Inside OLE 2nd Edition Code Corrections

Created May 1996

Code is shown in **10pt Courier New Bold**, **with corrections and additions in blue**.

A special thanks to Thomas Steiner who sent me many code corrections.

For issues you have with any changes, or if you find other bugs that are not corrected by these changes, please email kraigb@microsoft.com.

INOLE-A.MAK, lines 96-105

The original file refers to OC30.LIB and OC30U.LIB within the *!ifdef NOOCLIB* condition. OC30*.LIB was the original import library for the OLE Controls (ActiveX Controls) support DLL, OC30*.DLL. After Inside OLE was published, this support code was moved into OLEPRO32.DLL (OLEPROP.DLL on Win16) and so the import library is not OLEPRO32.LIB. Therefore the condition block inside this file should be:

```
!ifdef NOOCLIB
OCLIB   =
!else
!ifdef UNICODE
OCLIB   = OLEPRO32.LIB
!else
OCLIB   = OLEPRO32.LIB
!endif

!endif #NOOCLIB
```

Note that in Windows NT 4.0 and later the separate file OLEPRO32.DLL obsolete—the support APIs live inside OLEAUT32.DLL, so separate linking to OLEPRO32.LIB is unnecessary. However, it is still necessary for compilations targeting Windows95.

Also note that MKTYPLIB.EXE is being phased out as the MIDL compiler becomes the single IDL/ODL parsing tool for OLE which can produce marshalers and type libraries from the same IDL file. In short, the MIDL compiler now recognizes the "library" statement as in ODL and turns anything referenced in that block into a type library. MIDL also allows you to define an interface outside the library block to generate marshalers then refer to that interface inside the library block to produce type information for it. In this manner you only need to define the interface once. For more information, see the Win32 SDK documentation (Windows NT 4.0 and later) for the MIDL compiler.

INC\INOLE.H, lines 298-304

The *ReleaseInterface* macro is not perfectly safe (for threads or other concurrency issues). To make it thread safe, change it to contain the following code:

```
#define ReleaseInterface(p)\
            {\
            IUnknown *pt=(IUnknown *)p;\
            p=NULL;\
            if (NULL!=pt)\
                pt->Release();\
            }
```

This sequence prevents another piece of code from using the interface pointer but does it *before* calling *Release* as opposed to *after* the call as the original version does. This prevents problems where the act of calling *Release* causes the released object to make additional calls back to the client in which code may refer to the same interface pointer again. By setting that pointer to NULL first, then calling *Release,* any other calls that come back to the client from within that *Release* call will cause a fault, which is what we want to catch during development.

CLASSLIB\CCLIENT.CPP, line 89

The line that reads:

```
pszClass=SZCLASSSSDICLIENT;
```

should read:

```
pszClass=SZCLASSSDICLIENT;
```

that is, the original sources had an extra “S” in the constant name

CLASSLIB\CFRAME.CPP, line 813

Change this code:

```
, m_hWnd, AboutProc);
```

to this:

```
, m_hWnd, (DLGPROC)AboutProc);
```

Otherwise the code will not compile with STRICT defined.

INC\IDESCRIP.H

This file is replaced with the one generated from the change to the IDL; see CHAP09\IDESCRIP\IDESCRIP.IDL, line 22 below.

INC\INOLE.H, lines 298-305

Replace the macro definition:

```
#define ReleaseInterface(p)\
            {\
            if (NULL!=p)\
                {\
                p->Release();\
                p=NULL;\
                }\
            }
```

with this:

```
#define ReleaseInterface(p)\
            {\
            IUnknown *pt=(IUnknown *)p;\
            p=NULL;\
            if (NULL!=pt)\
                pt->Release();\
            }
```

This altered macro makes sure that the variable given for *p* is set to NULL *before* the *Release* call occurs. This prevents reentrancy problems where the call to *Release* ends up calling other code that uses the variable in *p* which can, of course, cause interesting problems.

INOLE\ANSI.CPP, line 74

The original code calculates a string length as follows:

```
cch=wcslen(pszW)+1;
```

However, this does not work correctly with MBCS ANSI characters. The correct code is as follows:

```
//This calculates the number of MBCS characters we'll need
cch=1+WideCharToMultiByte(CP_ACP, 0, pszW, -1, NULL, 0, NULL, NULL);
```

INOLE\HELPERS.CPP, line 479, 541

After the line of code (line 479):

```
CoTaskMemFree(pszw);
```

add the line:

```
pszw=NULL;
```

Also place the line of code (line 541)

```
CoTaskMemFree(pszw);
```

under a condition to have it read:

```
if (NULL!=pszw)
    CoTaskMemFree(pszw);
```

This prevents the cleanup code at the end of the function from attempting to free the memory that has already been freed.

INTERFACE, various

Many of the files in this directory don't actually compile due to small typos in the code. The updated sources available with this document fixes the errors.

CHAP02\ENUMRECT\ENUMRECT.CPP, lines 150-154

The code:

```
while (pApp->m_pIEnumRect->Next(1, &rc, &cRect))
    {
    if (!pApp->m_pIEnumRect->Skip(2))
        break;
    }
```

should have an error code on the *Next* call:

```
while (NOERROR==pApp->m_pIEnumRect->Next(1, &rc, &cRect))
    {
    if (NOERROR!=pApp->m_pIEnumRect->Skip(2))
        break;
    }
```

This is because *Next* returns NOERROR which has the value of zero, so the original code never iterates over the full enumeration. An explicit check for NOERROR is therefore necessary. Note that using the SUCCEEDED macro is not correct in this case since *Next* is allowed to return S_FALSE which would also return TRUE from SUCCEEDED.

CHAP02\REUSE\ANIMAL.CPP, lines 47-51

The *CreateAnimal* function incorrectly returns FALSE in two cases instead of the appropriate HRESULT. See text correction for page 140. The correct error codes are E_OUTOFMEMORY and E_FAIL:

```
HRESULT CreateAnimal(IUnknown *pUnkOuter, REFIID riid, void **ppv)
    {
```

```
    CAnimal    *pObj;

    //If aggregation is on, check riid==IID_IUnknown
    if (NULL!=pUnkOuter && riid!=IID_IUnknown)
        return ResultFromScode(CLASS_E_NOAGGREGATION);

    pObj=new CAnimal(pUnkOuter);

    if (NULL==pObj)
        return E_OUTOFMEMORY;

    if (!pObj->Init())
        return E_FAIL;

    return pObj->QueryInterface(riid, (PPVOID)ppv);
    }
```

CHAP04\CONNECT\CONNECT.CPP, lines 91, 125

Insert code to disconnect any connected sinks from and object before that object is destroyed.

At lines 90-91, the code should read:

```
if (NULL!=pApp->m_pObj)
    {
    //Make sure the sinks disconnect
    pApp->Disconnect(SINK1);
    pApp->Disconnect(SINK2);
    pApp->m_pObj->Release();
    }
```

Then before line 125, make the same calls to *pApp->Disconnect*:

```
//Make sure the sinks disconnect
pApp->Disconnect(SINK1);
pApp->Disconnect(SINK2);

if (0==pApp->m_pObj->Release())
    ...
```

CHAP05\DKOALA1\DKOALA1.CPP, lines 101-102, 213

All in-process server samples in the *Inside OLE* samples should increment the global object count, *g_cObj* within *DllGetClassObject* to include class factories in the object count that determines the answer returned from *DllCanUnloadNow*. The original samples did not increment the count. Therefore amend lines 101-102 as follows:

```
if (FAILED(hr))
    delete pObj;
else
    g_cObj++;
```

In addition, add a call to *ObjectDestroyed* after *delete this* in *CKoalaClassFactory::Release* at line 213 to make *Release* appear as follows:

```
STDMETHODIMP_(ULONG) CKoalaClassFactory::Release(void)
    {
    if (0L!=--m_cRef)
        return m_cRef;

    delete this;
```

```
        ObjectDestroyed();
        return 0L;
        }
```

See text corrections for pages 241-242.

CHAP05\DKOALA2\DKOALA2.CPP, lines 105-106, 378

See CHAP05\DKOALA1\DKOALA1.CPP, lines 101-102, 213 above.

CHAP05\DKOALA3\DKOALA3.CPP, lines 198-199, 312

See CHAP05\DKOALA1\DKOALA1.CPP, lines 101-102, 213 above.

CHAP05\POLYLINE\DLLPOLY.CPP, lines 142, 250

See CHAP05\DKOALA1\DKOALA1.CPP, lines 101-102, 213 above. The code affected is to add this line at line 142:

```
g_cObj++;
```

and this line after line 250 (the *delete this*):

```
ObjectDestroyed();
```

CHAP06\EKOALA5\KOALA.CPP, line 208

Add one line to *QueryInterface* such that it returns the *IUnknown* pointer for *IAnimal* and *IKoala*. For a full discussion of this change, see the Appendix at the end of this document.

```
        if (IID_IUnknown==riid || IID_IMarshal==riid
            || IID_IKoala==riid ||  IID_IAnimal==riid)
            *ppv=this;
```

CHAP06\KOALAPRX\KOALAPRX.DLL, lines 100-101, 212

See CHAP05\DKOALA1\DKOALA1.CPP, lines 101-102, 213 above (changes happen to *CKoalaProxyFactory::Release* instead of *CKoalaClassFactory::Release*)

CHAP07\COSMO\POLYLINE.CPP, line 306

Change the error return code from POLYLINE_E_READFAILURE to POLYLINE_E_WRITEFAILURE as described also for page 383 of the text.

CHAP08\POLYLINE\DLLPOLY.CPP, lines 144, 251

See CHAP05\POLYLINE\DLLPOLY.CPP, lines 142, 250 above.

CHAP08\POLYLINE\IPERSTMI.CPP, lines 154, 196, 200

Delete the call to *pIStream->Release* from both *CImpIPersistStreamInit::Load* (line 154) and *CImpIPersistStreamInit::Save* functions. Also within *Save* explicitly set *m_pObj->m_fDirty* to FALSE when *fClearDirty* is TRUE. The functions should appear as follows:

```
STDMETHODIMP CImpIPersistStreamInit::Load(LPSTREAM pIStream)
    {
    POLYLINEDATA    pl;
    ULONG           cb;
    HRESULT         hr;
```

```
    if (NULL==pIStream)
        return ResultFromScode(E_POINTER);

    //Read all the data into the POLYLINEDATA structure.
    hr=pIStream->Read(&pl, CBPOLYLINEDATA, &cb);

    if (FAILED(hr) || CBPOLYLINEDATA!=cb)
        return hr;

    m_pObj->m_pImpIPolyline->DataSet(&pl, TRUE, TRUE);
    return NOERROR;
    }


STDMETHODIMP CImpIPersistStreamInit::Save(LPSTREAM pIStream
    , BOOL fClearDirty)
    {
    POLYLINEDATA    pl;
    ULONG           cb;
    HRESULT         hr;

    if (NULL==pIStream)
        return ResultFromScode(E_POINTER);

    m_pObj->m_pImpIPolyline->DataGet(&pl);

    hr=pIStream->Write(&pl, CBPOLYLINEDATA, &cb);
    //Call to Release removed from here
    if (FAILED(hr) || CBPOLYLINEDATA!=cb)
        return ResultFromScode(STG_E_WRITEFAULT);

    if (fClearDirty)
        m_pObj->m_fDirty=FALSE;

    return NOERROR;
    }
```

See also text corrections for pages 421-422

CHAP09\IDESCRIP\IDESCRIP.IDL, line 22

The IDL for the IDescription interface is lacking a *size_is(cch)* attribute on the *pszText* argument which ends up causing some crashing on Win95. In addition, the type for the string should be *LPOLESTR* (Unicode) instead of *LPTSTR* (ANSI or Unicode depending on compilation) to follow the OLE convention that all strings are in Unicode. The new IDL is as follows:

```
[uuid(00021152-0000-0000-c000-000000000046),
    object,
    pointer_default(unique)
]
interface IDescription : IUnknown
    {
    import "unknwn.idl";

    HRESULT GetText([in, out, size_is(cch)] LPOLESTR pszText, [in] ULONG
cch);
    }
```

This precipitates changes to INC\IDESCRIP.H, CHAP09\IDESCRIP\IDESCRIP.H, as well as CHAP09\LINKSRC and CHAP09\LINKUSER samples, as described below.

CHAP09\LINKSRC\OBJECTS.H, line 247

Change "LPTSTR" to "LPOLESTR" to reflect the change to CHAP09\IDESCRIP\IDESCRIP.IDL line 22 above. This line now reads:

```
STDMETHODIMP GetText(LPOLESTR, ULONG);
```

CHAP09\LINKSRC\IDESCRIP.CPP, lines 63-103

To reflect the change made to CHAP09\IDESCRIP\IDESCRIP.IDL line 22 above, this function now appears as follows:

```
/*
 * CImpIDescription::GetText
 *
 * Purpose:
 *  Fills a buffer with our text description.
 *
 * Parameters:
 *  pszText         LPOLESTR to the buffer to fill
 *  cch             ULONG specifying the length of pszText
 *
 * Return Value:
 *  HRESULT         NOERROR if successful, error otherwise.
 */

HRESULT CImpIDescription::GetText(LPOLESTR pszText, ULONG cch)
    {
    HRESULT     hr;
    IStream    *pIStream;

    /*
     * The description text for this object is contained in
     * a stream called "Description" (constant SZDESCRIPTION
     * has this string) found in whatever storage we happen
     * to have.  This implementation is ignorant of the actual
     * object that is exposing it.
     */

    if (NULL==m_pIStorage)
        return ResultFromScode(E_FAIL);

    hr=m_pIStorage->OpenStream(SZDESCRIPTION, 0, STGM_DIRECT
        | STGM_READ | STGM_SHARE_EXCLUSIVE, 0, &pIStream);

    if (FAILED(hr))
        return ResultFromScode(E_FAIL);

   #ifdef WIN32ANSI
    char sz[512];
    hr=pIStream->Read((void *)sz, cch*sizeof(TCHAR), NULL);
    MultiByteToWideChar(CP_ACP, 0, sz, -1, pszText, cch);
   #else
    hr=pIStream->Read((void *)pszText, cch*sizeof(WCHAR), NULL);
   #endif
    pIStream->Release();
```

```
    return SUCCEEDED(hr) ? NOERROR : ResultFromScode(E_FAIL);
    }
```

That is, since the *pszText* argument is *always* Unicode now, ANSI compilations need to convert the text read from storage (which is stored in ANSI) into Unicode, which wasn't necessary before.

Note that LinkSrc will generate a file with ANSI characters or Unicode characters in its streams depending on whether you compile ANSI or Unicode. The change above to *IDescription::GetText* requires the implementation to convert to Unicode only in ANSI compilation.

CHAP09\LINKSRC\LINKSRC.RC, line 14

Inside this line to enable the sample to build under the VC++ 2.x or 4.x IDE, otherwise you pull in too many headers in the resource compiler:

```
#define RPC_NO_WINDOWS_H
```

This is not needed for command-line compilations.

CHAP09\LINKUSER\LINKUSER.CPP, line 501

Replace the single call:

```
 hr=pIDescription->GetText(szText, cch);
```

with code that always assumes Unicode text, according to the change made to CHAP09\IDESCRIP\IDESCRIP.IDL line 22 above:

```
#ifdef WIN32ANSI
 WCHAR       szw[cch];
 hr=pIDescription->GetText(szw, cch);
 WideCharToMultiByte(CP_ACP, 0, szw, -1, szText, cch, NULL, NULL);
#else
 hr=pIDescription->GetText(szText, cch);
#endif
```

CHAP09\LINKUSER\LINKUSER.RC, line 15

Inside this line to enable the sample to build under the VC++ 2.x or 4.x IDE, otherwise you pull in too many headers in the resource compiler:

```
#define RPC_NO_WINDOWS_H
```

This is not needed for command-line compilations.

CHAP10\DDATAOBJ\DDATAOBJ.DLL, lines 119-120, 235

See CHAP05\DKOALA1\DKOALA1.CPP, lines 101-102, 213 above (changes happen to *CDataObjectClassFactory::Release* instead of *CKoalaClassFactory::Release*)

CHAP10\POLYLINE\DLLPOLY.CPP, lines 144, 252

See CHAP05\POLYLINE\DLLPOLY.CPP, lines 142, 250 above.

CHAP10\POLYLINE\IPERSTMI.CPP, lines 154, 202, 207

See CHAP08\POLYLINE\IPERSTMI.CPP, lines 154, 196, 200 above.

CHAP12\DATATRAN\DATATRAN.CPP, lines 113, 227

See CHAP05\POLYLINE\DLLPOLY.CPP, lines 142, 250 above.

CHAP12\PATRON\PAGEMOUS.CPP, line 465

There are some specialty mouse drivers that alter the frequency of the WM_NCHITTEST message being sent to a window. With some of these drivers, this message is set within mouse-tracking loops inside Patron to the effect that resizing and object is not possible—*CPage::OnNCHitTest* is called too often, wiping out the value in *m_uSizingFlags*.

To account for this behavior, alter the line of code that exists at this location from:

```
if (m_fSizePending)
    return;
```

to:

```
if (m_fSizePending || m_fTracking)
    return;
```

CHAP13\PATRON\PAGEMOUS.CPP, line 526

For the same reasons described above for CHAP12\PATRON\PAGEMOUS.CPP line 465, add these lines of code. In the original code for Patron in Chapters 13 and beyond the check on m_fSizePending does not appear, but should to match the code in Chapter 12. Therefore this entire code block is an addition:

```
/*
 * Ignore this message if it occurs during tracking to adjust
 * for the behavior of oddball mouse drivers.
 */
if (m_fSizePending || m_fTracking)
    return;
```

This change also affects PAGEMOUS.CPP in Chapters 17 (line 638), 20 (line 636), 21 (line 634), 22 (672), and 24 (line 673). In these later chapters, the entire two lines above, including the check on

CHAP14\BEEPER1\DBEEPER.CPP, lines 93-94,184

See CHAP05\DKOALA1\DKOALA1.CPP, lines 101-102, 213 above (changes happen to *CBeeperClassFactory::Release* instead of *CKoalaClassFactory::Release*)

CHAP14\BEEPER1\BEEPER.CPP, line 576

The code under the *default* case is missing a *return*. The code should read as follows:

```
default:
    return ResultFromScode(DISP_E_MEMBERNOTFOUND);
```

CHAP14\BEEPER2\DBEEPER.CPP, lines 93-94, 184

See CHAP14\BEEPER1\DBEEPER.CPP, lines 93-94,184 above.

CHAP14\BEEPER2\WIN32.REG and WIN16.REG, lines 21-22

The DIR and HELPDIR entries are now places under the version number are they should be. See text error for page 170. In the file itself, these two entries are fixed and moved below the version key entry.

CHAP14\BEEPER3\DBEEPER.CPP, lines 99-100, 190

See CHAP14\BEEPER1\DBEEPER.CPP, lines 93-94,184 above.

CHAP14\BEEPER3\WIN32.REG and WIN16.REG, lines 21-22

The DIR and HELPDIR entries are now places under the version number are they should be. See text error for page 170. In the file itself, these two entries are fixed and moved below the version key entry.

CHAP14\BEEPER3a\DBEEPER.CPP, lines 99-100, 190

See CHAP14\BEEPER1\DBEEPER.CPP, lines 93-94,184 above.

CHAP14\BEEPER3a\WIN32.REG and WIN16.REG, lines 21-22

The DIR and HELPDIR entries are now places under the version number are they should be. See text error for page 170. In the file itself, these two entries are fixed and moved below the version key entry.

CHAP14\BEEPER4\DBEEPER.CPP, lines 99-100, 190

See CHAP14\BEEPER1\DBEEPER.CPP, lines 93-94,184 above.

CHAP14\BEEPER4\WIN32.REG and WIN16.REG, lines 21-22

The DIR and HELPDIR entries are now places under the version number are they should be. See text error for page 170. In the file itself, these two entries are fixed and moved below the version key entry.

CHAP14\BEEPER5\DBEEPER.CPP, lines 89-90, 180

See CHAP14\BEEPER1\DBEEPER.CPP, lines 93-94,184 above.

CHAP14\BEEPER5\WIN32.REG and WIN16.REG, lines 21-22

The DIR and HELPDIR entries are now places under the version number are they should be. See text error for page 170. In the file itself, these two entries are fixed and moved below the version key entry.

CHAP14\COSMO\ICLASSF.CPP, line 81

The first line of *CClassFactory::Release* reading:

```
if (0!=--m_cRef)
```

should be:

```
if (0=--m_cRef)
```

That is, = instead of !=.

CHAP14\COSMO\AUTOAPP.CPP, line 80

Replace the line:

```
//CCosmoFrame deletes this object during shutdown
```

with:

```
delete this;
```

This matches the change made to CHAP14\COSMO\COSMO.CPP line 152 below.

CHAP14\COSMO\COSMO.CPP, line 152

Delete the line of code that reads:

```
DeleteInterfaceImp(m_pAutoApp);
```

This change matches that made to CHAP14\COSMO\AUTOAPP.CPP, line 80 above.

CHAP14\COSMO\COSMO.CPP, line 295

Insert an *AddRef* call on the application object to balance the Release that's going to happen in *IExternalConnection::ReleaseConnection:*

```
RegisterActiveObject((IUnknown *)m_pAutoApp
    , CLSID_Cosmo2Application, ACTIVEOBJECT_STRONG
   , &m_dwActiveApp);

/*
 * This is necessary to balance the Release in
 * IExternalConnection::ReleaseConnection
 */
m_pAutoApp->AddRef();
```

CHAP15\AUTOCLI\AUTOCLI.CPP, lines 101, 183

Add a *VariantInit(&va)* at these line before the calls to *Invoke*. This should be done when local servers are used, otherwise the *Invoke* call returns an error.

CHAP15\AUTOCLI\AUTOCLI.CPP, lines 241-247

Replace this code:

```
/*
 * Get the current LCID to send to the automation object.
 * Note that depending on the Beeper installed, this may or
 * may not work, especially if you are in a non-english or
 * non-German speaking locale.
 */
m_lcid=GetUserDefaultLCID();
```

with this:

```
m_lcid=LANG_NEUTRAL;
```

This reflects the statement on page 744 of the text "With AutoCli, we always use basic English with LANGID_ENGLISH and SUBLANGID_NEUTRAL. The LCID for this is stored in the variable *m_lcid* in the application's constructor, *CApp::CApp*, and is passed later to *IDispatch* methods." Without this change, the sample does not work on machines in Germany, for instance.

CHAP15\AUTOCLI\AUTOCLI.CPP, line 482

The calculated value of *cch* based on *wcslen(pExInfo->bstrDescription)* is one too short. It needs an extra "1" added to account for the null terminator:

```
cch=wcslen(pExInfo->bstrDescription)+1;
```

CHAP16\BEEPER6\DBEEPER.CPP, lines 93-94, 184

See CHAP14\BEEPER1\DBEEPER.CPP, lines 93-94,184 above.

CHAP16\BEEPER6\WIN32.REG and WIN16.REG, lines 21-22

The DIR and HELPDIR entries are now places under the version number are they should be. See text error for page 170. In the file itself, these two entries are fixed and moved below the version key entry.

CHAP16\BEEPPROP\BEEPPROP.CPP, lines 99-100, 164

See CHAP14\BEEPER1\DBEEPER.CPP, lines 93-94,184 above except that the *ObjectDestroyed* call in *CBeeperPPFactory::Release* is instead:

```
g_cObj--;
```

CHAP17\PATRON\PAGEMOUS.CPP, line 638

See CHAP13\PATRON\PAGEMOUSE.CPP, line 526 above.

CHAP18\COSMO\ICLASSF.CPP, line 77

The first line of *CFigureClassFactory::Release* reading:

```
if (0!=--m_cRef)
```

should be:

```
if (0=--m_cRef)
```

That is, = instead of !=.

CHAP19\HCOSMO\HCOSMO.CPP, lines 100, 208

See CHAP05\POLYLINE\DLLPOLY.CPP, lines 142, 250 above.

CHAP19\HCOSMO\IADVSINK.CPP, lines 78, 84

The advise sink that HCosmo uses to receive data change notifications from the default handler is conceptually a separate object although the *IAdviseSink* interface is implemented on the Figure object itself. While the *QueryInterface* function for this interface makes the advise sink look like a separate object, its *AddRef* and *Release* members incorrectly include the Figure object in the reference count. This leads to a memory leak where the object is never fully released, due to the call to *IDataObject::DAdvise* that is made from *CFigure::Init* (FIGURE.CPP line 216).

The advise sink, however, is completely manage by the Figure object which creates and destroys it as needed. Therefore remove the call to *m_pUnkOuter->AddRef()* from *CImpIAdviseSink::AddRef* (line 78) as well as the *m_pUnkOuter->Release()* call from *CImpIAdviseSink::Release* (line 84). These functions then appear as follows:

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

STDMETHODIMP_(ULONG) CImpIAdviseSink::Release(void)
    {
    return --m_cRef;
    }
```

The object itself will be destroyed through a combination of the *m_pDefIDataObject->Release()* call (FIGURE.CPP line 79) and the *DeleteInterfaceImp(m_pImpIAdviseSink)* call (FIGURE.CPP line 98), both inside *CFigure::~CFigure*.

CHAP19\POLYLINE\DLLPOLY.CPP, lines 147, 275

See CHAP05\POLYLINE\DLLPOLY.CPP, lines 142, 250 above.

CHAP19\POLYLINE\POLYLINE.CPP, lines 102-113

Add a safety reference count around the calls that release aggregated pointers obtained from the default handler. This is necessary to prevent reentrancy crashes when other memory allocators (such as MFC's debug allocator) are in use.

```
/*
 * In aggregation, release cached pointers but
 * AddRef the controlling unknown first.  The
 * extra reference count protects us from reentrancy.
 */

m_cRef++;

pIUnknown->AddRef();
pIUnknown->AddRef();
pIUnknown->AddRef();

ReleaseInterface(m_pDefIViewObject);
ReleaseInterface(m_pDefIDataObject);
ReleaseInterface(m_pDefIPersistStorage);

m_cRef--;
```

The same fix also applies to Polyline in Chapters 21, 23, and 24.

CHAP19\POLYLINE\IPERSTMI.CPP, lines 154, 196, 201

See CHAP08\POLYLINE\IPERSTMI.CPP, lines 154, 196, 200 above.

CHAP20\LNKASSIS\DLLASSIS.CPP, lines 86, 178

See CHAP05\POLYLINE\DLLPOLY.CPP, lines 142, 250 above.

CHAP20\PATRON\DOCUMENT.CPP, line 506

Move this line:

```
Rename(pszFile); //Update caption bar
```

from line 506 (after *FDirtySet(FALSE)*) to line 493, before *pIStorage->Commit(STGC_DEFAULT);*. This change prevents Patron from thinking the document is dirty after a save.

CHAP20\PATRON\PAGEMOUS.CPP, line 636

See CHAP13\PATRON\PAGEMOUSE.CPP, line 526 above.

CHAP21\COSMO\ICLASSF.CPP, line 77

See CHAP18\COSMO\ICLASSF.CPP, line 77

CHAP21\PATRON\DOCUMENT.CPP, lines 654-657

Move these lines:

```
//CHAPTER20MOD
if (m_fRename)
    Rename(pszFile); //Update caption bar
//End CHAPTER20MOD
```

from lines 654-657 (after *FDirtySet(FALSE)*) to line 640, before *pIStorage->Commit(STGC_DEFAULT);*. This change prevents Patron from thinking the document is dirty after a save.

CHAP21\PATRON\PAGEMOUS.CPP, line 634

See CHAP13\PATRON\PAGEMOUSE.CPP, line 526 above.

CHAP21\POLYLINE\DLLPOLY.CPP, lines 145, 273

See CHAP05\POLYLINE\DLLPOLY.CPP, lines 142, 250 above.

CHAP21\POLYLINE\POLYLINE.CPP, lines 101-112

See CHAP19\POLYLINE\POLYLINE.CPP lines 102-113 above.

CHAP21\POLYLINE\IPERSTMI.CPP, lines 154, 196, 201

See CHAP08\POLYLINE\IPERSTMI.CPP, lines 154, 196, 200 above.

CHAP22\PATRON\DOCUMENT.CPP, lines 700-701

Move these lines:

```
if (m_fRename)
    Rename(pszFile); //Update caption bar
```

from lines 700-701 (after *FDirtySet(FALSE)*) to line 686, before *pIStorage->Commit(STGC_DEFAULT);*. This change prevents Patron from thinking the document is dirty after a save.

CHAP22\PATRON\PAGEMOUS.CPP, line 672

See CHAP13\PATRON\PAGEMOUSE.CPP, line 526 above.

CHAP23\COSMO\ICLASSF.CPP, line 77

See CHAP18\COSMO\ICLASSF.CPP, line 77

CHAP23\POLYLINE\DLLPOLY.CPP, lines 150, 278

See CHAP05\POLYLINE\DLLPOLY.CPP, lines 142, 250 above.

CHAP23\POLYLINE\POLYLINE.CPP, lines 114-125

See CHAP19\POLYLINE\POLYLINE.CPP lines 102-113 above.

CHAP23\POLYLINE\POLYLINE.CPP, lines 994-1021, 1079-1083, 1115, 1169

The *CPolyline::InPlaceActivate* calls *IOleInPlaceFrame::SetActiveObject* and *IOleInPlaceUIWindow::SetActiveObject* even when the object is *not* doing UI-active but only in-place active. These steps should be done only within *CPolyline::UIActivate.* Therefore these lines of code (994-1021) have been moved to line 1115 (of the original file). In addition, the reverse calls to *SetActiveObject* originally found in *CPolyline::InPlaceDeactivate* (lines 1079-1083) have been moved to line 1169 inside *CPolyline::UIDeactivate*.

CHAP23\POLYLINE\IOLEOBJ.CPP, lines 320-321

Add the following code (in blue) to the OLEIVERB_SHOW case in *CImpIOleObject::DoVerb* to match the hiding of the hatch window in OLEIVERB_HIDE:

```
if (NULL!=m_pObj->m_pIOleIPSite)
    {
    if (NULL!=m_pObj->m_pHW)
        ShowWindow(m_pObj->m_pHW->Window(), SW_SHOW);

    return NOERROR; //Already active
    }
```

CHAP23\POLYLINE\IPERSTMI.CPP, lines 154, 196, 201

See CHAP08\POLYLINE\IPERSTMI.CPP, lines 154, 196, 200 above.

CHAP24\PATRON\DOCUMENT.CPP, lines 691-692

See CHAP24\PATRON\DOCUMENT.CPP, lines 700-701 above. Destination line is 677.

CHAP24\PATRON\PAGEMOUS.CPP, line 673

See CHAP13\PATRON\PAGEMOUSE.CPP, line 526 above.

CHAP24\POLYLINE\DLLPOLY.CPP, lines 148, 276

See CHAP05\POLYLINE\DLLPOLY.CPP, lines 142, 250 above.

CHAP24\POLYLINE\POLYLINE.CPP, lines 128-139

See CHAP19\POLYLINE\POLYLINE.CPP lines 102-113 above.

CHAP23\POLYLINE\IOLEOBJ.CPP, lines 332-333

Same as for CHAP23\POLYLINE\IOLEOBJ.CPP, lines 320-321 above.

CHAP24\POLYLINE\POLYLINE.CPP, lines 1085-1086

Delete these two lines (shown below) that are redundant with *CPolyline::UIActivate* and belong inside *UIActivate* entirely:

```
//Critical for accelerators to work initially.
SetFocus(hWndHW);
```

CHAP23\POLYLINE\POLYLINE.CPP, lines 1088-1115, 1173-1177, 1213, 1260

See CHAP23\POLYLINE\POLYLINE.CPP, lines 994-1021, 1079-1083, 1115, 1169

CHAP24\POLYLINE\IPERSTMI.CPP, lines 154, 196, 201

See CHAP08\POLYLINE\IPERSTMI.CPP, lines 154, 196, 200 above.

CHAP24\POLYLINE\WIN32.REG and WIN16.REG, lines 41-42

The DIR and HELPDIR entries are now places under the version number are they should be. See text error for page 170. In the file itself, these two entries are fixed and moved below the version key entry.

CHAP24\POLYPROP\POLYPROP.CPP, lines 101-102, 164

See CHAP14\BEEPER1\DBEEPER.CPP, lines 93-94,184 above except that the *ObjectDestroyed* call in *CPolyPPFactory::Release* is instead:

```
g_cObj--;
```

Appendix: EKOALA5 Custom Marshaling Sample Error

This is an extract from the Microsoft Knowledge Base describing an incorrect usage of custom marshaling with the EKOALA5 and KOALAPRX samples of Chapter 5. The contents of this article apply to any product code based on these samples. Thanks to Jeffrey Saarhoff for this information.

SYMPTOMS

It is only possible to instantiate the Koala object implemented by the EKOALA5 sample if the client initially asks for the IUnknown interface. In this case, subsequent calls to QueryInterface to retrieve pointers to the custom interfaces *IKoala* or *IAnimal* will also succeed. This can be observed by using the OBJUSER3 client sample included with Chapter 6.

However, if the client initially asks for *IKoala* or *IAnimal*, then *IClassFactory::CreateInstance* and *CoCreateInstance* will fail with E_NOINTERFACE. This can be observed by modifying OBJUSER3 to ask for one of these interfaces when calling *CoCreateInstance*.

CAUSE

EKOALA5 does not actually implement *IKoala* or *IAnimal* as interfaces, and returns E_NOINTERFACE when queried for those interfaces. This causes EKOALA5's *IClassFactory::CreateInstance* and *CoCreateInstance* to fail.

Note that when an object is custom marshaled, its interfaces need only exist in the client's process. Thus, the proxy must implement the object's interfaces, but it is optional for the server (implementation dependent).

The error in this case is that the server claims the object does not support *IKoala* and *IAnimal* when in fact it does (through the proxy).

RESOLUTION

One solution would be to add interface implementations for *IKoala* and *IAnimal* to EKOALA5 and modify *CKoala::HandleCall* to simply call the appropriate interface method for each request.

However, an easier solution is to modify *CKoala::QueryInterface* to return *IUnknown* when queried for *IKoala* or *IAnimal*, even though the *CKoala* does not implement these interfaces.

CAUTION: This solution is specific to the peculiar architecture of the EKOALA5 sample. Returning *IUnknown* in place of unimplemented interfaces can easily cause a crash in other situations.

In KOALA.CPP:

```
STDMETHODIMP CKoala::QueryInterface(REFIID riid, PPVOID ppv)
    {
    *ppv=NULL;

    if (IID_IUnknown==riid || IID_IMarshal==riid)
        *ppv=this;
    else if (IID_IKoala==riid ||  IID_IAnimal==riid)
        *ppv=this;

    if (NULL!=*ppv)
        {
        ((LPUNKNOWN)*ppv)->AddRef();
        return NOERROR;
```

```
    }

return ResultFromScode(E_NOINTERFACE);
}
```

Note that the OLE stub in the server process will never directly call interface methods on an object that is custom marshaled, except for *IUnknown* and *IMarshal* methods. Other interface method calls are handled in the client process by the proxy (perhaps using private communication with the server). Thus even if the server does not implement certain interfaces supported by the object, as is the case with EKOALA5, it is safe to return *IUnknown* when queried for those interfaces.

In general, it is incorrect for a server to disallow a request for an interface from *CoCreateInstance* but allow it from the proxy.

MORE INFORMATION

The EKOALA5 server and corresponding proxy KOALAPRX implement custom marshaling for the Koala object, which provides the *IKoala* and *IAnimal* custom interfaces. These components are designed so that the proxy implements the custom interfaces in the client's process, but delegates to the server to do the actual work involved for some, but not all, of the interface methods. The server does not implement the interfaces themselves, but merely responds to specific requests from the proxy.

When creating the object, the requested interface is passed as a parameter to the server's *IClassFactory::CreateInstance* method. In EKOALA5, this method creates the object and then calls *QueryInterface*. If the requested interface is *IKoala* or *IAnimal*, *QueryInterface* fails causing *IClassFactory::CreateInstance* and *CoCreateInstance* to fail.

If the initially requested interface is *IUnknown*, *QueryInterface* and *CreateInstance* succeed. The *IUnknown* interface is returned first to the OLE stub, which then queries for *IMarshal* and proceeds to custom marshal the object back to the client by calling the *IMarshal* methods. On the client side, the proxy (KOALAPRX) is loaded and the object is unmarshaled. Any subsequent calls to *QueryInterface* are handled by the proxy, which implements *IKoala* and *IAnimal* so everything works normally from that point.

With the modifications noted in the "Resolution" section of this article, the creation process works the same when the client asks for *IKoala* or *IAnimal* using *CoCreateInstance* as it does when the client asks for *IUnknown*.