its dirty flag whenever it is saved, as long as the storage saved to is the current working storage after the save. In other words, the dirty flag is cleared after a successful "Save" or "Save As" operation, but not after a "Save A Copy As . . ." operation.

See Also

IAdviseSink::OnDataChange, **IDataObject::DAdvise**, **IPersistStorage::Save**

IPersistStorage::Load

Initializes an object from its existing storage.

```
HRESULT Load(
   IStorage *pStg         //Pointer to existing storage object for the component object
  );
```

Parameter

pStg
[in]Points to the existing storage object from which the component object is to be loaded.

Return Values

S_OK
The object was successfully loaded.

CO_E_ALREADYINITIALIZED
The object has already been initialized by a previous call to the **IPersistStorage::Load** method or the **IPersistStorage::InitNew** method.

E_OUTOFMEMORY
The object was not loaded due to lack of memory.

E_FAIL
The object was not loaded due to some reason besides a lack of memory.

Comments

This method initializes a component object from an existing storage object. The component object is placed in the loaded state if this method is called by the container application. If called by the default handler, this method places the component object in the running state.

Either the default handler or the component object itself can hold onto the **IStorage** interface while the object is loaded or running.

Notes to Callers

Rather than calling **IPersistStorage::Load** directly, you typically call the **OleLoad** helper function which performs the following steps:

1. Create an uninitialized instance of the component object class
2. Query the new instance for the **IPersistStorage** interface
3. Call the **IPersistStorage::Load** method to initialize the component from the existing storage

You also call this method indirectly when you call the **OleCreateFromData** function or the **OleCreateFromFile** function to insert a component object into a compound file (as in a drag-and-drop or clipboard paste operation).

The container should cache the **IPersistStorage** interface for use in later operations on the component object.

Notes to Implementors

Your implementation should perform the following steps to load a component object:

1. Open the component object's streams in the storage object, and read the necessary data into the object's internal data structures.
2. Clear the component object's dirty flag.
3. Call the **IPersistStorage::AddRef** method and cache the passed in storage pointer.
4. Keep open and cache the pointers to any streams or storages that the component object will need to

save itself to this storage.

5. Perform any other default initialization required for the component object.

Steps 3 and 4 are particularly important for ensuring that the component can save itself in low memory situations. By holding onto pointers to the storage and streams, the component can guarantee that a save operation to this storage will not fail due to insufficient memory.

Your implementation of this method should return the CO_E_ALREADYINITIALIZED error code if it receives a call to either the **IPersistStorage::InitNew** method or the **IPersistStorage::Load** method after it is already initialized.

See Also

GetConvertStg, **IPersistStorage::InitNew**, **OleLoad**, **ReadFmtUserTypeStg**, **SetConvertStg**, **WriteFmtUserTypeStg**

IPersistStorage::Save

Saves an object, and any nested objects that it contains, into the specified storage object. The object is placed in NoScribble mode, and it must not write to the specified storage until it receives a call to its **IPersistStorage::SaveCompleted** method.

```
HRESULT Save(
  IStorage *pStgSave,         //Pointer to storage object
  BOOL fSameAsLoad            //Indicates whether the specified storage object is the current one
 );
```

Parameters

pStgSave
[in]Points to the storage object into which the component object is to be saved.

fSameAsLoad
[in]Indicates whether the specified storage object is the current one, which was passed to the object by one of the following calls:

- **IPersistStorage::InitNew** when it was created.
- **IPersistStorage::Load** when it was loaded.
- **IPersistStorage::SaveCompleted** when it was saved to a storage different from its current storage.

This parameter is set to FALSE when performing a Save As or Save A Copy To operation or when performing a full save. In the latter case, this method saves to a temporary file, deletes the original file, and renames the temporary file.

This parameter is set to TRUE to perform a full save in a low-memory situation or to perform a fast incremental save in which only the dirty components are saved.

Return Values

S_OK
The object was successfully saved.

STG_E_MEDIUMFULL
The object was not saved because of a lack of space on the disk.

E_FAIL
The object could not be saved due to errors other than a lack of disk space.

Comments

This method saves an object, and any nested objects it contains, into the specified storage object. It also places the component object into NoScribble mode. Thus, the component object cannot write to its storage object until a subsequent call to the **IPersistStorage::SaveCompleted** method returns the object to Normal mode.

If the storage object is the same as the one it was loaded or created from, the save operation may be able to write incremental changes to the storage object. Otherwise, a full save must be done.

This method recursively calls the **IPersistStorage::Save** method, the **OleSave** function, or the **IStorage::CopyTo** method to save its nested objects.

This method does not call the **IStorage::Commit** method. Nor does it write the class identifier to the storage object. Both of these tasks are the responsibilities of the caller.

Notes to Callers

Rather than calling **IPersistStorage::Load** directly, you typically call the **OleSave** helper function which performs the following steps:

1. Call the **WriteClassStg** function to write the class identifier for the object to the storage.
2. Call the **IPersistStorage::Save** method.
3. If needed, call the **IStorage::Commit** method on the storage object.

Then, a container application performs any other operations necessary to complete the save and calls the **SaveCompleted** method for each component object.

If an embedded object passed the **IPersistStorage::Save** method to its nested objects, it must wait until it receives a call to its **IPersistStorage::SaveCompleted** method before calling the **IPersistStorage::SaveCompleted** method for its nested objects.

See Also

IPersistStorage::InitNew, **IPersistStorage::Load**, **IPersistStorage::SaveCompleted**, **IStorage::Commit**, **IStorage::CopyTo**, **OleSave**, **WriteClassStg**, **WriteFmtUserTypeStg**

IPersistStorage::SaveCompleted

Notifies the object that it can revert from NoScribble or HandsOff mode, in which it must not write to its storage object, to Normal mode, in which it can. The component object enters NoScribble mode when it receives an **IPersistStorage::Save** call.

```
HRESULT SaveCompleted(
  IStorage *pStgNew        //Pointer to the current storage object
 );
```

Parameter

pStgNew
[in]Points to the new storage object, if different from the storage object prior to saving. This pointer can be NULL if the current storage object does not change during the save operation. If the object is in HandsOff mode, this parameter must be non-NULL.

Return Values

S_OK
The object was successfully returned to Normal mode.

E_OUTOFMEMORY
The object remained in HandsOff mode or NoScribble mode due to a lack of memory. Typically, this error occurs when the object is not able to open the necessary streams and storage objects in *pStgNew*.

E_INVALIDARG
The *pStgNew* parameter is not valid. Typically, this error occurs if *pStgNew* is NULL when the object is in HandsOff mode.

E_UNEXPECTED
The object is in Normal mode, and there was no previous call to **IPersistStorage::Save** or **IPersistStorage::HandsOffStorage**.

Comments

This method notifies an object that it can revert to Normal mode and can once again write to its storage object. The object exits NoScribble mode or HandsOff mode.

If the object is reverting from HandsOff mode, the *pStgNew* parameter must be non-NULL. In HandsOffFromNormal mode, this parameter is the new storage object that replaces the one that was revoked by the **IPersistStorage::HandsOffStorage** method. The data in the storage object is a copy of the data from the revoked storage object. In HandsOffAfterSave mode, the data is the same as the data that was most recently saved. It is not the same as the data in the revoked storage object.

If the object is reverting from NoScribble mode, the *pStgNew* parameter can be NULL or non-NULL. If NULL, the component object once again has access to its storage object. If it is not NULL, the component object should simulate receiving a call to its **IPersistStorage::HandsOffStorage** method. If the component object cannot simulate this call, its container must be prepared to actually call the **IPersistStorage::HandsOffStorage** method.

The **IPersistStorage::SaveCompleted** method must recursively call any nested objects that are loaded or running.

If this method returns with an error code, the object is not returned to Normal mode. Thus, the container object can attempt different save strategies.

See Also

IAdviseSink::OnSave, **IOleObject::Close**, **IPersistStorage::HandsOffStorage**, **IPersistStorage::Save**, **IRootStorage::SwitchToFile**, **OleSave**

IPersistStream

The **IPersistStream** interface provides methods for saving and loading objects that use a simple serial stream for their storage needs. The primary purpose of this interface is to support OLE moniker implementations. Each of the OLE-provided moniker interfaces provide an **IPersistStream** implementation through which the moniker saves or loads itself. An instance of the OLE generic composite moniker class calls the **IPersistStream** methods of its component monikers to load or save the components in the proper sequence in a single stream.

OLE containers with embedded and linked component objects do not use this interface; they use the **IPersistStorage** interface instead.

When to Implement

Typically, applications do not implement the **IPersistStream** interface. However, the **IMoniker** interface is derived from the **IPersistStream** interface, so you must implement the **IPersistStream** interface if you are implementing a new moniker class.

When to Use

Typically, applications do not use the **IPersistStream** interface directly. The default link handler uses the **IPersistStream** interface to save and load the monikers that identify the link source. These monikers are stored in a stream in the storage for the linked object. If you are writing a custom link handler for your class of components, you will need to use the **IPersistStream** interface to load and save these monikers.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IPersistStream Methods	**Description**
IsDirty	Checks the object for changes since it was last saved.
Load	Initializes an object from the stream where it was previously saved.
Save	Saves an object into the specified stream and indicates whether the object should reset its dirty flag.
GetSizeMax	Return the size in bytes of the stream needed to save the object.

See Also

IMoniker

IPersistStream::GetSizeMax

Returns the size in bytes of the stream needed to save the object.

```
HRESULT GetSizeMax(
  ULARGE_INTEGER *pcbSize        //Pointer to size of stream needed to save object
 );
```

Parameter

pcbSize
[out]Points to a 64-bit unsigned integer value indicating the size in bytes of the stream needed to save this object.

Return Value

S_OK
The size was successfully returned.

Comments

This method returns the size needed to save an object. You can call this method to determine the size and set the necessary buffers before calling the **IPersistStream::Save** method.

Notes to Implementors

The **GetSizeMax** implementation should return a conservative estimate of the necessary size because the caller might call the **IPersistStream::Save** method with a non-growable stream.

See Also

IPersistStream::Save

IPersistStream::IsDirty

Checks the object for changes since it was last saved.

HRESULT IsDirty(*void***);**

Return Values

S_OK
: The object has changed since it was last saved.

S_FALSE
: The object has not changed since the last save.

Comments

This method checks whether an object has changed since it was last saved so you can avoid losing information in objects that have not yet been saved. The dirty flag for an object is conditionally cleared in the **IPersistStream::Save** method.

Notes to Callers

You should treat any error return codes as an indication that the object has changed. In other words, unless this method explicitly returns S_FALSE, you must assume that the object needs to be saved.

Note that the OLE-provided implementations of the **IPersistStream::IsDirty** method in the OLE-provided moniker interfaces always return S_FALSE because their internal state never changes.

See Also

IPersistStream::Save

IPersistStream::Load

Initializes an object from the stream where it was previously saved.

```
HRESULT Load(
  IStream *pStm        //Pointer to the stream from which the object should be loaded
 );
```

Parameter

pStm
[in]Points to the stream from which the object should be loaded.

Return Values

S_OK
The object was successfully loaded.

E_OUTOFMEMORY
The object was not loaded due to a lack of memory.

E_FAIL
The object was not loaded due to some reason other than a lack of memory.

Comments

This method loads an object from its associated stream. The seek pointer is set as it was in the most recent **IPersistStream::Save** method. This method can seek and read from the stream, but cannot write to it.

On exit, the seek pointer must be in the same position it was in on entry, immediately past the end of the data.

Notes to Callers

Rather than calling **IPersistStream::Load** directly, you typically use the **OleLoadFromStream** function which performs the following steps:

1. Calls the **ReadClassStm** function to get the class identifier from the stream.
2. Calls the **CoCreateInstance** function to create an instance of the object.
3. Queries the instance for **IPersistStream**.
4. Calls **IPersistStream::Load**.

The **OleLoadFromStream** function assumes that objects are stored in the stream with a class identifier followed by the object data. This is the storage pattern used by the generic, composite-moniker implementation provided by OLE.

If the objects are not stored using this pattern, you must call the methods separately yourself.

See Also

CoCreateInstance, **OleLoadFromStream**, **ReadClassStm**

IPersistStream::Save

Saves an object to the specified stream.

```
HRESULT Save(
   IStream *pStm,          //Pointer to the stream where the object is to be saved
   BOOL fClearDirty        //Specifies whether to clear the dirty flag
 );
```

Parameters

pStm
[in]Points to the stream into which the object should be saved.

fClearDirty
[in]Indicates whether to clear the dirty flag after the save is complete. If TRUE, the flag should be cleared. If FALSE, the flag should be left unchanged.

Return Values

S_OK
The object was successfully saved to the stream.

STG_E_CANTSAVE
The object could not save itself to the stream. This error could indicate, for example, that the object contains another object that is not serializable to a stream or that an **IStream::Write** call returned STG_E_CANTSAVE.

STG_E_MEDIUMFULL
The object could not be saved because there is no space left on the storage device.

Comments

This method saves an object into the specified stream and indicates whether the object should reset its dirty flag.

The seek pointer is positioned at the location in the stream at which the object should begin writing its data. The object calls the **IStream::Write** method to write its data.

On exit, the seek pointer must be positioned immediately past the object data. The position of the seek pointer is undefined if an error returns.

Notes to Callers

Rather than calling **IPersistStream::Save** directly, you typically use the **OleSaveToStream** helper function which performs the following steps:

1. Calls **IPersistStream::GetClassID** to get the object's class identifier.
2. Calls the **WriteClassStm** function to write the object's class identifier to the stream.
3. Calls **IPersistStream::Save**.

If you call these methods directly, you can write other data into the stream after the class identifier before calling **IPersistStream::Save**.

The OLE-provided implementation of **IPersistStream** follows this same pattern.

Notes to Implementors

The **IPersistStream::Save** method does not write the class identifier to the stream. The caller is responsible for writing the class identifier.

The **IPersistStream::Save** method can read from, write to, and seek in the stream; but it must not seek to a location in the stream before that of the seek pointer on entry.

See Also

IPersist::GetClassID, **IStream::Write**, **OleSaveToStream**

IRootStorage

The **IRootStorage** interface contains a single method that switches a storage object to a different underlying file and saves the storage object to that file. The save operation occurs even with low memory conditions and uncommitted changes to the storage object. A subsequent call to **IStorage::Commit** is guaranteed to not consume any additional memory.

When to Implement

Storage objects that are based on a file should implement **IRootStorage** in addition to the **IStorage** interface. For storage objects that are not file-based, this interface is not necessary.

OLE provides an implementation of a storage object, including the **IRootStorage** interface, as part of its compound file implementation.

When to Use

The primary use for the **IRootStorage** interface is to save a storage object to a file during low memory conditions. Typically, the container application calls the **IRootStorage** interface to switch to a new file.

You can call **IStorage::QueryInterface** with *IID_IRootStorage* to obtain a pointer to the **IRootStorage** interface.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IRootStorage Method	**Description**
SwitchToFile	Copy the file underlying this root storage object, then associate this storage with the copied file.

See Also

IStorage, **StgCreateDocfile**

IRootStorage::SwitchToFile

Copies the current file associated with the storage object to a new file. The new file is then used for the storage object and any uncommitted changes.

```
HRESULT SwitchToFile(
  LPOLESTR pszFile        //Filename for the new file
 );
```

Parameter

pszFile
Specifies the filename for the new file. It cannot be the name of an existing file. If NULL, this method creates a temporary file with a unique name, and you can call **IStorage::Stat** to retrieve the name of the temporary file.

Return Values

S_OK
The file was successfully copied.

STG_E_MEDIUMFULL
The file was not copied because of insufficient space on the storage device.

STG_E_ACCESSDENIED
The file was not copied because the caller does not have permission to access storage device.

STG_E_INVALIDPOINTER
The file was not copied because the *pszFile* pointer is invalid.

STG_E_FILEALREADYEXISTS
The file was not copied because the new filename (*pszFile*) points to an existing file.

Comments

The **IRootStorage::SwitchToFile** method copies the file associated with the storage object. An OLE container calls **IRootStorage::SwitchToFile** to perform a full save on a file in a low-memory situation. Typically, this is done only after a normal full save operation (i.e., save to temporary file, delete original file, rename temporary file) has failed with an E_OUTOFMEMORY error.

It is illegal to call **IRootStorage::SwitchToFile** if the storage object or anything contained within it has been marshalled to another process. As a consequence, the container must call the **IPersistStorage::HandsOffStorage** method for any element within the storage object that is loaded or running. The **HandsOffStorage** method forces the element to release its storage pointers and enter the hands-off storage mode. The container must also release all pointers to streams or storages that are contained in this root storage. After the full save operation is completed, the container returns the contained elements to normal storage mode.

Notes to Implementors

If you are implementing your own storage objects, the **IRootStorage** methods (including **QueryInterface**, **IUnknown::AddRef**, and **IUnknown::Release**) must not consume additional memory or file handles.

See Also

IPersistStorage::HandsOffStorage, **IPersistStorage::SaveCompleted**, **IStorage::Commit**, **IStorage::Stat**

IROTData

The **IROTData** interface is implemented by monikers to enable the Running Object Table (ROT) to compare monikers against each other.

The ROT uses the **IROTData** interface to test whether two monikers are equal. The ROT must do this when, for example, it checks whether a specified moniker is registered as running.

When to Implement

You must implement **IROTData** if you are writing your own moniker class (that is, writing your own implementation of the **IMoniker** interface), and if your monikers are meant to be registered in the ROT.

When to Use

You typically do not need to use this interface. This interface is used by the system's implementation of the ROT.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IROTData Method	**Description**
GetComparisonData	Retrieve data to allow moniker to be compared with another.

See Also

IMoniker, **IRunningObjectTable**

IROTData::GetComparisonData

Retrieves data from a moniker that can be used to test the moniker for equality against another moniker.

```
HRESULT GetComparisonData(
  PVOID *ppvData,         //Buffer that receives the comparison data
  ULONG cbMax,            //Specifies length of buffer
  PULONG pcbData          //Receives the length of the comparison data
 );
```

Parameters

ppvData
[out] Points to a buffer that receives the comparison data.

cbMax
[in] Specifies the length of the buffer specified in *ppvData*.

pcbData
[out] Receives the length of the comparison data.

Return Values

S_OK
The comparison data was successfully returned.

E_OUTOFMEMORY
The buffer contained insufficient space to return the comparison data.

Comments

The **IROTData::GetComparisonData** method is primarily called by the Running Object Table (ROT). The comparison data returned by the method is tested for binary equality against the comparison data returned by another moniker. The *pcbData* parameter enables the ROT to locate the end of the data returned.

Notes to Implementors

The comparison data that you return must uniquely identify the moniker, while still being as short as possible. The comparison data should include information about the internal state of the moniker, as well as the moniker's CLSID. For example, the comparison data for a file moniker would include the pathname stored within the moniker as well as the CLSID of the file moniker implementation. This makes it possible to distinguish two monikers that happen to store similar state information but are instances of different moniker classes.

The comparison data for a moniker cannot exceed 2048 bytes in length. For composite monikers, the total length of the comparison data for all of its components cannot exceed 2048 bytes; consequently, if your moniker can be a component within a composite moniker, the comparison data you return must be significantly less than 2048 bytes.

If your comparison data is longer than the value specified by the *cbMax* parameter, you must return an error. Note that when **IROTData::GetComparisionData** is called on the components of a composite moniker, the value of *cbMax* becomes smaller for each moniker in sequence.

See Also

IMoniker, **IRunningObjectTable**

IRunnableObject

The **IRunnableObject** interface enables a container to control the running of its embedded objects. In the case of an object implemented with a local server, calling IRunnableObject::Run launches the server's .EXE file. In the case of an object implemented with an in-process server, calling the **Run** method causes the object .DLL file to transition into the running state.

When to Implement

Object handlers should implement **IRunnableObject** to provide their containers with a way to run them and manage their running state. DLL object applications should implement **IRunnableObject** to support silent updates of their objects.

When to Use

Containers call **IRunnableObject** to determine if an embedded object is running, to force an object to run, to lock an object into the running state, or to inform an object handler whether its object is being run as either a simple embedding or as a link source.

Methods VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IRunnableObject Methods	Description
GetRunningClass	Returns CLSID of a running object.
Run	Forces an object to run.
IsRunning	Determines if an object is running.
LockRunning	Locks an object into running state.
SetContainedObject	Indicates that an object is embedded.

IRunnableObject::GetRunningClass

Returns the CLSID of a running object.

```
HRESULT GetRunningClass(
  LPCLSID lpClsid         //Pointer to an object's CLSID
 );
```

Parameter

lpClsid
[out] Points to the object's class ID.

Return Values

S_OK
CLSID was returned successfully.

E_INVALIDARG
An argument is invalid.

E_UNEXPECTED
An unexpected error occurred.

Comments

If an embedded document was created by an application that is not available on the user's computer, the document, by a call to **CoTreatAsClass**, may be able to display itself for editing by emulating a class that *is* supported on the user's machine. In this case, the CLSID returned by a call to **IRunnableObject::GetRunningClass** will be that of the class being emulated, rather than the document's native class.

See Also

CoTreatAsClass

IRunnableObject::IsRunning

Determines whether an object is currently in the running state.

BOOL IsRunning();

Return Values

TRUE
: The object is in the running state.

FALSE
: The object is not in the running state.

Comments

A container application could call **IRunnableObject::IsRunning** when it needs to know if the server is immediately available. For example, a container's implementation of the **IOleItemContainer::GetObject** method would return an error if the server is not running and the bindspeed parameter specifies BINDSPEED_IMMEDIATE.

An object handler could call **IRunnableObject::IsRunning** when it wants to avoid conflicts with a running server or when the running server might have more up-to-date information. For example, a handler's implementation of **IOleObject::GetExtent** would delegate to the object server if it is running, because the server's information might be more current than that in the handler's cache.

OleIsRunning is a helper function that conveniently repackages the functionality offered by **IRunnableObject::IsRunning**. With the release of OLE 2.01, the implementation of **OleIsRunning** was changed so that it calls **QueryInterface**, asks for **IRunnableObject**, and then calls **IRunnableObject::IsRunning**. In other words, you can use the interface and the helper function interchangeably.

See Also

OleIsRunning

IRunnableObject::LockRunning

Locks an already-running object into its running state or unlocks it from its running state.

```
HRESULT LockRunning(
    BOOL fLock,                   //Flag indicating whether object is locked
    BOOL fLastUnlockCloses        //Flag indicating whether to close object
  );
```

Parameters

fLock
[in] TRUE locks the object into its running state. FALSE unlocks the object from its running state.

fLastUnlockCloses
[in] TRUE specifies that if the connection being released is the last external lock on the object, the object should close. FALSE specifies that the object should remain open until closed by the user or another process.

Return Values

S_OK
If the value of *fLock* is TRUE, the object was successfully locked; if the value of *fLock* is FALSE, the object was successfully unlocked.

E_FAIL
The object was not running.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

Comments

Most implementations of **IRunnableObject::LockRunning** call **CoLockObjectExternal**.

OleLockRunning is a helper function that conveniently repackages the functionality offered by **IRunnableObject::LockRunning**. With the release of OLE 2.01, the implementation of **OleLockRunning** was changed to call **QueryInterface**, ask for **IRunnableObject**, and then call **IRunnableObject::LockRunning**. In other words, you can use the interface and the helper function interchangeably.

See Also

CoLockObjectExternal

IRunnableObject::Run

Runs an object.

```
HRESULT Run(
    LPBC lpbc          //Pointer to binding context
  );
```

Parameter

lpbc
    Points to the binding context of the run operation. May be NULL.

Return Values

S_OK
    The object was successfully placed in the running state.

E_OUTOFMEMORY
    Out of memory.

E_UNEXPECTED
    An unexpected error occurred.

Comments

Containers call **IRunnableObject::Run** to force their objects to enter the running state. If the object is not already running, calling **IRunnableObject::Run** can be an expensive operation, on the order of many seconds. If the object is already running, then this method has no effect on the object.

Notes to Callers

When called on a linked object that has been converted to a new class since the link was last activated, **IRunnableObject::Run** may return OLE_E_CLASSDIFF. In this case, the client should call **IOleLink::BindToSource**.

OleRun is a helper function that conveniently repackages the functionality offered by **IRunnableObject::Run**. With the release of OLE 2.01, the implementation of **OleRun** was changed so that it calls **QueryInterface**, asks for **IRunnableObject**, and then calls **IRunnableObject::Run**. In other words, you can use the interface and the helper function interchangeably.

Notes to Implementors

The object should register in the running object table if it has a moniker assigned. The object should not hold any strong locks on itself; instead, it should remain in the unstable, unlocked state. The object should be locked when the first external connection is made to the object.

An embedded object must hold a lock on its embedding container while it is in the running state. The Default handler provided by OLE 2 takes care of locking the embedding container on behalf of objects implemented by an EXE object application. Objects implemented by a DLL object application must explicitly put a lock on their embedding containers, which they do by first calling **IOleClientSite::Getcontainer** to get a pointer to the container, then calling **IOleContainer::LockContainer** to actually place the lock. This lock must be released when **IOleObject::Close** is called.

See Also

IOleLink::BindToSource, **OleRun**

IRunnableObject::SetContainedObject

Notifies an object that it is embedded in an OLE container, which ensures that reference counting is done correctly for containers that support links to embedded objects.

```
HRESULT SetContainedObject(
  BOOL fContained         //Flag indicating whether object is embedded
 );
```

Parameter

fContained
[in] TRUE specifies that the object is contained in an OLE container. FALSE indicates that it is not.

Return Values

S_OK
Object has been marked as a contained embedding.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
The argument is invalid.

E_UNEXPECTED
An unexpected error occurred.

Comments

The **IRunnableObject::SetContainedObject** method enables a container to inform an object handler that it is embedded in the container, rather than acting as a link. This call changes the container's reference on the object from strong, the default for external connections, to weak. When the object is running visibly, this method is of little significance because the end-user has a lock on the object. During a silent update of an embedded link source, however, the container should not be able to hold an object in the running state after the link has been broken. For this reason, the container's reference to the object must be weak.

Notes to Callers

A container application must call **IRunnableObject::SetContainedObject** if it supports linking to embedded objects. It normally makes the call immediately after calling **OleLoad** or **OleCreate** and never calls the method again, even before it closes. Moreover, a container almost always calls this method with *fContained* set to TRUE. The use of this method with *fContained* set to FALSE is rare.

Calling **IRunnableObject::SetContainedObject** is optional only when you know that the embedded object will not be referenced by any client other than the container. If your container application does not support linking to embedded objects; it is preferable, but not necessary, to call **IRunnableObject::SetContainedObject**.

OleSetContainedObject is a helper function that conveniently repackages the functionality offered by **IRunnableObject::SetContainedObject**. With the release of OLE 2.01, the implementation of **OleSetContainedObject** was changed to call **QueryInterface**, ask for **IRunnableObject**, and then call **IRunnableObject::SetContainedObject**. In other words, you can use the interface and the helper function interchangeably.

See Also

OleSetContainedObject, OleNoteObjectVisible, CoLockObjectExternal

IRunningObjectTable

The **IRunningObjectTable** interface manages access to the Running Object Table (ROT), a globally accessible lookup table on each workstation. A workstation's ROT keeps track of those objects that can be identified by a moniker and that are currently running on the workstation. When a client tries to bind a moniker to an object, the moniker checks the ROT to see if the object is already running; this allows the moniker to bind to the current instance instead of loading a new one.

The ROT contains entries of the form:

```
(pmkObjectName, pUnkObject)
```

The *pmkObjectName* element is a pointer to the moniker that identifies the running object. The *pUnkObject* element is a pointer to the running object itself. During the binding process, monikers consult the *pmkObjectName* entries in the Running Object Table to see if an object is already running.

Objects that can be named by monikers must be registered with the ROT when they are loaded and their registration must be revoked when they are no longer running.

When to Implement

You do not need to implement this interface. The system provides an implementation of the Running Object Table that is suitable for all situations.

When to Use

You typically use the ROT if you're a moniker provider (that is, you hand out monikers identifying your objects to make them accessible to others) or if you're writing your own moniker class (that is, implementing the **IMoniker** interface).

If you are a moniker provider, you register your objects with the ROT when they begin running and revoke their registrations when they are no longer running. This enables the monikers that you hand out to be bound to running objects. You should also use the ROT to record the object's last modification time. You can get an **IRunningObjectTable** interface pointer to the local ROT by calling the **GetRunningObjectTable** API function.

The most common type of moniker provider is a compound-document link source. This includes server applications that support linking to their documents (or portions of a document) and container applications that support linking to embeddings within their documents. Server applications that do not support linking can also use the ROT to cooperate with container applications that support linking to embeddings.

If you are writing your own moniker class, you use the ROT to determine whether a object is running and to retrieve the object's last modification time. You can get an **IRunningObjectTable** interface pointer to the local ROT by calling the **IBindCtx::GetRunningObjectTable** method on the bind context for the current binding operation. Moniker implementations should always use the bind context to acquire a pointer to the ROT; this allows future implementations of **IBindCtx** to modify binding behavior. Note that you must also implement the **IROTData** interface on your moniker class in order to allow your monikers to be registered with the ROT.

Methods in VTable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IRunningObjectTable Methods	**Description**

Register	Registers an object with the ROT.
Revoke	Revokes an object's registration with the ROT.
IsRunning	Checks whether an object is running.
GetObject	Returns a pointer to an object given its moniker.
NoteChangeTime	Notifies the ROT that an object has changed.
GetTimeOfLastChange	Returns the time an object was last changed.
EnumRunning	Returns an enumerator for the ROT.

See Also

IBindCtx::GetRunningObjectTable, IROTData, GetRunningObjectTable

IRunningObjectTable::EnumRunning

Returns an enumerator that can list the monikers of all the objects currently registered in the Running Object Table (ROT).

```
HRESULT EnumRunning(
  IEnumMoniker **ppenumMoniker        //Receives enumerator for ROT
 );
```

Parameter

ppenumMoniker
[out] Receives a pointer to an **IEnumMoniker** interface to the enumerator. If an error occurs; the implementation sets **ppenumMoniker* to NULL. If **ppenumMoniker* is non-NULL, the implementation calls **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK
Indicates that an enumerator was successfully returned.

E_OUTOFMEMORY
Indicates insufficient memory.

Comments

The *ppenumMoniker* enumerator does not enumerate monikers that are registered in the ROT after the enumerator has been created.

The **EnumRunning** method is intended primarily for the use by the system in implementing the Alert Object Table. Note that OLE 2 does not include an implementation of the Alert Object Table.

See Also

IEnum*XXXX*

IRunningObjectTable::GetObject

Determines whether the object identified by the specified moniker is running, and if it is, retrieves a pointer to that object. This method looks for the moniker in the Running Object Table (ROT), and retrieves the pointer registered there.

```
HRESULT GetObject(
   IMoniker *pmkObjectName,        //Moniker identifying object desired
   IUnknown **ppunkObject          //Receives pointer to the object
  );
```

Parameters

pmkObjectName
[in] Points to the moniker to search for in the Running Object Table.

ppunkObject
[out] Receives a pointer to the running object. If the object is not running or if an error occurs, the implementation sets **ppunkObject* to NULL. If **ppunkObject* is non-NULL, the implementation calls **IUnknown::AddRef** on the parameter; it is the caller's responsibility to call **IUnknown::Release**.

Return Values

S_OK
Indicates that *pmkObjectName* was found in the ROT and a pointer was returned.

S_FALSE
Indicates that there is no entry for *pmkObjectName* in the ROT, or that the object it identifies is no longer running (in which case, the entry is revoked).

Comments

This method checks the ROT for the moniker specified by *pmkObjectName*. If that moniker had previously been registered with a call to **IRunningObjectTable::Register**, this method returns the pointer that was registered at that time.

Notes to Callers

Generally, you call the **IRunningObjectTable::GetObject** method only if you are writing your own moniker class (that is, implementing the **IMoniker** interface). You typically call this method from your implementation of **IMoniker::BindToObject**.

However, note that not all implementations of **IMoniker::BindToObject** need to call this method. If you expect your moniker to have a prefix (indicated by a non-NULL *pmkToLeft* parameter to **IMoniker::BindToObject**), you should not check the ROT. The reason for this is that only complete monikers are registered with the ROT, and if your moniker has a prefix, your moniker is part of a composite and thus not complete. Instead, your moniker should request services from the object identified by the prefix (for example, the container of the object identified by your moniker).

See Also

IMoniker::BindToObject

IRunningObjectTable::GetTimeOfLastChange

Returns the time that an object was last modified. The object must have previously been registered with the Running Object Table (ROT). This method looks for the last change time recorded in the ROT.

```
HRESULT GetTimeOfLastChange(
  IMoniker *pmkObjectName,     //Moniker identifying object whose status is desired
  FILETIME *pfiletime          //Object's last change time
 );
```

Parameters

pmkObjectName
[in] Points to the moniker to search for in the ROT.

pfiletime
[out] Points to a **FILETIME** structure that receives the object's last change time.

Return Values

S_OK
Indicates that the last change time was successfully retrieved.

S_FALSE
Indicates that there is no entry for *pmkObjectName* in the ROT, or that the object it identifies is no longer running (in which case, the entry is revoked).

Comments

This method returns the change time that was last reported for this object by a call to **IRunningObjectTable::NoteChangeTime**. If **IRunningObjectTable::NoteChangeTime** has not been called previously, the method returns the time that was recorded when the object was registered.

This method is provided to enable checking whether a connection between two objects (represented by one object holding a moniker that identifies the other) is up-to-date. For example, if one object is holding cached information about the other object, this method can be used to check whether the object has been modified since the cache was last updated. See **IMoniker::GetTimeOfLastChange**.

Notes to Callers

Generally, you call **IRunningObjectTable::GetTimeOfLastChange** only if you are writing your own moniker class (that is, implementing the **IMoniker** interface). You typically call this method from your implementation of **IMoniker::GetTimeOfLastChange**. However, you should do so only if the *pmkToLeft* parameter of **IMoniker::GetTimeOfLastChange** is NULL. Otherwise, you should call **IMoniker::GetTimeOfLastChange** on your *pmkToLeft* parameter instead.

See Also

IMoniker::GetTimeOfLastChange, **IRunningObjectTable::NoteChangeTime**

IRunningObjectTable::IsRunning

Determines whether the object identified by the specified moniker is currently running. This method looks for the moniker in the Running Object Table (ROT).

```
HRESULT IsRunning(
   IMoniker *pmkObjectName        //Moniker identifying the object whose status is desired
  );
```

Parameter

pmkObjectName
[in] Points to the moniker to search for in the Running Object Table.

Return Values

S_OK
Indicates that the object identified by *pmkObjectName* is running.

S_FALSE
Indicates that there is no entry for *pmkObjectName* in the ROT, or that the object it identifies is no longer running (in which case, the entry is revoked).

Comments

This method simply indicates whether a object is running. To retrieve a pointer to a running object, use the **IRunningObjectTable::GetObject** method.

Notes to Callers

Generally, you call the **IRunningObjectTable::IsRunning** method only if you are writing your own moniker class (that is, implementing the **IMoniker** interface). You typically call this method from your implementation of **IMoniker::IsRunning**. However, you should do so only if the *pmkToLeft* parameter of **IMoniker::IsRunning** is NULL. Otherwise, you should call **IMoniker::IsRunning** on your *pmkToLeft* parameter instead.

See Also

IMoniker::IsRunning

IRunningObjectTable::NoteChangeTime

Records the time that a running object was last modified. The object must have previously been registered with the Running Object Table (ROT). This method stores the time of last change in the ROT.

```
HRESULT NoteChangeTime(
   DWORD dwRegister,        //Value identifying registration being updated
   FILETIME *pfiletime      //Time containing object's last change time
  );
```

Parameters

dwRegister
[in] Specifies a value identifying the ROT entry of the changed object. This value was previously returned by **IRunningObjectTable::Register**.

pfiletime
[in] Points to a **FILETIME** structure containing the object's last change time.

Return Values

S_OK
Indicates that the change time was recorded successfully.

E_INVALIDARG
Indicates that *dwRegister* is not a valid ROT entry or *pfiletime* is an invalid pointer.

Comments

The time recorded by this method can be retrieved by calling **IRunningObjectTable::GetTimeOfLastChange**.

This method is provided to enable a program to check whether a connection between two objects (represented by one object holding a moniker that identifies the other) is up-to-date. For example, if one object is holding cached information about the other object, this method can be used to check whether the object has been modified since the cache was last updated. See **IMoniker::GetTimeOfLastChange**.

Notes to Callers

If you're a moniker provider (that is, you hand out monikers identifying your objects to make them accessible to others), you must call the **IRunningObjectTable::NoteChangeTime** method whenever your objects are modified. You must have previously called **IRunningObjectTable::Register** and stored the identifier returned by that method; you use that identifier when calling **IRunningObjectTable::NoteChangeTime**.

The most common type of moniker provider is a compound-document link source. This includes server applications that support linking to their documents (or portions of a document) and container applications that support linking to embeddings within their documents. Server applications that do not support linking can also use the ROT to cooperate with container applications that support linking to embeddings.

When an object is first registered in the ROT, the ROT records its last change time as the value returned by calling **IMoniker::GetTimeOfLastChange** on the moniker being registered.

See Also

IRunningObjectTable::GetTimeOfLastChange, **IMoniker::GetTimeOfLastChange**

IRunningObjectTable::Register

Registers an object and its identifying moniker in the Running Object Table (ROT).

```
HRESULT Register(
  DWORD grfFlags,                 //Specifies a weak or a strong reference
  IUnknown *punkObject,           //Pointer to the object being registered
  IMoniker *pmkObjectName,        //Moniker that identifies the object being registered
  DWORD *pdwRegister              //Receives value identifying the registration
);
```

Parameters

grfFlags
[in] Specifies whether the ROT's reference to *punkObject* is weak or strong. This value must be either zero, indicating a weak reference that does not call **IUnknown::AddRef**; or ROTFLAGS_REGISTRATIONKEEPSALIVE, indicating a strong reference that calls **IUnknown::AddRef** and can keep the object running. If a strong reference is registered, a strong reference is released when the object's registration is revoked. Most callers specify zero, indicating a weak reference.

punkObject
[in] Points to the object that is being registered as running.

pmkObjectName
[in] Points to the moniker that identifies *punkObject*.

pdwRegister
[out] Receives a 32-bit value that can be used to identify this ROT entry in subsequent calls to **IRunningObjectTable::Revoke** or **IRunningObjectTable::NoteChangeTime**. The caller cannot specify NULL for this parameter. If an error occurs, **pdwRegister* is set to zero.

Return Values

S_OK
Indicates that the object was successfully registered.

MK_S_MONIKERALREADYREGISTERED
Indicates that the moniker/object pair was successfully registered, but that another object (possibly the same object) has already been registered with the same moniker.

E_INVALIDARG
Indicates one or more invalid arguments.

E_OUTOFMEMORY
Indicates failure due to insufficient memory.

Comments

This method registers a pointer to an object under a moniker that identifies the object. The moniker is used as the key when the table is searched with **IRunningObjectTable::GetObject**.

Registering a second object with the same moniker, or re-registering the same object with the same moniker, creates a second entry in the ROT. In this case, **IRunningObjectTable::Register** returns MK_S_MONIKERALREADYREGISTERED. Each call to **IRunningObjectTable::Register** must be matched by a call to **IRunningObjectTable::Revoke** because even duplicate entries have different *pdwRegister* identifiers. A problem with duplicate registrations is that there is no way to determine which object will be returned if the moniker is specified in a subsequent call to **IRunningObjectTable::IsRunning**.

Notes to Callers

If you're a moniker provider (that is, you hand out monikers identifying your objects to make them accessible to others), you must call the **IRunningObjectTable::Register** method to register your objects when they begin running. You must also call this method if you rename your objects while they are loaded.

The most common type of moniker provider is a compound-document link source. This includes server applications that support linking to their documents (or portions of a document) and container applications that support linking to embeddings within their documents. Server applications that do not support linking can also use the ROT to cooperate with container applications that support linking to embeddings.

If you're writing a server application, you should register an object with the ROT when it begins running, typically in your implementation of **IOleObject::DoVerb**. The object must be registered under its full moniker, which requires getting the moniker of its container document using **IOleClientSite::GetMoniker**. You should also revoke and re-register the object in your implementation of **IOleObject::SetMoniker**, which is called if the container document is renamed.

If you're writing a container application that supports linking to embeddings, you should register your document with the ROT when it is loaded. If your document is renamed, you should revoke and re-register it with the ROT and call **IOleObject::SetMoniker** for any embedded objects in the document to give them an opportunity to re-register themselves.

You must cache the identifier returned in *pdwRegister*, and use it in a call to **IRunningObjectTable::Revoke** to revoke the registration when the object is no longer running or when its moniker changes. This revocation is important because there is no way for the system to automatically remove entries from the ROT.

The system's implementation of **IRunningObjectTable::Register** calls **IMoniker::Reduce** on the *pmkObjectName* parameter to ensure that the moniker is fully reduced before registration. If a object is known by more than one fully reduced moniker, then it should be registered under all such monikers.

See Also

IMoniker::Reduce, **IOleClientSite::GetMoniker**, **IOleObject::SetMoniker**, **IRunningObjectTable::IsRunning**, **IRunningObjectTable::Revoke**

IRunningObjectTable::Revoke

Removes from the Running Object Table (ROT) an entry that was previously registered by a call to **IRunningObjectTable::Register**.

```
HRESULT Revoke(
  DWORD dwRegister      //Value identifying registration to be revoked
 );
```

Parameter

dwRegister
[in] Specifies a value identifying the ROT entry to revoke. This value was previously returned by **IRunningObjectTable::Register**.

Return Values

S_OK
Indicates that the object's registration was successfully revoked.

E_INVALIDARG
Indicates that *dwRegister* is not valid.

Comments

This method undoes the effect of a call to **IRunningObjectTable::Register**, removing both the moniker and the pointer to the object identified by that moniker.

Notes to Callers

If you're a moniker provider (that is, you hand out monikers identifying your objects to make them accessible to others), you must call the **IRunningObjectTable::Revoke** method to revoke the registration of your objects when they stop running. You must have previously called **IRunningObjectTable::Register** and stored the identifier returned by that method; you use that identifier when calling **IRunningObjectTable::Revoke**.

The most common type of moniker provider is a compound-document link source. This includes server applications that support linking to their documents (or portions of a document) and container applications that support linking to embeddings within their documents. Server applications that do not support linking can also use the ROT to cooperate with container applications that support linking to embeddings.

If you're writing a container application, you must revoke a document's registration when the document is closed. You must also revoke a document's registration before re-registering it when it is renamed.

If you're writing a server application, you must revoke an object's registration when the object is closed. You must also revoke an object's registration before re-registering it when its container document is renamed (see **IOleObject::SetMoniker**).

See Also

IOleObject::SetMoniker, **IRunningObjectTable::Register**

IStdMarshalInfo

The **IStdMarshalInfo** interface returns the CLSID identifying the handler to be used in the destination process during standard marshaling.

Even if an object uses standard marshaling instead of custom marshaling, it can still specify a handler that should be loaded in the client process. Such a handler would handle certain requests locally and delegate others back to the original object (using standard marshaling). To get the CLSID of the handler to be loaded, OLE queries the object for the **IStdMarshalInfo** interface and then the IPersist interface. If neither interface is supported, a standard handler is used.

When to Implement

If you are writing a server application that supports class emulation (that is, if your server can manipulate objects of another type in response to the Activate As option in the Convert dialog box), you must implement the **IStdMarshalInfo** interface

Note that your handler must aggregate the default handler.

When to Use

You typically don't call this interface yourself. OLE queries for this interface when performing standard marshaling.

Methods in VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IStdMarshalInfo Method	Description
GetClassForHandler	

See Also

IMarshal

IStdMarshalInfo::GetClassForHandler

Retrieves the CLSID of the object handler to be used in the destination process during standard marshaling.

```
HRESULT GetClassForHandler(
   DWORD dwDestContext,          //Specifies the destination process
   void * pvDestContext,         //Reserved for future use
   CLSID * pClsid                //Receives the CLSID
  );
```

Parameters

dwDestContext
[in] Specifies the destination context, that is, the process in which the unmarshaling will be done. The legal values for *dwDestContext* are taken from the enumeration **MSHCTX**. For information on the **MSHCTX** enumeration, see the "Data Structures" section.

pvDestContext
[in] Reserved for use with future **MSHCTX** values.

pClsid
[out] On exit, contains the handler's CLSID.

Return Values

S_OK
The CLSID was retrieved successfully.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
One or more arguments are invalid.

E_UNEXPECTED
An unexpected error occurred.

Comments

Notes to Implementors

Your implementation of **IStdMarshalInfo::GetClassForHandler** must return your own CLSID. This allows an object created by a different server to behave as one your server created.

IStorage

The **IStorage** interface supports the creation and management of structured storage objects. These storage objects provide hierarchical storage of information within a single file similar to a file system within a file. Storage objects are like the directories in a file system. They provide the structure in a structured storage object. They can contain other storage objects which can be nested indefinitely, or stream objects. Stream objects are like the files within a file system. They contain the actual data content in a storage object.

The **IStorage** interface provides methods for creating and managing the root storage object, child storage objects, and stream objects. These methods can create, open, enumerate, move, copy, rename, or delete the elements in the storage object.

An application must release its **IStorage** pointers when it is done with the storage object to deallocate memory used. The **IStorage** interface provides a **IUnknown::Release** method. There are also methods for changing the date and time of an element.

Storage objects can be opened in direct or transacted mode. In direct mode, changes are immediately reflected in the element that changed. In transacted mode, changes are not reflected in the element until explicitly committed. This feature allows you to revert to a previous version without committing the changes. The **IStorage** interface provides methods for committing changes and reverting to previous versions of the storage object.

Storage objects and their elements have a set of access flags that control how the elements can be used. For example, a stream can be opened in read only mode or read/write mode. See the **STGM** enumeration values for more information on these flags. Storage objects and their elements also have a set of statistics defined by the **STATSTG** structure.

When to Implement

OLE provides an implementation of the **IStorage** interface in compound files. The compound file implementation supports transacted access. OLE provides a set of helper APIs for using the compound file implementation of storage objects.

Container or object applications can implement their own storage objects instead of using the OLE-provided implementation.

When to Use

Container applications use storage objects for compound files. For example, a word processing document could contain a chart. The chart would be provided its own storage object within the root storage object of the containing application while document pages are stored in a separate storage object in the same root storage object.

Since storage object pointers can be marshaled to other processes, applications can share the data in storage objects without having to use global memory. For the OLE-provided implementation of compound files, the custom marshaling facilities in OLE create a remote version of the original object in the new process when the two processes have shared memory access. Thus, the remote version does not need to communicate with the original process to carry out its functions.

While the **IStorage** and **IStream** interfaces are used to deal with the storage object and its elements, the **IPersistStorage** interface actually serializes the storage object and its elements to a disk file.

Methods VTable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.

Release	Decrements the reference count.

IStorage Methods	Description
CreateStream	Creates and opens a stream object with the specified name contained in this storage object.
OpenStream	Opens an existing stream object within this storage object using the specified access permissions in *grfMode*.
CreateStorage	Creates and opens a new storage object within this storage object.
OpenStorage	Opens an existing storage object with the specified name according to the specified access mode.
CopyTo	Copies the entire contents of this open storage object into another storage object. The layout of the destination storage object may differ.
MoveElementTo	Copies or moves a substorage or stream from this storage object to another storage object.
Commit	Reflects changes for a transacted storage object to the parent level.
Revert	Discards all changes that have been made to to the storage object since the last commit operation.
EnumElements	Returns an enumerator object that can be used to enumerate the storage and stream objects contained within this storage object.
DestroyElement	Removes the specified element from this storage object.
RenameElement	Renames the specified element in this storage object.
SetElementTimes	Sets the modification, access, and creation times of the indicated storage element, if supported by the underlying file system.
SetClass	Assigns the specified CLSID to this storage object.
SetStateBits	Stores up to 32 bits of state information in this storage object.
Stat	Returns the STATSTG structure for this open storage object.

IStorage::Commit

Ensures that any changes made to a storage object open in transacted mode are reflected in the parent storage. If the storage object is the root storage, the **IStorage::Commit** method reflects the changes in the actual device, for example, a file on disk. If the storage object is the root storage and is open in direct mode, this method has no effect except for flushing all memory buffers to the disk. For non-root storage objects in direct mode, this method has no effect.

```
HRESULT Commit(
   DWORD grfCommitFlags        //Specifies how changes are to be committed
  );
```

Parameter

grfCommitFlags
[in]Controls how the changes are committed to the storage object. See the **STGC** enumeration for a definition of these values.

Return Values

S_OK
Changes to the storage object were successfully committed to the parent level.

STG_E_INVALIDFLAG
The value for the *grfCommitFlags* parameter is not valid.

STG_E_INVALIDPARAMETER
One of the parameters was not valid.

STG_E_NOTCURRENT
Another open instance of the storage object has committed changes. Thus, the current commit operation may overwrite previous changes.

STG_E_MEDIUMFULL
No space left on device to commit.

STG_E_TOOMANYOPENFILES
The commit operation could not be completed because there are too many open files.

STG_E_REVERTED
The storage object has been invalidated by a revert operation above it in the transaction tree.

Comments

Storage objects can be opened in direct or transacted mode. In direct mode, changes are immediately reflected in the storage object. In transacted mode, changes are accumulated and are not reflected in the storage object until an explicit commit operation is done. The commit operation publishes the currently known changes in this storage object and its children to the next higher level in the storage hierarchy. Transacted mode allows an efficient way to revert to a previous version undoing changes prior to the commit operation.

Changes that have been made to this storage object since it was opened or since the last commit operation are reflected in the storage object. The changes are published to the next level of the storage hierarchy. If the parent is opened in transacted mode, the parent may still revert at a later time, rolling back the changes to this storage object.

Calling **IStorage::Commit** has no effect on currently-opened nested elements of this storage object. They are still valid and can be used. However, the **IStorage::Commit** method does not automatically commit changes to these nested elements. The commit operation publishes only known changes to the next higher level of the storage hierarchy. Thus, transactions to nested levels must be committed to this storage object before they can be committed to higher levels.

For storage objects opened in direct mode, the commit operation does nothing except in the case of a root storage object. In this case, the commit operation is similar to a flush buffer operation ensuring that changes in memory buffers are written to the underlying storage device.

In commit operations, steps must be taken to ensure the robustness of data during the commit process:

- When committing changes to root storage objects, the caller must take care to ensure that the operation successfully completes or that, when failing, the old committed contents of the **IStorage** are still intact and can be restored.
- If this storage object was opened with some of its items excluded, then the caller is responsible for rewriting them before calling commit. Write mode is required on the storage opening for the commit to succeed.
- Unless they prohibit multiple simultaneous writers on the same storage object, applications will usually want to specify at least STGC_ONLYIFCURRENT in the *grfCommitFlags* parameter to prevent the changes made by one writer from inadvertently overwriting the changes made by another.

Note to Callers

The OLE-provided compound files use a two phase commit process unless STGC_OVERWRITE is specified in the *grfCommitFlags* parameter. This two-phase process ensures the robustness of data in case the commit operation fails. First, all new data is written to unused space in the underlying file. If necessary, new space is allocated to the file. Once this step has been successfully completed, a table in the file is updated using a single sector write to indicate that the new data is to be used in place of the old. The old data becomes free space to be used at the next commit. Thus, the old data is available and can be restored in case an error occurs when committing changes. If STGC_OVERWRITE is specified, a single phase commit operation is used.

See Also

STGC, **IStorage::Revert**

IStorage::CopyTo

Copies the entire contents of this open storage object into another storage object.

```
HRESULT CopyTo(
  DWORD ciidExclude,            //Number of elements in rgiidExclude
  IID const * rgiidExclude,     //Array of interface identifiers (IIDs)
  SNB snbExclude,               //Points to a block of stream names in the storage object
  IStorage * pstgDest           //Points to destination storage object
 );
```

Parameters

ciidExclude
: [in]The number of elements in the array pointed to by *rgiidExclude*. If *rgiidExclude* is NULL, then *ciidExclude* is ignored.

rgiidExclude
: [in]Specifies an array of interface identifiers the caller takes responsibility for copying from the source to the destination. This array can include interfaces that this storage object does not support. It can be NULL, indicating that all other interfaces are to be copied. If non-NULL, an array length of zero indicates that no other interfaces are to be copied, only the state exposed by the **IStorage** object.

snbExclude
: [in]Specifies a block of elements in this storage object that are not to be copied to the destination. These elements are not created in the destination. If IID_IStorage is in the *rgiidExclude* array, then this parameter is ignored. This parameter may be NULL.

pstgDest
: [in]Points to the open storage object into which this storage object is to be copied. The destination storage object can be a different implementation of the **IStorage** interface from the source storage object. Thus, **IStorage::CopyTo** can only use publicly available methods of the destination storage object. If *pstgDest* is open in transacted mode, it can be reverted by calling its **IStorage::Revert** method.

Return Values

S_OK
: The storage object was successfully copied.

STG_S_DESTLACKSINTERFACE
: The destination lacks an interface that the source requested to be copied. The lack of an interface does not prevent other parts of the copy operation to be successfully completed.

STG_E_ACCESSDENIED
: The destination storage object is a child of the source storage object.

STG_E_INSUFFICIENTMEMORY
: The copy was not completed due to a lack of memory.

STG_E_INVALIDPOINTER
: The pointer specified for the storage object was invalid.

STG_E_INVALIDPARAMETER
: One of the parameters was invalid.

STG_E_TOOMANYOPENFILES
: The copy was not completed because there are too many open files.

STG_E_MEDIUMFULL
: The copy was not completed because the storage medium is full.

Comments

This method merges elements contained in the source storage object with those already present in the destination. The layout of the destination storage object may differ from the source storage object.

The copy process is recursive, invoking **IStorage::CopyTo** and **IStream::CopyTo** on the elements nested inside the source.

If an attempt is made to copy a stream on top of an existing stream with the same name, the existing destination stream is first removed and then replaced with the source stream. Attempting to copy a source storage object on top of an existing destination storage object does not remove the existing storage. Thus, after the copy operation, the destination **IStorage** contains older elements if they were not replaced by newer ones.

Note to Callers

The following example illustrates using the **CopyTo** method to copy everything possible from the source to the destination:

```
pstg->CopyTo(0, Null, Null, pstgDest)
```

This is the most commonly used form of this operation, used in most Full Save and SaveAs scenarios.

See Also

IStorage::MoveElementTo, **IStorage::Revert**

IStorage::CreateStorage

Creates and opens a new storage object nested within this storage object.

```
HRESULT CreateStorage(
    const wchar_t * pwcsName,      //Points to the name of the new storage object
    DWORD grfMode,                 //Access mode for the new storage object
    DWORD reserved1,               //Reserved; must be zero
    DWORD reserved2,               //Reserved; must be zero
    IStorage ** ppstg              //Points to new storage object
  );
```

Parameters

pwcsName
[in]Points to the name of the newly created storage object. This name can be used later to open or reopen the storage object.

grfMode
[in]Specifies the access mode to use when opening the newly created storage object. See the **STGM** enumeration values for descriptions of the possible values.

reserved1
[in]Reserved for future use; must be zero.

reserved2
[in]Reserved for future use; must be zero.

ppstg
[out]Points to the location where the new **IStorage** interface pointer is returned; only valid if the operation is successful. This parameter is set to NULL if an error occurs.

Return Values

S_OK
The storage object was created successfully.

STG_E_ACCESSDENIED
Insufficient permissions to create storage object.

STG_E_FILEALREADYEXISTS
The name specified for the storage object already exists in the storage object and the *grfmode* flag includes the flag STGM_FAILIFTHERE.

STG_E_INSUFFICIENTMEMORY
The storage object was not created due to a lack of memory.

STG_E_INVALIDFLAG
The value specified for the *grfMode* flag is not a valid **STGM** enumeration value.

STG_E_INVALIDFUNCTION
The specified combination of *grfMode* flags is not supported.

STG_E_INVALIDNAME
Invalid value for *pwcsName*.

STG_E_INVALIDPOINTER
The pointer specified for the storage object was invalid.

STG_E_INVALIDPARAMETER
One of the parameters was invalid.

STG_E_REVERTED
The object has been invalidated by a revert operation above it in the transaction tree.

STG_E_TOOMANYOPENFILES
The storage object was not created because there are too many open files.

STG_S_CONVERTED

The existing stream with the specified name was replaced with a new storage object containing a single stream called CONTENTS. The new storage object will be added.

Comments

If an element with the name specified in the *pwcsName* parameter already exists and the *grfMode* parameter includes the STGM_CREATE flag, the existing element is replaced by the new storage object. If the *grfMode* parameter includes the STGM_CONVERT flag, the existing element is converted to a stream object named CONTENTS and the new storage object is created containing the CONTENTS stream object. The destruction of the old element and the creation of the new storage object are both subject to the transaction mode on the parent storage object.

If the storage object already exists and *grfMode* is set to STGM_FAILIFTHERE, this method fails with the return value STG_E-FILEALREADYEXISTS.

Note to Callers

The OLE-provided compound file implementation of the **IStorage::CreateStorage** method does not support the STGM_DELETEONRELEASE flag. Other implementations of **IStorage** might support this flag.

See Also

IStorage::OpenStorage

IStorage::CreateStream

Creates and opens a stream object with the specified name contained in this storage object. All elements within a storage object – including both streams and other storage objects – are kept in the same name space.

```
HRESULT CreateStream(
   const wchar_t * pwcsName,        //Points to the name of the new stream
   DWORD grfMode,                   //Access mode for the new stream
   DWORD reserved1,                 //Reserved; must be zero
   DWORD reserved2,                 //Reserved; must be zero
   IStream ** ppstm                 //Points to new stream object
  );
```

Parameters

pwcsName
[in]Points to the name of the newly created stream. This name can be used later to open or reopen the stream.

grfMode
[in]Specifies the access mode to use when opening the newly created stream. See the **STGM** enumeration values for descriptions of the possible values.

reserved1
[in]Reserved for future use; must be zero.

reserved2
[in]Reserved for future use; must be zero.

ppstm
[out]Points to the location where the new **IStream** interface pointer is returned; only valid if the operation is successful. This parameter is set to NULL if an error occurs.

Return Values

S_OK
The new stream was successfully created

STG_E_ACCESSDENIED
Insufficient permissions to create stream.

STG_E_FILEALREADYEXISTS
The name specified for the stream already exists in the storage object and the *grfmode* flag includes the flag STGM_FAILIFTHERE.

STG_E_INSUFFICIENTMEMORY
The stream was not created due to a lack of memory.

STG_E_INVALIDFLAG
The value specified for the *grfMode* flag is not a valid **STGM** enumeration value.

STG_E_INVALIDFUNCTION
The specified combination of *grfMode* flags is not supported. For example, if this method is called without the STGM_SHARE_EXCLUSIVE flag.

STG_E_INVALIDNAME
Invalid value for *pwcsName*.

STG_E_INVALIDPOINTER
The pointer specified for the stream object was invalid.

STG_E_INVALIDPARAMETER
One of the parameters was invalid.

STG_E_REVERTED

The object has been invalidated by a revert operation above it in the transaction tree.

STG_E_TOOMANYOPENFILES
The stream was not created because there are too many open files.

Comments

If a stream with the name specified in the *pwcsName* parameter already exists and the *grfMode* parameter includes the STGM_CREATE flag, the existing stream is replaced by a newly created one. The destruction of the old stream and the creation of the new stream object are both subject to the transaction mode on the parent storage object.

If the stream already exists and *grfMode* is set to STGM_FAILIFTHERE, this method fails with the return value STG_E_FILEALREADYEXISTS.

Note to Callers

The OLE-provided compound file implementation of the **IStorage::CreateStream** method does not support the following behaviors:

- The STGM_DELETEONRELEASE flag is not supported.
- Transacted mode is not supported for stream objects.
- Opening the same stream more than once from the same storage is not supported. The STGM_SHARE_EXCLUSIVE flag must be specified.

Other implementations of **IStorage** might support these features.

See Also

IStorage::OpenStream, **IStream**

IStorage::DestroyElement

Removes the specified element from this storage object.

```
HRESULT DestroyElement(
  const wchar_t * pwcsName        //Points to the name of the element to be removed
 );
```

Parameter

pwcsName
[in]Points to the name of the element to be removed from this storage object.

Return Values

S_OK
The element was successfully removed.

STG_E_ACCESSDENIED
The caller does not have sufficient permissions for removing the element.

STG_E_FILENOTFOUND
The element with the specified name does not exist.

STG_E_INSUFFICIENTMEMORY
The element was not removed due to a lack of memory.

STG_E_INVALIDNAME
Invalid value for *pwcsName*.

STG_E_INVALIDPOINTER
The pointer specified for the element was invalid.

STG_E_INVALIDPARAMETER
One of the parameters was invalid.

STG_E_REVERTED
The object has been invalidated by a revert operation above it in the transaction tree.

STG_E_TOOMANYOPENFILES
The element was not removed because there are too many open files.

Comments

An existing open instance of this element from the parent becomes invalid after the **IStorage::DestroyElement** call completes.

The removal is subject to committing the changes if the storage object is opened in transacted mode.

IStorage::EnumElements

Returns an enumerator object that can be used to enumerate the storage and stream objects contained within this storage object.

```
HRESULT EnumElements(
    DWORD reserved1,              //Reserved; must be zero
    void * reserved2,             //Reserved; must be NULL
    DWORD reserved3,              //Reserved; must be zero
    IEnumSTATSTG ** ppenum        //Location for IEnumSTATSTG
  );
```

Parameters

reserved1
[in]Reserved for future use; must be zero.

reserved2
[in]Reserved for future use; must be NULL.

reserved3
[in]Reserved for future use; must be zero.

ppenum
[out]Points to the location where the new enumerator object should be returned.

Return Values

S_OK
The enumerator object was successfully returned.

STG_E_INSUFFICIENTMEMORY
The enumerator object could not be created due to lack of memory.

STG_E_INVALIDPARAMETER
One of the parameters was not valid.

STG_E_REVERTED
The object has been invalidated by a revert operation above it in the transaction tree.

Comments

The enumerator object returned by this method implements the **IEnumSTATSTG** interface, one of the standard enumerator interfaces that contain the **Next**, **Reset**, **Clone**, and **Skip** methods. **IEnumSTATSTG** enumerates the data stored in an array of **STATSTG** structures.

The storage object must be open in read mode to allow the enumeration of its elements.

The order in which the elements are enumerated and whether the enumerator is a snapshot or always reflects the current state of the storage object, depends on the **IStorage** implementation. The OLE-provided compound file implementation takes a snapshot.

See Also

IEnumXXXX, **IEnumSTATSTG**, **STATSTG**

IStorage::MoveElementTo

Copies or moves a substorage or stream from this storage object to another storage object.

```
HRESULT MoveElementTo(
    const wchar_t pwcsName,          //Name of the element to be moved
    IStorage * pstgDest,             //Points to destination storage object
    LPWSTR pwcsNewName,              //Points to new name of element in destination
    DWORD grfFlags                   //Specifies a copy or a move
  );
```

Parameters

pwcsName
[in]Specifies the name of the element in this storage object to be moved or copied.

pstgDest
[in]Points to the destination storage object.

pwcsNewName
[in]Points to the new name for the element in its new storage object.

grfFlags
[in]Specifies whether the operation should be a move (STGMOVE_MOVE) or a copy (STGMOVE_COPY). See the **STGMOVE** enumeration.

Return Values

S_OK
The storage object was successfully copied or moved.

STG_E_ACCESSDENIED
The destination storage object is a child of the source storage object.

STG_E_FILENOTFOUND
The element with the specified name does not exist.

STG_E_FILEALREADYEXISTS
The specified file already exists.

STG_E_INSUFFICIENTMEMORY
The copy or move was not completed due to a lack of memory.

STG_E_INVALIDFLAG
The value for the *grfFlags* parameter is not valid.

STG_E_INVALIDNAME
Invalid value for *pwcsName*.

STG_E_INVALIDPOINTER
The pointer specified for the storage object was invalid.

STG_E_INVALIDPARAMETER
One of the parameters was invalid.

STG_E_REVERTED
The object has been invalidated by a revert operation above it in the transaction tree.

STG_E_TOOMANYOPENFILES
The copy or move was not completed because there are too many open files.

Comments

The **IStorage::MoveElementTo** method is typically the same as invoking the **IStorage::CopyTo** method on the indicated element and then removing the source element. In this case, the **MoveElementTo** method uses only the publicly available functions of the destination storage object to carry out the move.

If the source and destination storage objects have special knowledge about each other's implementation (for example, they could be different instances of the same implementation), this operation can be implemented efficiently.

See Also

STGMOVE, **IStorage::CopyTo**

IStorage::OpenStorage

Opens an existing storage object with the specified name according to the specified access mode.

```
HRESULT OpenStorage(
   const wchar_t * pwcsName,        //Points to the name of the storage object to open
   IStorage * pstgPriority,         //Points to previous opening of the storage object
   DWORD grfMode,                   //Access mode for the new storage object
   SNB snbExclude,                  //Points to a block of stream names in the storage object
   DWORD reserved,                  //Reserved; must be zero
   IStorage ** ppstg                //Points to opened storage object
  );
```

Parameters

pwcsName
[in]Points to the name of the storage object to open. It is ignored if *pstgPriority* is non-NULL.

pstgPriority
[in]If the *pstgPriority* parameter is not NULL, it is a pointer to a previous opening of an element of the storage object, usually one that was opened in priority mode. The storage object should be closed and re-opened according to *grfMode*. When the **IStorage::OpenStorage** method returns, *pstgPriority* is no longer valid. Use the value returned in the *ppstg* parameter. If the *pstgPriority* parameter is NULL, it is ignored.

grfMode
[in]Specifies the access mode to use when opening the storage object. See the **STGM** enumeration values for descriptions of the possible values.

snbExclude
[in]Either NULL or a non-NULL value that points to a block of stream names in this storage object that are to be emptied as the object is opened. This exclusion happens independent of whether a snapshot copy happens on the open.

reserved
[in]Reserved for future use; must be zero.

ppstg
[out]Points to the location where the **IStorage** interface pointer is returned; only valid if the operation is successful. This parameter is set to NULL if an error occurs.

Return Values

S_OK
The storage object was opened successfully.

STG_E_ACCESSDENIED
Insufficient permissions to open storage object.

STG_E_FILENOTFOUND
The storage object with the specified name does not exist.

STG_E_INSUFFICIENTMEMORY
The storage object was not opened due to a lack of memory.

STG_E_INVALIDFLAG
The value specified for the *grfMode* flag is not a valid **STGM** enumeration value.

STG_E_INVALIDFUNCTION
The specified combination of *grfMode* flags is not supported.

STG_E_INVALIDNAME
Invalid value for *pwcsName*.

STG_E_INVALIDPOINTER
The pointer specified for the storage object was invalid.

STG_E_INVALIDPARAMETER
One of the parameters was invalid.

STG_E_REVERTED
The object has been invalidated by a revert operation above it in the transaction tree.

STG_E_TOOMANYOPENFILES
The storage object was not created because there are too many open files.

STG_S_CONVERTED
The existing stream with the specified name was replaced with a new storage object containing a single stream called CONTENTS. The new storage object will be added.

Comments

Root storage objects can be opened with STGM_DELETEONRELEASE, in which case the object is destroyed when it receives its final release. This is useful for creating temporary storage objects.

Note to Callers

The OLE-provided compound file implementation of the **IStorage::OpenStorage** method does not support the following behaviors:

- The STGM_PRIORITY flag is not supported for non-root storages.
- Opening the same storage object more than once from the same parent storage is not supported. The STGM_SHARE_EXCLUSIVE flag must be specified.

Other implementations of **IStorage** might support these features.

See Also

IStorage::CreateStorage

IStorage::OpenStream

Opens an existing stream object within this storage object using the specified access permissions in *grfMode*.

```
HRESULT OpenStream(
    const wchar_t * pwcsName,        //Points to name of stream to open
    void * reserved1,                //Reserved; must be NULL
    DWORD grfMode,                   //Access mode for the new stream
    DWORD reserved2,                 //Reserved; must be zero
    IStream ** ppstm                 //Points to opened stream object
  );
```

Parameters

pwcsName
[in]Points to the name of the stream to open.

reserved1
[in]Reserved for future use; must be NULL.

grfMode
[in]Specifies the access mode to use when opening the stream. See the **STGM** enumeration values for descriptions of the possible values.

reserved2
[in]Reserved for future use; must be zero.

ppstm
[out]Points to the location where the opened **IStream** interface pointer is returned; only valid if the operation is successful. This parameter is set to NULL if an error occurs.

Return Values

S_OK
The stream was successfully opened.

STG_E_ACCESSDENIED
Insufficient permissions to open stream.

STG_E_FILENOTFOUND
The stream with specified name does not exist.

STG_E_INSUFFICIENTMEMORY
The stream was not opened due to a lack of memory.

STG_E_INVALIDFLAG
The value specified for the *grfMode* flag is not a valid **STGM** enumeration value.

STG_E_INVALIDFUNCTION
The specified combination of *grfMode* flags is not supported. For example, if this method is called without the STGM_SHARE_EXCLUSIVE flag.

STG_E_INVALIDNAME
Invalid value for *pwcsName*.

STG_E_INVALIDPOINTER
The pointer specified for the stream object was invalid.

STG_E_INVALIDPARAMETER
One of the parameters was invalid.

STG_E_REVERTED
The object has been invalidated by a revert operation above it in the transaction tree.

STG_E_TOOMANYOPENFILES
The stream was not opened because there are too many open files.

Comments

There are restrictions on the permissions that can be given in *grfMode*. For example, the permissions on this storage object restrict the permissions on its streams.

Note to Callers

The OLE-provided compound file implementation of the **IStorage::OpenStream** method does not support the following behaviors:

- The STGM_DELETEONRELEASE flag is not supported.
- Transacted mode is not supported for stream objects.
- Opening the same stream more than once from the same storage is not supported. The STGM_SHARE_EXCLUSIVE flag must be specified.

Other implementations of **IStorage** might support these features.

See Also

IStorage::CreateStream, **IStream**

IStorage::RenameElement

Renames the specified element in this storage object.

```
HRESULT RenameElement(
  const wchar_t * pwcsOldName,        //Points to the name of the element to be changed
  const wchar_t * pwcsNewName         //Points to the new name for the specified element
 );
```

Parameters

pwcsOldName
[in]Points to the name of the element to be changed.

pwcsNewName
[in]Points to the new name for the specified element.

Return Values

S_OK
The element was successfully renamed.

STG_E_ACCESSDENIED
The caller does not have sufficient permissions for renaming the element.

STG_E_FILENOTFOUND
The element with the specified old name does not exist.

STG_E_FILEALREADYEXISTS
The element specified by the new name already exists.

STG_E_INSUFFICIENTMEMORY
The element was not renamed due to a lack of memory.

STG_E_INVALIDNAME
Invalid value for one of the names.

STG_E_INVALIDPOINTER
The pointer specified for the element was invalid.

STG_E_INVALIDPARAMETER
One of the parameters was invalid.

STG_E_REVERTED
The object has been invalidated by a revert operation above it in the transaction tree.

STG_E_TOOMANYOPENFILES
The element was not renamed because there are too many open files.

Comments

An element in a storage object cannot be renamed while it is open. The rename operation is subject to committing the changes if the storage is open in transacted mode.

The **IStorage::RenameElement** method is not guaranteed to work in low memory with storage objects open in transacted mode. It may work in direct mode.

IStorage::Revert

Discards all changes that have been made to the storage object since the last commit.

HRESULT Revert(*void***);**

Return Values

S_OK
 The revert operation was successful.

STG_E_INSUFFICIENTMEMORY
 The revert operation could not be completed due to a lack of memory.

STG_E_TOOMANYOPENFILES
 The revert operation could not be completed because there are too many open files.

STG_E_REVERTED
 The object has been invalidated by a revert operation above it in the transaction tree.

Comments

For storage objects opened in transacted mode, the **IStorage::Revert** method discards any uncommitted changes to this storage object or changes that have been committed to this storage object from nested elements.

After this method finishes, any existing elements that were opened from the reverted storage object are invalid and can no longer be used. The error STG_E_REVERTED is returned on all calls except **IStorage::Release** using these openings.

For storage objects opened in direct mode, this method has no effect.

See Also

IStorage::Commit

IStorage::SetClass

Assigns the specified CLSID to this storage object.

```
HRESULT SetClass(
   REFCLSID clsid         //Class identifier to be assigned to the storage object
  );
```

Parameter

clsid
[in]The class identifier that is to be associated with the storage object.

Return Values

S_OK
The CLSID was successfully assigned.

STG_E_ACCESSDENIED
The caller does not have sufficient permissions for assigning a class identifier to the storage object.

STG_E_MEDIUMFULL
Not enough space was left on device to complete the operation.

STG_E_REVERTED
The object has been invalidated by a revert operation above it in the transaction tree.

Comments

When first created, a storage object has an associated CLSID of CLSID_NULL. You call this method to assign a class identifier to the storage object.

You can retrieve the current CLSID of a storage object with the **IStorage::Stat** method.

See Also

IStorage::Stat

IStorage::SetElementTimes

Sets the modification, access, and creation times of the specified storage element, if supported by the underlying file system.

```
HRESULT SetElementTimes(
  const wchar_t * pwcsName,      //Points to name of element to be changed
  FILETIME const * pctime,       //New creation time for element, or NULL
  FILETIME const * patime,       //New access time for element, or NULL
  FILETIME const * pmtime        //New modification time for element, or NULL
 );
```

Parameters

pwcsName
[in]The name of the storage object element whose times are to be modified. If NULL, the time is set on the root storage rather than one of its elements.

pctime
[in]Either the new creation time for the element or NULL if the creation time is not to be modified.

patime
[in]Either the new access time for the element or NULL if the access time is not to be modified.

pmtime
[in]Either the new modification time for the element or NULL if the modification time is not to be modified.

Return Values

S_OK
The time values were successfully set.

STG_E_ACCESSDENIED
The caller does not have sufficient permissions for changing the element.

STG_E_FILENOTFOUND
The element with the specified name does not exist.

STG_E_INSUFFICIENTMEMORY
The element was not changed due to a lack of memory.

STG_E_INVALIDNAME
Invalid value for the element name.

STG_E_INVALIDPOINTER
The pointer specified for the element was invalid.

STG_E_INVALIDPARAMETER
One of the parameters was invalid.

STG_E_REVERTED
The object has been invalidated by a revert operation above it in the transaction tree.

STG_E_TOOMANYOPENFILES
The element was not changed because there are too many open files.

Comments

This method sets time statistics for the specified storage element within this storage object.

Not all file systems support all of the time values. This method sets those times that are supported and ignores the rest. Each of the time value parameters can be NULL; indicating that no modification should occur.

You can retrieve these time values with the **IStorage::Stat** method.

Note to Callers

The OLE-provided compound file implementation maintains modification and change times for internal storage objects. For root storage objects, whatever is supported by the underlying file system (or **ILockBytes**) is supported. The compound file implementation does not maintain any time stamps for internal streams. Unsupported time stamps are reported as zero, enabling the caller to test for support.

See Also

IStorage::Stat

IStorage::SetStateBits

Stores up to 32 bits of state information in this storage object.

```
HRESULT SetStateBits(
   DWORD grfStateBits,          //Specifies new values of bits
   DWORD grfMask                //Specifies mask that indicates which bits are significant
  );
```

Parameters

grfStateBits

[in]Specifies the new values of the bits to set. No legal values are defined for these bits; they are all reserved for future use and must not be used by applications. See the **STGSTATE** enumeration for a definition of these values.

grfMask

[in]A binary mask indicating which bits in *grfStateBits* are significant in this call. See the **STGSTATE** enumeration for a definition of these values.

Return Values

S_OK

The state information was successfully set.

STG_E_ACCESSDENIED

The caller does not have sufficient permissions for changing this storage object.

STG_E_INVALIDFLAG

The value for the *grfStateBits* or *grfMask* parameters are not valid.

STG_E_INVALIDPARAMETER

One of the parameters was invalid.

Comments

You can retrieve the state information with the **IStorage::Stat** method. When a storage object is first created, the value of this state information is zero. The values for the state bits are not defined currently and should not be used by applications.

The state set by this method is for external use only. The OLE-provided compound file implementation does not perform any action based on the state.

See Also

STATSTATE, **IStorage::Stat**

IStorage::Stat

Returns the **STATSTG** structure for this open storage object.

```
HRESULT Stat(
  STATSTG * pstatstg,          //Location for STATSTG structure
  DWORD grfStatFlag            //Values taken from the STATFLAG enumeration
 );
```

Parameters

pstatstg
[out]Points to a **STATSTG** structure where this method places information about the open storage object. This parameter is NULL if an error occurs.

grfStatFlag
[in]Specifies that some of the fields in the **STATSTG** structure are not returned, thus saving a memory allocation operation. Values are taken from the **STATFLAG** enumeration.

Return Values

S_OK
The **STATSTG** structure was successfully returned at the specified location.

STG_E_ACCESSDENIED
The caller does not have sufficient permissions for accessing statistics for this storage object.

STG_E_INSUFFICIENTMEMORY
The **STATSTG** structure was not returned due to a lack of memory.

STG_E_INVALIDFLAG
The value for the *grfStateFlag* parameter is not valid.

STG_E_INVALIDPARAMETER
One of the parameters was invalid.

See Also

STATFLAG, **STATSTG**, **IStorage::SetClass**, **IStorage::SetElementTimes**, **IStorage::SetStateBits**

IStream

The **IStream** interface supports reading and writing data to stream objects. Stream objects are elements nested within a storage object. They are similar to standard files. You use stream objects to store the data for your application.

The **IStream** interface provides methods similar to the MS-DOS FAT file functions. For example, each stream object has its own access rights and a seek pointer. The main difference between a stream object and a DOS file is that streams are not open using a file handle. Instead, you use the **IStream** interface pointer.

The methods in this interface present your object data as a contiguous sequence of bytes which you can read or write. There are also methods for committing and reverting changes on streams open in transacted mode and methods for restricting access to a range of bytes in the stream.

Streams can remain open for long periods of time without consuming file system resources. The **IStream::Release** method is similar to a close function on a file. Once released, the stream object is no longer valid and cannot be used.

When to Implement

OLE provides an implementation of the **IStream** interface that you can use for saving object data. The specification of **IStream** defines more functionality that the OLE implementation supports. For example, the **IStream** interface defines streams up to 2 (64) bytes in length requiring a 64-bit seek pointer. However, the OLE implementation only supports streams up to 2 (32) bytes in length and read and write operations are always limited to 2 (32) bytes at a time. The OLE implementation also does not support stream transactioning or region locking.

Container or object applications can implement their own stream objects instead of using the OLE-provided implementation.

When to Use

If you are writing an object application, you can use the **IStream** interface to store the data for your object.

Since stream objects can be marshaled to other processes, applications can share the data in storage objects without having to use global memory. For the OLE-provided implementation of stream objects, the custom marshaling facilities in OLE create a remote version of the original object in the new process when the two processes have shared memory access. Thus, the remote version does not need to communicate with the original process to carry out its functions.

The remote version of the stream object shares the same seek pointer as the original stream. If you do not want to share the seek pointer, you should use the **IStream::Clone** method to provide a copy of the stream object for the remote process.

Note If you are creating a stream object that is larger than the heap in your machine's memory and you are using an HGLOBAL, the stream object calls **GlobalRealloc** internally whenever it needs more memory. Because **GlobalRealloc** always copies data from the source to the destination, increasing a stream object from 20M to 25M, for example, consumes immense amounts of time. This is due to the size of the increments copied and is worsened if there is less than 45M of memory on the machine because of disk swapping.

The preferred solution is to implement an **IStream** that uses memory allocated by **VirtualAlloc** instead of **GlobalAlloc**. This can reserve a large chunk of virtual address space and then commit memory within that address space as required. No data copying occurs and memory is committed only as it is needed.

Another alternative is to call the **IStream::SetSize** method on the stream object to increase the memory allocation in advance. This is not, however, as efficient as using **VirtualAlloc** as described above.

Methods in Vtable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IStream Methods	Description
Read	Reads a specified number of bytes from the stream object into memory starting at the current seek pointer.
Write	Writes a specified number from bytes into the stream object starting at the current seek pointer.
Seek	Changes the seek pointer to a new location relative to the beginning of the stream, the end of the stream, or the current seek pointer.
SetSize	Changes the size of the stream object.
CopyTo	Copies a specified number of bytes from the current seek pointer in the stream to the current seek pointer in another stream.
Commit	Ensures that any changes made to a stream object open in transacted mode are reflected in the parent storage object.
Revert	Discards all changes that have been made to a transacted stream since the last **IStream::Commit** call.
LockRegion	Restricts access to a specified range of bytes in the stream. Supporting this functionality is optional since some file systems do not provide it.
UnlockRegion	Removes the access restriction on a range of bytes previously restricted with **IStream::LockRegion**.
Stat	Retrieves the **STATSTG** structure for this stream.
Clone	Creates a new stream object that references the same bytes as the original stream but provides a separate seek pointer to those bytes.

IStream::Clone

Creates a new stream object with its own seek pointer that references the same bytes as the original stream.

```
HRESULT Clone(
   IStream ** ppstm          //Points to location for pointer to the new stream object
  );
```

Parameter

ppstm
[out]Points to location for the pointer to the new stream object. If an error occurs, this parameter is NULL.

Return Values

S_OK
The stream was successfully cloned.

STG_E_INSUFFICIENT_MEMORY
The stream was not cloned due to a lack of memory.

STG_E_INVALIDPOINTER
The *ppStm* pointer is not valid.

Comments

This method creates a new stream object for accessing the same bytes but using a separate seek pointer. The new stream object sees the same data as the source stream object. Changes written to one object are immediately visible in the other. Range locking is shared between the stream objects.

The initial setting of the seek pointer in the cloned stream instance is the same as the current setting of the seek pointer in the original stream at the time of the clone operation.

See Also

IStream::CopyTo

IStream::Commit

Ensures that any changes made to a stream object open in transacted mode are reflected in the parent storage. If the stream object is open in direct mode, **IStream::Commit** has no effect other than flushing all memory buffers to the next level storage object. The OLE-provided implementation of streams does not support opening streams in transacted mode.

```
HRESULT Commit(
  DWORD grfCommitFlags        //Specifies how changes are committed
 );
```

Parameter

grfCommitFlags
[in]Controls how the changes for the stream object are committed. See the **STGC** enumeration for a definition of these values.

Return Values

S_OK
Changes to the stream object were successfully committed to the parent level.

STG_E_MEDIUMFULL
The commit operation failed due to lack of space on the storage device.

Comments

This method ensures that changes to a stream object opened in transacted mode are reflected in the parent storage. Changes that have been made to the stream since it was opened or last committed are reflected to the parent storage object. If the parent is opened in transacted mode, the parent may still revert at a later time rolling back the changes to this stream object.

If the stream is open in direct mode, this method ensures that any memory buffers have been flushed out to the underlying storage object. This is much like a flush in traditional file systems.

The **IStream::Commit** method is useful on a direct mode stream when the implementation of the **IStream** interface is a wrapper for underlying file system APIs. In this case, **IStream::Commit** would be connected to the file system's flush call.

Notes to Callers

The OLE-provided implementation of stream objects does not support transacted streams. Also, stream objects only exist within storage objects in this implementation. Thus, the OLE implementation of **IStream::Commit** has no effect other than flushing internal memory buffers to the parent storage object. With this implementation, it does not matter if you commit changes to streams. You can just commit changes for storage objects.

See Also

IStorage::Commit

IStream::CopyTo

Copies a specified number of bytes from the current seek pointer in the stream to the current seek pointer in another stream.

```
HRESULT CopyTo(
  IStream * pstm,                     //Points to the destination stream
  ULARGE_INTEGER cb,                  //Specifies the number of bytes to copy
  ULARGE_INTEGER * pcbRead,           //Pointer to the actual number of bytes read from the source
  ULARGE_INTEGER * pcbWritten         //Pointer to the actual number of bytes written to the destination
 );
```

Parameters

pstm
[in]Points to the destination stream. The stream pointed to by *pstm* can be a new stream or a clone of the source stream.

cb
[in]Specifies the number of bytes to copy from the source stream.

pcbRead
[out]Pointer to the location where this method writes the actual number of bytes read from the source. You can set this pointer to NULL to indicate that you are not interested in this value. In this case, this method does not provide the actual number of bytes read.

pcbWritten
[out]Pointer to the location where this method writes the actual number of bytes written to the destination. You can set this pointer to NULL to indicate that you are not interested in this value. In this case, this method does not provide the actual number of bytes written.

Return Values

S_OK
The stream object was successfully copied.

STG_E_INVALIDPOINTER
The value of one of the pointer parameters is not valid.

STG_E_MEDIUMFULL
The stream is not copied because there is no space left on the storage device.

Comments

This method copies the specified bytes from one stream to another. The seek pointer in each stream instance is adjusted for the number of bytes read or written. This method is equivalent to reading *cb* bytes into memory using **IStream::Read** and then immediately writing them to the destination stream using **IStream::Write**, although **IStream::CopyTo** will be more efficient.

The destination stream can be a clone of the source stream created by calling the **IStream::Clone** method.

If **IStream::CopyTo** returns an error, you cannot assume that the seek pointers are valid for either the source or destination. Additionally, the values of *pcbRead* and *pcbWritten* are not meaningful even though they are returned.

If **IStream::CopyTo** returns successfully, the actual number of bytes read and written are the same.

To copy the remainder of the source from the current seek pointer, specify the maximum large integer value for the *cb* parameter. If the seek pointer is the beginning of the stream, this techniques copies the entire stream.

See Also

IStream::Read, **IStream::Write**, **IStream::Clone**

IStream::LockRegion

Restricts access to a specified range of bytes in the stream. Supporting this functionality is optional since some file systems do not provide it.

```
HRESULT LockRegion(
   ULARGE_INTEGER libOffset,        //Specifies the byte offset for the beginning of the range
   ULARGE_INTEGER cb,               //Specifies the length of the range in bytes
   DWORD dwLockType                 //Specifies the restriction on accessing the specified range
  );
```

Parameters

libOffset
[in]Specifies the byte offset for the beginning of the range.

cb
[in]Specifies, in bytes, the length of the range to be restricted.

dwLockType
[in]Specifies the restrictions being requested on accessing the range.

Return Values

S_OK
The specified range of bytes was locked.

STG_E_INVALIDFUNCTION
Locking is not supported at all or the specific type of lock requested is not supported.

STG_E_LOCKVIOLATION
Requested lock is supported, but cannot be granted because of an existing lock.

Comments

The byte range can extend past the current end of the stream. Locking beyond the end of a stream is useful as a method of communication between different instances of the stream without changing data that is actually part of the stream.

Three types of locking can be supported: locking to exclude other writers, locking to exclude other readers or writers, and locking that allows only one requestor to obtain a lock on the given range, which is usually an alias for one of the other two lock types. A given stream instance might support either of the first two types, or both. The lock type is specified by *dwLockType*, using a value from the **LOCKTYPE** enumeration.

Any region locked with **IStream::LockRegion** must later be explicitly unlocked by calling **IStream::UnlockRegion** with exactly the same values for the *libOffset*, *cb*, and *dwLockTyp*e parameters. The region must be unlocked before the stream is released. Two adjacent regions cannot be locked separately and then unlocked with a single unlock call.

Notes to Callers

Since range locking is not supported by the OLE-provided implementation of streams, calling **IStream::LockRegion** has no effect for this implementation.

Since the type of locking supported is optional and can vary in different implementations of **IStream**, you must provide code to deal with the STG_E_INVALIDFUNCTION error.

Notes to Implementors

Support for this method is optional for implementations of stream objects since it may not be supported by the underlying file system. The type of locking supported is also optional. The STG_E_INVALIDFUNCTION error is returned if the requested type of locking is not supported.

See Also

LOCKTYPE, **IStream::UnlockRegion**

IStream::Read

Reads a specified number of bytes from the stream object into memory starting at the current seek pointer.

```
HRESULT Read(
    void * pv,              //Pointer to buffer into which the stream is read
    ULONG cb,               //Specifies the number of bytes to read
    ULONG * pcbRead         //Pointer to location that contains actual number of bytes read
  );
```

Parameters

pv
[in]Points to the buffer into which the stream is read. If an error occurs, this value is NULL.

cb
[in]Specifies the number of bytes of data to attempt to read from the stream object.

pcbRead
[out]Pointer to a location where this method writes the actual number of bytes read from the stream object. You can set this pointer to NULL to indicate that you are not interested in this value. In this case, this method does not provide the actual number of bytes read.

Return Values

S_OK
Data was successfully read from the stream object.

S_FALSE
The data could not be read from the stream object.

STG_E_ACCESSDENIED
The caller does not have sufficient permissions for reading this stream object.

STG_E_INVALIDPOINTER
One of the pointer values is invalid.

STG_E_REVERTED
The object has been invalidated by a revert operation above it in the transaction tree.

Comments

This method reads bytes from this stream object into memory. The stream object must be opened in STGM_READ mode. This method adjusts the seek pointer by the actual number of bytes read.

The number of bytes actually read is returned in the *pcbRead* parameter.

Notes to Callers

The actual number of bytes read can be fewer than the number of bytes requested if an error occurs or if the end of the stream is reached during the read operation.

The OLE-provided implementation of **IStream** returns S_OK if the end of the stream was reached during the read. (This is the same as the "end of file" behavior found in the MS-DOS FAT file system.

Some implementations might return an error if the end of the stream is reached during the read. You must be prepared to deal with the error return or S_OK return values on end of stream reads.

See Also

STGMOVE, **IStorage::OpenStream**, **IStream::Write**

IStream::Revert

Discards all changes that have been made to a transacted stream since the last **IStream::Commit** call. For streams open in direct mode and streams using the OLE implementation of **IStream::Revert**, this method has no effect.

HRESULT Revert(*void***);**

Return Values

S_OK
The stream was successfully reverted to its previous version.

STG_E_REVERTED
The object has been invalidated by a revert operation above it in the transaction tree.

Comments

This method discards changes made to a transacted stream since the last commit operation.

The OLE-provided implementation does not support transacted streams, and this method has no effect in that implementation.

See Also

IStream::Commit

IStream::Seek

Changes the seek pointer to a new location relative to the beginning of the stream, to the end of the stream, or to the current seek pointer.

```
HRESULT Seek(
   LARGE_INTEGER dlibMove,                 //Offset relative to dwOrigin
   DWORD dwOrigin,                         //Specifies the origin for the offset
   ULARGE_INTEGER * plibNewPosition        //Pointer to location containing new seek pointer
 );
```

Parameters

dlibMove

[in]Displacement to be added to the location indicated by *dwOrigin*. If *dwOrigin* is STREAM_SEEK_SET, then this is interpreted as an unsigned value rather than signed.

dwOrigin

[in]Specifies the origin for the displacement specified in *dlibMove.* The origin can the be beginning of the file, the current seek pointer, or the end of the file. See the **STREAM_SEEK** enumeration for the values.

plibNewPosition

[out]Pointer to the location where this method writes the value of the new seek pointer from the beginning of the stream. You can set this pointer to NULL to indicate that you are not interested in this value. In this case, this method does not provide the new seek pointer.

Return Values

S_OK

The seek pointer has been successfully adjusted.

STG_E_INVALIDPOINTER

The value of the *plibNewPosition* parameter is not valid.

STG_E_INVALIDFUNCTION

The value of the *dwOrigin* parameter is not valid.

Comments

This method changes the seek pointer so subsequent reads and writes can take place at a different location in the stream object. It is an error to seek before the beginning of the stream. However, it is not an error to seek past the end of the stream. Seeking past the end of the stream is useful for subsequent writes, as the stream will at that time be extended to the seek position immediately before the write is done.

You can also use this method to obtain the current value of the seek pointer by calling this method with the *dwOrigin* parameter set to STREAM_SEEK_CUR and the *dlibMove* parameter set to 0 so the seek pointer is not changed. The current seek pointer is returned in the *plibNewPosition* parameter.

See Also

STREAM_SEEK, **IStream::Read**, **IStream::Write**

IStream::SetSize

Changes the size of the stream object.

```
HRESULT SetSize(
   ULARGE_INTEGER libNewSize       //Specifies the new size of the stream object
  );
```

Parameter

libNewSize
[in]Specifies the new size of the stream as a number of bytes.

Return Values

S_OK
The size of the stream object was successfully changed.

STG_E_MEDIUMFULL
The stream size is not changed because there is no space left on the storage device.

STG_E_INVALIDFUNCTION
The value of the *libNewSize* parameter is not valid. Since streams cannot be greater than 2 (32) bytes in the OLE-provided implementation, the high DWORD of *libNewSize* must be 0. If it is nonzero, this parameter is not valid.

Comments

This method changes the size of the stream object. You can use this method to preallocate space for the stream. If the *libNewSize* parameteris larger than the current stream size, the stream is extended to the indicated size by filling the intervening space with bytes of undefined value. This operation is similar to the **IStream::Write** method if the seek pointer is past the current end-of-stream.

If the *libNewSize* parameter is smaller than the current stream, then the stream is truncated to the indicated size.

The seek pointer is not affected by the change in stream size.

Calling **IStream::SetSize** is an effective way of trying to obtain a large chunk of contiguous space. However, in the OLE-provided implementation of streams, there is no guarantee that the space will be contiguous.

See Also

IStream::Write

IStream::Stat

Retrieves the **STATSTG** structure for this stream.

```
HRESULT Stat(
  STATSTG * pstatstg,         //Location for STATSTG structure
  DWORD grfStatFlag           //Values taken from the STATFLAG enumeration
 );
```

Parameters

pstatstg
: [out]Points to a **STATSTG** structure where this method places information about this stream object. This pointer is NULL if an error occurs.

grfStatFlag
: [in]Specifies that this method does not return some of the fields in the **STATSTG** structure, thus saving a memory allocation operation. Values are taken from the **STATFLAG** enumeration.

Return Value

S_OK
: The **STATSTG** structure was successfully returned at the specified location.

STG_E_ACCESSDENIED
: The caller does not have sufficient permissions for accessing statistics for this storage object.

STG_E_INSUFFICIENTMEMORY
: The **STATSTG** structure was not returned due to a lack of memory.

STG_E_INVALIDFLAG
: The value for the *grfStateFlag* parameter is not valid.

STG_E_INVALIDPOINTER
: The *pStatStg* pointer is not valid.

See Also

STATFLAG, **STATSTG**

IStream::UnlockRegion

Removes the access restriction on a range of bytes previously restricted with **IStream::LockRegion**.

```
HRESULT UnlockRegion(
   ULARGE_INTEGER libOffset,     //Specifies the byte offset for the beginning of the range
   ULARGE_INTEGER cb,            //Specifies the length of the range in bytes
   DWORD dwLockType              //Specifies the access restriction previously placed on the range
  );
```

Parameters

libOffset
[in]Specifies the byte offset for the beginning of the range.

cb
[in]Specifies, in bytes, the length of the range to be restricted.

dwLockType
[in]Specifies the access restrictions previously placed on the range.

Return Values

S_OK
The byte range was unlocked.

STG_E_INVALIDFUNCTION
Locking is not supported at all or the specific type of lock requested is not supported.

STG_E_LOCKVIOLATION
The requested unlock cannot be granted.

Comments

This method unlocks a region previously locked with the **IStream::LockRegion** method. Locked regions must later be explicitly unlocked by calling **IStream::UnlockRegion** with exactly the same values for the *libOffset*, *cb*, and *dwLockTyp*e parameters. The region must be unlocked before the stream is released. Two adjacent regions cannot be locked separately and then unlocked with a single unlock call.

See Also

LOCKTYPE, **IStream::LockRegion**

IStream::Write

Writes a specified number from bytes into the stream object starting at the current seek pointer.

```
HRESULT Write(
    void const* pv,             //Pointer to buffer from which stream is written
    ULONG cb,                   //Specifies the number of bytes to write
    ULONG * pcbWritten          //Specifies the actual number of bytes written
  );
```

Parameters

pv
[in]Points to the buffer from which the stream should be written.

cb
[in]Specifies the number of bytes of data to attempt to write into the stream.

pcbWritten
[out]Pointer to a location where this method writes the actual number of bytes written to the stream object. You can set this pointer to NULL to indicate that you are not interested in this value. In this case, this method does not provide the actual number of bytes written.

Return Values

S_OK
The data was successfully written into the stream object.

STG_E_MEDIUMFULL
The write operation was not completed because there is no space left on the storage device.

STG_E_ACCESSDENIED
The caller does not have sufficient permissions for writing this stream object.

STG_E_CANTSAVE
Data cannot be written for reasons other than no access or space.

STG_E_INVALIDPOINTER
One of the pointer values is invalid.

STG_E_REVERTED
The object has been invalidated by a revert operation above it in the transaction tree.

STG_E_WRITEFAULT
The write operation was not completed due to a disk error.

Comments

This method writes the specified data to a stream object. The seek pointer is adjusted for the number of bytes actually written. The number of bytes actually written is returned in the *pcbWrite* parameter. If the byte count is zero bytes, the write operation has no effect.

If the seek pointer is currently past the end of the stream and the byte count is non-zero, this method increases the size of the stream to the seek pointer and writes the specified bytes starting at the seek pointer. The fill bytes written to the stream are not initialized to any particular value. This is the same as the end-of-file behavior in the MS-DOS FAT file system.

With a zero byte count and a seek pointer past the end of the stream, this method does not create the fill bytes to increase the stream to the seek pointer. In this case, you must call the **IStream::SetSize** method to increase the size of the stream and write the fill bytes.

Notes to Callers

In the OLE-provided implementation, stream objects are not sparse. Any fill bytes are eventually allocated on the disk and assigned to the stream.

The *pcbWrite* parameter can have a value even if an error occurs.

IUnknown

The **IUnknown** interface lets clients get pointers to other interfaces on a given object through the **QueryInterface** method, and manage the existence of the object through the **IUnknown::AddRef** and **IUnknown::Release** methods. All other COM interfaces are inherited, directly or indirectly, from **IUnknown**. Therefore, the three methods in **IUnknown** are the first entries in the VTable for every interface.

When to Implement

You must implement **IUnknown** as part of every interface. If you are using C++ multiple inheritance to implement multiple interfaces, the various interfaces can share one implementation of **IUnknown**. If you are using nested classes to implement multiple interfaces, you must implement **IUnknown** once for each interface you implement.

When to Use

Use **IUnknown** methods to switch between interfaces on an object, add references, and release objects.

Methods in Vtable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IUnknown::AddRef

The **IUnknown::AddRef** method increments the reference count for a the calling interface on an object. It should be called for every new copy of a pointer to an interface on a given object.

ULONG AddRef(*void***);**

Return Value

Returns an integer from 1 to n, the value of the new reference count. This information is meant to be used for diagnostic/testing purposes only, because, in certain situations, the value may be unstable.

Comments

Objects use a reference counting mechanism to ensure that the lifetime of the object includes the lifetime of references to it. You use **IUnknown::AddRef** to stabilize a copy of an interface pointer. It can also be called when the life of a cloned pointer must extend beyond the lifetime of the original pointer. The cloned pointer must be released by calling **IUnknown::Release.**

Objects must be able to maintain (2 (31))-1 outstanding pointer references. Therefore, the internal reference counter that **IUnknown::AddRef** maintains must be a 32-bit unsigned integer.

Notes to Callers

Call this function for every new copy of an interface pointer that you make. For example, if you are passing a copy of a pointer back from a function, you must call **IUnknown::AddRef** on that pointer. You must also call **IUnknown::AddRef** on a pointer before passing it as an in-out parameter to a function; the function will call **IUnknown::Release** before copying the out-value on top of it.

See Also

IUnknown::Release

IUnknown::QueryInterface

Returns a pointer to a specified interface on a component to which a client currently holds an interface pointer. This function must call **IUnknown::AddRef** on the pointer it returns.

```
HRESULT QueryInterface(
   REFIID iid,             //Specifies the requested interface's IID
   void ** ppvObject       //Receives an indirect pointer to the object
  );
```

Parameters

iid
[in] Specifies the IID of the interface being requested.

ppvObject
[out] Receives a pointer to an interface pointer to the object on return. If the interface specified in *riid* is not supported by the object, *ppvObj* is set to **NULL**.

Return Value

S_OK if the interface is supported, S_FALSE if not.

Comments

The QueryInterface method gives a client access to other interfaces on an object.

For any one object, a specific query for the **IUnknown** interface on any of the object's interfaces must always return the same pointer value. This allows a client to determine whether two pointers point to the same component by calling **QueryInterface** on both and comparing the results. It is specifically not the case that queries for interfaces (even the same interface through the same pointer) must return the same pointer value.

There are four requirements for implementations of **QueryInterface** (In these cases, "must succeed" means "must succeed barring catastrophic failure."):

- The set of interfaces accessible on an object through **IUnknown::QueryInterface** must be static, not dynamic. This means that if a call to **QueryInterface** for a pointer to a specified interface succeeds the first time, it must succeed again, and if it fails the first time, it must fail on all subsequent queries.
- It must be symmetric – if a client holds a pointer to an interface on an object, and queries for that interface, the call must succeed.
- It must be reflexive – if a client holding a pointer to one interface queries successfully for another, a query through the obtained pointer for the first interface must succeed.
- It must be transitive – if a client holding a pointer to one interface queries successfully for a second, and through that pointer queries successfully for a third interface, a query for the first interface through the pointer for the third interface must succeed.

IUnknown::Release

Decrements the reference count for the calling interface on a object. If the reference count on the object falls to 0, the object is freed from memory.

ULONG Release(*void***);**

Return Value

Returns the resulting value of the reference count, which is used for diagnostic/testing purposes only. If you need to know that resources have been freed, use an interface with higher-level semantics.

Comments

If **IUnknown::AddRef** has been called on this object's interface *n* times and this is the *n+1th* call to **IUnknown::Release**, the implementation of **IUnknown::AddRef** must cause the interface pointer to free itself. When the released pointer is the only existing reference to an object (whether the object supports single or multiple interfaces), the implementation must free the object.

Note Aggregation of objects restricts the ability to recover interface pointers.

Notes to Callers

Call this function when you no longer need to use an interface pointer. If you are writing a function that takes an in-out parameter, call **IUnknown::Release** on the pointer you are passing in before copying the out-value on top of it.

See Also

IUnknown::AddRef

IViewObject

The **IViewObject** interface enables an object to display itself directly without passing a data object to the caller. In addition, this interface can create and manage a connection with an advise sink so the caller can be notified of changes in the view object.

The caller can request specific representations and specific target devices. For example, a caller can ask for either an object's content or an iconic representation. Also, the caller can ask the object to compose a picture for a target device that is independent of the drawing device context. As a result, the picture can be composed for one target device and drawn on another device context. For example, to provide a print preview operation, you can compose the drawing for a printer target device but actually draw the representation on the display.

The **IViewObject** interface is similar to **IDataObject**; except that **IViewObject** places a representation of the data onto a device context while **IDataObject** places the representation onto a transfer medium.

Unlike most other interfaces, **IViewObject** cannot be marshaled to another process. This is because device contexts are only effective in the context of one process.

When to Implement

Object handlers and in-process servers that manage their own presentations implement **IViewObject** for use by compound document containers. OLE provides an **IViewObject** implementation for its default object handler's cache.

When to Use

You call **IViewObject** from a container application if you need to draw a contained object on a specific device context. For example, if you want to print the object to a printer, you call the **Draw** method in the **IViewObject** interface.

Methods in VTable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IViewObject Methods	**Description**
Draw	Draws a representation of the object onto a device context.
GetColorSet	Returns the logical palette the object uses for drawing.
Freeze	Freezes the drawn representation of an object so it will not change until a subsequent **Unfreeze**.
Unfreeze	Unfreezes the drawn representation of an object.
SetAdvise	Sets up a connection between the view object and an advise sink so that the advise sink can receive notifications of changes in the view object.
GetAdvise	Returns the information on the most recent **SetAdvise**.

IViewObject::Draw

Draws a representation of an object onto the specified device context.

```
HRESULT Draw(
   DWORD dwAspect,                      //Specifies how the object is to be represented
   LONG lindex,                         //Specifies the part of the object to draw
   void * pvAspect,                     //Always NULL
   DVTARGETDEVICE * ptd,                //Specifies the target device in a structure
   HDC hicTargetDev,                    //Specifies the information context for the target device ptd
   HDC hdcDraw,                         //Identifies the device context on which to draw
   const LPRECTL lprcBounds,            //Specifies the rectangle in which the object is drawn
   const LPRECTL lprcWBounds,           //Specifies the window extent and window origin when drawing a metafile
   BOOL (*) (DWORD) pfnContinue,        //Callback function for canceling or continuing the drawing
   DWORD dwContinue                     //Parameter to pass to the callback function pfnContinue
  );
```

Parameters

dwAspect

[in]Specifies how the object is to be represented. Representations include content, an icon, a thumbnail, or a printed document. Valid values are taken from the enumeration **DVASPECT**. See the **DVASPECT** enumeration for more information.

lindex

[in]Indicates the portion of the object that is of interest for the draw operation. Its interpretation varies depending on the value in the *dwAspect* parameter. See the **DVASPECT** enumeration for more information.

pvAspect

[in]Provides more information about the view of the object specified in *dwAspect*. Since none of the current aspects support additional information; *pvAspect* must always be NULL.

ptd

[in]Points to the target device structure that describes the device for which the object is to be rendered. If NULL, the view should be rendered for the default target device (typically the display). A value other than NULL is interpreted in conjunction with *hicTargetDev* and *hdcDraw*. For example, if *hdcDraw* specifies a printer as the device context, the *ptd* parameter points to a structure describing that printer device. The data may actually be printed if *hicTargetDev* is a valid value or it may be displayed in print preview mode if *hicTargetDev* is NULL.

hicTargetDev

[in]Specifies the information context for the target device indicated by the *ptd* parameter from which the object can extract device metrics and test the device's capabilities. If *ptd* is NULL; the object should ignore the value in the *hicTargetDev* parameter.

hdcDraw

[in]Specifies the device context on which to draw.

lprcBounds

[in]Points to a **RECTL** structure specifying the rectangle on *hdcDraw* and in which the object should be drawn. This parameter controls the positioning and stretching of the object.

lprcWBounds

[in]If *hdcDraw* is a metafile device context, the *lprcWBounds* parameter points to a **RECTL** structure specifying the bounding rectangle in the underlying metafile. The rectangle structure contains the window extent and window origin. These values are useful for drawing metafiles. The rectangle indicated by *lprcBounds* is nested inside this *lprcWBounds* rectangle; they are in the same coordinate space.

If *hdcDraw* is not a metafile device context; *lprcWBounds* will be NULL.

pfnContinue
[in]Points to a callback function that the view object should call periodically during a lengthy drawing operation to determine whether the operation should continue or be canceled. This function returns TRUE to continue drawing. It returns FALSE to stop the drawing in which case **IViewObject::Draw** returns DRAW_E_ABORT.

dwContinue
[in]Contains a value to pass as a parameter to the function pointed to by the *pfnContinue* parameter. Typically, *dwContinue* is a pointer to an application-defined structure needed inside the callback function.

Return Values

S_OK
The object was drawn successfully.

E_INVALIDARG
Invalid window boundaries specified for metafile device context.

E_OUTOFMEMORY
Ran out of memory.

OLE_E_BLANK
No data to draw from.

DRAW_E_ABORT
Draw operation aborted.

VIEW_E_DRAW
Error in drawing.

DV_E_LINDEX
Invalid value for *lindex*; currently only -1 is supported.

DV_E_DVASPECT
Invalid value for *dwAspect*.

OLE_E_INVALIDRECT
Invalid rectangle.

Comments

A container application issues a call to **IViewObject::Draw** to create a representation of a contained object. This method draws the specified piece (*lindex*) of the specified view (*dwAspect* and *pvAspect*) on the specified device context (*hdcDraw*). Formatting, fonts, and other rendering decisions are made on the basis of the target device specified by the *ptd* parameter.

There is a relationship between the *dwDrawAspect* value and *lprcbounds* value. The *lprcbounds* value specifies the rectangle on *hdcDraw* into which the drawing is to be mapped. For DVASPECT_THUMBNAIL, DVASPECT_ICON, and DVASPECT_SMALLICON, the object draws whatever it wants to draw, and it maps it into the space given in the best way. Some objects might scale to fit while some might scale to fit but preserve aspect ratio. In addition; some might scale so the drawing appears at full width; but the bottom is cropped. The container can suggest a size via **IOleObject::SetExtent**, but it has no control over the rendering size. In the case of DVASPECT_CONTENT, the **Draw** implementation should either use the extents given by **IOleObject::SetExtent** or use the bounding rectangle given in the *lprcBounds* parameter.

Note to Callers

The value of *hicTargetDevice* is typically an information context for the target device. However, it may be a full device context instead.

Note to Implementors

If you are writing an object handler (such as the default handler) that implements **IViewObject::Draw** by playing a metafile, you have to treat **SetPaletteEntries** metafile records in a special way because of Windows' behavior. The Windows function **PlayMetaFile** sets these palette entries to the foreground. You must override this default by setting them to the background palette. Use **EnumMetaFile** to do this. Enhanced metafiles are always recorded with the background palette so there is no need to do it manually.

See Also

DVASPECT, **OleDraw**

IViewObject::Freeze

Freezes a certain aspect of the object's presentation so that it does not change until the **IViewObject::Unfreeze** method is called. The most common use of this method is for banded printing.

```
HRESULT Freeze(
    DWORD dwAspect,         //Specifies how the object is to be represented
    LONG lindex,            //Specifies the part of the object to draw
    void * pvAspect,        //Always NULL
    DWORD * pdwFreeze       //Points to location containing an identifier to be used in a subsequent call to
                            Unfreeze
  );
```

Parameters

dwAspect
[in]Specifies how the object is to be represented. Representations include content, an icon, a thumbnail, or a printed document. Valid values are taken from the enumeration **DVASPECT**. See the **DVASPECT** enumeration for more information.

lindex
[in]Indicates the portion of the object that is of interest for the draw operation. Its interpretation varies with *dwAspect.* See the **DVASPECT** enumeration for more information.

pvAspect
[in]Provides more information about the view of the object specified in *dwAspect*. Since none of the current aspects support additional information; *pvAspect* must always be NULL.

pdwFreeze
[out]Points to location where an identifying key is returned. This unique key is later used to cancel the freeze by calling **IViewObject::Unfreeze**. This key is an index that the default cache uses to keep track of which object is frozen.

Return Values

S_OK
The presentation was successfully frozen.

VIEW_S_ALREADYFROZEN
Presentation has already been frozen. The value of *pdwFreeze* is the identifying key of the already frozen object.

OLE_E_BLANK
Presentation not in cache.

DV_E_LINDEX
Invalid value for *lindex*; currently; only -1 is supported.

DV_E_DVASPECT
Invalid value for *dwAspect*.

Comments

The **IViewObject::Freeze** method causes the view object to freeze its drawn representation until a subsequent call to **IViewObject::Unfreeze** releases it. After calling **IViewObject::Freeze**; successive calls to **IViewObject::Draw** with the same parameters produce the same picture until **IViewObject::Unfreeze** is called.

IViewObject::Freeze is not part of the persistent state of the object and does not continue across unloads and reloads of the object.

The most common use of this method is for banded printing.

While in a frozen state; view notifications are not sent. Pending view notifications are deferred to the subsequent call to **IViewObject::Unfreeze**.

See Also

DVASPECT, **IViewObject::Unfreeze**

IViewObject::GetAdvise

Retrieves the existing advisory connection on the object if there is one. This method simply returns the parameters used in the most recent call to the **IViewObject::SetAdvise** method.

```
HRESULT GetAdvise(
  DWORD * pdwAspect,          //Points to location of dwAspect parameter from previous SetAdvise
  DWORD * padvf,              //Points to location of advf parameter from previous SetAdvise
  IAdviseSink ** ppAdvSink    //Points to location of pAdvSink parameter from previous SetAdvise
 );
```

Parameters

pdwAspect
[out]Points to location where the *dwAspect* parameter from the previous **SetAdvise** call is returned. If this pointer is NULL, the caller does not permit this value to be returned.

padvf
[out]Points to location where the *advf* parameter from the previous **SetAdvise** call is returned. If this pointer is NULL, the caller does not permit this value to be returned.

ppAdvSink
[out]Points to location where the *pAdvSink* parameter from the previous **SetAdvise** call is returned. If this pointer is NULL, the caller does not permit this value to be returned.

Return Values

S_OK
The existing advisory connection was retrieved.

E_INVALIDARG
An invalid *advf* flag was specified.

E_OUTOFMEMORY
Ran out of memory.

See Also

ADVF, **IAdviseSink**, **IViewObject::SetAdvise**

IViewObject::GetColorSet

Returns the logical palette that the object will use for drawing in its **IViewObject::Draw** method with the corresponding parameters.

```
HRESULT GetColorSet(
    DWORD dwAspect,                 //Specifies how the object is to be represented
    LONG lindex,                    //Specifies the part of the object to draw
    void * pvAspect,                //Always NULL
    DVTARGETDEVICE * ptd,           //Specifies the target device in a structure
    HDC hicTargetDev,               //Specifies the information context for the target device ptd
    LOGPALETTE ** ppColorSet        //Points to LOGPALETTE structure
  );
```

Parameters

dwAspect
[in]Specifies how the object is to be represented. Representations include content, an icon, a thumbnail, or a printed document. Valid values are taken from the enumeration **DVASPECT**. See the **DVASPECT** enumeration for more information.

lindex
[in]Indicates the portion of the object that is of interest for the draw operation. Its interpretation varies with *dwAspect.* See the **DVASPECT** enumeration for more information.

pvAspect
[in]Provides more information about the view of the object specified in *dwAspect*. Since none of the current aspects support additional information; *pvAspect* must always be NULL.

ptd
[in]Points to the target device structure that describes the device for which the object is to be rendered. If NULL, the view should be rendered for the default target device (typically the display). A value other than NULL is interpreted in conjunction with *hicTargetDev* and *hdcDraw*. For example, if *hdcDraw* specifies a printer as the device context, *ptd* points to a structure describing that printer device. The data may actually be printed if *hicTargetDev* is a valid value or it may be displayed in print preview mode if *hicTargetDev* is NULL.

hicTargetDev
[in]Specifies the information context for the target device indicated by the *ptd* parameter from which the object can extract device metrics and test the device's capabilities. If *ptd* is NULL, the object should ignore the *hicTargetDev* parameter.

ppColorSet
[out]Points to location where a **LOGPALETTE** structure is returned. The **LOGPALETTE** structure contains the set of colors that would be used if **IViewObject::Draw** were called with the same parameters for *dwAspect*, *lindex*, *pvAspect*, *ptd*, and *hicTargetDev*. A NULL pointer to the **LOGPALETTE** structure means that the object does not use a palette.

Return Values

S_OK
The set of colors was returned successfully.

S_FALSE
Set of colors is empty or the object will not give out the information.

E_INVALIDARG
One or more arguments are invalid.

OLE_E_BLANK
No presentation data for object.

E_UNEXPECTED

An unexpected error occurred.

DV_E_LINDEX
Invalid value for *lindex*; currently only -1 is supported.

DV_E_DVASPECT
Invalid value for *dwAspect*.

Comments

The **IViewObject::GetColorSet** method recursively queries any nested objects and returns a color set that represents the union of all colors requested. The color set eventually percolates to the top-level container that owns the window frame. This container can call **IViewObject::GetColorSet** on each of its embedded objects to obtain all the colors needed to draw the embedded objects. The container can use the color sets obtained in conjunction with other colors it needs for itself to set the overall color palette.

The OLE-provided implementation of **IViewObject::GetColorSet** looks at the data it has on hand to draw the picture. If CF_DIB is the drawing format; the palette found in the bitmap is used. For a regular bitmap; no color information is returned. If the drawing format is a metafile, the object handler enumerates the metafile looking for a CreatePalette metafile record. If one is found; the handler uses it as the color set.

Note to Implementors

Object applications that rely on the default handler for drawing and that use metafiles for doing so should provide a SetPaletteEntries record when they generate their metafiles. If a SetPaletteEntries record is not found; the default object handler returns S_FALSE.

See Also

DVASPECT

IViewObject::SetAdvise

Sets up a connection between the view object and an advise sink so that the advise sink can be notified about changes in the object's view.

```
HRESULT SetAdvise(
  DWORD dwAspect,           //Specifies the view object for which notification is being
                            requested
  DWORD advf,               //Specifies information about the advise sink
  IAdviseSink * pAdvSink    //Points to the advise sink that is to receive change notifications
 );
```

Parameters

dwAspect
[in]Specifies the view for which the advisory connection is being set up. Valid values are taken from the enumeration **DVASPECT**. See the **DVASPECT** enumeration for more information.

advf
[in]Contains a group of flags for controlling the advisory connection. Valid values are from the enumeration **ADVF**. However, only some of the possible **ADVF** values are relevant for this method. The following table briefly describes the relevant values. See the **ADVF** enumeration for a more detailed description.

ADVF Value	Description
ADVF_ONLYONCE	Causes the advisory connection to be destroyed after the first notification is sent.
ADVF_PRIMEFIRST	Causes an initial notification to be sent regardless of whether data has changed from its current state.

Note that the ADVF_ONLYONCE and ADVF_PRIMEFIRST can be combined to provide an asynchronous call to **IDataObject::GetData**.

pAdvSink
Points to the advisory sink that is to be informed of changes. A NULL value deletes any existing advisory connection.

Return Values

S_OK
The advisory connection was successfully established.

E_INVALIDARG
An invalid *advf* flag was specified.

E_OUTOFMEMORY
Ran out of memory.

OLE_E_ADVISENOTSUPPORTED
Advisory notifications are not supported.

DV_E_DVASPECT
Invalid value for *dwAspect*.

Comments

A container application that is requesting a draw operation on a view object can also register with the **IViewObject::SetAdvise** method to be notified when the presentation of the view object changes. To find out about when an object's underlying data changes, you must call **IDataObject::DAdvise** separately.

To remove an existing advisory connection, call the **IViewObject::SetAdvise** method with *pAdvSink* set to NULL.

If the view object changes, a call is made to the appropriate advise sink through its **IAdviseSink::OnViewChange** method.

At any time; a given view object can support only one advisory connection. Therefore, when **IViewObject::SetAdvise** is called and the view object is already holding onto an advise sink pointer, OLE releases the existing pointer before the new one is registered.

See Also

ADVF, **IAdviseSink**, **IViewObject::GetAdvise**

IViewObject::Unfreeze

Releases a previously frozen drawing. The most common use of this method is for banded printing.

```
HRESULT Unfreeze(
   DWORD dwFreeze        //Identifies the view object that is frozen
  );
```

Parameter

dwFreeze
[in]Contains a key previously returned from **IViewObject::Freeze**. This key determines which object to unfreeze.

Return Values

S_OK
The drawing was unfrozen successfully.

OLE_E_NOCONNECTION
Error in the unfreezing process or the object is currently not frozen.

See Also

IViewObject::Freeze

IViewObject2

The **IViewObject2** interface is an extension to the **IViewObject** interface which returns the size of the drawing for a given view of an object. You can prevent the object from being run if it isn't already running by calling this method instead of **IOleObject::GetExtent**.

Like the **IViewObject** interface, **IViewObject2** cannot be marshaled to another process. This is because device contexts are only effective in the context of one process.

The OLE provided default implementation provides the size of the object in the cache.

When to Implement

Object handlers and in-process servers that manage their own presentations implement **IViewObject2** for use by compound document containers.

When to Use

A container application or object handler calls the **GetExtent** method in the **IViewObject2** interface to get the object's size from its cache.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IViewObject Methods	**Description**
Draw	Draws a representation of the object onto a device context.
GetColorSet	Returns the logical palette the object uses for drawing.
Freeze	Freezes the drawn representation of an object so it will not change until a subsequent **Unfreeze**.
Unfreeze	Unfreezes the drawn representation of an object.
SetAdvise	Sets up a connection between the view object and an advise sink so that the advise sink can receive notifications of changes in the view object.
GetAdvise	Returns the information on the most recent **SetAdvise**.

IViewObject2 Method	**Description**
GetExtent	Returns the size of the view object from the cache.

IViewObject2::GetExtent

Returns the size that the specified view object will be drawn on the specified target device.

```
HRESULT GetExtent(
  DWORD dwAspect,             //Specifies the view object for which the size is being requested
  DWORD lindex,               //Specifies the part of the object to draw
  DVTARGETDEVICE ptd,         //Specifies the target device in a structure
  LPSIZEL lpsizel             //Points to size of object
 );
```

Parameters

dwAspect
: [in]Specifies the requested view of the object whose size is of interest. Valid values are taken from the enumeration **DVASPECT**. See the **DVASPECT** enumeration for more information.

lindex
: [in]Indicates the portion of the object that is of interest. Currently only -1 is valid.

ptd
: [in]Points to the target device for which the object's size should be returned.

lpsizel
: Points to location where the object's size is returned.

Return Values

S_OK
: The object's extent was successfully returned.

OLE_E_BLANK
: An appropriate cache is not available.

E_OUTOFMEMORY
: Insufficient memory to execute this operation.

Comments

The OLE-provided implementation of **IViewObject2::GetExtent** searches the cache for the size of the view object.

The **GetExtent** method in the **IOleObject** interface provides some of the same information as **IViewObject2::GetExtent**.

Note to Callers

To prevent the object from being run if it isn't already running, you can call **IViewObject2::GetExtent** rather than **IOleObject::GetExtent** to determine the size of the presentation to be drawn.

See Also

IOleObject::GetExtent, **IViewObject**

Legal Information

Information in this online help system is subject to change without notice and does not represent a commitment on the part of Microsoft Corporation. The software and/or files described in this online help system are furnished under a license agreement or nondisclosure agreement. The software and/or files may be used or copied only in accordance with the terms of the agreement. The purchaser may make one copy of the software for backup purposes. No part of this online help system may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or information and retrieval systems, for any purpose other than the purchaser's personal use, without the written permission of Microsoft Corporation.

Microsoft may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property rights except as expressly provided in any written license agreement from Microsoft.

Microsoft, MS, MS-DOS, Visual Basic, Win32, Win32s, and Windows are registered trademarks, and Visual C++ and Windows NT are trademarks of Microsoft Corporation.

Apple and Macintosh are registered trademarks of Apple Computer, Inc. Corel is a registered trademark of Corel Systems Corporation. IBM is a registered trademark of International Business Machines Corporation. Ami Pro, Lotus, and 1-2-3 are registered trademarks of Lotus Development Corporation. WordPerfect is a registered trademark of Novell, Inc.

BIND_OPTS

The **BIND_OPTS** structure contains parameters used during a moniker binding operation. A **BIND_OPTS** structure is stored in a bind context; the same bind context is used by each component of a composite moniker during binding, allowing the same parameters to be passed to all components of a composite moniker. See **IBindCtx** for more information about bind contexts.

If you're a moniker client (that is, you use a moniker to acquire an interface pointer to an object), you typically do not need to specify values for the fields of this structure. The **CreateBindCtx** API function creates a bind context with the bind options set to default values that are suitable for most situations; the **BindMoniker** API function does the same thing when creating a bind context for use in binding a moniker. If you want to modify the values of these bind options, you can do so by passing a **BIND_OPTS** structure to the **IBindCtx::SetBindOptions** method. Moniker implementors can pass a **BIND_OPTS** structure to the **IBindCtx::GetBindOptions** method to retrieve the values of these bind options.

The **BIND_OPTS** structure is defined in OBJIDL.IDL.

```
typedef struct tagBIND_OPTS
{
     DWORD cbStruct;
     DWORD grfFlags;
     DWORD grfMode;
     DWORD dwTickCountDeadline;
} BIND_OPTS, *LPBIND_OPTS;
```

Members

cbStruct
Specifies the size of this structure in bytes (that is, the size of the **BIND_OPTS** structure).

grfFlags
Specifies flags that control aspects of moniker binding operations. This value is any combination of the bit flags in the **BINDFLAGS** enumeration. New values may be defined in the future, so moniker implementations should ignore any bits in this field that they do not understand. The **CreateBindCtx** API function initializes this field to zero.

grfMode
Specifies flags that should be used when opening the file that contains the object identified by the moniker. The values are taken from the **STGM** enumeration. The binding operation uses these flags in the call to **IPersistFile::Load** when loading the file. If the object is already running, these flags are ignored by the binding operation. The **CreateBindCtx** API function initializes this field to STGM_READWRITE.

dwTickCountDeadline
Specifies the clock time (in milliseconds, as returned by the **GetTickCount** API function) by which the caller would like the binding operation to be completed. This member lets the caller limit the execution time of an operation when speed is of primary importance. A value of zero indicates that there is no deadline. Callers most often use this capability when calling the **IMoniker::GetTimeOfLastChange** method, though it can be usefully applied to other operations as well. The **CreateBindCtx** API function initializes this field to zero.

Typical deadlines allow for a few hundred milliseconds of execution. This deadline is a recommendation, not a requirement; however, operations that exceed their deadline by a large amount may cause delays for the end user. Each moniker implementation should try to complete its operation by the deadline, or fail with the error MK_E_EXCEEDEDDEADLINE.

If a binding operation exceeds its deadline because one or more objects that it needs are not running, the moniker implementation should register the objects responsible in the bind context using the **IBindCtx::RegisterObjectParam**. The objects should be registered under the parameter names "ExceededDeadline", "ExceededDeadline1", "ExceededDeadline2", and so on. If the caller later finds the object in the Running Object Table, the caller can retry the binding operation.

The **GetTickCount** API function indicates the number of milliseconds since system startup, and wraps back to zero after 2^31 milliseconds. Consequently, callers should be careful not to inadvertently pass a zero value (which indicates no deadline), and moniker implementations should be aware of clock wrapping problems (see the **GetTickCount** API function for more information).

See Also

BIND_FLAGS, **CreateBindCtx**, **IBindCtx::SetBindOptions**

DVTARGETDEVICE

Use the **DVTARGETDEVICE** structure to specify information about the target device for which data is being composed. **DVTARGETDEVICE** contains enough information about a Windows target device so a handle to a device context (hDC) can be created using the Windows **CreateDC** function.

```
typedef struct tagDVTARGETDEVICE
{
     DWORD tdSize;
     WORD  tdDriverNameOffset;
     WORD  tdDeviceNameOffset;
     WORD  tdPortNameOffset;
     WORD  tdExtDevmodeOffset;
     BYTE  tdData[1];
}DVTARGETDEVICE;
```

Members

tdSize
Specifies the size in bytes of the **DVTARGETDEVICE** structure. The initial size is included so the structure can be copied more easily.

tdDriverNameOffset
Specifies the offset in bytes from the beginning of the structure to the device driver name, which is stored as a NULL-terminated string in the *tdData* buffer.

tdDeviceNameOffset
Specifies the offset in bytes from the beginning of the structure to the device name, which is stored as a NULL-terminated string in the *tdData* buffer. This value can be zero to indicate no device name.

tdPortNameOffset
Specifies the offset in bytes from the beginning of the structure to the port name, which is stored as a NULL-terminated string in the *tdData* buffer. This value can be zero to indicate no port name.

tdExtDevmodeOffset
Specifies the offset in bytes from the beginning of the structure to the **DEVMODE** structure retrieved by the calling **ExtDeviceMode**.

tdData
Specifies an array of bytes containing data for the target device. It is not necessary to include empty strings in *tdData* (for names where the offset value is zero).

Comments

Some OLE 1 client applications incorrectly construct target devices by allocating too few bytes in the **DEVMODE** structure for the OLETARGETDEVICE. They typically only supply the number of bytes in the DEVMODE.*dmSize* member. The number of bytes to be allocated should be the sum of DEVMODE.*dmSize* + DEVMODE.*dmDriverExtra*. When a call is made to the **CreateDC** API function with an incorrect target device, the printer driver tries to access the additional bytes and unpredictable results can occur. To protect against a crash and make the additional bytes available, OLE pads the size of OLE 2 target devices created from OLE 1 target devices.

See Also

FORMATETC, **IEnumFORMATETC**, **IViewObject**, **OleConvertOLESTREAMToIStorage**

FILETIME

The **FILETIME** data structure is a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601. It is the means by which Win32 determines the date and time. **FILETIME** is used by the **CoDosDateTimeToFileTime**, **CoFileTimeToDosDateTime**, and **CoFileTimeNow** functions. It is defined as follows:

```
typedef struct STRUCT tagFILETIME
{
     DWORD  dwLowDateTime;
     DWORD  dwHighDateTime;
} FILETIME;
```

Members

dwLowDateTime
The low 32 bits of the Win32 date/time value.

dwHighDateTime
The upper 32 bits of the Win32 date/time value.

Comments

The **FILETIME** data structure is used in the time conversion functions between DOS and Win32.

See Also

CoDosDateTimeToFileTime, **CoFileTimeNow**, **CoFileTimeToDosDateTime**

FORMATETC

The **FORMATETC** structure is a generalized Clipboard format. It is enhanced to encompass a target device, the aspect or view of the data, and a storage medium indicator. Where one might expect to find a Clipboard format, OLE uses a **FORMATETC** data structure instead.

Defined in the **IEnumFORMATETC** interface.

```
typedef struct tagFORMATETC
{
     CLIPFORMAT  cfFormat;
     DVTARGETDEVICE  *ptd;
     DWORD  dwAspect;
     LONG  lindex;
     DWORD  tymed;
}FORMATETC;  *LPFORMATETC;
```

Members

cfFormat

Indicates the particular clipboard format of interest. There are three types of formats recognized by OLE:

- Standard interchange formats, such as CF_TEXT.
- Private application formats understood only by the application offering the format, or by other applications offering similar features.
- OLE formats, which are used to create linked or embedded objects.

ptd

Points to a **DVTARGETDEVICE** structure containing information about the target device for which the data is being composed. A NULL value is used whenever the specified data format is independent of the target device or when the caller doesn't care what device is used. In the latter case, if the data requires a target device, the object should pick an appropriate default device (often the display for visual components). Data obtained from an object with a NULL target device, such as most metafiles, is independent of the target device. The resulting data is usually the same as it would be if the user chose the Save As command from the File menu and selected an interchange format.

dwAspect

Specifies one of the **DVASPECT** enumeration constants that indicate how much detail should be contained in the rendering. A single clipboard format can support multiple aspects or views of the object. Most data and presentation transfer and caching methods pass aspect information. For example, a caller might request an object's iconic picture, using the metafile clipboard format to retrieve it. Note that only one **DVASPECT** value can be used in *dwAspect*. That is, *dwAspect* cannot be the result of a BOOLEAN OR operation on several **DVASPECT** values.

lindex

Identifies part of the aspect when the data must be split across page boundaries. The most common value is -1, which identifies all of the data. For the aspects DVASPECT_THUMBNAIL and DVASPECT_ICON, *lindex* is ignored.

tymed

Specifies one of the **TYMED** enumeration constants which indicate the type of storage medium used to transfer the object's data. Data can be transferred using whatever medium makes sense for the object. For example, data can be passed using global memory, a disk file, or structured storage objects. For more information, see the **TYMED** enumeration.

Comments

The **FORMATETC** structure is used by methods in the data transfer and presentation interfaces as a parameter specifying the data being transferred. For example, the **IDataObject::GetData** method uses the **FORMATETC** structure to indicate exactly what kind of data the caller is requesting.

See Also

DVASPECT, **IDataAdviseHolder**, **IDataObject**, **IEnumFORMATETC**, **IOleCache**, **OleCreate**, **OleCreateFromData**, **OleCreateLinkFromData**, **OleCreateStaticFromData**, **OleCreateLink**, **OleCreateLinkToFile**, **OleCreateFromFile**, **STATDATA**, **STGMEDIUM**, **TYMED**

INTERFACEINFO

The **INTERFACEINFO** structure contains information about incoming calls. The structure is defined as follows:

```
typedef struct tagINTERFACEINFO
{
    LPUNKNOWN  pUnk;
    IID  iid;
    WORD  wMethod;
} INTERFACEINFO,  * LPINTERFACEINFO;
```

Members

pUnk
Points to the object.

iid
Indicates the interface id (iid).

wMethod
Indicates the interface method.

See Also

IMessageFilter::HandleIncomingCall

OBJECTDESCRIPTOR

The **OBJECTDESCRIPTOR** structure is the data structure used for the CF_OBJECTDESRIPTOR and CF_LINKSRCDESCRIPTOR file formats. These formats provide user interface information during data transfer operations, for example, the Paste Special dialog box or target feedback information during drag and drop operations.

Defined in the **IOLETypes** pseudo-interface (*oletyp.idl*).

```
typedef struct tagOBJECTDESCRIPTOR
{
      ULONG cbSize;
      CLSID clsid;
      DWORD dwDrawAspect;
      SIZEL sizel;
      POINTL pointl;
      DWORD dwStatus;
      DWORD dwFullUserTypeName;
      DWORD dwSrcOfCopy;
      /* variable sized string data may appear here */
} OBJECTDESCRIPTOR;
```

Members

cbSize
Specifies the size of structure in bytes

clsid
Specifies the CLSID of the object being transferred. The *clsid* is used to obtain the icon for the Display As Icon option in the Paste Special dialog box and is applicable only if the Embed Source or Embedded Object formats are offered. If neither is offered, the value of *clsid* should be CLSID_NULL. The *clsid* can be retrieved by the source by loading the object and calling the **IOleObject::GetUserClassID** method. Note that for link objects, this value is not the same as the value returned by the **IPersist::GetClassID** method.

dwDrawAspect
Specifies the display aspect of the object. Typically, this value is DVASPECT_CONTENT or DVASPECT_ICON. If the source application did not draw the object originally, the *dwDrawAspect* field contains a zero value (which is not the same as DVASPECT_CONTENT).

sizel
Specifies the true extents of the object (without cropping or scaling) in HIMETRIC units. Setting this field is optional. The value can be (0,0) for applications that do not draw the object being transferred. This field is used primarily by targets of drag-and-drop operations so they can give appropriate feedback to the user.

pointl
Specifies the offset in HIMETRIC units from the upper-left corner of the object where a drag-and-drop operation was initiated. This field is only meaningful for a drag-and-drop transfer operation since it corresponds to the point where the mouse was clicked to initiate the drag-and-drop operation. The value is (0,0) for other transfer situations, such as a Clipboard copy and paste.

dwStatus
Contains a copy of the status flags for the object. These flags are defined by the **OLEMISC** enumeration. If an embedded object is being transferred, they are returned by calling the **IOleObject::GetMiscStatus** method.

dwFullUserTypeName
Specifies the offset for finding the full user type name of the object being transferred. It specifies the offset, in bytes, from the beginning of the **OBJECTDESCRIPTOR** data structure to the null-

terminated string that specifies the full user type name of the object being transferred. The value is zero if the string is not present. This string is used by the destination of a data transfer to create labels in the Paste Special dialog box. The destination application must be able to handle the cases when this string is omitted.

dwSrcOfCopy

Specifies the offset, in bytes, from the beginning of the data structure to the null-terminated string that specifies the source of the transfer. The *dwSrcOfCopy* field is typically implemented as the display name of the temporary moniker that identifies the data source. The value for *dwSrcOfCopy* is displayed in the Source line of the Paste Special dialog box. A zero value indicates that the string is not present. If *dwSrcOfCopy* is zero, the string "Unknown Source" is displayed in the Paste Special dialog box.

See Also

IDataObject, **FORMATETC**

OLEINPLACEFRAMEINFO

The **OLEINPLACEFRAMEINFO** structure contains information about the accelerators supported by a container during an in-place session. The structure is defined in the **IOleInPlaceFrame** interface (*inplcf.idl*), and it is used in the **IOleInPlaceSite::GetWindowContext** method and the **OleTranslateAccelerator** API function.

```
typedef struct tagOIFI
{
     UINT cb;
     BOOL fMDIApp;
     HWND hwndFrame;
     HACCEL haccel;
     UINT cAccelEntries;
} OLEINPLACEFRAMEINFO, *LPOLEINPLACEFRAMEINFO;
```

Members

cb
Specifies the size in bytes of this structure. The object server must specify sizeof(**OLEINPLACEFRAMEINFO**) in the structure it passes to **IOleInPlaceSite::GetWindowContext**. The container can then use this size to determine the structure's version.

fMDIApp
Indicates whether the container is an MDI application.

hwndFrame
Specifies a handle to the container's top-level frame window.

haccel
Specifies the handle to the accelerator table that the container wants to use during an in-place editing session.

cAccelEntries
Specifies the number of accelerators in *haccel*.

Comments

When an object is being in-place activated, its server calls the container's **IOleInPlaceSite::GetWindowContext** method which fills in an **OLEINPLACEFRAMEINFO** structure. During an in-place session, the message loop of an EXE server passes a pointer to the **OLEINPLACEFRAMEINFO** structure to **OleTranslateAccelerator**. OLE uses the information in this structure to determine whether a message maps to one of the container's accelerators.

See Also

IOleInPlaceSite::GetWindowContext, **OleTranslateAccelerator**

OLEMENUGROUPWIDTHS

The **OLEMENUGROUPWIDTHS** structure is the mechanism for building a shared menu. It indicates the number of menu items in each of the six menu groups of a menu shared between a container and an object server during an in-place editing session.

The structure is defined in the **IOleInPlaceFrame** interface (*inplcf.idl*). It is used in the **IOleInPlaceFrame::InsertMenus** and **ICDStandardForm::SetMenu** methods, and the **OleCreateMenuDescriptor** API function.

```
typedef struct tagOleMenuGroupWidths
{
     LONG width[6];
} OLEMENUGROUPWIDTHS, * LPOLEMENUGROUPWIDTHS;
```

Member

width

Specifies an array whose elements contain the number of menu items in each of the six menu groups of a shared in-place editing menu. Each menu group can have any number of menu items. The container uses elements 0, 2, and 4 to indicate the number of menu items in its File, View, and Window menu groups. The object server uses elements 1, 3, and 5 to indicate the number of menu items in its Edit, Object, and Help menu groups.

Comments

A container application and an object server use this structure to build a shared menu. The object server initializes to zeros the array elements in an **OLEMENUGROUPWIDTHS** structure and passes a pointer to it along with a menu handle to the container in a call to **IOleInPlaceFrame::InsertMenus**. The container adds its menu items to the menu, and fills in the structure with the number of items in each of its groups (indexes 0, 2, and 4). The server then uses the group width values returned by the container to insert its menu items in the appropriate position in the menu. The server fills in the structure with the number of items in each of its groups (indexes 1, 3, and 5), and then passes the structure to OLE in a call to the **OleCreateMenuDescriptor** API function. This enables OLE to intercept the container's menu messages and redirect the messages generated by the server's menus.

See Also

IOleInPlaceFrame::InsertMenus, **OleCreateMenuDescriptor**

OLEUIBUSY

The **OLEUIBUSY** structure contains information that the OLE 2.01 User Interface Library uses to initialize the Busy dialog box, and space for the library to return information when the dialog is dismissed.

```
typedef struct tagOLEUIBUSY
{
// These IN fields are standard across all OLEUI dialog functions.
     DWORD  cbStruct;
     DWORD  dwFlags;
     HWND  hWndOwner;
     LPCSTR  lpszCaption;
     LPFNOLEUIHOOK  lpfnHook;
     LPARAM  lCustData;
     HINSTANCE  hInstance;
     LPCSTR  lpszTemplate;
     HRSRC  hResource;

// Specifics for OLEUIBUSY.
     HTASK  hTask;
     HWND FAR *  lphWndDialog;
} OLEUICHANGEICON, *POLEUICHANGEICON, FAR *LPOLEUICHANGEICON;
```

Members

cbStruct
The size of the structure in bytes. This field must be filled on input.

dwFlags
On input, *dwFlags* specifies the initialization and creation flags. On exit, it specifies the user's choices. It may be a combination of the following flags:

BZ_DISABLECANCELBUTTON
Input only: This flag disables the Cancel button.

BZ_DISABLESWITCHTOBUTTON
Input only: This flag disables the Switch To... button.

BZ_DISABLERETRYBUTTON
Input only: This flag disables the Retry button.

BZ_NOTRESPONDINGDIALOG
Input only: This flag generates a Not Responding dialog instead of a Busy dialog. The text is slightly different, and the Cancel button is disabled.

hWndOwner
Identifies the window that owns the dialog box. It should not be NULL.

lpszCaption
Points to a string to be used as the title of the dialog box. If NULL, then the library uses Busy.

lpfnHook
Points to a hook function that processes messages intended for the dialog box. The hook function must return zero to pass a message that it didn't process back to the dialog box procedure in the library. The hook function must return a non zero value to prevent the library's dialog box procedure from processing a message it has already processed.

lCustData
Specifies application-defined data that the library passes to the hook function pointed to by the *lpfnHook* member. The library passes a pointer to the **OLEUIBUSY** structure in the *lParam* parameter of the WM_INITDIALOG message; this pointer can be used to retrieve the *lCustData*

member.

hInstance

Identifies an instance that contains a dialog box template specified by the *lpTemplateName* member.

lpszTemplate

Points to a null-terminated string that specifies the name of the resource file for the dialog box template that is to be substituted for the library's Busy dialog box template.

hResource

A customized template handle.

hTask Input only:

A handle to the task that is blocking.

lphWndDialog

The dialog box's HWND is placed here.

See Also

OleUIBusy

OLEUICHANGEICON

The **OLEUICHANGEICON** structure contains information that the OLE 2.01 User Interface Library uses to initialize the Change Icon dialog box, and it contains space for the library to return information when the dialog box is dismissed.

```
typedef struct tagOLEUICHANGEICON
{
// These IN fields are standard across all OLEUI dialog functions.
     DWORD  cbStruct;
     DWORD  dwFlags;
     HWND  hWndOwner;
     LPCSTR  lpszCaption;
     LPFNOLEUIHOOK  lpfnHook;
     LPARAM  lCustData;
     HINSTANCE  hInstance;
     LPCSTR  lpszTemplate;
     HRSRC  hResource;

 // Specifics for OLEUICHANGEICON.
     HGLOBAL  hMetaPict;
     CLSID  clsid;
     char  szIconExe[OLEUI_CCHPATHMAX];
     int  cchIconExe;
} OLEUICHANGEICON, *POLEUICHANGEICON, FAR *LPOLEUICHANGEICON;
```

Members

cbStruct
The size of the structure in bytes. This field must be filled on input.

dwFlags
On input, *dwFlags* specifies the initialization and creation flags. On exit, it specifies the user's choices. It can be a combination of the following flags:

CIF_SHOWHELP
Specifies that the dialog box will display a Help button.

CIF_SELECTCURRENT
On input, selects the Current radio button on initialization. On exit, specifies that the user selected Current.

CIF_SELECTDEFAULT
On input, selects the Default radio button on initialization. On exit, specifies that the user selected Default.

CIF_SELECTFROMFILE
On input, selects the From File radio button on initialization. On exit, specifies that the user selected From File.

CIF_USEICONEXE
Input only. Extracts the icon from the executable specified in the *szIconExe* member, instead of retrieving it from the class. This is useful for OLE embedding or linking to non-OLE files.

hWndOwner
Identifies the window that owns the dialog box. It should not be NULL.

lpszCaption
Points to a string to be used as the title of the dialog box. If NULL, then the library uses Change Icon.

lpfnHook

Points to a hook function that processes messages intended for the dialog box. The hook function must return zero to pass a message that it didn't process back to the dialog box procedure in the library. The hook function must return a non-zero value to prevent the library's dialog box procedure from processing a message it has already processed.

lCustData

Specifies application-defined data that the library passes to the hook function pointed to by the *lpfnHook* member. The library passes a pointer to the **OLEUICHANGEICON** structure in the *lParam* parameter of the WM_INITDIALOG message; this pointer can be used to retrieve the *lCustData* member.

hInstance

Identifies an instance that contains a dialog box template specified by the *lpTemplateName* member.

lpszTemplate

Points to a null-terminated string that specifies the name of the resource file for the dialog box template that is to be substituted for the library's Change Icon dialog box template.

hResource

A customized template handle.

hMetaPict

The current and final image. The source of the icon is embedded in the metafile itself.

clsid

Input only. The class to use to get the Default icon.

szIconExe

Input only. The executable to extract the default icon from. This member is ignored unless CIF_USEICONEXE is included in the *dwFlags* parameter and an attempt to retrieve the class icon from the specified CLSID fails.

cchIconExe

Input only. The number of characters in *szIconExe*. This member is ignored unless CIF_USEICONEXE is included in the *dwFlags* member.

See Also

OleUIChangeIcon

OLEUICHANGESOURCE

This structure is used to initialize the standard Change Source dialog box. It allows the user to modify the destination or source of a link. This may simply entail selecting a different file name for the link, or possibly changing the item reference within the file, for example, changing the destination range of cells within spreadsheet that the link is to.

```
typedef struct tagOLEUICHANGESOURCEW
{
// These IN fields are standard across all OLEUI dialog functions.     DWORD
cbStruct;
     DWORDT  dwFlags;
     HWND    hWndOwner;
     LPCWSTR  lpszCaption;
     LPFNOLEUIHOOK  lpfnHook;
     LPARAM  lCustData;
     HINSTANCE  hInstance;
     LPCWSTR  lpszTemplate;
     HRSRC  hResource;

// INTERNAL ONLY: do not modify these members
     OPENFILENAMEW*  lpOFN;
     DWORD  dwReserved1[4];

// Specifics for OLEUICHANGESOURCE.
     LPOLEUILINKCONTAINERW  lpOleUILinkContainer;
     DWORD  dwLink;
     LPTSTR  lpszDisplayName;
     ULONG  nFileLength;
     LPTSTR  lpszFrom;
     LPTSTR  lpszTo;
} OLEUICHANGESOURCEW, *POLEUICHANGESOURCEW, FAR *LPOLEUICHANGESOURCEW;
```

Members

cbStruct
The size of the structure in bytes.

dwFlags
On input, this field specifies the initialization and creation flags. On exit, it specifies the user's choices. It may be a combination of the following flags:

CSF_SHOWHELP
Enables or shows the Help button.

CSF_VALIDSOURCE
Indicates that the link was validated.

CSF_ONLYGETSOURCE
Disables automatic validation of the link source when the user presses OK. If you specify this flag, you should validate the source when the dialog box returns OK.

hWndOwner
Identifies the window that owns the dialog box.

lpszCaption
Points to a string to be used as the title of the dialog box. If NULL, then the library uses Change Source.

lpfnHook

Points to a hook function that processes messages intended for the dialog box. The hook function must return zero to pass a message that it didn't process back to the dialog box procedure in the library. The hook function must return a non zero value to prevent the library's dialog box procedure from processing a message it has already processed.

lCustData

Specifies application-defined data that the library passes to the hook function pointed to by the *lpfnHook* member. The library passes a pointer to the **OLEUICHANGEICON** structure in the *lParam* parameter of the WM_INITDIALOG message; this pointer can be used to retrieve the *lCustData* member.

hInstance

Identifies a instance that contains a dialog box template specified by the *lpszTemplate* member. This member is ignored if the *lpszTemplate* member is NULL or invalid.

lpszTemplate

Points to a null-terminated string that specifies the name of the resource file for the dialog box template that is to be substituted for the library's Convert dialog box template.

hResource

The resource handle for a custom dialog. If this member is NULL, then the library uses the standard Convert dialog template, or if it is valid, the template named by the *lpszTemplate* member.

lpOFN

Points to the **OPENFILENAME** structure, which contains information used by the operating system to initialize the system-defined Open or Save As dialog boxes.

dwReserved1[4]

Reserved for future use.

lpOleUILinkContainer

Pointsto the container's implementation of the **IOleUILinkContainer** Interface; used to validate the link source. The Edit Links dialog uses this to allow the container to manipulate its links.

dwLink

A container-defined unique 32-bit link identifier used to validate link sources. Used by *lpOleUILinkContainer*.

lpszDisplayName

Points to the complete source display name.

nFileLength

This is the file moniker portion of *lpszDisplayName*.

lpszFrom

The prefix of the source that was changed from.

lpszTo

The prefix of the source to be changed to.

See Also

IOleUILinkContainer, **OleUIChangeSource**

OLEUICONVERT

The **OLEUICONVERT** structure contains information that the OLE 2.01 User Interface Library uses to initialize the Convert dialog box, and space for the library to return information when the dialog is dismissed.

```
typedef struct tagOLEUICONVERT
{
// These IN fields are standard across all OLEUI dialog functions.
     DWORD  cbStruct
     DWORD  dwFlags
     HWND  hWndOwner
     LPCSTR  lpszCaption
     LPFNOLEUIHOOK  lpfnHook
     LPARAM  lCustData
     HINSTANCE hInstance
     LPCSTR  lpszTemplate
     HRSRC  hResource

// Specifics for OLEUICONVERT.
     CLSID  clsid;
     CLSID  clsidConvertDefault;
     CLSID  clsidActivateDefault;
     CLSID  clsidNew;
     DWORD  dvAspect;
     WORD  wFormat;
     BOOL  fIsLinkedObject;
     HGLOBAL  hMetaPict;
     LPTSTR  lpszUserType;
     BOOL  fObjectsIconChanged;
     LPTSTR  lpszDefLabel
     UINT  cClsidExclude
     LPCLSID  lpClsidExclude
} OLEUICONVERT, *POLEUICONVERT, FAR *LPOLEUICONVERT;
```

Members

cbStruct

The size of the structure, in bytes. This field must be filled on input.

dwFlags

On input, this field specifies the initialization and creation flags. On exit, it specifies the user's choices. It may be a combination of the following flags:

CF_SHOWHELPBUTTON

Specifies that the dialog will display a Help button. This flag is set on input.

CF_SETCONVERTDEFAULT

Specifies that the class whose CLSID is specified by *clsidConvertDefault* will be used as the default selection. This selection appears in the class listbox when the Convert To radio button is selected. This flag is set on input.

CF_SETACTIVATEDEFAULT

Specifies that the class whose CLSID is specified by *clsidActivateDefault* will be used as the default selection. This selection appears in the class listbox when the Activate As radio button is selected. This flag is set on input.

CF_SELECTCONVERTTO

On input, this flag specifies that Convert To will be initially selected (default behavior). This flag is set on output if Convert To was selected when the user dismissed the dialog.

CF_SELECTACTIVATEAS
On input, this flag specifies that Activate As will be initially selected. This flag is set on output if Activate As was selected when the user dismissed the dialog.

CF_DISABLEDISPLAYASICON
Specifies that the Display As Icon button will be disabled on initialization.

CF_DISABLEACTIVATEAS
Specifies that the Activate As radio button will be disabled on initialization.

CF_HIDECHANGEICON
Hides the Change Icon button in the Convert dialog.

CF_CONVERTONLY
Disables the Activate As radio button in the Convert dialog.

hWndOwner
Identifies the window that owns the dialog box. It should not be NULL.

lpszCaption
Points to a string to be used as the title of the dialog box. If NULL, then the library uses Convert.

lpfnHook
Points to a hook function that processes messages intended for the dialog box. The hook function must return zero to pass a message that it didn't process back to the dialog box procedure in the library. The hook function must return a non-zero value to prevent the library's dialog box procedure from processing a message it has already processed.

lCustData
Specifies application-defined data that the library passes to the hook function pointed to by the *lpfnHook* member. The library passes a pointer to the **OLEUICONVERT** structure in the *lParam* parameter of the WM_INITDIALOG message; this pointer can be used to retrieve the *lCustData* member.

hInstance
Identifies a instance that contains a dialog box template specified by the *lpszTemplate* member. This member is ignored if the *lpszTemplate* member is NULL or invalid.

lpszTemplate
Points to a null-terminated string that specifies the name of the resource file for the dialog box template that is to be substituted for the library's Convert dialog box template.

hResource
The resource handle for a custom dialog. If this member is NULL, then the library uses the standard Convert dialog template, or if it is valid, the template named by the *lpszTemplate* member.

clsid
The CLSID of the object to be converted or activated. This member is set on input.

clsidConvertDefault
The CLSID to use as the default class when Convert To is selected. This member is ignored if the *dwFlags* member does not include CF_SETCONVERTDEFAULT. This member is set on input.

clsidActivateDefault
The CLSID to use as the default class when Activate As is selected. This member is ignored if the *dwFlags* member does not include CF_SETACTIVATEDEFAULT. This member is set on input.

clsidNew
The CLSID of the selected class. This member is set on output.

dvAspect
The aspect of the object. This must be either DVASPECT_CONTENT or DVASPECT_ICON. If *dvAspect* is DVASPECT_ICON on input, then the Display As Icon box is checked and the object's icon is displayed. This member is set on input and output.

wFormat

The data format of the object to be converted or activated.

fIsLinkedObject

TRUE if the object is linked. This member is set on input.

hMetaPict

The METAFILEPICT containing the iconic aspect. This member is set on input and output.

lpszUserType

The User Type name of the object to be converted or activated. If this value is NULL, then the dialog will retrieve the User Type name from the registry. This string is freed on exit.

fObjectsIconChanged

TRUE if the object's icon changed. (that is, if **OleUIChangeIcon** was called and not canceled.). This member is set on output.

lpszDefLabel

The default label to use for the icon. If NULL, the short user type name will be used. If the object is a link, the caller should pass the Display Name of the link source. This is freed on exit.

cClsidExclude

The number of CLSIDs in *lpClsidExclude*.

lpClsidExclude

The list of CLSIDs to exclude from the list.

See Also

OleUIConvert, **OleUIChangeIcon**

OLEUIEDITLINKS

The **OLEUIEDITLINKS** structure contains information that the OLE 2.01 User Interface Library uses to initialize the Edit Links dialog box, and contains space for the library to return information when the dialog is dismissed.

```
typedef struct tagOLEUIEDITLINKS
{
// These IN fields are standard across all OLEUI dialog functions.
     DWORD  cbStruct;
     DWORD  dwFlags;
     HWND  hWndOwner;
     LPCSTR  lpszCaption;
     LPFNOLEUIHOOK  lpfnHook;
     LPARAM  lCustData;
     HINSTANCE  hInstance;
     LPCSTR  lpszTemplate;
     HRSRC  hResource;

// Specifics for OLEUIEDITLINKS.
     LPOLEUILINKCONTAINER    lpOleUILinkContainer;
} OLEUIEDITLINKS, *POLEUIEDITLINKS, FAR *LPOLEUIEDITLINKS;
```

Members

cbStruct
The size of the structure in bytes. This field must be filled on input.

dwFlags
On input, *dwFlags* specifies the initialization and creation flags. It may be a combination of the following flags:

ELF_SHOWHELP
Specifies that the dialog will display a Help button.

ELF_DISABLEUPDATENOW
Specifies that the Update Now button will be disabled on initialization.

ELF_DISABLEOPENSOURCE
Specifies that the Open Source button will be disabled on initialization.

ELF_DISABLECHANGESOURCE
Specifies that the Change Source button will be disabled on initialization.

ELF_DISABLECANCELLINK
Specifies that the Cancel Link button will be disabled on initialization.

hWndOwner
Identifies the window that owns the dialog box. It should not be NULL.

lpszCaption
Points to a string to be used as the title of the dialog box. If NULL, then the library uses Links.

lpfnHook
Points to a hook function that processes messages intended for the dialog box. The hook function must return zero to pass a message that it didn't process back to the dialog box procedure in the library. The hook function must return a non-zero value to prevent the library's dialog box procedure from processing a message it has already processed.

lCustData
Specifies application-defined data that the library passes to the hook function pointed to by the *lpfnHook* member. The library passes a pointer to the **OLEUIEDITLINKS** structure in the *lParam* parameter of the WM_INITDIALOG message; this pointer can be used to retrieve the *lCustData*

member.

hInstance

Identifies an instance that contains a dialog box template specified by the *lpTemplateName* member.

lpszTemplate

Points to a null-terminated string that specifies the name of the resource file for the dialog box template that is to be substituted for the library's Edit Links dialog box template.

hResource

A customized template handle.

lpOleUILinkContainer

Points to the container's implementation of the **IOleUILinkContainer** Interface. The Edit Links dialog uses this to allow the container to manipulate its links.

See Also

IOleUILinkContainer, **OleUIEditLinks**

OLEUIGNRLPROPS

This structure is used to initialize the General tab of the Object Properties dialog box. A reference to it is passed in as part of the **OLEUIOBJECTPROPS** structure to the **OleUIObjectProperties** function. This tab shows the type and size of an OLE embedding and allows it the user to tunnel to the convert dialog box. This tab also shows the link destination if the object is a link.

```
typedef struct tagOLEUIGNRLPROPS
{
// These IN fields are standard across all OLEUI property pages. DWORD
cbStruct;
     DWORD  dwFlags;
     DWORD  dwReserved1[2];
     LPFNOLEUIHOOK  lpfnHook;
     LPARAM  lCustData;
     DWORD  dwReserved2[3];

     struct tagOLEUIOBJECTPROPSW* lpOP;
} OLEUIGNRLPROPSW, *POLEUIGNRLPROPSW, FAR* LPOLEUIGNRLPROPSW;
```

Members

cbStruct
The size of the structure in bytes. This field must be filled on input.

dwFlags
There are currently no flags associated with this member. It should be set to 0 (zero).

dwReserved1[2]
Reserved for future use.

lpfnHook
Points to a hook function that processes messages intended for the dialog box. The hook function must return zero to pass a message that it didn't process back to the dialog box procedure in the library. The hook function must return a non zero value to prevent the library's dialog box procedure from processing a message it has already processed.

lCustData
Specifies application-defined data that the library passes to the hook function pointed to by the *lpfnHook* member during WM_INITDIALOG.

dwReserved2[3]
Reserved for future use.

struct tagOLEUIOBJECTPROPSW* *lpOP*
Used internally.

See Also

OleUIObjectProperties, **OLEUIOBJECTPROPS**

OLEUIINSERTOBJECT

The OLEUIINSERTOBJECT structure contains information that the OLE 2.01 User Interface Library uses to initialize the Insert Object dialog box, and space for the library to return information when the dialog is dismissed.

```
typedef struct tagOLEUIINSERTOBJECT
{
// These IN fields are standard across all OLEUI dialog functions.
     DWORD  cbStruct;
     DWORD  dwFlags;
     HWND  hWndOwner;
     LPCSTR  lpszCaption;
     LPFNOLEUIHOOK  lpfnHook;
     LPARAM  lCustData;
     HINSTANCE  hInstance;
     LPCSTR  lpszTemplate;
     HRSRC  hResource;
     CLSID  clsid;

// Specifics for OLEUIINSERTOBJECT.
     LPTSTR  lpszFile;
     UINT  cchFile;
     UINT  cClsidExclude;
     LPCLSID  lpClsidExclude;
     IID  iid;

// Specific to create objects if flags say so
     DWORD  oleRender;
     LPFORMATETC  lpFormatEtc;
     LPOLECLIENTSITE  lpIOleClientSite;
     LPSTORAGE  lpIStorage;
     LPVOID FAR *  ppvObj;
     SCODE  sc;
     HGLOBAL  hMetaPict;
} OLEUIINSERTOBJECT, *POLEUIINSERTOBJECT, FAR *LPOLEUIINSERTOBJECT;
```

Members

cbStruct
Size of the structure in bytes. This field must be filled on input.

dwFlags
On input, specifies the initialization and creation flags. On exit, specifies the user's choices. It can be a combination of the following flags:

IOF_SHOWHELP
Specifies that the dialog will display a Help button.

IOF_SELECTCREATENEW
Specifies that the Create New radio button will initially be checked. This cannot be used with IOF_SELECTCREATEFROMFILE.

IOF_SELECTCREATEFROMFILE
Specifies that the Create From File radio button will initially be checked. This cannot be used with IOF_SELECTCREATENEW.

IOF_CHECKLINK
Specifies that the Link check box will initially be checked.

IOF_CHECKDISPLAYASICON
Specifies that the Display As Icon check box will initially be checked, the current icon will be displayed, and the Change Icon button will be enabled.

IOF_CREATENEWOBJECT
Specifies that a new object should be created when the user selects OK to dismiss the dialog box and the Create New radio button was selected.

IOF_CREATEFILEOBJECT
Specifies that a new object should be created from the specified file when the user selects OK to dismiss the dialog box and the Create From File radio button was selected.

IOF_CREATELINKOBJECT
Specifies that a new linked object should be created when the user selects OK to dismiss the dialog box and the user checked the Link check box.

IOF_DISABLELINK
Specifies that the Link check box will be disabled on initialization.

IOF_VERIFYSERVERSEXIST
Specifies that the dialog box should validate the classes it adds to the listbox by ensuring that the server specified in the registration database exists. This is a significant performance factor.

IOF_DISABLEDISPLAYASICON
Specifies that the Display As Icon check box will be disabled on initialization.

IOF_HIDECHANGEICON
Hides the Change Icon button in the Insert Object dialog.

IOF_SHOWINSERTCONTROL
Displays the Insert Control radio button.

IOF_SELECTCREATECONTROL
Displays the Create Control radio button.

hWndOwner
Identifies the window that owns the dialog box. It should not be NULL.

lpszCaption
Points to a string to be used as the title of the dialog box. If NULL, then the library uses Insert Object.

lpfnHook
Points to a hook function that processes messages intended for the dialog box. The hook function must return zero to pass a message that it didn't process back to the dialog box procedure in the library. The hook function must return a non zero value to prevent the library's dialog box procedure from processing a message it has already processed.

lCustData
Specifies application-defined data that the library passes to the hook function pointed to by the *lpfnHook* member. The library passes a pointer to the OLEUIINSERTOBJECT structure in the *lParam* parameter of the WM_INITDIALOG message; this pointer can be used to retrieve the *lCustData* member.

hInstance
Identifies an instance that contains a dialog box template specified by the *lpTemplateName* member.

lpszTemplate
Points to a null-terminated string that specifies the name of the resource file for the dialog box template that is to be substituted for the library's Insert Object dialog box template.

hResource
A customized template handle.

clsid
CLSID for class of the object to be inserted. Filled on output.

lpszFile

Points to the name of the file linked to or insert. Filled on output.

cchFile

The size of *lpszFile* buffer; will not exceed OLEUI_CCHPATHMAX.

cClsidExclude

The number of CLSIDs included in the *lpClsidExclude* list. Filled on input.

lpClsidExclude

A list of CLSIDs to exclude from listing.

iid

The identifier of the requested interface. If **OleUIInsertObject** creates the object, then it will return a pointer to this interface. This parameter is ignored if **OleUIInsertObject** does not create the object.

oleRender

The rendering option. If **OleUIInsertObject** creates the object, then it selects the rendering option when it creates the object. This parameter is ignored if **OleUIInsertObject** does not create the object.

lpFormatEtc

The desired format. If **OleUIInsertObject** creates the object, then it selects the format when it creates the object. This parameter is ignored if **OleUIInsertObject** does not create the object.

lpIOleClientSite

Points to the client site to be used for the object. This parameter is ignored if **OleUIInsertObject** does not create the object.

lpIStorage

Points to the storage to be used for the object This parameter is ignored if **OleUIInsertObject** does not create the object.

ppvObj

Points to the location where the pointer to the object is returned. This parameter is ignored if **OleUIInsertObject** does not create the object.

sc

The result of creation calls. This parameter is ignored if **OleUIInsertObject** does not create the object.

hMetaPict

MetafilePict structure containing the iconic aspect, if it wasn't placed in the object's cache.

See Also

OleUIInsertObject

OLEUILINKPROPS

This structure is used to initialize the Link tab of the Object Properties dialog box.   A reference to it is passed in as part of the OLEUIOBJECTPROPS structure to the **OleUIObjectProperties** function. This tab shows the location, update status, and update time for a link. It allows the user to change the source of the link, toggle its update status between automatic and manual update, open the source, force an update of the link, or break the link (convert it to a static picture).

```
// These IN fields are standard across all OLEUI property pages.

typedef struct tagOLEUILINKPROPSW
{
// These IN fields are standard across all OLEUI property pages.
     DWORD  cbStruct;
     DWORD  dwFlags;
     DWORD  dwReserved1[2];
     LPFNOLEUIHOOK  lpfnHook;
     LPARAM  lCustData;
     DWORD  dwReserved2[3];

     struct tagOLEUIOBJECTPROPSW* lpOP;
} OLEUILINKPROPSW, *POLEUILINKPROPSW, FAR* LPOLEUILINKPROPSW;
```

cbStruct
  Size of the structure in bytes.

dwFlags
  Contains in-out flags specific to the links page.

dwReserved1[2];
  Reserved for future use.

lpfnHook
  Hook callback (not used in this dialog).

lCustData
  Custom data to pass to hook (not used in this dialog).

dwReserved2[3]
  Reserved for future use.

struct tagOLEUIOBJECTPROPS* *lpOP;*
  Used internally.

See Also

OleUIObjectProperties, **OLEUIOBJECTPROPS**

OLEUIOBJECTPROPS

This structure is used to initialize the standard Object Properties dialog box. It contains references to interfaces used to gather information about the embedding or link, references to three structures that are used to initialize the default tabs – General (OLEUIGNRLPROPS), View (OLEUIVIEWPROPS), and Link (OLEUILINKPROPS), if appropriate – and a standard property-sheet extensibility interface that allows the caller to add additional custom property sheets to the dialog.

```
typedef struct tagOLEUIOBJECTPROPS
{

// These IN fields are standard across all OLEUI property sheets.

     DWORD cbStruct;
     DWORD dwFlags;

// Standard PROPSHEETHEADER used for extensibility
     LPPROPSHEETHEADER       lpPS;

// Data which allows manipulation of the object
     DWORD dwObject;
     LPOLEUIOBJINFO          lpObjInfo;

// Data which allows manipulation of the link
     DWORD dwLink;
     LPOLEUILINKINFO         lpLinkInfo;

// Data specfic to each page
     LPOLEUIGNRLPROPS        lpGP;
     LPOLEUIVIEWPROPS        lpVP;
     LPOLEUILINKPROPS        lpLP;

} OLEUIOBJECTPROPS, *POLEUIOBJECTPROPS, FAR* LPOLEUIOBJECTPROPS;
```

cbStruct
Size of the structure in bytes.

dwFlags
These are the in-out: global flags for the property sheet.

OPF_OBJECTISLINK
The object is a link object and therefore has a link property page.

OPF_NOFILLDEFAULT
Do not fill in default values for the object.

OPF_SHOWHELP
Specifies that the dialog box will display a Help button.

OPF_DISABLECONVERT
Disable the Convert button on the general property page.

lpPS
[in]The standard property sheet header (PROPSHEETHEADER), used for extensibility.

//Data which allows manipulation of the object
dwObject
[in] identifier for the object

lpObjInfo
[in] interface to manipulate object

//Data which allows manipulation of the link

dwLink;

[in] A container-defined unique 32-bit identifier for a single link. Containers can use the pointer to the link's container site for this value.

lpLinkInfo

[in] interface to manipulate link

// Data specific to each page

lpGP

[in] General page data.

lpVP

[in] View page data

lpLP

[in] Link page data.

See Also

OleUIObjectProperties, OLEUIGNRLPROPS, **OLEUIVIEWPROPS**, **OLEUILINKPROPS**

OLEUIPASTEENTRY

This structure is an array of OLEUIPASTEENTRY entries specified in the OLEUIPASTESPECIAL structure for the Paste Special dialog box. Each entry includes a FORMATETC which specifies the formats that are acceptable, a string that is to represent the format in the dialog box's listbox, a string to customize the result text of the dialog, and a set of flags from the OLEUIPASTEFLAG enumeration. The flags indicate if the entry is valid for pasting only, linking only or both pasting and linking. If the entry is valid for linking, the flags indicate which link types are acceptable by OR'ing together the appropriate OLEUIPASTE_LINKTYPE<#> values. These values correspond to the array of link types as follows:

OLEUIPASTE_LINKTYPE1=arrLinkTypes[0]
OLEUIPASTE_LINKTYPE2=arrLinkTypes[1]
OLEUIPASTE_LINKTYPE3=arrLinkTypes[2]
OLEUIPASTE_LINKTYPE4=arrLinkTypes[3]
OLEUIPASTE_LINKTYPE5=arrLinkTypes[4]
OLEUIPASTE_LINKTYPE6=arrLinkTypes[5]
OLEUIPASTE_LINKTYPE7=arrLinkTypes[6]
OLEUIPASTE_LINKTYPE8=arrLinkTypes[7]

arrLinkTypes[] is an array of registered clipboard formats for linking. A maximum of eight link types are allowed.

```
typedef struct tagOLEUIPASTEENTRY
{
     FORMATETC   fmtetc;
     LPCSTR  lpstrFormatName;
     LPCSTR  lpstrResultText;
     DWORD  dwFlags;
     DWORD  dwScratchSpace;
} OLEUIPASTEENTRY, *POLEUIPASTEENTRY, FAR *LPOLEUIPASTEENTRY;
```

Members

fmtetc
The format that is acceptable. The Paste Special dialog checks if this format is offered by the object on the clipboard and if so, offers it for selection to the user.

lpstrFormatName
The string that represents the format to the user. Any %s in this string is replaced by the *FullUserTypeName* of the object on the clipboard and the resulting string is placed in the list box of the dialog. Only one %s is allowed. The presence or absence of %s specifies whether the result-text is to indicate that data is being pasted or that an object that can be activated by an application is being pasted. If %s is present, the resulting text says that an object is being pasted. Otherwise it says that data is being pasted.

lpstrResultText
The string used to customize the resulting text of the dialog when the user selects the format corresponding to this entry. Any %s in this string is replaced by the application name or *FullUserTypeName* of the object on the clipboard. Only one %s is allowed.

dwFlags
The values from OLEUIPASTEFLAG enumeration.

dScratchSpace
The scratch space available to routines that loop through an IEnumFORMATETC* to mark if the *PasteEntry* format is available. This field CAN be left uninitialized.

See Also

OLEUIPASTEFLAG, OleUIPasteSpecial, OLEUIPASTESPECIAL

OLEUIPASTESPECIAL

The OLEUIPASTESPECIAL structure contains information that the OLE 2.01 User Interface Library uses to initialize the Paste Special dialog box, as well as space for the library to return information when the dialog is dismissed.

```
typedef struct tagOLEUIPASTESPECIAL
{
// These IN fields are standard across all OLEUI dialog functions.     DWORD
cbStruct;
     DWORD  dwFlags;
     HWND  hWndOwner;
     LPCSTR  lpszCaption;
     LPFNOLEUIHOOK  lpfnHook;
     LPARAM  lCustData;
     HINSTANCE  hInstance;
     LPCSTR  lpszTemplate;
     HRSRC  hResource;

// Specifics for OLEUIPASTESPECIAL.
     LPDATAOBJECT  lpSrcDataObj;
     LPOLEUIPASTEENTRY  arrPasteEntries;
     int  cPasteEntries;
     UINT FAR *  arrLinkTypes;
     int  cLinkTypes;
     UINT  cClsidExclude;
     LPCLSID  lpClsidExclude;
     int  nSelectedIndex;
     BOOL    fLink;
     HGLOBAL       hMetaPict;
     SIZEL  sizel;
} OLEUIPASTESPECIAL, *POLEUIPASTESPECIAL, FAR *LPOLEUIPASTESPECIAL;
```

Members

cbStruct
The size of the structure, in bytes. This field must be filled on input.

dwFlags
On input, *dwFlags* specifies the initialization and creation flags. On exit, it specifies the user's choices. It may be a combination of the following flags:

PSF_SHOWHELP
Specifies that the dialog will display a Help button.

PSF_SELECTPASTE
Selects the Paste radio button at dialog startup. This is the default, if PSF_SELECTPASTE or PSF_SELECTPASTELINK are not specified. Also, it specifies the state of the button on dialog termination. IN/OUT flag.

PSF_SELECTPASTELINK
Selects the PasteLink radio button at dialog startup. Also, specifies the state of the button on dialog termination. IN/OUT flag.

PSF_CHECKDISPLAYASICON
Specifies whether the Display As Icon radio button was checked on dialog termination. OUT flag.

PSF_DISABLEDISPLAYASICON
Specifies that the Display As Icon check box will be disabled on initialization.

HIDECHANGEICON

Used to disable the change-icon button in the dialog, which is available to users when they're pasting an OLE object by default. See STAYONCLIPBOARDCHANGE otherwise.

STAYONCLIPBOARDCHANGE

Used to tell the dialog to stay up if the clipboard changes while the dialog is up. If the user switches to another application and copies or cuts something, the dialog will, by default, perform a cancel operation, which will remove the dialog since the options it's in the middle of presenting to the user are no longer up-to-date with respect to what's really on the clipboard.

NOREFRESHDATAOBJECT

Used in conjunction with STAYONCLIPBOARDCHANGE (it doesn't do anything otherwise). If the clipboard changes while the dialog box is up and STAYONCLIPBOARDCHANGE is specified, then NOREFRESHDATAOBJECT indicates that the dialog should NOT refresh the contents of the dialog to reflect the new contents of the clipboard. This is useful if the application is using the paste-special dialog on an IDataObject besides the one on the clipboard, for example, as part of right-mouse drag/drop operation.

hWndOwner

Identifies the window that owns the dialog box. It should not be NULL.

lpszCaption

Points to a string to be used as the title of the dialog box. If NULL, then the library uses Paste Special.

lpfnHook

Points to a hook function that processes messages intended for the dialog box. The hook function must return zero to pass a message that it didn't process back to the dialog box procedure in the library. The hook function must return a non zero value to prevent the library's dialog box procedure from processing a message it has already processed.

lCustData

Specifies application-defined data that the library passes to the hook function pointed to by the *lpfnHook* member. The library passes a pointer to the OLEUIPASTESPECIAL structure in the *lParam* parameter of the WM_INITDIALOG message; this pointer can be used to retrieve the *lCustData* member.

hInstance

Identifies a instance that contains a dialog box template specified by the *lpTemplateName* member.

lpszTemplate

Points to a null-terminated string that specifies the name of the resource file for the dialog box template that is to be substituted for the library's Paste Special dialog box template.

hResource

Customized template handle.

lpsrcDataObj

The source IDataObject* (on the clipboard) for data to paste. This field is filled on input. If *lpSrcDataObj* is NULL when **OleUIPasteSpecial** is called, then **OleUIPasteSpecial** will attempt to retrieve a pointer to an IDataObject from the clipboard. If **OleUIPasteSpecial** succeeds, it is the caller's responsibility to free the IDataObject returned in *lpSrcDataObj*.

arrPasteEntries

The OLEUIPASTEENTRY array which specifies acceptable formats. This field is filled on input.

cPasteEntries

The number of OLEUIPASTEENTRY array entries. This field is filled on input.

arrLinkTypes

A list of link types that are acceptable. Link types are referred to using OLEUIPASTEFLAG in arrPasteEntries. This field is filled on input.

cLinkTypes

The number of link types. This field is filled on input.

cClsidExclude
The number of CLSIDs in *lpClsidExclude*. This field is filled on input.

lpClsidExclude
An array of CLSIDs to exclude from the list of available server objects for a Paste operation. Note that this does not affect Paste Link. An application can prevent embedding into itself by listing its own CLSID in this list. This field is filled on input.

nSelectedIndex
Index of *arrPasteEntries*[] that the user selected. This field is filled on output.

fLink
Indicates whether Paste or Paste Link was selected by the user. This field is filled on output.

hMetaPict
The handle to the Metafile containing the icon and icon title selected by the user. This field is filled on output.

sizel
The size of object as displayed in its source, if the display aspect chosen by the user matches the aspect displayed in the source. If the user chooses a different aspect, then *sizel.cx* and *sizel.cy* are both set to zero. The size of the object as it is displayed in the source is retrieved from the ObjectDescriptor if *fLink* is FALSE and from the LinkSrcDescriptor if *fLink* is TRUE. This field is filled on output.

See Also

OleUIPasteSpecial, **OLEUIPASTEENTRY**, **OLEUIPASTEFLAG**

OLEUIVIEWPROPS

This structure is used to initialize the View tab of the Object properties dialog box. A reference to it is passed in as part of the **OLEUIOBJECTPROPS** structure to the **OleUIObjectProperties** function. This tab allows the user to toggle between "content" and "iconic" views of the object, and change its scaling within the container. It also allows the user to tunnel to the change icon dialog when the object is being displayed iconically.

```
// These IN fields are standard across all OLEUI property pages.

typedef struct tagOLEUIVIEWPROPSA
{
// These IN fields are standard across all OLEUI property pages.
     DWORD  cbStruct;
     DWORD  dwFlags;
     DWORD  dwReserved1[2];
     LPFNOLEUIHOOK  lpfnHook;
     LPARAM  lCustData;
     DWORD  dwReserved2[3];

struct tagOLEUIOBJECTPROPS* lpOP;

int   nScaleMin;
int   nScaleMax;

} OLEUIVIEWPROPSA, *POLEUIVIEWPROPSA, FAR* LPOLEUIVIEWPROPSA;
```

Members

cbStruct
Size of the structure in bytes.

dwFlags
IN-OUT: flags specific to view page

VPF_SELECTRELATIVE
[in] Relative to origin.

VPF_DISABLERELATIVE
[in] Disable relative to origin.

VPF_DISABLESCALE
[in] Disable scale option.

dwReserved1[2]
Reserved for future use.

lpfnHook
Hook callback (not used in this dialog).

lCustDataa
Custom data to pass to hook (not used in this dialog).

dwReserved2[3];
Reserved for future use.

struct tagOLEUIOBJECTPROPS* *lpOP*;
Used internally.

nScaleMin
Minimum value for the scale range.

nScaleMax

Maximum value for the scale range.

See Also

OleUIObjectProperties, **OLEUIOBJECTPROPS**

OLEVERB

The **OLEVERB** structure is used in the **IEnumOLEVERB** interface to return information describing a verb for an object. Defined in the **IEnumOLEVERB** interface (*eoverb.idl*).

```
typedef struct tagOLEVERB
{
     LONG lVerb;
     LPWSTR lpszVerbName;
     DWORD fuFlags;
     DWORD grfAttribs;
} OLEVERB, * LPOLEVERB;
```

Members

lVerb
Specifies the integer ID associated with this verb.

lpszVerbName
Points to a string that contains the verb's name.

fuFlags
grfAttribs
Specifies some combination of the verb attributes in the **OLEVERBATTRIB** enumeration.

See Also

IEnumOLEVERB, **IOleObject**

RemSNB

The **RemSNB** structure is used for marshaling the **SNB** data type.

Defined in the **IStorage** interface (*storag.idl*).

```
typedef struct tagRemSNB {
     unsigned long ulCntStr;
     unsigned long ulCntChar;
     [size_is(ulCntChar)] wchar_t rgString[];
} RemSNB;
typedef [transmit_as(RemSNB)] wchar_t **SNB;
```

Members

ulCntStr
Specifies the number of strings in the *rgString* buffer.

ulCntChar
Specifies the size in bytes of the *rgString* buffer.

rgString
Points to an array of bytes containing the stream name strings from the SNB.

See Also

IStorage

SNB

A string name block (SNB) is a pointer to an array of pointers to strings, that ends in a NULL pointer. String name blocks are used the **IStorage** interface and API calls that open storage objects. The strings point to contained storage objects or streams that are to be excluded in the open calls.

```
typedef OLESTR **SNB
```

Comments

The SNB should be created by allocating a contiguous block of memory in which the pointers to strings are followed by a NULL pointer, which is then followed by the actual strings.

The marshalling of a string name block is based on the assumption that the SNB passed in was created this way. Although it could be stored in other ways, the SNB created in this manner has the advantage of requiring only one allocation operation and one freeing of memory for all the strings.

See Also

IStorage

STATDATA

The **STATDATA** structure is the data structure used to specify each advisory connection. It is used for enumerating current advisory connections. It holds data returned by the **IEnumSTATDATA** enumerator. This enumerator interface is returned by **IDataObject:DAdvise**. Each advisory connection is specified by a unique **STATDATA** structure.

Defined in *com.h*.

```
typedef struct tagSTATDATA
{
      FORMATETC formatetc;
      DWORD grfAdvf;
      IAdviseSink* pAdvSink;
      DWORD dwConnection;
} STATDATA;
```

Members

formatetc

Specifies the **FORMATETC** structure for the data of interest to the advise sink. The advise sink receives notification of changes to the data specified by this **FORMATETC** structure.

grfAdvf

Specifies the **ADVF** enumeration value that determines when the advisory sink is notified of changes in the data.

pAdvSink

Specifies the pointer for the **IAdviseSink** interface that will receive change notifications.

dwConnection

Specifies the token that uniquely identifies the advisory connection. This token is returned by the method that sets up the advisory connection.

See Also

IEnumSTATDATA

STATSTG

The **STATSTG** structure contains statistical information about an open storage, stream, or byte array object. This structure is used in the **IEnumSTATSTG**, **ILockBytes**, **IStorage**, and **IStream** interfaces.

Defined in the **IStream** interface (*stream.idl*).

```
typedef struct tagSTATSTG
{
     LPWSTR pwcsName;
     DWORD type;
     ULARGE_INTEGER cbSize;
     FILETIME mtime;
     FILETIME ctime;
     FILETIME atime;
     DWORD grfMode;
     DWORD grfLocksSupported;
     CLSID clsid;
     DWORD grfStateBits;
     DWORD dwStgFmt;
} STATSTG;
```

Members

pwcsName
Points to a NULL-terminated string containing the name. Space for this string is allocated by the method called and freed by the caller. You can specify not to return this member by specifying the STATFLAG_NONAME value when you call a method that returns a **STATSTG** structure.

type
Indicates the type of storage object. This is one of the values from the **STGTY** enumeration.

cbSize
Specifies the size in bytes of the stream or byte array. Undefined for storage objects; zero is returned for storage objects.

mtime
Indicates the last modification time for this storage, stream, or byte array.

ctime
Indicates the creation time for this storage, stream, or byte array.

atime
Indicates the last access time for this storage, stream or byte array.

grfMode
Indicates the access mode specified when the object was opened. This member is only valid in calls to **Stat** methods.

grfLocksSupported
Indicates the types of region locking supported by the stream or byte array. See the **LOCKTYPES** enumeration for the values available. This member is not used for storage objects.

clsid
Indicates the class identifier for the storage object; set to CLSID_NULL for new storage objects. This member is not used for streams or byte arrays.

grfStateBits
Indicates the current state bits of the storage object, that is, the value most recently set by the **IStorage::SetStateBits** method. This member is not valid for streams or byte arrays.

dwStgFmt
Indicates the format of the storage object. This is one of the values from the **STGFMT** enumeration.

See Also

[IStorage::SetElementTimes]

STGMEDIUM

The **STGMEDIUM** structure is a generalized global memory handle used for data transfer operations by the **IAdviseSink**, **IDataObject**, and **IOleCache** interfaces.

Defined in the **IAdviseSink** interface (*advsnk.idl*).

```
typedef struct tagSTGMEDIUM
{
     DWORD tymed;
     [switch_type(DWORD), switch_is((DWORD) tymed)]
     union {
           [case(TYMED_GDI)] HBITMAP hBitmap;
           [case(TYMED_MFPICT)] HMETAFILEPICT hMetafilePict;
           [case(TYMED_ENHMF)] HENHMETAFILE hEnhMetaFile;
           [case(TYMED_HGLOBAL)] HGLOBAL hGlobal;
           [case(TYMED_FILE)] LPWSTR lpszFileName;
           [case(TYMED_ISTREAM)] IStream *pstm;
           [case(TYMED_ISTORAGE)] IStorage *pstg;
           [default] ;
     };
     [unique] IUnknown *pUnkForRelease;
}STGMEDIUM;
typedef STGMEDIUM *LPSTGMEDIUM;
```

Members

tymed
Indicates the type of storage medium. The marshaling and unmarshaling routines use this value to determine which union member was used. This value must be one of the elements of the **TYMED** enumeration.

union member
Specifies a handle, string, or interface pointer that the receiving process can use to access the data being transferred. If *tymed* is TYMED_NULL, the union member is undefined; otherwise, it is one of the following:

hBitmap
Specifies a bitmap handle. The *tymed* member is TYMED_GDI.

hMetafilePict
Specifies a metafile handle. The *tymed* member is TYMED_MFPICT.

hEnhMetaFile
Specifies an enhanced metafile handle. The *tymed* member is TYMED_ENHMF.

hGlobal
Specifies a global memory handle. The *tymed* member is TYMED_HGLOBAL.

lpszFileName
Specifies the path of a disk file that contains the data. The *tymed* member is TYMED_FILE.

pstm
Specifies an **IStream** instance. The *tymed* member is TYMED_ISTREAM.

pstg
Specifies an **IStorage** instance. The *tymed* member is TYMED_ISTORAGE.

pUnkForRelease
Points to an interface instance that allows the sending process to control the way the storage is released when the receiving process calls the **ReleaseStgMedium** API function. If *pUnkForRelease* is NULL, **ReleaseStgMedium** uses default procedures to release the storage; otherwise,

ReleaseStgMedium uses the specified **IUnknown** interface.

See Also

FORMATETC, **IAdviseSink**, **IDataObject**, **IOleCache**, **ReleaseStgMedium**

Microsoft Win32 Developer’s Reference

You have requested information from the **Microsoft Win32 Developer’s Reference**. One or more of these help files is not available on your system.