## OLE 2.01 User Interface Library
**Topics**

Data Transfer
Debug
Icon Support
Library Management
Memory Management
Message Filter
Miscellaneous
Monikers and Linking
Object Feedback
Registration Database
Summary Info and Properties
Storage
Transformations

## Summary Info and Properties

OleStdInitSummaryInfo
OleStdFreeSummaryInfo
OleStdClearSummaryInfo
OleStdReadSummaryInfo
OleStdWriteSummaryInfo
OleStdGetSecurityProperty
OleStdSetSecurityProperty
OleStdGetStringProperty
OleStdSetStringProperty
OleStdGetStringZProperty
OleStdGetDocProperty
OleStdSetDocProperty
OleStdGetThumbNailProperty
OleStdSetThumbNailProperty
OleStdGetDateProperty
OleStdSetDateProperty

**Object Feedback**

Inplace Hatch Border Support
OleUIDrawHandles
OleUIDrawShading
OleUIShowObject

## Inplace Support

CreateHatchWindow
GetHatchRect
GetHatchWidth
RegisterHatchWindowClass
HatchWndProc
SetHatchRect
SetHatchWindowSize

## Data Transfer

Device Contexts
ObjectDescriptor
OleStdEnumFmtEtc_Create
OleStdEnumFmtEtc_Destroy
OleStdGetData
OleStdGetDropEffect
OleStdGetLinkSourceData
OleStdGetMetafilePictFromOleObject
OleStdGetOleObjectData
OleStdGetPriorityClipboardFormat
OleStdIsDuplicateFormat
OleStdQueryFormatMedium
OleStdGetObjectDescriptorFromOleObject
OleStdQueryLinkSourceData

## OleStdMsgFilter

OleStdMsgFilter_Create
OleStdMsgFilter_EnableBusyDialog
OleStdMsgFilter_EnableNotRespondingDialog
OleStdMsgFile_SetHandleInComingCallbackProc
OleStdMsgFilter_GetInComingStatus
OleStdMsgFilter_SetInComingStatus
OleStdMsgFilter_SetParentWindow
OleStdMsgFilter

## Debugging Macros

OLEDBG_BEGIN1
OLEDBG_BEGIN2
OLEDBG_BEGIN3
OLEDBG_BEGIN4
OLEDBG_BEGIN
OLEDBG_END1
OLEDBG_END2
OLEDBG_END3
OLEDBG_END4
OLEDBG_END
OLEDBGDATA_MAIN
OLEDBGDATA
OleDbgOut1
OleDbgOut2
OleDbgOut3
OleDbgOut4
OleDbgOutHResult
OleDbgOutNoPrefix1
OleDbgOutNoPrefix2
OleDbgOutNoPrefix3
OleDbgOutNoPrefix4
OleDbgOutNoPrefix
OleDbgOut
OleDbgOutRect1
OleDbgOutRect2
OleDbgOutRect3
OleDbgOutRect4
OleDbgOutRect
OleDbgOutRefCnt1
OleDbgOutRefCnt2
OleDbgOutRefCnt3
OleDbgOutRefCnt4
OleDbgOutRefCnt
OleDbgOutScode

## Monikers and Linking

OleStdCreateTempFileMoniker
OleStdGetFirstMoniker
OleStdGetItemToken
OleStdGetLenFilePrefixOfMoniker
OleStdNoteFileChangeTime
OleStdNoteObjectChangeTime
OleStdRegisterAsRunning
OleStdRevokeAsRunning
OleStdMkParseDisplayName

## Storage

OleStdCommitStorage
OleStdCreateChildStorage
OleStdCreateRootStorage
OleStdCreateStorageOnHGlobal
OleStdCreateTempStorage
OleStdOpenChildStorage
OleStdOpenRootStorage
OpenOrCreateRootStorage

**ObjectDescriptor**
OleStdFillObjectDescriptorFromData
OleStdGetObjectDescriptorDataFromOleObject
OleStdGetObjectDescriptorData
OleStdQueryObjectDescriptorData

## Memory Management

OleStdCopyString
OleStdFree
OleStdFreeString
OleStdGetSize
OleStdMalloc
OleStdRealloc

## Device Contexts

OleStdCreateDC
OleStdCreateIC
OleStdCreateTargetDevice
OleStdDeleteTargetDevice
ResetOrigDC
SetDCToAnisotropic
SetDCToDrawInHimetricRect

## Icon Support

GetIconOfClass
GetIconOfFile
HIconFromClass
Icon Metafile Format
OleStdCopyMetafilePict
OleStdIconLabelTextOut
OleStdSetIconInCache
OleUIMetafilePictExtractIcon
OleUIMetafilePictExtractIconSource
OleUIMetafilePictExtractLabel
OleUIMetafilePictFromIconAndLabel
OleUIMetafilePictIconDraw
OleUIMetafilePictIconFree

## Library Management

OleUIInitialize
OleUIUnInitialize
OleUILockLibrary
OleUICanUnloadNow

## Registration Database

FServerFromClass
GetAssociatedExecutable
OleStdGetAuxUserType
OleStdGetDefaultFileFormatOfClass
OleStdGetMiscStatusOfClass
OleStdGetTreatAsFmtUserType
OleStdGetUserTypeOfClass
UClassFromDescription
UDescriptionFromClass

## Transformations

XformHeightInHimetricToPixels
XformHeightInPixelsToHimetric
XformWidthInHimetricToPixels
XformWidthInPixelsToHimetric

## Miscellaneous

Browse
ChopText
ErrorWithFile
GetTaskInfo
HourGlassOff
HourGlassOn
OleStdCreateDbAlloc
OleStdCheckVtbl
OleStdDoConvert
OleStdDoTreatAs
OleStdInitVtbl
OleStdIsOleLink
OleStdMarkPasteEntryList
OleStdNullMethod
OleStdQueryInterface
OleStdSetupAdvises
OleStdSwitchDisplayAspect
OleStdVerifyRelease
OpenFileError
ParseCmdLine
ReplaceCharWithNull

## GetTaskInfo

**BOOL GetTaskInfo(***hWnd, htask, lplpszTaskName, lplpszWindowName, lphWnd***)**

**HWND** *hWnd*; /* hWnd of calling window */
**HTASK** *htask*; /* hTask to retrieve information about */
**LPSTR FAR*** *lplpszTaskName*; /* pointer to location to return name of hTask */
**LPSTR FAR*** *lplpszWindowName*; /* pointer to location to return top-level window title */
**HWND FAR*** *lphWnd*; /* pointer to location to return first top-level window handle */

GetTaskInfo returns information about the specified task and places the module name, window name and top-level HWND for the task in the specified pointers.

| **Parameter** | **Description** |
|---|---|
| *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 |

**Returns**
TRUE if top-level HWND for specified task is found, FALSE otherwise.

**Comments**
The two string pointers allocated in this routine are the responsibility of the CALLER to de-allocate.

## OleUIMetafilePictIconFree

**STDAPI_(void) OleUIMetafilePictIconFree(HGLOBAL *hMetaPict*)**

**HGLOBAL** *hMetaPict*; /* handle of metafilepict to free */

OleUIMetafilePictIconFree deletes the metafile contained in a METAFILEPICT structure and frees the memory for the structure itself.

| **Parameter** | **Description** |
|---|---|
| *hMetaPict* | HGLOBAL metafilepict structure created in OleUIMetafilePictFromIconAndLabel |

**Returns**
None

**See Also**
OleUIMetafilePictIconDraw, GetIconOfFile, GetIconOfClass

## OleUIMetafilePictIconDraw

**STDAPI_(BOOL) OleUIMetafilePictIconDraw(***hDC, pRect, hMetaPict, fIconOnly***)**

**HDC** *hDC;* /* Device context to draw the metafilepict into */
**LPRECT** *pRect;* /* Bounding rectangle */
**HGLOBAL** *hMetaPict;* /* MetafilePict to draw */
**BOOL** *fIconOnly;* /* TRUE to draw the icon only (no label); FALSE otherwise */

Draws the metafile from OleUIMetafilePictFromIconAndLabel, either with the label or without.

| **Parameter** | **Description** |
|---|---|
| *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. |

**Returns**
TRUE if the function is successful, FALSE if the given metafilepict is invalid.

**See Also**
OleUIMetafilePictFromIconAndLabel, GetIconOfFile, GetIconOfClass

## OleUIMetafilePictExtractLabel

**STDAPI_(UINT) OleUIMetafilePictExtractLabel(***hMetaPict, lpszLabel, cchLabel, lpWrapIndex***)**

```
HGLOBAL hMetaPict;      /* Metafile to extract label from */
LPSTR lpszLabel;        /* Address to return label */
UINT cchLabel;          /* Length of buffer pointed to by lpszLabel */
LPDWORD lpWrapIndex;    /* Pointer to return the index of the first character in last line */
```

OleUIMetafilePictExtractLabel retrieves the label string from metafile representation of an icon.

| Parameter | Description |
|---|---|
| *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. |

**Returns**
Number of characters copied.

**See Also**
OleUIMetafilePictFromIconAndLabel, GetIconOfFile, GetIconOfClass

## OleUIMetafilePictExtractIcon

**STDAPI_(HICON) OleUIMetafilePictExtractIcon(***hMetaPict***)**

**HGLOBAL** *hMetaPict;* /* Metafile to extract icon from */

Retrieves the icon from metafile into which DrawIcon was done before.

| **Parameter** | **Description** |
|---|---|
| *hMetaPict* | HGLOBAL to the METAFILEPICT containing the metafile. |

**Returns**
HICON of icon recreated from the data in the metafile.

**See Also**
OleUIMetafilePictFromIconAndLabel, GetIconOfFile, GetIconOfClass

## OleUIMetafilePictExtractIconSource

**STDAPI_(BOOL) OleUIMetafilePictExtractIconSource(***hMetaPict, lpszSource, piIcon***)**

**HGLOBAL** *hMetaPict;* /* Metafile to extract icon source from */
**LPSTR** *lpszSource;* /* pointer to return icon source */
**UINT FAR ***piIcon;* /* pointer to return icon index within source */

Retrieves the filename and index of the icon source from a metafile created with OleUIMetafilePictFromIconAndLabel.

| **Parameter** | **Description** |
|---|---|
| *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 |

**Returns**
TRUE if the records were found, FALSE otherwise.

**See Also**
OleUIMetafilePictFromIconAndLabel, GetIconOfFile, GetIconOfClass

## 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
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.

## GetIconOfFile

**STDAPI_(HGLOBAL) GetIconOfFile(HINSTANCE hInst, LPSTR lpszPath, BOOL fUseFileAsLabel)**

GetIconOfFile returns a hMetaPict containing an icon and label (filename) for the specified filename.

| **Parameter** | **Description** |
|---|---|
| *hinst* | Current HINSTANCE |
| *lpszPath* | LPSTR path including filename to use |
| *fUseFileAsLabel* | BOOL TRUE if the icon's label is the filename, FALSE if the short user type name should be the label. |

**Returns**

hMetaPict containing the icon and label - if there's no class in reg db for the file in lpszPath, then we use "Document" as the label and the plain Document icon as the icon. If lpszPath is NULL, then we return NULL.

**See Also**

OleUIMetafilePictFromIconAndLabel, GetIconOfClass

## GetAssociatedExecutable

**BOOL FAR PASCAL GetAssociatedExecutable(LPSTR lpszExtension, LPSTR lpszExecutable)**

Finds the executable associated with the provided extension

| Parameter | Description |
|---|---|
| *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. |

**Returns**

TRUE if we found an exe, FALSE if we didn't.

## GetIconOfClass

**STDAPI_(HGLOBAL) GetIconOfClass(HINSTANCE hInst, REFCLSID rclsid, LPSTR lpszLabel, BOOL fUseTypeAsLabel)**

Returns a hMetaPict containing an icon and label (human-readable form of class) for the specified clsid.

| **Parameter** | **Description** |
|---|---|
| *hinst* | Current HInstance |
| *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. |

**Returns**

hMetaPict containing the icon and label - if we don't find the clsid in the reg db then we return NULL.

**See Also**

OleUIMetafilePictFromIconAndLabel, GetIconOfFile

## OleUIMetafilePictFromIconAndLabel

**STDAPI_(HGLOBAL) OleUIMetafilePictFromIconAndLabel(HICON hIcon, LPSTR pszLabel, LPSTR pszSourceFile, UINT iIcon)**

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.

| **Parameter** | **Description** |
|---|---|
| *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. |

**Returns**

Global memory handle containing a METAFILEPICT where the metafile uses the MM_ANISOTROPIC mapping mode. The extents reflect both icon and label.

**See Also**

GetIconOfFile, GetIconOfClass

## OleStdIconLabelTextOut

**STDAPI_(UINT) OleStdIconLabelTextOut(HDC hDC, HFONT hFont, int nXStart, int nYStart, UINT fuOptions, RECT FAR * lpRect, LPSTR lpszString, UINT cchString, int FAR * lpDX)**

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.

| **Parameter** | **Description** |
|---|---|
| *hDC* | device context to draw into; if this is NULL, then we don't output 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 |

**Returns**

Index of beginning of last line (0 if there's only one line of text).

**See Also**

OleUIMetafilePictFromIconAndLabel, GetIconOfFile, GetIconOfClass

## OleStdGetUserTypeOfClass

**STDAPI_(UINT) OleStdGetUserTypeOfClass(REFCLSID rclsid, LPSTR lpszUserType, UINT cch, HKEY hKey)**

Returns the user type (human readable class name) of the specified class.

| **Parameter** | **Description** |
|---|---|
| *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). |

**Returns**

Number of characters in returned string. 0 on error.

**See Also**

OleStdGetAuxUserType

## OleStdGetAuxUserType

**STDAPI_(UINT) OleStdGetAuxUserType(REFCLSID rclsid, WORD wAuxUserType, LPSTR lpszAuxUserType, int cch, HKEY hKey)**

Returns the specified AuxUserType from the registration database.

| **Parameter** | **Description** |
|---|---|
| *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 |

**Returns**

Number of characters in returned string. 0 on error.

**See Also**

OleStdGetUserTypeOfClass

## XformWidthInPixelsToHimetric

**STDAPI_(int) XformWidthInPixelsToHimetric(HDC hDC, int iWidthInPix)**
Converts the specified width value from pixels to logical Himetric units.

| **Parameter** | **Description** |
|---|---|
| *hDC* | HDC providing reference to the pixel mapping. If NULL, a screen DC is used. |
| *iWidthInPix* | int containing the value to convert. |

**Comments**
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:

$$\text{DistInHiMetric} = \frac{\text{HIMETRIC\_PER\_INCH} * \text{DistInPix}}{\text{PIXELS\_PER\_LOGICAL\_IN}}$$

**Returns**
Converted value of iWidthInPix.

**See Also**
XformWidthInHimetricToPixels, XformHeightInHimetricToPixels, XformHeightInPixelsToHimetric

## XformWidthInHimetricToPixels

**STDAPI_(int) XformWidthInHimetricToPixels(HDC hDC, int iWidthInHiMetric)**

Converts the specified width value from logical Himetric units to pixels.

| **Parameter** | **Description** |
|---|---|
| *hDC* | HDC providing reference to the pixel mapping. If NULL, a screen DC is used. |
| *iWidthInHiMetric* | int containing the value to convert. |

**Comments**

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:

$$\text{DistInHiMetric} = \frac{(\text{HIMETRIC\_PER\_INCH} * \text{DistInPix})}{\text{PIXELS\_PER\_LOGICAL\_IN}}$$

**Returns**

Converted value of iWidthInHiMetric.

**See Also**

XformWidthInPixelsToHimetric, XformHeightInHimetricToPixels, XformHeightInPixelsToHimetric

## XformHeightInPixelsToHimetric

**STDAPI_(int) XformHeightInPixelsToHimetric(HDC hDC, int iHeightInPix)**

Converts the specified height value from pixels to logical Himetric units.

| **Parameter** | **Description** |
|---|---|
| *hDC* | HDC providing reference to the pixel mapping. If NULL, a screen DC is used. |
| *iHeightInPixels* | int containing the value to convert. |

**Comments**

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:

$$\text{DistInHiMetric} = \frac{(\text{HIMETRIC\_PER\_INCH} * \text{DistInPix})}{\text{PIXELS\_PER\_LOGICAL\_IN}}$$

**Returns**

Converted value of iHeightInPixels.

**See Also**

XformWidthInHimetricToPixels, XformHeightInHimetricToPixels, XformWidthInPixelsToHimetric

## XformHeightInHimetricToPixels

**STDAPI_(int) XformHeightInHimetricToPixels(HDC hDC, int iHeightInHiMetric)**

Converts the specified height value from logical Himetric units to pixels.

| **Parameter** | **Description** |
|---|---|
| *hDC* | HDC providing reference to the pixel mapping. If NULL, a screen DC is used. |
| *iHeightInHiMetric* | int containing the value to convert. |

**Comments**

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:

$$\text{DistInHiMetric} = \frac{(\text{HIMETRIC\_PER\_INCH} * \text{DistInPix})}{\text{PIXELS\_PER\_LOGICAL\_IN}}$$

**Returns**

Converted value of iHeightInHiMetric.

**See Also**

XformWidthInHimetricToPixels, XformWidthInPixelsToHimetric, XformHeightInPixelsToHimetric

## RegisterHatchWindowClass

**STDAPI_(BOOL) RegisterHatchWindowClass(HINSTANCE hInst)**

Registers the hatch window class.

| **Parameter** | **Description** |
|---|---|
| *hInst* | Process instance |

**Returns**

TRUE if the window class is successfully registered;  FALSE otherwise.

**See Also**

CreateHatchWindow, GetHatchWidth,

## CreateHatchWindow

**STDAPI_(HWND) CreateHatchWindow(HWND hWndParent, HINSTANCE hInst)**

Creates a hatch window.

| **Parameter** | **Description** |
|---|---|
| *hWndParent* | parent of hatch window |
| *hInst* | instance handle |

**Returns**

If successful, CreateHatchWindow returns a pointer to the created hatch window. It returns NULL if the function fails.

**See Also**

RegisterHatchWindowClass, GetHatchWidth,

## GetHatchWidth

**STDAPI_(UINT) GetHatchWidth(HWND hWndHatch)**

Get width of hatch window's hatch border.

| **Parameter** | **Description** |
|---|---|
| *hWndHatch* | hatch window handle |

**Returns**

Width of the window's hatch border.

**See Also**

RegisterHatchWindowClass, CreateHatchWindow, GetHatchRect, SetHatchRect

## GetHatchRect

**STDAPI_(void) GetHatchRect(HWND hWndHatch, LPRECT lprcHatchRect)**

Get a hatch window's hatch rectangle. This is the size of the hatch window if it were not clipped by the ClipRect.

| **Parameter** | **Description** |
|---|---|
| *hWndHatch* | hatch window handle |
| *lprcHatchRect* | pointer to RECT structure to return hatch rect |

**Returns**
None.

**See Also**
RegisterHatchWindowClass, CreateHatchWindow, SetHatchRect

## SetHatchRect

**STDAPI_(void) SetHatchRect(HWND hWndHatch, LPRECT lprcHatchRect)**

Store hatch rect with HatchRect window. This rect is the size of the hatch window if it were not clipped by the ClipRect.

| **Parameter** | **Description** |
|---|---|
| *hWndHatch* | hatch window handle |
| lprcHatchRect | pointer to RECT structure to return hatch rect |

**Returns**
**None.**

**See Also**
RegisterHatchWindowClass, CreateHatchWindow, GetHatchRect, SetHatchWindowSize

## SetHatchWindowSize

**STDAPI_(void) SetHatchWindowSize(HWND hWndHatch, LPRECT lprcIPObjRect, LPRECT lprcClipRect, LPPOINT lpptOffset)**

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).

| **Parameter** | **Description** |
|---|---|
| *hWndHatch* | hatch window handle |
| *lprcIPObjRect* | pointer to RECT structure containing full size of in-place server object window |
| *lprcClipRect* | pointer to RECT structure containing 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) |

**Comment**

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.

**Returns**

None.

**See Also**

RegisterHatchWindowClass, CreateHatchWindow, GetHatchRect,

## HatchWndProc

**LRESULT FAR PASCAL __export HatchWndProc(HWND hWnd, UINT Message, WPARAM wParam, LPARAM lParam)**

HatchWndProc is the window procedure for hatch windows.

**Returns**
The return value is message-dependent.

**See Also**
RegisterHatchWindowClass, CreateHatchWindow, GetHatchRect,

## HourGlassOn

**HCURSOR WINAPI HourGlassOn(void)**

Shows the hourglass cursor returning the last cursor in use.

**Returns**
Cursor in use prior to showing the hourglass.

**See Also**
HourGlassOn

## HourGlassOff
**void WINAPI HourGlassOff(HCURSOR hCur)**

Restores the hourglass cursor to a previous cursor.

| **Parameter** | **Description** |
|---|---|
| *hCur* | HCURSOR as returned from HourGlassOn |

**Returns**
None.

**See Also**
HourGlassOff

## Browse

**BOOL WINAPI Browse(HWND hWndOwner, LPSTR lpszFile, LPSTR lpszInitialDir, UINT cchFile, UINT iFilterString, DWORD dwOfnFlags)**

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 REVIEW TBD. Otherwise that string is loaded from resources and used.

| **Parameter** | **Description** |
|---|---|
| *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. |
| *dwOfnFlags* | DWORD flags to OR with OFN_HIDEREADONLY |

**Returns**

TRUE if the user selected a file and pressed OK. FALSE otherwise, including if the user presses Cancel.

## ReplaceCharWithNull

**int WINAPI ReplaceCharWithNull(LPSTR psz, int ch)**

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.

| **Parameter** | **Description** |
|---|---|
| *psz* | LPSTR to the string to process. |
| *ch* | int character to replace. |

**Returns**

Number of characters replaced. -1 if psz is NULL.

## ErrorWithFile

**int WINAPI ErrorWithFile(HWND hWnd, HINSTANCE hInst, UINT idsErr, LPSTR pszFile, UINT uFlags)**

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.

| **Parameter** | **Description** |
|---|---|
| *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. |

**Returns**

Return value from MessageBox.

## HIconFromClass

**HICON WINAPI HIconFromClass(LPSTR pszClass)**

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.

| **Parameter** | **Description** |
|---|---|
| *pszClass* | LPSTR giving the object class to look up. |

**Returns**

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.

**See Also**

GetIconOfFile, GetIconOfClass

## FServerFromClass

**BOOL WINAPI FServerFromClass(LPSTR pszClass, LPSTR pszEXE, UINT cch)**

Looks up the classname in the registration database and retrieves the name under LocalServer.

| Parameter | Description |
|---|---|
| *pszClass* | LPSTR to the classname to look up. |
| *pszEXE* | LPSTR at which to store the server name |
| *cch* | UINT size of pszEXE |

**Returns**

TRUE if one or more characters were loaded into pszEXE; FALSE otherwise.

## UClassFromDescription

**UINT WINAPI UClassFromDescription(LPSTR psz, LPSTR pszClass, UINT cb)**

Looks up the actual OLE class name in the registration database for the given descriptive name chosen from a listbox.

| Parameter | Description |
|---|---|
| *psz* | LPSTR to the descriptive name. |
| *pszClass* | LPSTR in which to store the class name. |
| *cb* | UINT maximum length of pszClass. |

**Returns**

Number of characters copied to pszClass. 0 on failure.

## UDescriptionFromClass

**UINT WINAPI UDescriptionFromClass(LPSTR pszClass, LPSTR psz, UINT cb)**

Looks up the actual OLE descriptive name name in the registration database for the given class name.

| **Parameter** | **Description** |
|---|---|
| *pszClass* | LPSTR to the class name. |
| *psz* | LPSTR in which to store the descriptive name. |
| *cb* | UINT maximum length of psz. |

**Returns**

Number of characters copied to pszClass.  0 on failure.

## ChopText

**LPSTR WINAPI ChopText(HWND hWnd, int nWidth, LPSTR lpch)**

Parse a string (pathname) and convert it to be within a specified length by chopping the least significant part

| **Parameter** | **Description** |
|---|---|
| *hWnd* | window handle in which the string resides |
| *nWidth* | max width of string in pixels. If nWidth is NULL, then ChopText uses the width of hWnd. |
| *lpch* | pointer to beginning of the string |

**Returns**

pointer to the modified string

## OpenFileError

**void WINAPI OpenFileError(HWND hDlg, UINT nErrCode, LPSTR lpszFile)**

Display message for error returned from OpenFile.

| **Parameter** | **Description** |
|---|---|
| *hDlg* | HWND of the dialog. |
| *nErrCode* | UINT error code returned in OFSTRUCT passed to OpenFile |
| *lpszFile* | LPSTR file name passed to OpenFile |

**Comments**

OpenFileError reports specific errors for "Access Denied", "Sharing Violation", "File/Path Not Found".

**Returns**

None.

## OleStdGetMiscStatusOfClass

**STDAPI_(BOOL) OleStdGetMiscStatusOfClass(REFCLSID rclsid, HKEY hKey, DWORD FAR * lpdwValue)**

Returns the value of the misc status for the given clsid.

| Parameter | Description |
|---|---|
| *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). |
| *lpdword* | pointer to where to return the misc status. |

**Returns**

TRUE on success, FALSE on failure.

## OleStdGetDefaultFileFormatOfClass

**STDAPI_(CLIPFORMAT) OleStdGetDefaultFileFormatOfClass(REFCLSIDrclsid, HKEY hKey)**

Returns the default file format of the specified class. This is entered in REGDB as follows:
CLSID\{...}\DataFormats\DefaultFile = <cfFmt>

| **Parameter** | **Description** |
|---|---|
| *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). |

**Returns**

If successful, then the default file format; otherwise, NULL.

## OleUIInitialize

**STDAPI_(BOOL) OleUIInitialize(HINSTANCE hInstance, HINSTANCE hPrevInst,LPSTR lpszClassIconBox,LPSTR)**

Initializes the OLE2UI library by loading resources and registering the necessary window messages and clipboard formats.

| **Parameter** | **Description** |
|---|---|
| *hInst* | HINSTANCE to use for loading resources. If you statically link ole2ui into your application, then this should be your application's hInstance. |
| *hPrevInst* | HINSTANCE of the application's previous instance. If you statically link to ole2ui, then you should use WinMain's hinstPrevious parameter. |
| *lpszClassIconBox* | LPSTR containing the class name for the IconBox custom control. You should use the symbol SZCLASSICONBOX from UICLASS.H, which is generated by the ole2ui library's makefile. This class name is used to register the IconBox custom control; each application must use a unique name for this class. |
| *lpszClassResImage* | LPSTR containing the class name for the ResultImage custom control. You should use the symbol SZCLASSRESULTIMAGE from UICLASS.H, which is generated by the ole2ui library's makefile. See comments below. |

**Comments**

Applications that statically link with the OLE2UI library must call OleUIInitialize before using any library functions; however, an application that dynamically links with the library should not call OleUIInitialize, because it is called by the DLL's LibMain function.

The *lpszClassIconBox* and *lpszClassResImage* strings are used to register the IconBox and ResultImage custom controls used by the library's dialogs. Since these classes are registered as global classes, these strings must be unique to the DLL or application that is registering the class.

The *hPrevInst* parameter is used to determine if the library should register the custom control classes. If *hPrevInst* is NULL (called by a DLL version or by the first instance of a statically-linked version), then the library registers the classes. If *hPrevInst* is non-NULL (for later instances of a statically-linked version); then the classes are not registered.

**Returns**

TRUE if the library is successfully initialized; FALSE otherwise.

**See Also**

OleUIUninitialize

## OleUIUninitialize

**STDAPI_(BOOL) OleUIUninitialize()**

Uninitializes the OLE2UI library.

**Comments**

This function is called from WEP(); so if your application uses the library as a DLL, then it does not need to call OleUIUninitialize. If your application uses the library as a static linked library, then it must call OleUIUninitialize before the application exits.

**Returns**

TRUE

**See Also**

OleUIInitialize

## OleStdCreateDC

**STDAPI_(HDC) OleStdCreateDC(DVTARGETDEVICE FAR* ptd)**

Creates a DC for the specified target device.

| **Parameter** | **Description** |
|---|---|
| *ptd* | Pointer to target device to create DC for. |

**Returns**

S_OK if successful; NULL if the DC couldn't be created.

**See Also**

OleStdCreateIC

## OleStdCreateIC

**STDAPI_(HDC) OleStdCreateIC(DVTARGETDEVICE FAR* ptd)**

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.

| **Parameter** | **Description** |
|---|---|
| *ptd* | Pointer to target device to create an IC for. |

**Returns**

S_OK if successfull; NULL if the DC couldn't be created.

**See Also**

OleStdCreateDC

## OleStdCreateTargetDevice

**STDAPI_(DVTARGETDEVICE FAR*) OleStdCreateTargetDevice(LPPRINTDLG lpPrintDlg)**

Creates an OLE target device (DVTARGETDEVICE) based on the information in the PRINTDLG structure.

| **Parameter** | **Description** |
|---|---|
| *lpPrintDlg* | Pointer to PRINTDLG structure, containing information to use to create target device. |

**Returns**
Pointer to newly allocated target device, If successful.

**Comments**
Caller must free the returned target device using the current memory allocator (You can use OleStdFree).

**See Also**
OleStdDeleteTargetDevice

## OleStdDeleteTargetDevice

**STDAPI_(BOOL) OleStdDeleteTargetDevice(DVTARGETDEVICE FAR* ptd)**

Uses OleStdFree to free the specified target device.

| **Parameter** | **Description** |
|---|---|
| *ptd* | Pointer to target device to delete. |

**Returns**

TRUE

**See Also**

OleStdCreateTargetDevice

## SetDCToAnisotropic

**STDAPI_(int) SetDCToAnisotropic(HDC hDC, LPRECT lprcPhysical, LPRECT lprcLogical, LPRECT lprcWindowOld, LPRECT lprcViewportOld)**

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:


Origin = P3
X extent = P2x - P3x
Y extent = P2y - P3y

| **Parameter** | **Description** |
|---|---|
| *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 |

**Returns**
The original mapping mode of the DC.

**See Also**
SetDCToDrawInHimetricRect

## SetDCToDrawInHimetricRect

**STDAPI_(int) SetDCToDrawInHimetricRect(HDC hDC, LPRECT lprcPix, LPRECT lprcHiMetric, LPRECT lprcWindowOld, LPRECT lprcViewportOld)**

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:


Origin = P3
X extent = P2x - P3x
Y extent = P2y - P3y

| **Parameter** | **Description** |
|---|---|
| *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 |

**Returns**
The original mapping mode of the DC.

**See Also**
SetDCToAnisotropic

## ResetOrigDC

**STDAPI_(int) ResetOrigDC(HDC hDC, int nMapModeOld, LPRECT lprcWindowOld, LPRECT lprcViewportOld)**

Restores a DC set to draw in himetric from SetDCToDrawInHimetricRect.

| **Parameter** | **Description** |
|---|---|
| *hDC* | HDC to restore |
| *nMapModeOld* | int original mapping mode of hDC |
| *lprcWindowOld* | LPRECT filled in SetDCToDrawInHimetricRect |
| *lprcViewportOld* | LPRECT filled in SetDCToDrawInHimetricRect |

**Returns**
Same as nMapModeOld.

**See Also**
SetDCToDrawInHimetricRect, SetDCToAnisotropic

## OleStdMkParseDisplayName

**STDAPI OleStdMkParseDisplayName(REFCLSID rClsid, LPBC lpbc, LPSTR lpszUserName, ULONG FAR* lpchEaten, LPMONIKER FAR* lplpmk)**

Parses a string into a moniker by calling MkParseDisplayName.

| **Parameter** | **Description** |
|---|---|
| *rClsid* | Original class of link source. CLSID_NULL if class of object is unknown. other parameters the same as MkParseDisplayName API. |
| *lpbc* | Pointer to the binding context in which to accumulate bound objects. |
| *szUserName* | Pointer to the display name to be parsed. |
| *lpchEaten* | On exit, the number of characters of the display name that were successfully parsed. |
| *lplpmk* | Pointer to the resulting moniker. |

**Returns**

NOERROR if string parsed successfully, otherwise OleStdMkParseDisplayName returns the error code returned by MkParseDisplayName.

**Comments**

If *rClsid* refers to an OLE 2.0 class, then OleStdMkParseDisplayName just calls MkParseDisplayName. However, if *rClsid* refers to an OLE 1.0 class, then the class' *ProgID* is retrieved and a string of the form "**!** *lpszUserName*" is created and passed to MkParseDisplayName. If that fails, then MkParseDisplayName is called with *lpszUserName*.

Prefixing *lpszUserName* with "**@***ProgID***!**" forces MkParseDisplayName to assume the file specified by *lpszUserName* is of that class referred to by *ProgID*. Note that this technique only works for OLE1 classes.

## OleUILockLibrary
**STDAPI OleUILockLibrary(BOOL fLock)**

Increments or decrements a lock count which prevents a DLL from being prematurely unloaded or from remaining loaded after it is no longer needed.

| **Parameter** | **Description** |
|---|---|
| *fLock* | TRUE to lock the library; FALSE otherwise. |

**Returns**
NOERROR if the DLL can safely unload; S_FALSE if the DLL should remain loaded.

**Comments**
Only In-Process (INPROC) server DLLs that use the OLE2UI library as a DLL should call **OleUILockLibrary**. In-Process server DLLs that use the OLE2UI library as a LIB should use OleUICanUnloadNow. Servers implemented as executables do not need to use either **OleUILockLibrary** or OleUICanUnloadNow

All OLE 2.x In-Process servers (ie., servers implemented as a DLL) that use the OLE2UI library as a DLL (ie., not a statically linked LIB) must call **OleUILockLibrary**. These server DLLs must call **OleUILockLibrary(TRUE)** on initialization and **OleUILockLibrary(FALSE)** on shutdown to prevent the ole2ui library's **DllCanUnloadNow** from returning NOERROR until the server DLL shuts down.

While the INPROC server DLL is loaded, OLE2UI DLL must remain loaded, which means that ole2ui.dll's **DllCanUnloadNow** must return S_FALSE. The server DLL can be unloaded by a call to CoFreeUnusedLibraries; however the OLE2UI library must remain in use (because an enumerator returned from OleStdEnumFmtEtc_Create still exists). Only after all explicit "OleUILockLibrary" locks and instances of objects created have been released will the OLE2UI's **DllCanUnloadNow** return NOERROR.

**See Also**
OleUICanUnloadNow

## OleUICanUnloadNow
**STDAPI OleUICanUnloadNow()**

Determines if it is safe for a DLL to unload.

**Returns**
NOERROR it is safe for the DLL to unload, S_FALSE if the DLL must stay loaded.

**Comments**
Only In-Process server DLLs that use the OLE2UI library as a LIB should use **OleUICanUnloadNow**. In-Process (INPROC) server DLLs that use the OLE2UI library as a DLL should call OleUILockLibrary. Servers implemented as executables do not need to use either OleUILockLibrary or **OleUICanUnloadNow**.

If you statically link to the OLE2UI library and you implement **DllCanUnloadNow**, then you must call **OleUICanUnloadNow** in your implementation of **DllCanUnloadNow** to determine if the server DLL can be safely unloaded or not.

**OleUICanUnloadNow** returns S_FALSE if there are any existing instance of objects created by the OLE2UI library functions (EnumFORMATETC objects created by OleStdEnumFmtEtc_Create, for example) which should prevent the server DLL from being unloaded.

**See Also**
OleUILockLibrary

## OleStdCheckVtbl

**STDAPI_(BOOL) OleStdCheckVtbl(LPVOID lpVtbl, UINT nSizeOfVtbl, LPSTR lpszIface)**

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.

| **Parameter** | **Description** |
|---|---|
| *lpVtbl* | Pointer to VTBL to check. |
| *nSizeOfVtbl* | Number of methods in lpVtbl. |
| *lpszIface* | String containing interface name to be used in debugging message. |

**Returns**

TRUE if all entries in Vtbl are valid; FALSE otherwise.

**See Also**

OleStdNullMethod

## OleStdNullMethod

**STDMETHODIMP OleStdNullMethod(LPUNKNOWN lpThis)**

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!

**See Also**

OleStdCheckVtbl

## OleStdRegisterAsRunning

**STDAPI_(void) OleStdRegisterAsRunning(LPUNKNOWN lpUnk, LPMONIKER lpmkFull, DWORD FAR* lpdwRegister)**

Registers an object in the Running Object Table by calling GetRunningObjectTable and IRunningObjectTable::Register.

| **Parameter** | **Description** |
|---|---|
| *lpUnk* | Pointer to running object's IUnknown interface. |
| *lpmkFull* | Pointer to moniker that will bind to newly running object. |
| *lpdwRegister* | Pointer to return registration value to be passed later to OleStdRevokeAsRunning |

**Returns**
None.

**See Also**
OleStdRevokeAsRunning

## OleStdRevokeAsRunning

**STDAPI_(void) OleStdRevokeAsRunning(DWORD FAR* lpdwRegister)**

Revokes an object's registration in the Running Object Table by calling GetRunningObjectTable and IRunningObjectTable::Revoke.

| **Parameter** | **Description** |
|---|---|
| *lpdwRegister* | Value returned from OleStdRegisterAsRunning. |

**Returns**
None.

**See Also**
OleStdRegisterAsRunning

## OleStdCreateTempFileMoniker

**STDAPI_(void) OleStdCreateTempFileMoniker(LPSTR lpszPrefixString, UINT FAR* lpuUnique, LPSTR lpszName, LPMONIKER FAR* lplpmk)**

Creates a unique file moniker to be used as the name of an untitled document (eg., Document1). Useful for applications that support linking to unsaved documents. Builds names of the form <Prefix><UniqueNumber>. Checks the Running Object Table to find the first unused name of this form.

| **Parameter** | **Description** |
|---|---|
| *lpszPrefixString* | Prefix to use to create temporary moniker. |
| *lpuUnique* | Unique identifer for moniker. In/Out - the application should keep this identifier to use repeatedly. This |
| *lpszName* | Address of buffer for created moniker (<Prefix><UniqueNumber>). |
| *lplpmk* | Pointer to moniker. |

**Returns**
None.

**Comment**
This function is analogous to the Windows function GetTempFileName.

## OleStdGetFirstMoniker

**STDAPI_(LPMONIKER) OleStdGetFirstMoniker(LPMONIKER lpmk)**

Returns the first piece of a moniker.

| Parameter | Description |
|---|---|
| lpmk | Pointer to moniker to return first piece of. |

**Returns**

Pointer to moniker containing first piece of the specified moniker.

**Comment**

If the given moniker is not a generic composite moniker, then an AddRef'ed pointer to the given moniker is returned.

## OleStdMarkPasteEntryList

**STDAPI_(void) OleStdMarkPasteEntryList(LPDATAOBJECT lpSrcDataObj, LPOLEUIPASTEENTRY lpPriorityList, int cEntries)**

Marks each entry in the PasteEntryList if its format is available from the specified source IDataObject*. The dwScratchSpace field of each PasteEntry is set to TRUE if available, else FALSE.

| **Parameter** | **Description** |
|---|---|
| *lpSrcDataObj* | source IDataObject* pointer |
| *lpPriorityList* | array of PasteEntry structures |
| *cEntries* | count of elements in PasteEntry array |

**Returns**

None.

## OleStdGetPriorityClipboardFormat

**STDAPI_(int) OleStdGetPriorityClipboardFormat(LPDATAOBJECT lpSrcDataObj, LPOLEUIPASTEENTRY lpPriorityList, int cEntries)**

Retrieves the first clipboard format in a list for which data exists in the source IDataObject*.

| **Parameter** | **Description** |
|---|---|
| *lpSrcDataObj* | source IDataObject* pointer |
| *lpPriorityList* | array of PasteEntry structures |
| *cEntries* | count of elements in PasteEntry array |

**Returns**

-1 if no acceptable match is found. Index of first acceptable match in the priority list.

## OleStdIsDuplicateFormat

**STDAPI_(BOOL) OleStdIsDuplicateFormat(LPFORMATETC lpFmtEtc, LPFORMATETC arrFmtEtc, int nFmtEtc)**

Determines if the cfFormat specfied in lpFmtEtc is also in arrFmtEtc.

| **Parameter** | **Description** |
|---|---|
| *lpFmtEtc* | Pointer to FORMATETC |
| *arrFmtEtc* | Array of FORMATETC structures. |
| *nFmtEtc* | Number of FORMATETC structures in arrFmtEtc. |

**Returns**

TRUE if the lpFmtEtc->cfFormat is found in the array of FormatEtc structures; FALSE otherwise.

## OleStdGetDropEffect

**STDAPI_(DWORD) OleStdGetDropEffect( DWORD grfKeyState )**

Converts a keyboard state into a DROPEFFECT.

| **Parameter** | **Description** |
|---|---|
| grfKeyState | Identifies the current state of the modifier keys |

**Returns**

DROPEFFECT value derived from the key state. The following is the standard interpretation:

| | |
|---|---|
| no modifier | Default Drop, NULL is returned |
| CTRL | DROPEFFECT_COPY returned |
| SHIFT | DROPEFFECT_MOVE returned |
| CTRL-SHIFT | DROPEFFECT_LINK returned |

**Comments**

The default drop 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.

## OleStdGetItemToken

**STDAPI_(ULONG) OleStdGetItemToken(LPSTR lpszSrc, LPSTR lpszDst, int nMaxChars)**

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.

| **Parameter** | **Description** |
|---|---|
| *lpszSrc* | Pointer to a source string |
| *lpszDst* | Pointer to destination buffer |
| *nMaxChars* | Number of characters in lpszSrc |

**Returns**

The number of charaters preceding the first token.

## OleStdCreateRootStorage

**STDAPI_(LPSTORAGE) OleStdCreateRootStorage(LPSTR lpszStgName, DWORD grfMode)**

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 embeddings. (CreateChildStorage can be used to create child storages.)

| **Parameter** | **Description** |
|---|---|
| *lpszStgName* | Pointer to pathname of the Storage to create. If NULL, then a temporary Storage is created with the STGM_DELETEONRELEASE flag. |
| *grfMode* | Access mode to use to create the Storage. |

**Returns**

Pointer to newly created Storage.

**Comment**

The root-level storage is opened in transacted mode.

## OleStdOpenRootStorage

**STDAPI_(LPSTORAGE) OleStdOpenRootStorage(LPSTR lpszStgName, DWORD grfMode)**

Opens 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.)

| **Parameter** | **Description** |
|---|---|
| *lpszStgName* | Pointer to pathname of the storage object to open. |
| *grfMode* | Access mode to use to open the storage object. This value is OR'd with STGM_TRANSACTED. |

**Comment**

The root-level storage is opened in transacted mode.

## OpenOrCreateRootStorage

**STDAPI_(LPSTORAGE) OleStdOpenOrCreateRootStorage(LPSTR lpszStgName, DWORD grfMode)**

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 opened, 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 embeddings. (CreateChildStorage can be used to create child storages.)

| **Parameter** | **Description** |
|---|---|
| *lpszStgName* | Pointer to the pathnmae of the storage object to open or create. |
| *grfMode* | Access modue to use to open or create the storage. |

**Returns**

Pointer to opened storage if successful; NULL otherwise.

**Comment**

The root-level storage is opened in transacted mode.

## OleStdCreateChildStorage

**STDAPI_(LPSTORAGE) OleStdCreateChildStorage(LPSTORAGE lpStg, LPSTR lpszStgName)**

Creates 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.

| **Parameter** | **Description** |
|---|---|
| *lpStg* | Storage to create in which to create the child Storage. |
| *lpszStgName* | Pointer to name of the child Storage to create. |

**Returns**

Pointer to the newly created child Storage if successfull; NULL otherwise.

**Comment**

The child storage is opened in transacted mode.

## OleStdOpenChildStorage

**STDAPI_(LPSTORAGE) OleStdOpenChildStorage(LPSTORAGE lpStg, LPSTR lpszStgName, DWORD grfMode)**

Opens 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.

| **Parameter** | **Description** |
|---|---|
| *lpStg* | Storage that contains the child Storage. |
| *lpStgName* | Pointer to the name of the child Storage to open. |
| *grfMode* | Access mode in which the child Storage will be opened. |

**Returns**

Pointer to the opened child storage if successful; NULL otherwise.

**Comment**

The child storage is opened in transacted mode.

## OleStdCommitStorage

**STDAPI_(BOOL) OleStdCommitStorage(LPSTORAGE lpStg)**

Commits 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.

| **Parameter** | **Description** |
|---|---|
| *lpStg* | Pointer to Storage to commit. |

**Returns**

TRUE if the Storage was committed; FALSE otherwise.

**Comments**

OleStdCommitStorage first attempts to perform this commit in a safe manner using (STGC_DEFAULT). If this fails, it then attempts to do the commit in a less robust manner (STGC_OVERWRITE).

## OleUIDrawHandles

**STDAPI_(void) OleUIDrawHandles(LPRECT lpRect, HDC hdc, DWORD dwFlags, UINT cSize, BOOL fDraw)**

Draw handles or/and boundary around Container Object when selected.

| **Parameter** | **Description** |
|---|---|
| lpRect | Dimensions of Container Object |
| hdc | HDC of Container Object (MM_TEXT mapping mode) |
| dwFlags | Exclusive flags:<br>OLEUI_HANDLES_INSIDE Draw handles on inside of rect<br>OLEUI_HANDLES_OUTSIDE Draw handles on outside of rect<br>Optional flags:<br>OLEUI_HANDLES_NOBORDER Draw handles only, no rect<br>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 |

**Returns**

None.

## OleUIDrawShading

**STDAPI_(void) OleUIDrawShading(LPRECT lpRect, HDC hdc, DWORD dwFlags, UINT cWidth)**

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.

| **Parameter** | **Description** | |
|---|---|---|
| *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 hex value |
| *cWidth* | Width of border in pixels | |

**Returns**

None.

## OleUIShowObject

**STDAPI_(void) OleUIShowObject(LPCRECT lprc, HDC hdc, BOOL fIsLink)**

Draw the ShowObject effect around the object

| Parameter | Description |
|---|---|
| *lprc* | rectangle for drawing |
| *hdc* | HDC for drawing |
| *fIsLink* | linked object (TRUE) or embedding object (FALSE) |

**Returns**

None.

## OleStdMsgFilter

Provides a standard implementation of the IMessageFilter interface.

## OleStdMsgFilter_Create

**STDAPI_(LPMESSAGEFILTER) OleStdMsgFilter_Create(HWND hWndParent, LPSTR szAppName, MSGPENDINGPROC lpfnCallback, LPFNOLEUIHOOK lpfnOleUIHook)**

Creates and initializes an instance of the standard message filter.

| **Parameter** | **Description** |
|---|---|
| *hWndParent* | Window to use as the parent for the Busy dialog. |
| *szAppName* | String containing name of application. |
| *lpfnCallback* | Address of Message Pending Callback |
| *lpfnOleUIHook* | Address of Hook Procedure used by the Busy dialog. Can be NULL. |

**Returns**
A pointer to the newly created standard message filter.

**See Also**
OleStdMsgFilter_SetParentWindow

## OleStdMsgFilter_SetInComingStatus

**STDAPI_(void) OleStdMsgFilter_SetInComingCallStatus(LPMESSAGEFILTER lpThis, DWORD dwInComingCallStatus)**

This function sets the value that is returned from the IMessageFilter::HandleInComing method.

| **Parameter** | **Description** |
|---|---|
| *lpThis* | Pointer to instance of MESSAGEFILTER |
| *dwInComingCallStatus* | Value to be returned from the IMessageFilter::HandleInComing method. |

**Returns**

None.

## OleStdMsgFilter_GetInComingStatus

**STDAPI_(DWORD) OleStdMsgFilter_GetInComingCallStatus(LPMESSAGEFILTER lpThis)**

This function returns the currentincoming call status. It can be used to disable/enable options in the calling application.

| **Parameter** | **Description** |
|---|---|
| *lpThis* | Pointer to instance of MESSAGEFILTER |

**Returns**

SERVERCALL_ISHANDLED, SERVERCALL_REJECTED, SERVERCALL_RETRYLATER, if successful; -1 otherwise

## OleStdMsgFilter_EnableBusyDialog

**STDAPI_(void) OleStdMsgFilter_EnableBusyDialog(LPMESSAGEFILTER lpThis, BOOL fEnable)**

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.

| **Parameter** | **Description** |
|---|---|
| *lpThis* | Pointer to instance of MESSAGEFILTER |
| *fEnable* | TRUE to enable the Busy Dialog; FALSE to disable it. |

**Returns**

None.

**Comments**

If the busy dialog is NOT enabled, then the rejected call is immediately cancelled WITHOUT prompting the user. In this situation OleStdMsgFilter_RetryRejectedCall always returns OLESTDCANCELRETRY, thus 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.

## OleStdMsgFilter_EnableNotRespondingDialog

**STDAPI_(void) OleStdMsgFilter_EnableNotRespondingDialog(LPMESSAGEFILTER lpThis, BOOL fEnable)**

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.

| **Parameter** | **Description** |
|---|---|
| *lpThis* | Pointer to instance of MESSAGEFILTER |
| *fEnable* | TRUE to enable the "Not Responding" dialog; FALSE to disable it. |

**Returns**

None.

## OleStdMsgFilter_SetParentWindow

**STDAPI_(HWND) OleStdMsgFilter_SetParentWindow(LPMESSAGEFILTER lpThis, HWND hWndParent)**

This function allows caller to set which window will be used as the parent for the Busy dialog.

| **Parameter** | **Description** |
|---|---|
| *lpThis* | Pointer to instance of MESSAGEFILTER |
| *hWndParent* | Window handle of parent for Busy dialog. |

**Returns**

HWND of previous parent window.

**Comment**

It is important for an in-place active server to reset this to its current in-place frame window when it is in-place activated. If the hWndParent is set to NULL, then the desktop window will be the parent of the dialog.

## OleStdEnumFmtEtc_Create

**STDAPI_(LPENUMFORMATETC) OleStdEnumFmtEtc_Create(WORD wCount, LPFORMATETC lpEtc)**

Creates an instance of the standard implementation of IEnumFmtEtc.

| **Parameter** | **Description** |
|---|---|
| *wCount* | Number of FORMATETC structures in lpEtc |
| *lpEtc* | Array of FORMATETC structures |

**Returns**

Pointer to the newly created enumerator.

## OleStdEnumFmtEtc_Destroy

**VOID OleStdEnumFmtEtc_Destroy(LPOLESTDENUMFMTETC lpEF)**

Destroys the specified instance of OLESTDENUMFMTETC

| **Parameter** | **Description** |
|---|---|
| *lpEF* | Pointer to enumerator to destroy. |

**Returns**
None.

## OLEDBGDATA_MAIN(szPrefix)

## OLEDBGDATA

Debugging Macros defined in olestd.h. When compiled without DEBUG defined, this macros compile away. With DEBUG defined, these macros declare a buffer to be used with the other debugging macros defined in olestd.h Include OLEDBGDATA_MAIN at the beginning of your application's main file, and OLEDBGDATA at the beginning of each file in which you use the debugging macros.

| **Parameter** | **Description** |
|---|---|
| *szPrefix* | Prefix to be output in debugging message. Short version of application name is recommended. |

### OleDbgOutHResult(lpsz,hr)

### OleDbgOutScode(lpsz,sc)

Debugging Macros defined in olestd.h. When compiled without DEBUG defined, this macros compile away. With DEBUG defined, these macros print a debugging message including the string specified by lpsz and the name of the specified Scode or Hresult.

**OleDbgOut(lpsz)**

**OleDbgOut1(lpsz)**

**OleDbgOut2(lpsz)**

**OleDbgOut3(lpsz)**

**OleDbgOut4(lpsz)**

Debugging Macros defined in olestd.h. When compiled without DEBUG defined, this macros compile away. With DEBUG defined, these macros output the text, prefixed by the prefix declared in OLEDBGDATA_MAIN, at the specified debugging level (ie, OleDbgOut1 outputs at debugging level 1.) If the debugging level is greater than the global debugging message level, then the message is not output. OleDbgOut outputs the message regardless of the globa debugging message level. These macros output at the current indentation level, and do not change it.

**OleDbgOutNoPrefix(lpsz)**

**OleDbgOutNoPrefix1(lpsz)**

**OleDbgOutNoPrefix2(lpsz)**

**OleDbgOutNoPrefix3(lpsz)**

**OleDbgOutNoPrefix4(lpsz)**

Debugging Macros defined in olestd.h. When compiled without DEBUG defined, this macros compile away. With DEBUG defined, these macros output the text at the specified debugging level (ie, OleDbgOutNoPrefix1 outputs at debugging level 1.) If the debugging level is greater than the global debugging message level, then the message is not output. OleDbgOut outputs the message regardless of the globa debugging message level. These macros output at the current indentation level, and do not change it.

## OLEDBG_BEGIN(lpsz)

## OLEDBG_BEGIN1(lpsz)

## OLEDBG_BEGIN2(lpsz)

## OLEDBG_BEGIN3(lpsz)

## OLEDBG_BEGIN4(lpsz)

Debugging Macros defined in olestd.h. When compiled without DEBUG defined, this macros compile away. With DEBUG defined, these macros output the text specified by lpsz, prefixed by the string specified by OLEDBGDATA_MAIN.

Each macro outputs text at the specified debugging level (ie, OLEDBG_BEGIN1 outputs at debugging level 1.), and increments the Indentation level by 1. If the debugging level is greater than the global debugging message level, then the message is not output. OLEDBG_BEGIN outputs a message regardless of the global debugging message level (useful for crucial functions).

These macros can be used to mark the entrance to functions and methods, and should be paired with matching calls to OLEDBG_END*.

## OLEDBG_END

## OLEDBG_END1

## OLEDBG_END2

## OLEDBG_END3

## OLEDBG_END4

Debugging Macros defined in olestd.h. When compiled without DEBUG defined, this macros compile away. With DEBUG defined, these macros output a string composed of the prefixe sspecified by OLEDBGDATA_MAIN and the word "End".

Each macro outputs text at the specified debugging level (ie, OLEDBG_END1 outputs at debugging level 1.), and decrements the Indentation level by 1. If the debugging level is greater than the global debugging message level, then the message is not output. OLEDBG_END outputs a message regardless of the global debugging message level (useful for crucial functions).

These macros can be used to mark the exit from functions and methods, and should be paired with matching calls to OLEDBG_BEGIN*.

**OleDbgOutRefCnt(lpsz,lpObj,refcnt)**

**OleDbgOutRefCnt1(lpsz,lpObj,refcnt)**

**OleDbgOutRefCnt2(lpsz,lpObj,refcnt)**

**OleDbgOutRefCnt3(lpsz,lpObj,refcnt)**

**OleDbgOutRefCnt4(lpsz,lpObj,refcnt)**

Debugging Macros defined in olestd.h. When compiled without DEBUG defined, this macros compile away. With DEBUG defined, these macros output a message composed of the prefix declared by OLEDBGDATA_MAIN, the string specified by lpsz, a pointer to an object specified by lpObj, and the object's reference count, specified by refcnt.

Each macro outputs text at the specified debugging level (ie, OleDbgOutRefCnt1 outputs at debugging level 1.), and does not change the Indentation level. If the specified debugging level is greater than the global debugging message level, then the message is not output. OleDbgOutRefCnt outputs a message regardless of the global debugging message level (useful for error conditions).

**OleDbgOutRect(lpsz,lpRect)**

**OleDbgOutRect1(lpsz,lpRect)**

**OleDbgOutRect2(lpsz,lpRect)**

**OleDbgOutRect3(lpsz,lpRect)**

**OleDbgOutRect4(lpsz,lpRect)**

Debugging Macros defined in olestd.h. When compiled without DEBUG defined, this macros compile away. With DEBUG defined, these macros output a message composed of the prefix declared by OLEDBGDATA_MAIN, the string specified by lpsz, and the values of the members (including the width and height) of the RECT structure specified by lpRect.

Each macro outputs text at the specified debugging level (ie, OleDbgOutRect1 outputs at debugging level 1.), and does not change the Indentation level. If the specified debugging level is greater than the global debugging message level, then the message is not output. OleDbgOutRect outputs a message regardless of the global debugging message level (useful for error conditions).

## OleStdSetupAdvises

**STDAPI_(BOOL) OleStdSetupAdvises(LPOLEOBJECT lpOleObject, DWORD dwDrawAspect, LPSTR lpszContainerApp, LPSTR lpszContainerObj, LPADVISESINK lpAdviseSink, BOOL fCreate)**

Setup the standard View and Ole advises required by a standard, compound document-oriented container. Such a container relies on OLE to manage the presentation of the OLE object. The container calls IViewObject::Draw to render (display) the object.

This helper routine performs the following tasks:

1. Setup View advise
2. Setup Ole advise
3. Call IOleObject::SetHostNames
4. Call OleSetContainedObject

| **Parameter** | **Description** |
|---|---|
| *lpOleObject* | pointer to the OLE object |
| *dwDrawAspect* | Aspect for object |
| *lpszContainerApp* | pointer to string containing name of container app |
| *lpszContainerObj* | pointer to string |
| *lpAdviseSink* | pointer to Advise |
| *fCreate* | TRUE if the object is being created; FALSE if an existing object is being loaded. See Comments below. |

**Comments**

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 containers might need to setup this advise for programmatic reasons.

This advise will be torn down automatically by the server when the object is released, so the connection ID is not stored.

Set *fCreate* to TRUE if the *lpOleObject* is being created. If *lpOleObject* is an existing object that is being loaded, then set *fCreate* to FALSE. If *fCreate* is TRUE, then the ADVF_PRIMEFIRST flag is used in the call to IViewObject::Advise so that the initial presentation will be sent immediately.

**Returns**

TRUE if successful; FALSE otherwise.

## OleStdSwitchDisplayAspect

**STDAPI OleStdSwitchDisplayAspect(LPOLEOBJECT lpOleObj, LPDWORD lpdwCurAspect, DWORD dwNewAspect, HGLOBAL hMetaPict, BOOL fDeleteOldAspect, BOOL fSetupViewAdvise, LPADVISESINK lpAdviseSink, BOOL FAR* lpfMustUpdate)**

Switch the currently cached display aspect between DVASPECT_ICON and DVASPECT_CONTENT.

When setting up icon aspect, any currently cached content cache is discarded and any advise connections for content aspect are broken.

| **Parameter** | **Description** |
|---|---|
| *lpOleObj* | pointer to the object whose display aspect is being changed. |
| *lpdwCurAspect* | pointer to object's current aspect. |
| *dwNewAspect* | aspect to switch to. |
| *hMetaPict* | Metafile to use as presentation for DVASPECT_ICON |
| *fDeleteOldAspect* | TRUE if old aspect should be deleted. |
| *fSetupViewAdvise* | TRUE is a View advise should be set up. |
| *lpAdviseSink* | pointer for advise; used only fSetupViewAdvise is TRUE |
| *lpfMustUpdate* | On exit, TRUE if the data must be updated. |

**Comments**

If we are setting up Icon aspect with a custom icon (ie., dwNewAspect is DVASPECT_ICON and hMetaPict is non-NULL), 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 we are setting up Icon aspect with a custom icon, then we add the icon to the cache. Otherwise, the cache is updated, running the object if necessary.

It is possible to retain the caches set up for the old aspect, but this increases the storage space required for the object and possibly requires additional overhead to maintain the unused caches. For these reasons the strategy to delete the previous caches is preferred. If it is a requirement to quickly switch between Icon and Content display, then it would be better to keep both aspect caches.

**Returns:**

NOERROR if the new display aspect was set up successfully. If an error occurred during the aspect switch, then the error return from IOleCache::Cache is returned from OleStdSwitchDisplayAspect. If an error occurs, then the current display aspect and cache contents are unchanged.

## OleStdSetIconInCache

**STDAPI OleStdSetIconInCache(LPOLEOBJECT lpOleObj, HGLOBAL hMetaPict)**

SetData a new icon into the existing DVASPECT_ICON cache.

| **Parameter** | **Description** |
|---|---|
| *lpOleObj* | pointer to OLE object |
| *hMetaPict* | Metafile containing icon to store in cache |

**Returns**

HRESULT returned from IOleCache::SetData

## OleStdDoConvert

**STDAPI OleStdDoConvert(LPSTORAGE lpStg, REFCLSID rClsidNew)**

Converts the object in the specified storage to a new class.

| **Parameter** | **Description** |
|---|---|
| *lpStg* | pointer to object's storage |
| *rClsidNew* | Class ID of class to convert the object to. |

**Returns**

NOERROR if successful.

**Comments**

OleStdDoConvert performs 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.

OleStdDoConvert performs step 2.

## OleStdGetTreatAsFmtUserType

**STDAPI_(BOOL) OleStdGetTreatAsFmtUserType(REFCLSID rclsidApp, LPSTORAGE lpStg, CLSID FAR* lpclsid, CLIPFORMAT FAR* lpcfFmt, LPSTR FAR* lplpszType)**

Determines if the application should perform a TreatAs (ActivateAs object or emulation) operation for the object that is stored in the storage.

| **Parameter** | **Description** |
|---|---|
| *rclsidApp* | Class ID of current application. |
| *lpStg* | pointer to object's storage. |
| *lpclsid* | pointer to Class ID contained in object's storage. |
| *lpcfFmt* | pointer to storage's clipboard format (to be returned from ReadFmtUserTypeStg) |
| *lplpszType* | pointer to storage's User Type (to be returned from ReadFmtUserTypeStg) |

**Comments**

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 Reg DB. If it cannot be found in the Reg DB, then the TreatAs operation can NOT be executed.

NOTE: lplpszType must be freed by caller.

**Returns**

TRUE if TreatAs should be performed. If TRUE is returned, then lpclsid, lplpszType, lpcfFmt are valid. FALSE if TreatAs should not be performed, in which case lplpszType and lpcfFmt will be NULL, and lpclsid will be CLSID_NULL.

## OleStdIsOleLink

**STDAPI_(BOOL) OleStdIsOleLink(LPUNKNOWN lpUnk)**

Determines if an object is an OLE link object. OleStdIsOleLink queries to see if IOleLink interface is supported. If so, the object is a link, otherwise it is not.

| **Parameter** | **Description** |
|---|---|
| lpUnk | pointer to object's IUnknown interface. |

**Returns**

TRUE if the OleObject is an OLE link object.

## OleStdQueryInterface

**STDAPI_(LPUNKNOWN) OleStdQueryInterface(LPUNKNOWN lpUnk, REFIID riid)**

Retrieves a pointer to the specified interface.

| Parameter | Description |
|---|---|
| *lpUnk* | pointer to object's IUnknown interface. |
| *riid* | ID of interface to which to retrieve a pointer. |

**Returns**

The desired interface pointer if exposed by the given object. Returns NULL if the interface is not available.

## OleStdGetData

**STDAPI_(HGLOBAL) OleStdGetData(LPDATAOBJECT lpDataObj, CLIPFORMAT cfFormat, DVTARGETDEVICE FAR* lpTargetDevice, DWORD dwAspect, LPSTGMEDIUM pMedium)**

Retrieve data from an IDataObject in a specified format on a global memory block.

| **Parameter** | **Description** |
|---|---|
| *lpDataObj* | object on which GetData should be called. |
| *cfFormat* | desired clipboard format (eg. CF_TEXT) |
| *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. |
| *lpMedium* | ptr to STGMEDIUM struct. The resultant medium from the IDataObject::GetData call is returned. |

**Returns**
If successful, the global memory handle of retrieved data block. NULL if not successful.

**Comments**
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.

## OleStdMalloc

**STDAPI_(LPVOID) OleStdMalloc(ULONG ulSize)**

Allocate memory using the currently active IMalloc* allocator.

| **Parameter** | **Description** |
|---|---|
| *ulSize* | Size, in bytes, of memory block to allocated. |

**Returns**

Pointer to allocated memory if successful; NULL otherwise.

## OleStdRealloc

**STDAPI_(LPVOID) OleStdRealloc(LPVOID pmem, ULONG ulSize)**

Reallocates memory using the currently active IMalloc* allocator.

| Parameter | Description |
|---|---|
| *pmem* | Pointer to already-allocated memory. |
| *ulSize* | Size, in bytes, of memory block to reallocate. |

**Returns**

Pointer to reallocated memory if successful; NULL otherwise.

## OleStdFree

**STDAPI_(void) OleStdFree(LPVOID pmem)**

Frees memory using the currently active IMalloc* allocator.

| Parameter | Description |
|---|---|
| *pmem* | Pointer to memory block to free. |

**Returns**

None.

## OleStdGetSize

**STDAPI_(ULONG) OleStdGetSize(LPVOID pmem)**

Get the size of a memory block that was allocated using the currently active IMalloc* allocator.

| Parameter | Description |
|---|---|
| pmem | Pointer to memory block to return the size of. |

**Returns**

Size of specified memory.

## OleStdFreeString

**STDAPI_(void) OleStdFreeString(LPSTR lpsz, LPMALLOC lpMalloc)**

Frees a string that was allocated with the currently active IMalloc* allocator.

| Parameter | Description |
|---|---|
| *lpsz* | Pointer to string to free. |
| *lpMalloc* | Current IMalloc*. If NULL, then OleStdFreeString will retrieve and use the active allocator. |

**Returns**

None.

**Comments**

If the caller has the current IMalloc* handy, then it can be passed as an argument, otherwise this function will retrieve the active allocator and use it.

## OleStdCopyString

**STDAPI_(LPSTR) OleStdCopyString(LPSTR lpszSrc, LPMALLOC lpMalloc)**

Copies a string into memory allocated with the currently active IMalloc* allocator.

| Parameter | Description |
|---|---|
| *lpszSrc* | Pointer to string to copy. |
| *lpMalloc* | Current IMalloc*. If NULL, then OleStdFreeString will retrieve and use the active allocator. |

**Returns**

Pointer to copied string.

**Comments**

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.

## OleStdCreateStorageOnHGlobal

**STDAPI_(LPSTORAGE) OleStdCreateStorageOnHGlobal(HANDLE hGlobal, BOOL DeleteOnRelease, DWORD grfMode)**

Creates a memory based IStorage*.

| **Parameter** | **Description** |
|---|---|
| *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 |

**Comments**

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.

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:
S_OK if successful.

## OleStdCreateTempStorage

**STDAPI_(LPSTORAGE) OleStdCreateTempStorage(BOOL fUseMemory, DWORD grfMode)**

Create a temporay IStorage* that will DeleteOnRelease. This can be either memory based or file based.

| **Parameter** | **Description** |
|---|---|
| *fUseMemory* | controls if memory-based or file-based stg is created |
| *grfMode* | storage mode flags |

**Returns**

Pointer to newly created IStorage if successful;  NULL otherwise.

## OleStdGetOleObjectData

**STDAPI OleStdGetOleObjectData(LPPERSISTSTORAGE lpPStg, LPFORMATETC lpformatetc,LPSTGMEDIUM lpMedium, BOOL fUseMemory)**

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. This function can be used to implement IDataObject::GetData and IDataObject::GetDataHere.

| **Parameter** | **Description** |
|---|---|
| *lpPStg* | Pointer to object's IPersistStorage interface. |
| *lpformatetc* | Pointer to FORMATETC passed to IDataObject::GetData or GetDataHere. |
| *lpMedium* | Pointer to MEDIUM passed to IDataObject::GetData or GetDataHere. |
| *fUseMemory* | Valid for GetData only. TRUE to allocated memory-based storage; FALSE to attempt to allocate file-based structured storage (IStorage *). |

**Comments**

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 written into the passed in IStorage. This is useful to support an IDataObject::GetDataHere call where the caller has allocated his own IStorage.

## OleStdGetLinkSourceData

**STDAPI OleStdGetLinkSourceData(LPMONIKER lpmk, LPCLSID lpClsID, LPFORMATETC lpformatetc, LPSTGMEDIUM lpMedium)**

Render CF_LINKSOURCE data on an TYMED_ISTREAM medium using a moniker and Class ID as input. This function can be used to implement IDataObject::GetData and IDataObject::GetDataHere.

| **Parameter** | **Description** |
|---|---|
| *lpmk* | Pointer to object's moniker. |
| *lpClsID* | Pointer to object's Class ID. |
| *lpformatetc* | Pointer to FORMATETC passed to IDataObject::GetData or GetDataHere. |
| *lpMedium* | Pointer to MEDIUM passed to IDataObject::GetData or GetDataHere. |

**Returns**

NOERROR if successful; otherwise, it returns the error from WrteClassStm.

**Comments**

If lpMedium->tymed == TYMED_NULL, then a delete-on-release memory-based stream is allocated. This is useful to support an IDataObject::GetData call where the callee must allocate the medium.

If lpMedium->tymed == TYMED_ISTREAM, then the data is written into the passed in IStream. This is useful to support an IDataObject::GetDataHere call where the caller has allocated his own IStream.

## OleStdGetObjectDescriptorData

**STDAPI_(HGLOBAL) OleStdGetObjectDescriptorData(CLSID clsid, DWORD dwAspect, SIZEL sizel, POINTL pointl, DWORD dwStatus, LPSTR lpszFullUserTypeName, LPSTR lpszSrcOfCopy)**

Fills and returns a OBJECTDESCRIPTOR structure.

| **Parameter** | **Description** |
|---|---|
| *clsid* | CLSID of object being transferred |
| *dwAspect* | Display Aspect of object |
| *sizel* | Size of object in HIMETRIC |
| *pointl* | Offset from upper-left corner of object where mouse went down for drag. Meaningful only when drag-drop is used. |
| *dwStatus* | OLEMISC flags |
| *lpszFullUserTypeName* | User Type Name |
| *lpszSrcOfCopy* | Source of Copy |

**Returns**
Handle to OBJECTDESCRIPTOR structure.

**See Also**
OBJECTDESCRIPTOR

## OleStdFillObjectDescriptorFromData

**STDAPI_(HGLOBAL) OleStdFillObjectDescriptorFromData(LPDATAOBJECT lpDataObject, LPSTGMEDIUM lpmedium, CLIPFORMAT FAR* lpcfFmt)**

Fills and returns a OBJECTDESCRIPTOR structure. The source object will offer OF_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.

| **Parameter** | **Description** |
|---|---|
| *lpDataObject* | Source object |
| *lpmedium* | Storage medium |
| *lpcfFmt* | Format offered by lpDataObject (OUT parameter) |

**Returns**

Handle to OBJECTDESCRIPTOR structure.

## OleStdQueryLinkSourceData

**STDAPI OleStdQueryLinkSourceData(LPFORMATETC lpformatetc)**

Determines whether the requested medium in the FORMATETC is acceptable for CF_LINKSOURCE. This function can be used to implement IDataObject::QueryGetData for format CF_LINKSOURCE.

| **Parameter** | **Description** |
|---|---|
| *lpformatetc* | Pointer to FORMATETC passed into IDataObject::QueryGetData. |

**Returns**

NOERROR if acceptable; otherwise DATA_E_FORMATETC

## OleStdQueryObjectDescriptorData

**STDAPI OleStdQueryObjectDescriptorData(LPFORMATETC lpformatetc)**

Determines whether the requested medium in the FORMATETC is acceptable for CF_EMBEDSOURCE or CF_EMBEDDEDOBJECT. This function can be used to implement IDataObject::QueryGetData for format CF_EMBEDSOURCE and CF_EMBEDDEDOBJECT.

| **Parameter** | **Description** |
|---|---|
| *lpformatetc* | Pointer to FORMATETC passed into IDataObject::QueryGetData. |

**Returns**

NOERROR if acceptable; otherwise DATA_E_FORMATETC

## OleStdQueryFormatMedium

**STDAPI OleStdQueryFormatMedium(LPFORMATETC lpformatetc, TYMED tymed)**

Determines whether TYMED matches one of the requested mediums in the FORMATETC. This function can be used to implement IDataObject::QueryGetData.

| **Parameter** | **Description** |
|---|---|
| *lpformatetc* | Pointer to FORMATETC passed into IDataObject::QueryGetData. |
| *tymed* | Supported TYMED. |

**Returns**

NOERROR if acceptable; otherwise DATA_E_FORMATETC

## OleStdCopyMetafilePict

**STDAPI_(BOOL) OleStdCopyMetafilePict(HANDLE hpictin, HANDLE FAR* phpictout)**

Makes a copy of a MetafilePict.

| Parameter | Description |
|---|---|
| *hpictin* | MetafilePict to copy. |
| *phpictout* | Pointer to where to return copy of hpictin. |

**Returns**

TRUE if successful, otherwise FALSE.

## OleStdGetMetafilePictFromOleObject

**STDAPI_(HANDLE) OleStdGetMetafilePictFromOleObject(LPOLEOBJECT lpOleObj, DWORD dwDrawAspect)**

Generate a MetafilePict by drawing the OLE object.

| **Parameter** | **Description** |
|---|---|
| *lpOleObj* | Pointer to OLE object whose metafile to return. |
| *dwDrawAspect* | Drawing aspect of object |

**Returns**

Handle of allocated METAFILEPICT

## OleStdVerifyRelease

**STDAPI_(ULONG) (LPUNKNOWN lpUnk, LPSTR lpszMsg)**

Calls Release on the object that is expected to go away. If the refcnt of the object did no go to 0 then gives a debug message.

| Parameter | Description |
|---|---|
| *lpUnk* | Pointer to object's IUnknown interface |
| *lpszMsg* | String pointer of message to display if the object's reference count was not 0 after calling Release. |

**Returns**

Value of the object's reference count.

## OleStdInitVtbl

**STDAPI_(void) OleStdInitVtbl(LPVOID lpVtbl, UINT nSizeOfVtbl)**

Initializes 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.

| **Parameter** | **Description** |
|---|---|
| *lpVtbl* | Pointer to VTBL to initialize. |
| nSizeOfVtbl | Number of methods in VTBL. |

**Returns**

None

**Comments**

This function does not initialize the Vtbl with useful 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.

## OleStdNoteFileChangeTime

**STDAPI_(void) OleStdNoteFileChangeTime(LPSTR lpszFileName, DWORD dwRegister)**

Notes 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.

**Returns**

None.

## OleStdNoteObjectChangeTime

**STDAPI_(void) OleStdNoteObjectChangeTime(DWORD dwRegister)**

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.

| **Parameter** | **Description** |
|---|---|
| *dwRegister* | Unique identifier returned by IRunningObjectTable::Register |

**Comment**

This function set the change time to the current time.

## OleStdGetLenFilePrefixOfMoniker

**STDAPI_(ULONG) OleStdGetLenFilePrefixOfMoniker(LPMONIKER lpmk)**

If the first piece of the Moniker is a FileMoniker, then return the length of the filename string.

| **Parameter** | **Description** |
|---|---|
| *lpmk* | pointer to moniker |

**Returns**

0 if moniker does NOT start with a FileMoniker. Otherwise, the length of filename prefix of the display name retrieved from the given (lpmk) moniker.

## OleStdDoTreatAs

**STDAPI OleStdDoTreatAsClass(LPSTR lpszUserType, REFCLSID rclsid, REFCLSID rclsidNew)**

Performs the container-side responsibilities for "ActivateAs" (aka.TreatAs) for an object.

| **Parameter** | **Description** |
|---|---|
| *lpszUserType* | Pointer to current User Type name. |
| *rclsid* | Class ID of current class. |
| *rclsidNew* | Class ID of class to treat as. |

**Returns**

S_OK if successful.

**Comments**

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.

OleStdDoTreatAsClass performs step 2.

Note that if the current class is not registered in the registration database, then a minimal entry for it will be added.

## OleStdGetObjectDescriptorDataFromOleObject

**STDAPI_(HGLOBAL) OleStdGetObjectDescriptorDataFromOleObject(LPOLEOBJECT lpOleObj, LPSTR lpszSrcOfCopy, DWORD dwAspect, POINTL pointl, LPSIZEL lpSizelHim)**

Fills and returns a OBJECTDESCRIPTOR structure. Information for the structure is obtained from an OLEOBJECT.

| **Parameter** | **Description** |
|---|---|
| *lpOleObj* | Pointer to OleObject from which ONJECTDESCRIPTOR info is obtained. |
| *lpszSrcOfCopy* | 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. |
| *dwAspectDisplay* | Aspect of object |
| *pointl* | Offset from upper-left corner of object where mouse wentdown for drag. Meaningful only when drag-drop is used. |
| *lpSizelHim* | Size of scaled object in container. If the object is scaled in the container, then the container should pass the extents that it uses to display the object. If the object is not being scaled, then *lpSizelHim* should be set to NULL. If *lpSizelHim* is NULL, IViewObject2::GetExtent is called to retrieve the object's extents. |

**Returns**

Handle to OBJECTDESCRIPTOR structure.

## OleStdMsgFile_SetHandleInComingCallbackProc

**STDAPI_(HANDLEINCOMINGCALLBACKPROC)**
**OleStdMsgFilter_SetHandleInComingCallbackProc(LPMESSAGEFILTER lpThis,**
**HANDLEINCOMINGCALLBACKPROC lpfnHandleInComingCallback)**

Installs or uninstalls a callback procedure to selectively handle or reject specific incoming method calls on particular interfaces.

| **Parameter** | **Description** |
|---|---|
| *lpThis* | Pointer to message filter interface |
| *lpfnHandleInComingCallback* | Pointer to callback procedure to be installed. If NULL, then the currently installed callback procedure is uninstalled. |

**Returns**
Pointer to previous callback procedure. NULL if there is no callback procedure was previously installed.

**Comments**
A callback procedure installed by calling OleStdMsgFilter_SetHandleInComingCallbackProc will override the dwIncomingCallStatus established by a call to OleStdMsgFilter_SetInComingStatus. Using OleStdMsgFilter_SetInComingStatus allows an app to reject or accept ALL in coming calls. Using a HandleInComingCallbackProc allows an app to selectively handle or reject particular method calls.

To uninstall a HandleInComingCallbackProc, call
OleStdMsgFilter_SetHandleInComingCallbackProc(NULL)

## ParseCmdLine

**STDAPI_(void) ParseCmdLine(LPSTR lpszLine, BOOL FAR* lpfEmbedFlag, LPSTR szFileName)**

Parses a Windows command line passed to an application in WinMain. If the embedding switch ("-Embedding" or "/Embedding") is found, then it sets *lpfEmbedFlag to TRUE. If a filename is included, then it is copied to szFileName.

| **Parameter** | **Description** |
|---|---|
| lpszLine | Command line from application's WinMain. |
| lpfEmbedFlag | Pointer to BOOL to be filled on output. TRUE if the embedding switch was found, FALSE otherwise. |
| szFileName | Buffer to filename from command line. Filled on output; will be NULL if the command line did not include a filename. |

**Returns**

none.

## OleStdCreateDbAlloc

**HRESULT OleStdCreateDbAlloc(ULONG reserved, IMalloc** ppmalloc)**

Create an instance of CDbAlloc -- a debug implementation of IMalloc.

| Parameter | Description |
|---|---|
| reserved | ULONG reserved for future use. Must be 0. |
| ppmalloc | IMalloc FAR* FAR* Pointer to an IMalloc interface. Points to newly created debug allocator object on output. |

**Returns**

NOERROR if an instance of CDbAlloc was successfully created, E_OUTOFMEMORY if the creation failed.

**Comments**

CDbAlloc is a simple wrapping of the C runtime memory allocator that includes memory leak and overwrite detection.

Memory 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.

CDbAlloc also includes additional parameter validation code, as well as additional checks to make sure that instances that are passed to Free() were actually allocated by the corresponding instance of the allocator.

The following code sample creates an instance of this debug allocator and uses the default output interface:

```
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;
}
```

## OleStdInitSummaryInfo

**STDAPI_(LPSUMINFO) OleStdInitSummaryInfo(int reserved)**

Allocates a Summary Info structure.

| **Parameter** | **Description** |
|---|---|
| reserved | INT reserved for future use; must be 0. |

**Returns**

Pointer to newly initialized Summary Info structure.
**

**Comments**

CoInitialize MUST be called before calling OleStdInitSummaryInfo. Memory is allocated using the currently active IMalloc allocator (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.

## OleStdFreeSummaryInfo

**STDAPI_(void) OleStdFreeSummaryInfo(LPSUMINFO FAR *lplp)**

Frees a Summary Info structure.

| **Parameter** | **Description** |
|---|---|
| lplp | Pointer to open Summary Info structure to free |

**Returns**

void

**Comments**

Memory is freed using the currently active IMalloc allocator (returned from 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 Also**

OleStdGetThumbNailProperty

## OleStdClearSummaryInfo

**STDAPI_(void) OleStdClearSummaryInfo(LPSUMINFO lp)**

Frees storage (memory) for all the properties of the specified LPSUMINFO.

| Parameter | Description |
|---|---|
| lp | Pointer to an open SUMINFO structure. |

**Returns**

none.

**Comments**

After calling OleStdClearSummaryInfo, you must call OleStdReadSummaryInfo to load the SUMINFO structure again.

**See Also**

OleStdReadSummaryInfo

## OleStdReadSummaryInfo

**STDAPI_(int) OleStdReadSummaryInfo(LPSTREAM lpStream, LPSUMINFO lp)**

Reads all Summary Info properties into memory (except thumbnail which is demand loaded).

| **Parameters** | **Description** |
|---|---|
| *lpStream* | Pointer to open SummaryInfo IStream* |
| *lp* | Pointer to open SUMINFO structure |

**Returns**

Returns 1 if all Summary Info properties (except Thumbnail) were successfully read into memory, 0 otherwise.

## OleStdWriteSummaryInfo

**STDAPI_(int) OleStdWriteSummaryInfo(LPSTREAM lpStream, LPSUMINFO lp)**

Write all Summary Info properties to the specified IStream.

| **Parameter** | **Description** |
|---|---|
| *lpStream* | Pointer to an open SummaryInfo IStream* |
| *lp* | Pointer to an open SUMINFO structure |

**Returns**

Returns 1 if all Summary Info properties were successfully written to the specified IStream, 0 otherwise.

## OleStdGetSecurityProperty

**STDAPI_(DWORD) OleStdGetSecurityProperty(LPSUMINFO lp)**

Retrieves the Security Property

| **Parameter** | **Description** |
|---|---|
| *lp* | Pointer to open SUMINFO structure |

**Returns**

DWORD specifying the security level stored in the specified SUMINFO structure. The possible values and their meanings are listed in the table below.

| | | |
|---|---|---|
| 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 (application-enforced) security level on the document, an application other than the originator of the document can adjust its user interface according to the document's properties. An application should not display any information about a password-protected document, and should not allow modifications to enforced-read-only or locked-for-annotations documents. If the user attempts to modify properties for a read-only-recommended document, the application should display a warning.

## OleStdSetSecurityProperty

**STDAPI_(int) OleStdSetSecurityProperty(LPSUMINFO lp, DWORD security)**

Set the Security Property

| **Parameter** | **Description** |
|---|---|
| *lp* | Pointer to an open SUMINFO structure. |
| *security* | Security level. See Comments below for values. |

**Returns**

Always returns 1.

**Comments**

The *security* parameter should be one of the following values:

| | | |
|---|---|---|
| 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 |

By noting the (application-enforced) security level on the document, an application other than the originator of the document can adjust its user interface according to the document's properties. An application should not display any information about a password-protected document, and should not allow modifications to enforced-read-only or locked-for-annotations documents. If the user attempts to modify properties for a read-only-recommended document, the application should display a warning.

## OleStdGetStringProperty

**STDAPI_(LPSTR) OleStdGetStringProperty(LPSUMINFO lp, DWORD pid)**

Retrieves a string property and returns NULL terminated string.

| Parameter | Description |
|---|---|
| *lp* | Pointer to open SUMINFO structure |
| *pid* | ID of String Property |

**Returns**

Pointer to the string stored in the specified string property.

**Comments**

The memory allocated for the returned string is freed by OleStdFreeSummaryInfo, so the caller should NOT the returned string.

## OleStdSetStringProperty

**STDAPI_(int) OleStdSetStringProperty(LPSUMINFO lp, DWORD pid, LPSTR lpsz)**

Set the specified string property

| Parameter | Description |
|---|---|
| *lp* | Pointer to an open SUMINFO structure |
| *pid* | ID of string property |
| *lpsz* | Pointer to NULL terminated string to be set as the new value for the specified string property. If lpsz is NULL, then the property is cleared. |

**Returns**

Returns 1 if the specified string property is set successfully, O otherwise, or if *pid* is invalid.

**Comments**

A copy of *lpsz* is stored as the string property; so the calling application can free lpsz without invalidating the string property.

## OleStdGetStringZProperty

**STDAPI_(LPSTZR) OleStdGetStringZProperty(LPSUMINFO lp, DWORD pid)**

Retrieves a string property.

| Parameter | Description |
|---|---|
| *lp* | Pointer to an open SUMINFO structure |
| *pid* | ID of string property to retrieve. |

**Returns**

NULL terminated string with leading byte count.

**Comments**

The returned string will be freed by OleStdFreeSummaryInfo, so the caller should NOT be freed the returned string.

## OleStdGetDocProperty

**STDAPI_(void) OleStdGetDocProperty(LPSUMINFO lp, DWORD FAR* nPage, DWORD FAR* nWords, DWORD FAR* nChars)**

Retrieves document properties

| **Parameter** | **Description** |
|---|---|
| *lp* | Pointer to an open SUMINFO structure |
| *nPage* | Pointer to DWORD to return the number of pages in document. OUT parameter. |
| *nWords* | Pointer to DWORD to return the number of words in document. OUT parameter. |
| *nChars* | Pointer to DWORD to return the number of characters in document. OUT parameter. |

**Returns**

None.

## OleStdSetDocProperty

Set document properties.

| Parameter | Description |
|---|---|
| *lp* | Pointer to an open SUMINFO structure |
| *nPage* | Number of pages in document |
| *nWords* | Number of words in document |
| *nChars* | Number of characters in document |

**Returns**
Always returns 1.

## OleStdGetThumbNailProperty

**STDAPI_(int) OleStdGetThumbNailProperty( LPSTREAM lps, LPSUMINFO lp, DWORD FAR* clipFormatNo, LPSTR FAR* lpszName, THUMBNAIL FAR* clip, DWORD FAR* byteCount, BOOL transferClip)**

Retrieves a Thumbnail property.

| **Parameter** | **Description** |
|---|---|
| *lps* | Pointer to an IStream. |
| *lp* | Pointer to an open SUMINFO structure |
| *clipFormatNo* | Clipboard format for thumbnail. Only VT_CF_WIN is implemented, so *clipFormatNo* should be set to CF_METAFILEPICT. |
| *lpszName* | Format name. NOT IMPLEMENTED |
| *clip* | Handle to thumbnail for VT_CF_WIN. *clip* will be handle to MetafilePict Only VT_CF_WIN is currently implemented |
| *byteCount* | Size of thumbnail stream. For VT_CF_WIN, this should be the sum of the size of the Metafile and the MetafilePict structure. |
| *transferClip* | Transfer ownership of thumbnail to caller. See Comments below. |

**Returns**

OLE thumbnail selector value. Will be one of the following values.

| | |
|---|---|
| VT_CF_WIN | Windows thumbnail (clipFormatNo is interpreted as Windows clipboard format) |
| VT_CF_FMTID | NOT IMPLEMENTED. Thumbnail format is specified by ID using *clipFormatNo*. (but NOT a Windows format ID) |
| VT_CF_NAME | NOT IMPLEMENTED. Thumbnail format is specified by name using *lpszName*. |
| VT_CF_EMPTY | Blank thumbnail (clip will be NULL) |
| VT_CF_OOM | Memory allocation failure |

**Comments**

Currently, VT_CF_WIN is the only supported clipboard format for thumbnail properties. OleStdSetThumbNailProperty does implement VT_CF_FMTID and VT_CF_NAME; however, OleStdGetThumbNailProperty, OleStdReadSummaryInfo and OleStdWriteSummaryInfo only support VT_CF_WIN.

On input, the thumbnail is read on demand; all 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.

## OleStdSetThumbNailProperty

**STDAPI_(int) OleStdSetThumbNailProperty(LPSTREAM lps, LPSUMINFO lp, int vtcfNo, DWORD clipFormatNo, LPSTR lpszName, THUMBNAIL clip, DWORD byteCount)**

Set a thumbnail property.

| **Parameters** | **Description** |
|---|---|
| *lps* | Pointer to an open SummaryInfo IStream |
| *lp* | Pointer to an open SUMINFO structure |
| *vtcfNo* | OLE thumbnail selector value. See Comments below for values. |
| *clipFormatNo* | Clipboard format for thumbnail used if *vtcfNo* is VT_CF_WIN or VT_CF_FMTID. The interpretation of cliipFormatNo depends on the *vtcfNo* specified. In most cases, *vtcfNo* is VT_CF_WIN and *clipFormatNo* is CF_METAFILEPICT. |
| *lpszName* | Format name if *vtcfNo* is VT_CF_NAME |
| *clip* | Handle to thumbnail for VT_CF_WIN clip will be handle to MetafilePict |
| *byteCount* | Size of thumbnail stream. If *vtcfNo* is VT_CF_WIN, then *byteCount* should be the sum of the size of the Metafile and the size of a MetafilePict structure. |

**Returns**

Returns 1 if the thumbnail property was set successfully; 0 otherwise.

**Comments**

The following table lists the possible values for *vtcfNo*:

| | |
|---|---|
| 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). |

VT_CF_WIN is currently the only supported value for *vtcfNo*. OleStdSetThumbNailProperty does implement VT_CF_FMTID and VT_CF_NAME, however, OleStdGetThumbNailProperty, OleStdReadSummaryInfo and OleStdWriteSummaryInfo only support VT_CF_WIN.

OleStdSetThumbnailProperty copies *lpszName* but saves the "clip" handle passed.

## OleStdGetDateProperty

**STDAPI_(void) OleStdGetDateProperty(LPSUMINFO lp, DWORD pid, int FAR* yr, int FAR* mo, int FAR* dy, DWORD FAR* sc)**

Retrieves a date property.

| Parameter | Description |
|---|---|
| *lp* | Pointer to an open SUMINFO structure |
| *pid* | ID of data property to retrieve |
| *yr* | Pointer to where to return the year. |
| *mo* | Pointer to where to return the month. |
| *dy* | Pointer to where to return the day. |
| *sc* | Pointer to where to return the seconds. |

**Returns**

None.

## OleStdSetDateProperty

**STDAPI_(int) OleStdSetDateProperty(LPSUMINFO lp, DWORD pid, int yr, int mo, int dy, int hr, int mn, int sc)**

Sets date property.

| Parameter | Description |
|---|---|
| *lp* | Pointer to an open SUMINFO structure |
| *pid* | ID of Property |
| *yr* | year |
| *mo* | month |
| *dy* | day |
| *hr* | hours |
| *mn* | minutes |
| *sc* | seconds |

**Returns**

Returns 1 if the date property was set successfully, 0 otherwise.

**Comments**

To clear a date property, set all parameters except *lp* and *pid* to 0.

The following example sets the PID_EDITTIME property to 12:30:23 Jan 1,1993i:

```
OleStdSetDateProperty(lpSumInfo, PID_EDITTIME, 1993, 1, 1, 12, 30, 23);
```