Uniform Data Transfer
-Clipboard Transfer and OLE Drag and Drop-Part 2

Program examples compiled using Visual C++ 6.0 compiler on Windows XP Pro machine with Service Pack 2. Topics and sub topics for this tutorial are listed below. Don't forget to read Tenouk's small disclaimer. The supplementary notes for this tutorial are cdib.h, cdib.cpp, formatetc and IDataObject.

Index:

MFC Drag and Drop

Drag and drop was the ultimate justification for the data object code you've been looking at. OLE supports this feature with its `IDropSource` and `IDropTarget` interfaces plus some library code that manages the drag-and-drop process. The MFC library offers good drag-and-drop support at the **view level**, so we'll use it. Be aware that drag-and-drop transfers are immediate and independent of the clipboard. If the user cancels the operation, there's no "memory" of the object being dragged.
Drag-and-drop transfers should work consistently between applications, between windows of the same application, and within a window. When the user starts the operation, the cursor should change to an arrow-rectangle combination. If the user holds down the **Ctrl** key, the cursor turns into a plus sign (+), which indicates that the object is being copied rather than moved.
MFC also supports drag-and-drop operations for items in compound documents. This is the next level up in MFC OLE support, and it's not covered in this Module. Look up the MSDN's OCLIENT example in the online documentation under Visual C++ Samples.

The Source Side of the Transfer

When your source program starts a drag-and-drop operation for a data object, it calls `COleDataSource::DoDragDrop`. This function internally creates an object of MFC class `COleDropSource`, which implements the `IOleDropSource` interface. `DoDragDrop()` is one of those functions that don't return for a while. It returns when the user drops the object or cancels the operation or when a specified number of milliseconds have elapsed.
If you're programming drag-and-drop operations to work with a `CRectTracker` object, you should call `DoDragDrop()` only when the user clicks inside the tracking rectangle, not on its border. `CRectTracker::HitTest` gives you that information. When you call `DoDragDrop()`, you need to set a flag that tells you whether the user is dropping the object into the same view (or document) that it was dragged from.

The Destination Side of the Transfer

If you want to use the MFC library's view class drag-and-drop support, you must add a data member of class `COleDropTarget` to your derived view class. This class implements the `IDropTarget` interface, and it holds an

`IDropSource` pointer that links back to the `COleDropSource` object. In your view's `OnInitialUpdate()` function, you call the `Register()` member function for the embedded `COleDropTarget` object.
After you have made your view a drop target, you must override four `CView` virtual functions, which the framework calls during the drag-and-drop operation. Here's a summary of what they should do, assuming that you're using a tracker.

Virtual function	Description
`OnDragEnter()`	Adjusts the focus rectangle and then calls `OnDragOver()`.
`OnDragOver()`	Moves the dotted focus rectangle and sets the drop effect (determines cursor shape).
`OnDragLeave()`	Cancels the transfer operation; returns the rectangle to its original position and size.
`OnDrop()`	Adjusts the focus rectangle and then calls the `DoPaste()` helper function to get formats from the data object.

Table 1.

The Drag-and-Drop Sequence

Figure 1 illustrates the MFC drag-and-drop process.

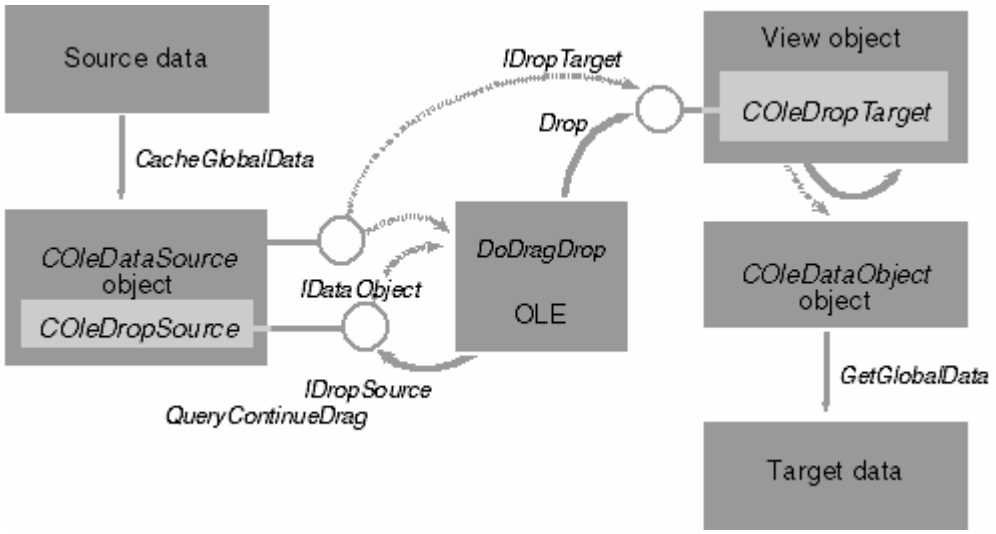

Figure 1: MFC OLE drag-and-drop processing.

Here's a summary of what's going on:

1. User presses the left mouse button in the source view window.
2. Mouse button handler calls `CRectTracker::HitTest` and finds out that the cursor was inside the tracker rectangle.
3. Handler stores formats in a `COleDataSource` object.
4. Handler calls `COleDataSource::DoDragDrop` for the data source.
5. User moves the cursor to the view window of the target application.
6. OLE calls `IDropTarget::OnDragEnter` and `OnDragOver()` for the `COleDropTarget` object, which calls the corresponding virtual functions in the target's view. The `OnDragOver()` function is passed a `COleDataObject` pointer for the source object, which the target tests for a format it can understand.
7. `OnDragOver()` returns a drop effect code, which OLE uses to set the cursor.
8. OLE calls `IDataSource::QueryContinueDrag` on the source side to find out whether the drag operation is still in progress. The MFC `COleDataSource` class responds appropriately.
9. User releases the mouse button to drop the object in the target view window.

10. OLE calls `IDropTarget::OnDrop`, which calls `OnDrop()` for the target's view. Because `OnDrop()` is passed a `COleDataObject` pointer, it can retrieve the desired format from that object.
11. When `OnDrop()` returns in the target program, `DoDragDrop()` can return in the source program.

The MYMFC30B Example: OLE Drag and Drop

This example picks up where the MYMFC30A example leaves off. It adds drag-and-drop support, using the existing `SaveDib()` and `DoPasteDib()` helper functions. All of the clipboard code is the same. You should be able to adapt MYMFC30B to other applications that require drag and drop for data objects.
To prepare MYMFC30B, open the mfcproject\Mymfc30b.dsw workspace and build the project. Run the application, and test drag and drop between child windows and between instances of the program.

The MYMFC30B Steps From Scratch

This is MDI application without ActiveX Controls and Automation support. The view base class is `CScrollView`.

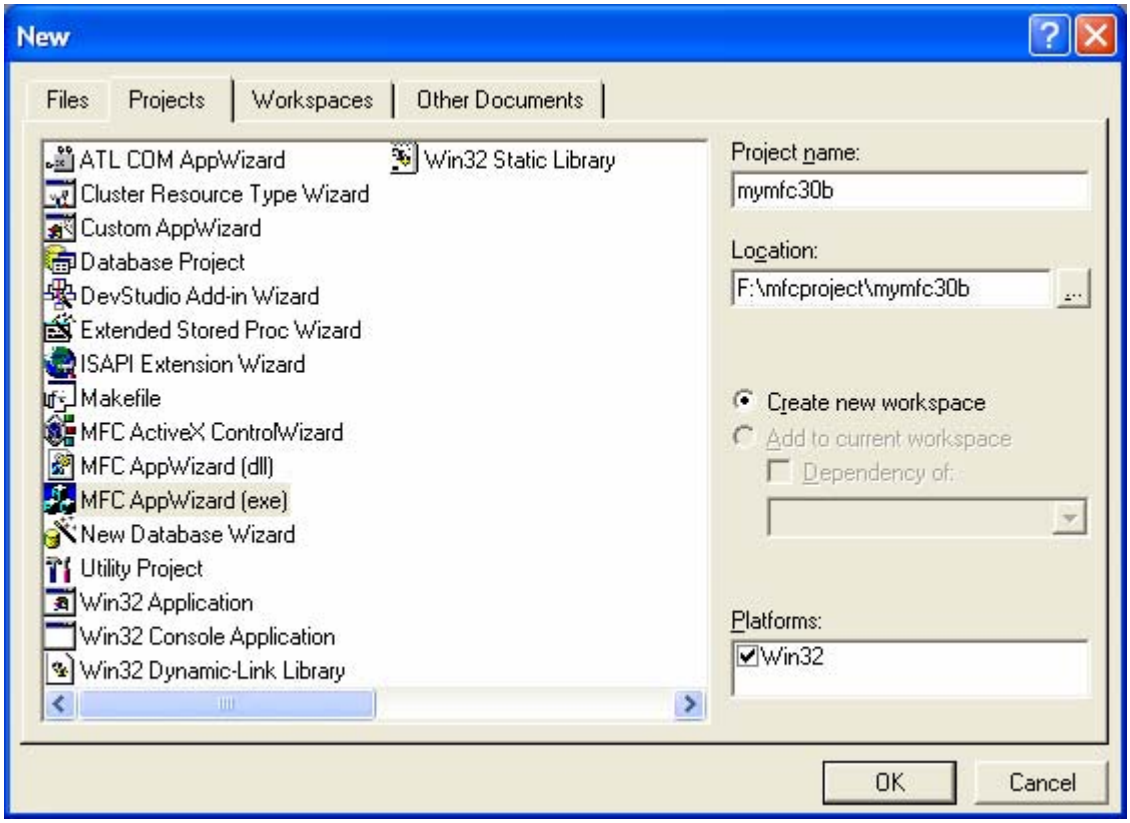

Figure 2: MYMFC30B – Visual C++ new project dialog.

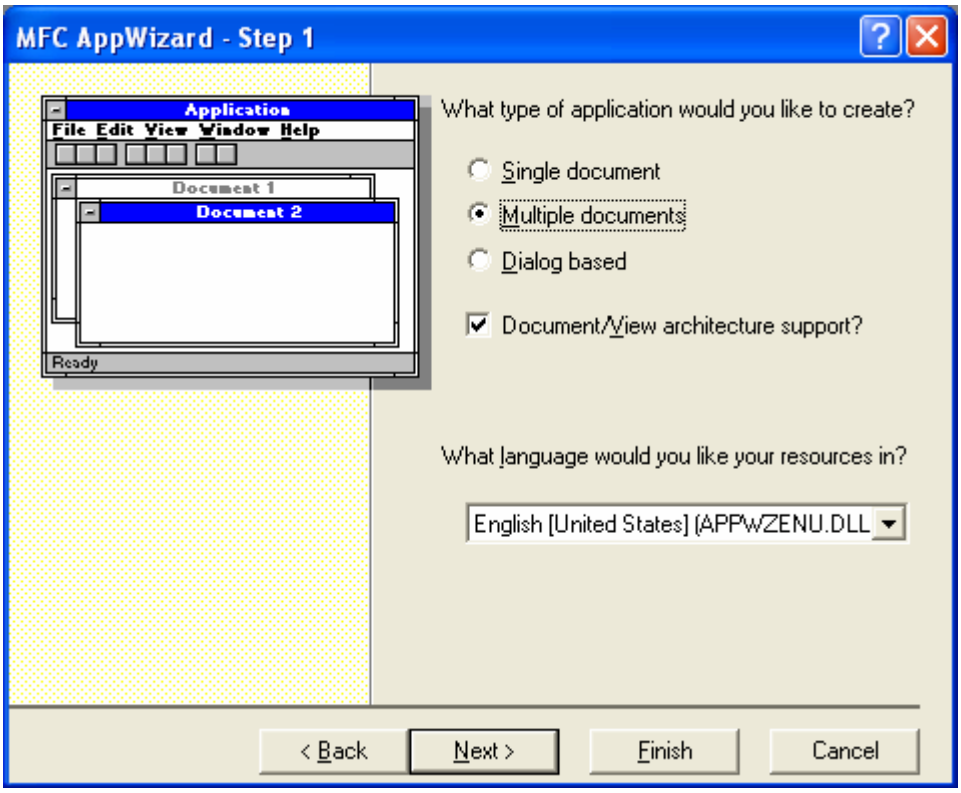

Figure 3: MYMFC30B – AppWizard step 1 of 6, select MDI application.

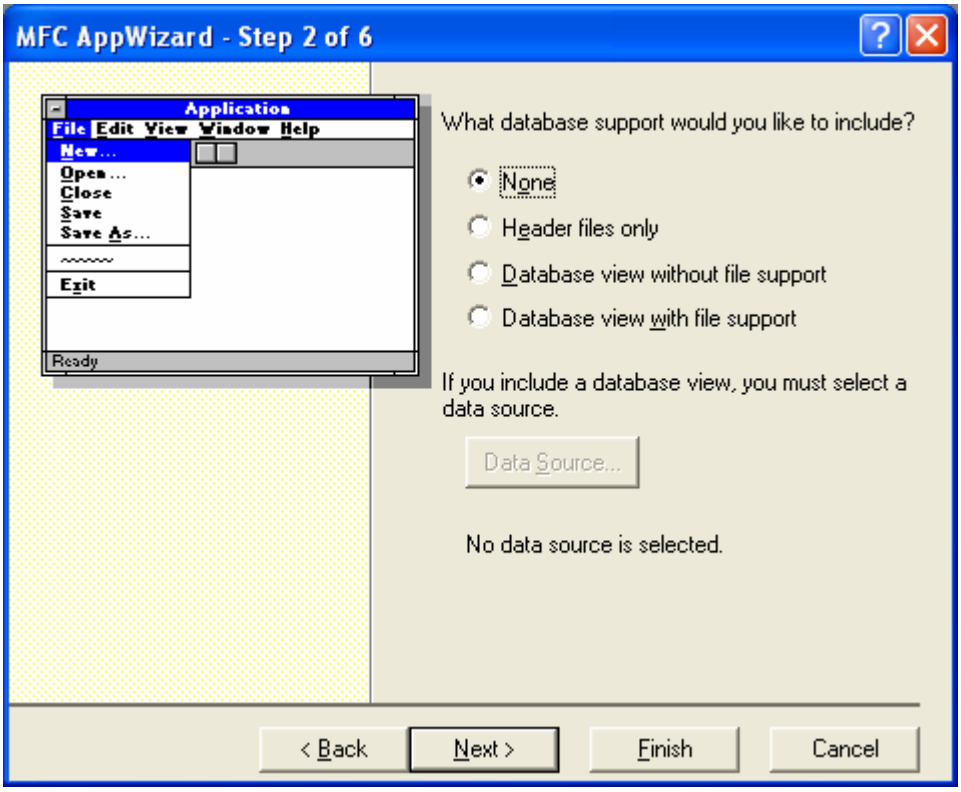

Figure 4: MYMFC30B – AppWizard step 2 of 6.

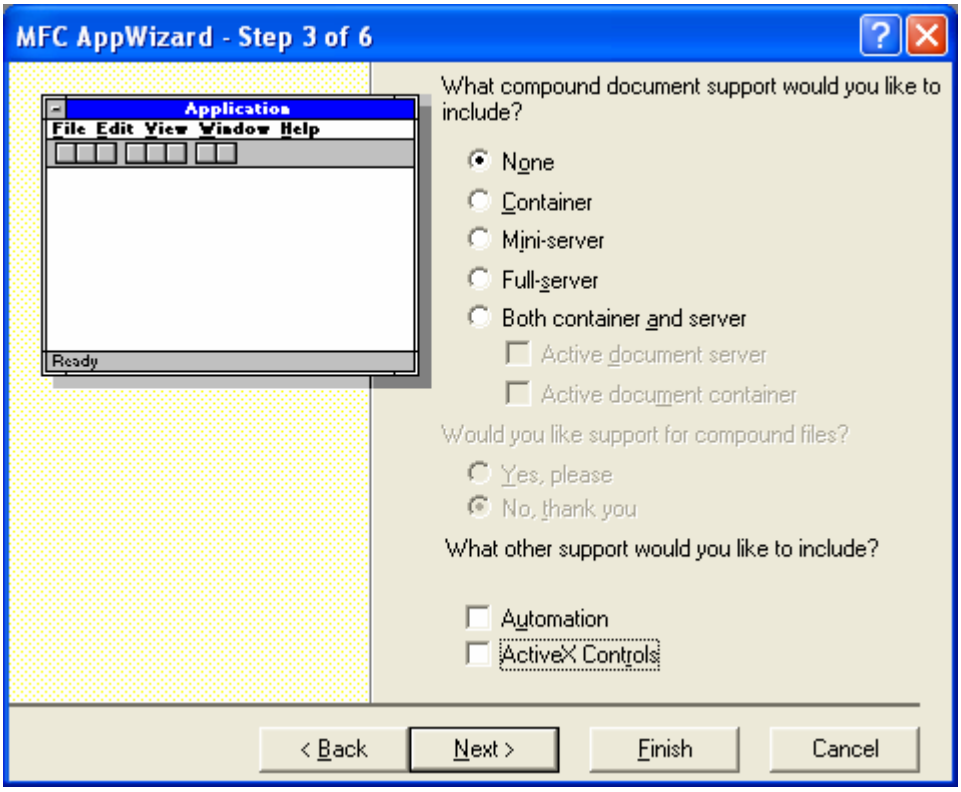

Figure 5: MYMFC30B – AppWizard step 3 of 6, deselect the Automation and ActiveX Controls options.

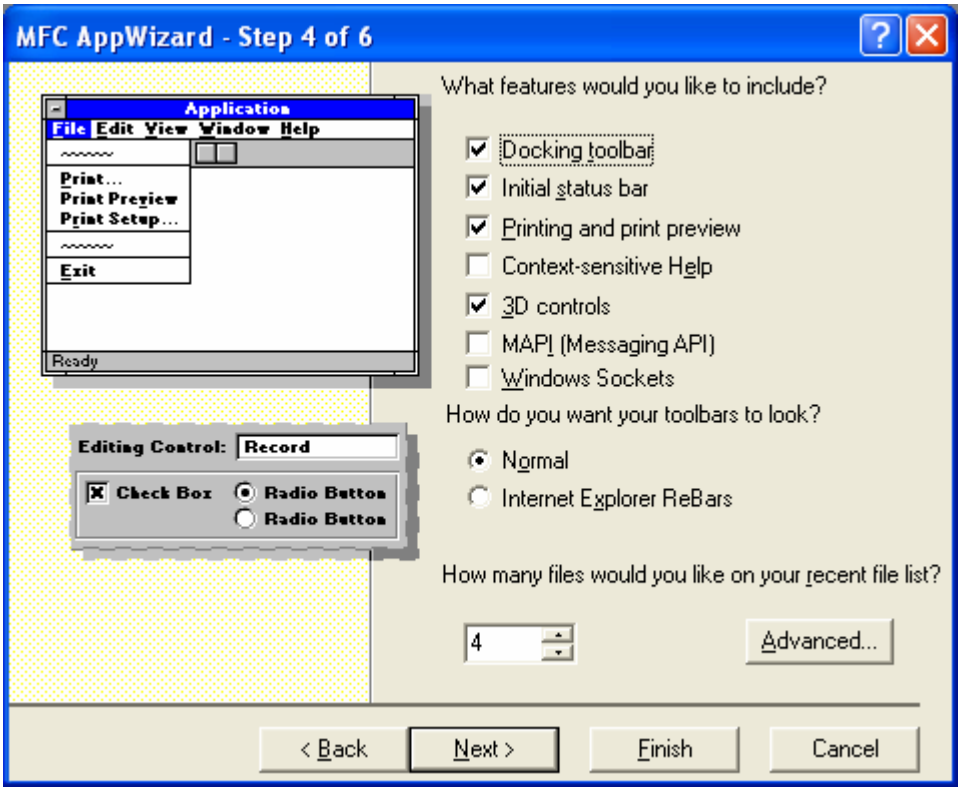

Figure 6: MYMFC30B – AppWizard step 4 of 6.

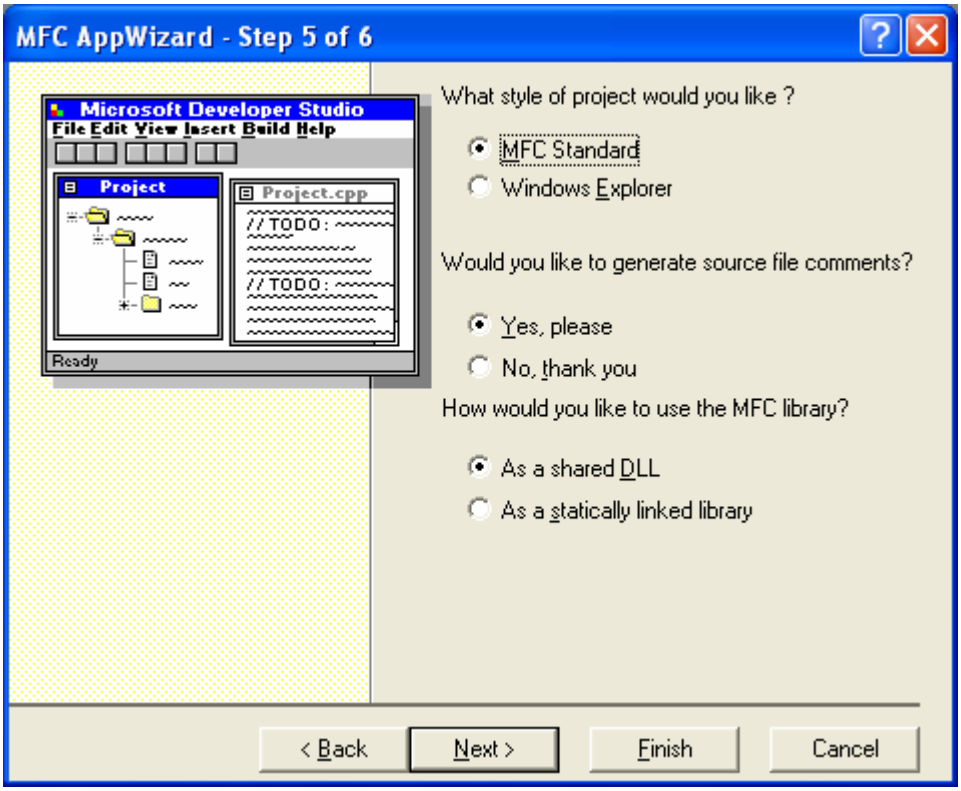

Figure 7: MYMFC30B – AppWizard step 5 of 6.

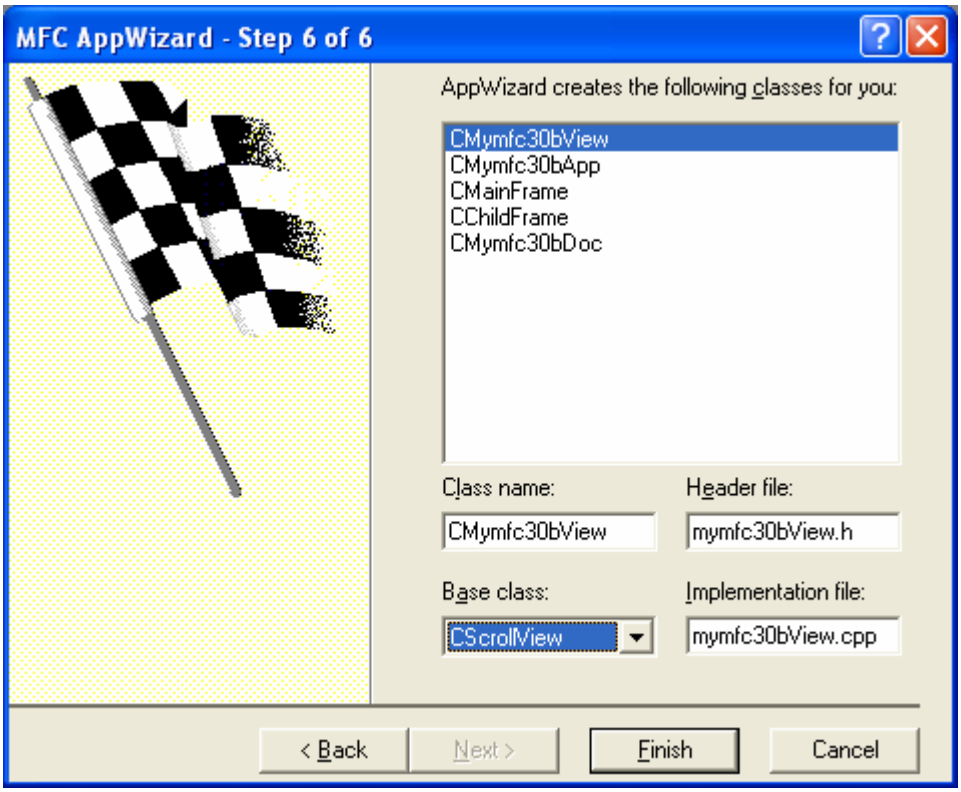

Figure 8: MYMFC30B – AppWizard step 4 of 6, change the view base class to `CScrollView`.

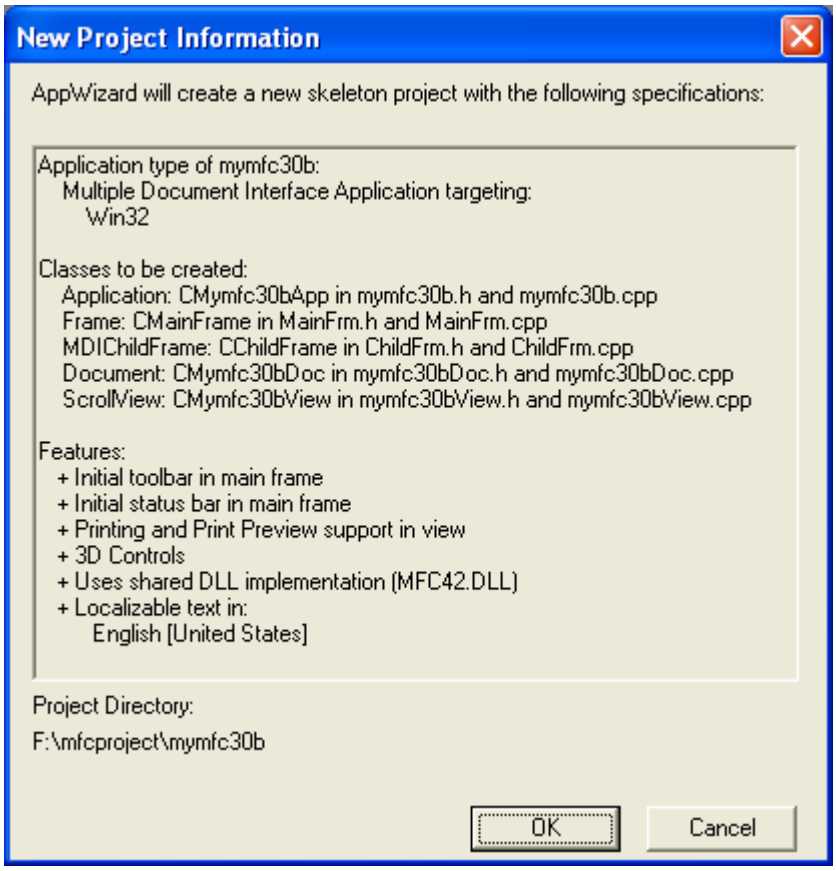

Figure 9: MYMFC30B project summary.

Add the `CDib` class. Copy the cdib.h and cdib.cpp from previous project and paste them to this project. Add those files by using the **Add to Project** menu as shown below.

Figure 10: Adding files for `Cdib` class.

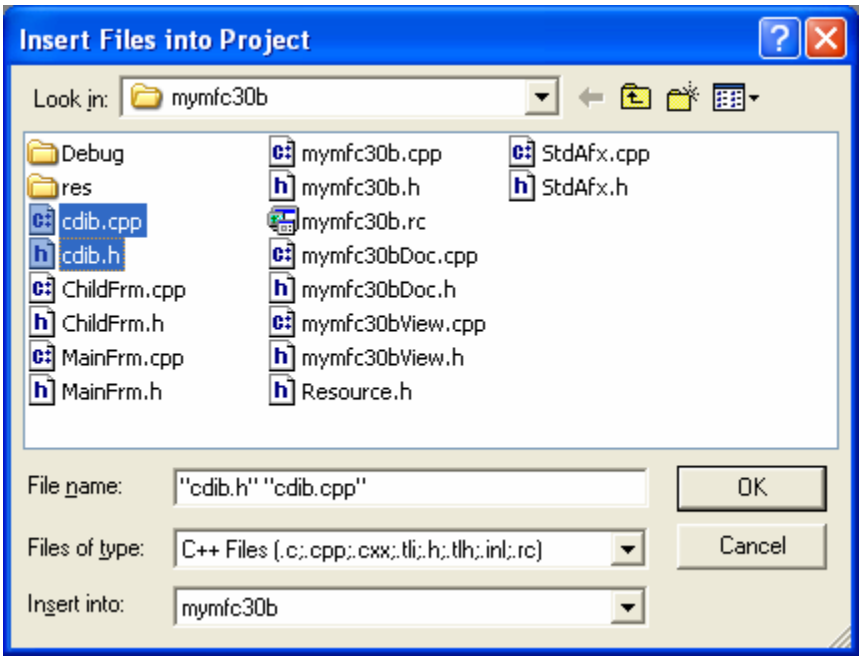

Figure 11: Selecting **cdib.cpp** and **cdib.h** files for `Cdib` class.

Add menu items under the **Edit** menu of the `IDR_MYMFC3TYPE`. Use the information in the following Table. Replace the **Undo** menu with the **Clear All**.

ID	Menu caption	Prompt
Separator	-	-
`ID_EDIT_CLEAR_ALL`	Clear A&ll	-
`ID_EDIT_COPYTO`	Copy T&o	Copy directly to a BMP file
`ID_EDIT_PASTEFROM`	P&aste From	Load a dib from a BMP file

Table 2.

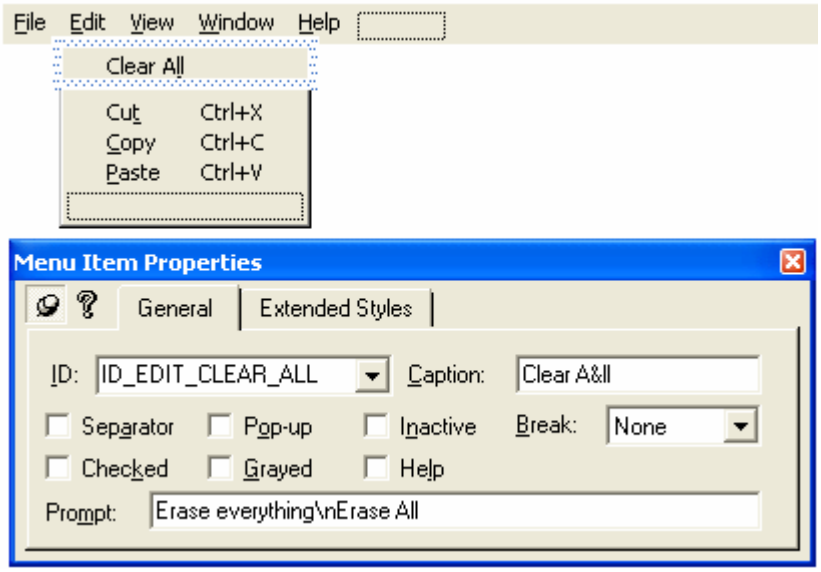

Figure 12: Replacing **Undo** with **Clear All** menu item.

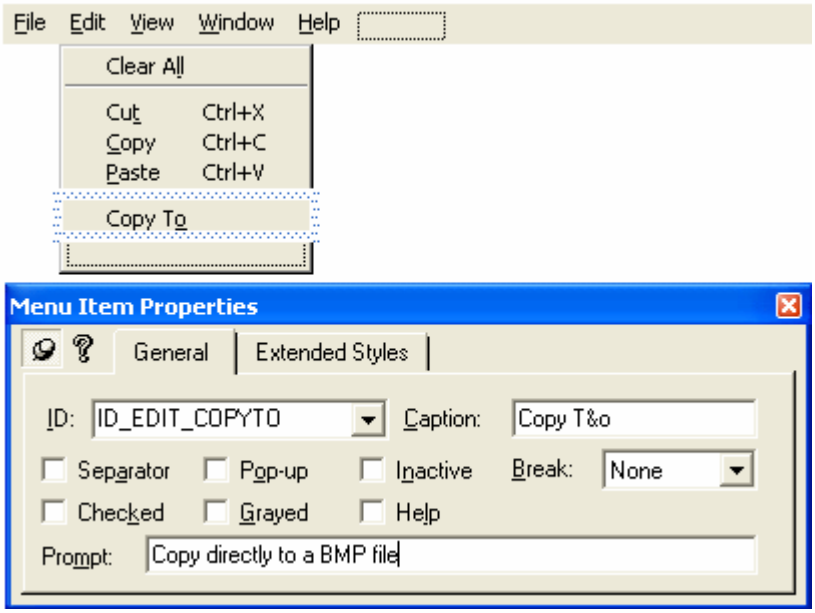

Figure 13: Adding **Copy To** menu item.

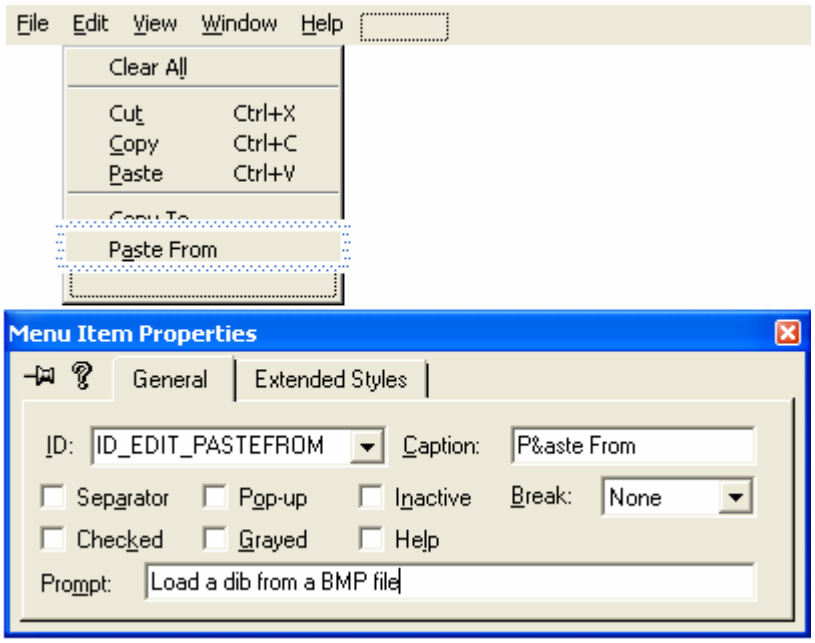

Figure 14: Adding **Paste From** menu item.

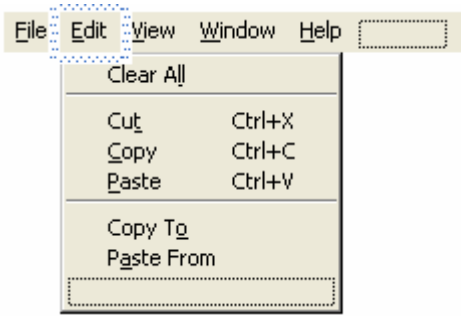

Figure 15: A completed menu items for MYMFC30B.

Add `WM_PALETTECHANGED` and `WM_QUERYNEWPALETTE` windows messages to the `CMainFrame` class.

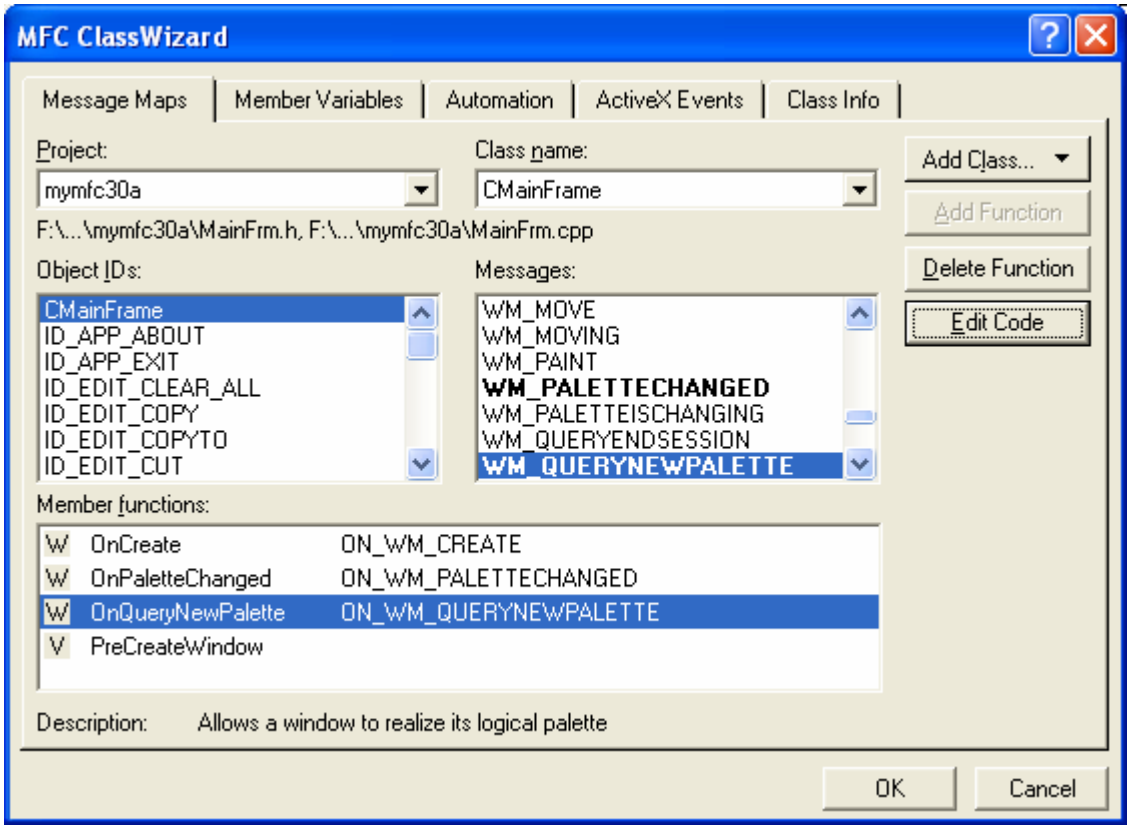

Figure 16: Adding Windows message handler to the `CMainFrame` class.

Add the following `#define` directive in **MainFrm.h**.

```
#define WM_VIEWPALETTECHANGED WM_USER + 5
```

```
#endif // _MSC_VER > 1000
#define WM_VIEWPALETTECHANGED   WM_USER + 5
class CMainFrame : public CMDIFrameWnd
```

Listing 1.

In **MainFrm.cpp**, add the following codes.

```
BOOL CMainFrame::OnQueryNewPalette()
{
	TRACE("CMainFrame::OnQueryNewPalette\n");
	CClientDC dc(this);
	// don't bother if we're not using a palette
	if((dc.GetDeviceCaps(RASTERCAPS) & RC_PALETTE) == 0) return TRUE;
	// determine the active view
	HWND hActiveView = NULL;
	CFrameWnd* pActiveFrm = GetActiveFrame();
	if(pActiveFrm != NULL) {
		CView* pActiveView = pActiveFrm->GetActiveView();
		if(pActiveView != NULL) {
			hActiveView = pActiveView->GetSafeHwnd();
		}
	}
	// iterate through all views
	BOOL bBackground;
	CView* pView;
	CDocument* pDoc;
	CDocTemplate* pTemplate;
	POSITION posView;
	POSITION posDoc;
	POSITION posTemplate = AfxGetApp()->GetFirstDocTemplatePosition();
	while(posTemplate != NULL)
	{
		pTemplate = AfxGetApp()->GetNextDocTemplate(posTemplate);
		posDoc = pTemplate->GetFirstDocPosition();
		while(posDoc != NULL)
		{
			pDoc = pTemplate->GetNextDoc(posDoc);
			posView = pDoc->GetFirstViewPosition();
			while(posView != NULL)
			{
				pView = pDoc->GetNextView(posView);
				bBackground = !(hActiveView == pView->GetSafeHwnd());
				// background mode if view is not the active view
				pView->SendMessage(WM_VIEWPALETTECHANGED, bBackground);
			}
		}
	}
	return TRUE;
}

void CMainFrame::OnPaletteChanged(CWnd* pFocusWnd)
{
	TRACE("CMainFrame::OnPaletteChanged\n");
	if(GetSafeHwnd() != pFocusWnd->GetSafeHwnd())
	{ OnQueryNewPalette(); }
}
```

```
void CMainFrame::OnPaletteChanged(CWnd* pFocusWnd)
{
    TRACE("CMainFrame::OnPaletteChanged\n");
    if(GetSafeHwnd() != pFocusWnd->GetSafeHwnd()) {
        OnQueryNewPalette();
    }
}
```

Listing 2.

Using ClassWizard, add the following virtual function to `CMymfc30bView` class.

- `OnDragEnter()`
- `OnDragLeave()`
- `OnDragOver()`
- `OnDrop()`
- `OnPrepareDC()`

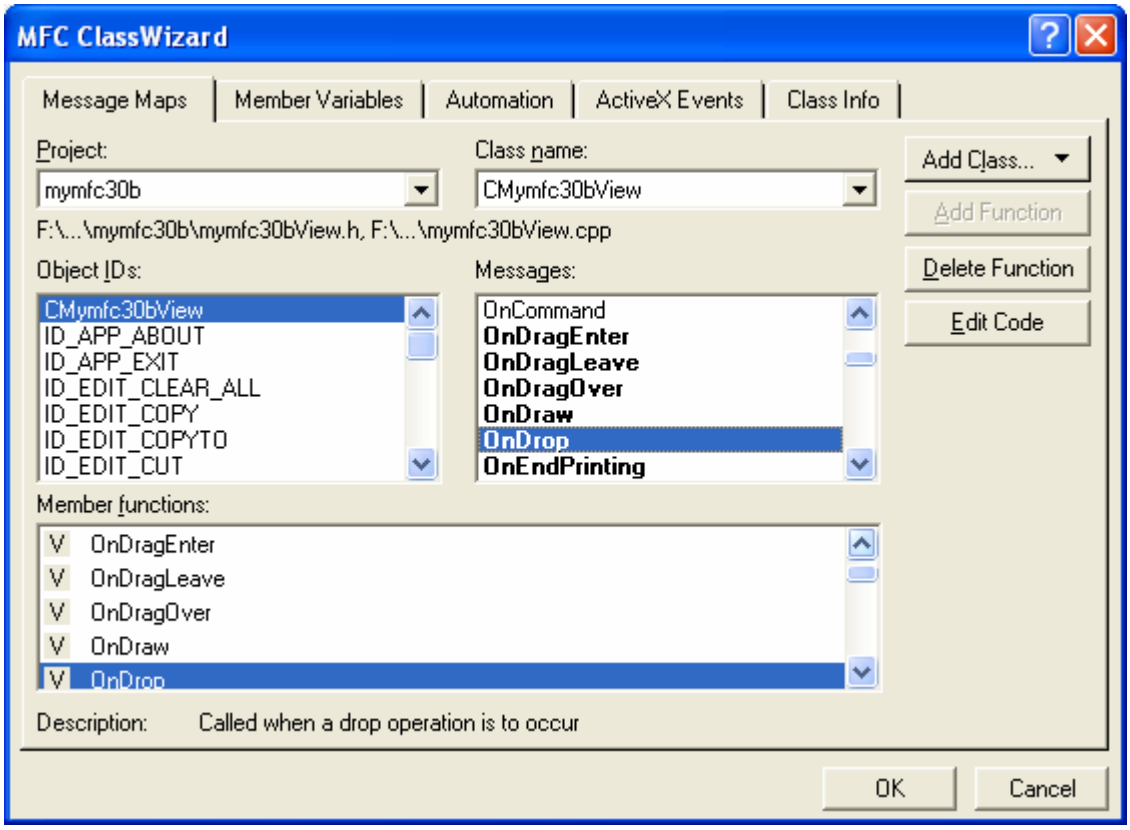

Figure 17: Adding virtual functions virtual function to `CMymfc30bView` class.

Add window messages to the `CMymfc30bView` class as shown below.

- `OnLButtonDown()`
- `OnSetCursor()`

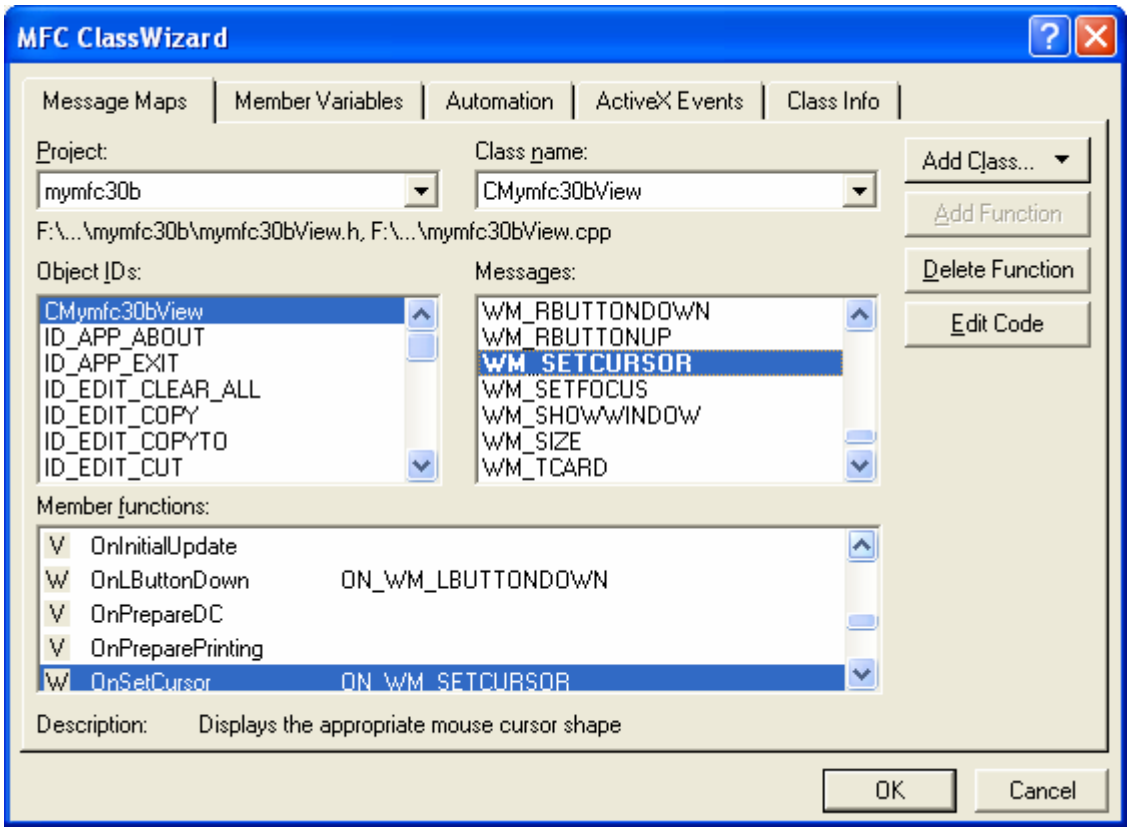

Figure 18: Adding window messages to the `CMymfc30bView` class.

Add command and update command handlers. Take note that the update command handler for `ID_EDIT_COPY`, `ID_EDIT_COPYTO` and `ID_EDIT_CUT` is same.

ID	Handler	
`ID_EDIT_COPY`	Command.	`OnEditCopy()`
`ID_EDIT_COPY`	update command	`OnUpdateEditCopy()`
`ID_EDIT_COPYTO`	Command.	`OnEditCopyto()`
`ID_EDIT_COPYTO`	update command	`OnUpdateEditCopy()`
`ID_EDIT_CUT`	Command.	`OnEditCut()`
`ID_EDIT_CUT`	update command	`OnUpdateEditCopy()`
`ID_EDIT_PASTE`	Command	`OnEditPaste()`
`ID_EDIT_PASTE`	Update command.	`OnUpdateEditPaste()`
`ID_EDIT_PASTEFROM`	Command.	`OnEditPastefrom()`

Table 3.

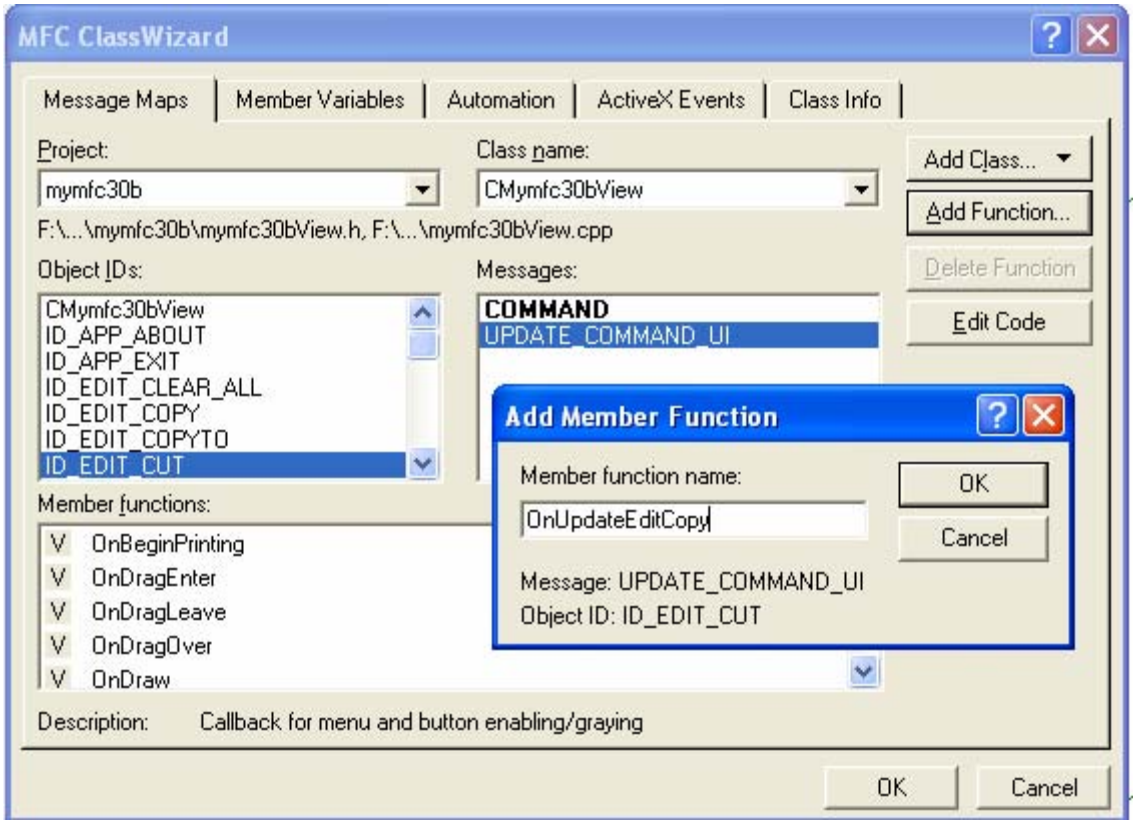

Figure 19: Adding update command handlers for `ID_EDIT_CUT`.

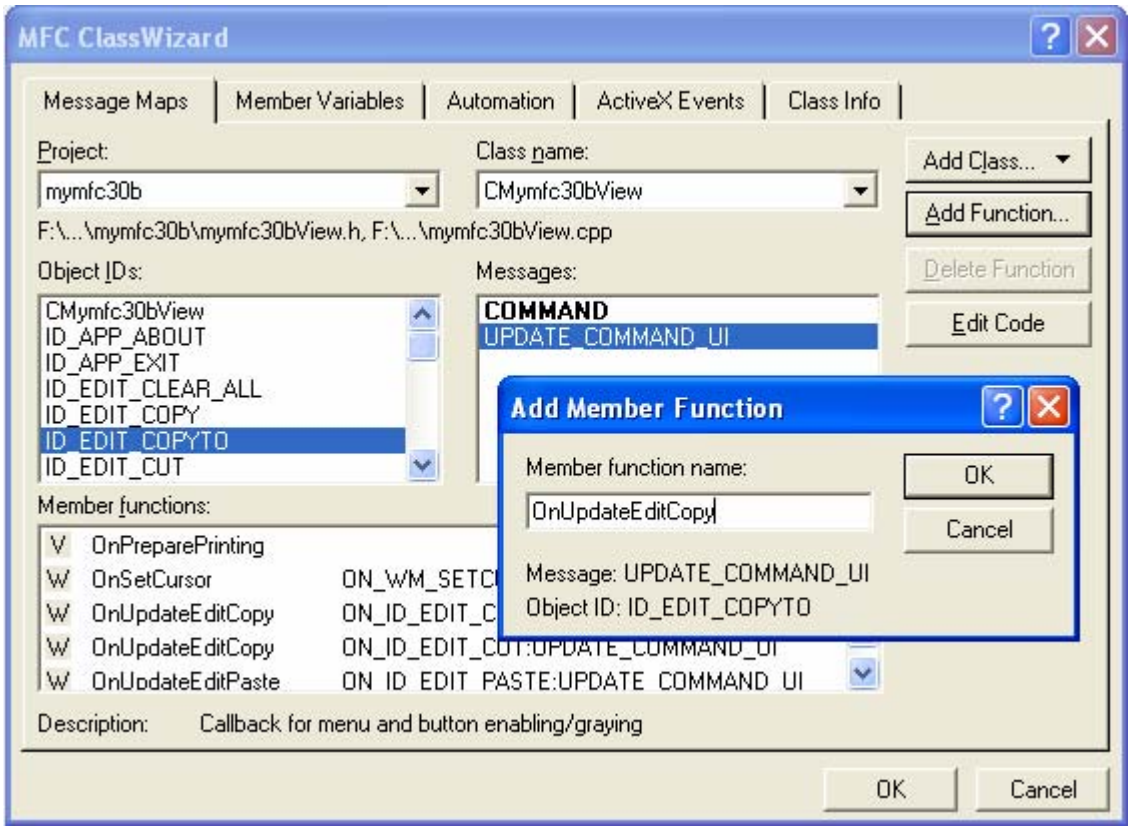

Figure 20: Adding update command handlers for `ID_EDIT_COPYTO`.

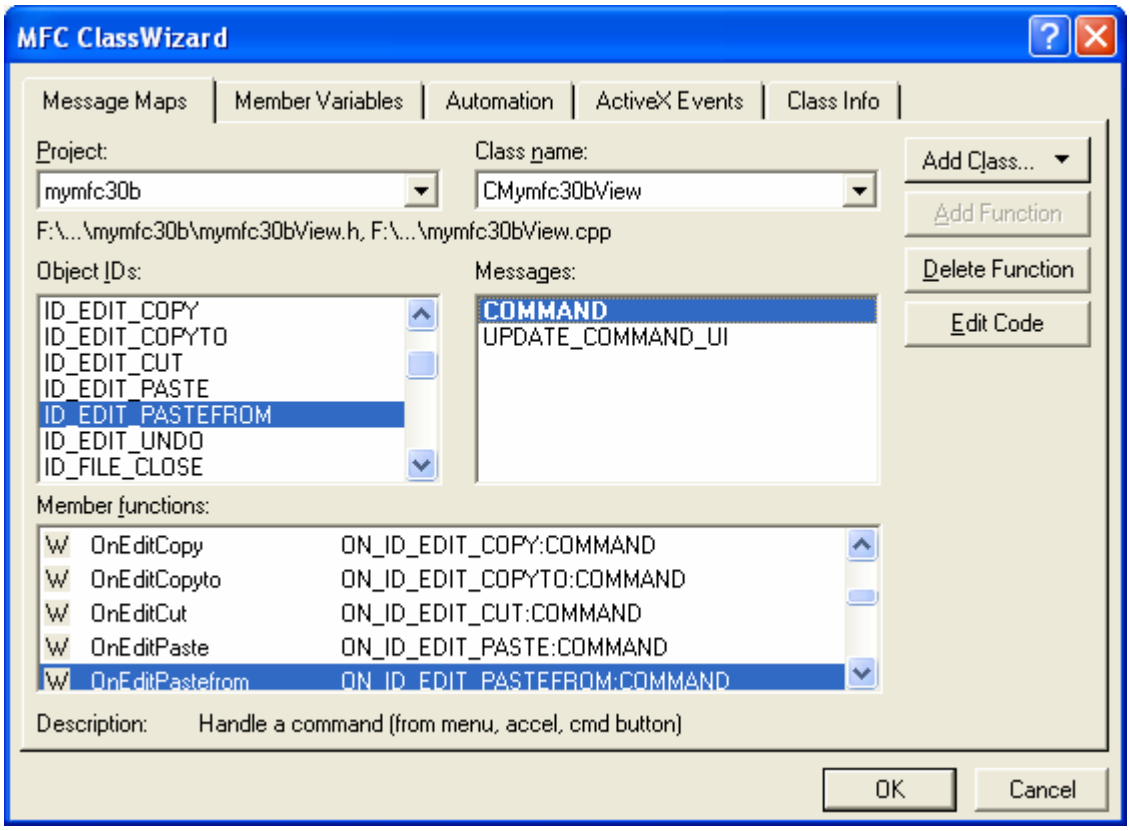

Figure 21: Adding command handlers for `ID_EDIT_PASTEFROM`.

Manually or using ClassView, add member variables as shown below in **mymfc30bView.h**. These variables are `private` by default.

```
    // for tracking
    CRectTracker m_tracker;
    CRect m_rectTracker;
    // for drag-and-drop
    CRect m_rectTrackerEnter; // original logical coords
    COleDropTarget m_dropTarget;
    CSize m_dragOffset; // device coords
    CSize m_sizeTotal;  // document size
```

```
class CMymfc30bView : public CScrollView
{
    // for tracking
    CRectTracker m_tracker;
    CRect m_rectTracker;
    // for drag-and-drop
    CRect m_rectTrackerEnter; // original logical coords
    COleDropTarget m_dropTarget;
    CSize m_dragOffset; // device coords
    CSize m_sizeTotal;  // document size
```

Listing 3.

Then using ClassView or manually add the following member functions.

```
private:
    BOOL DoPasteDib(COleDataObject* pDataObject);
    COleDataSource* SaveDib();
    void MoveTrackRect(CPoint point);
```

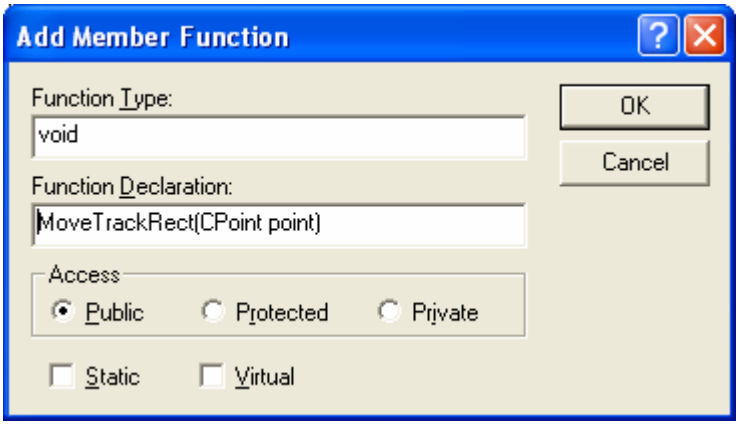

Figure 22: Adding a member function.

```
// Implementation
public:
    void MoveTrackRect(CPoint point);
    COleDataSource* SaveDib();
    BOOL DoPasteDib(COleDataObject* pDataObject);
    virtual ~CMymfc30bView();
#ifdef _DEBUG
    virtual void AssertValid() const;
```

Listing 4.

Add code to the implementation file **mymfc30bView.cpp**.

```
CMymfc30bView::CMymfc30bView() : m_sizeTotal(800, 1050), // 8-by-10.5 inches
                                                         //  when printed
    m_rectTracker(50, 50, 250, 250),
    m_dragOffset(0, 0),
    m_rectTrackerEnter(50, 50, 250, 250)
{
}
```

```
// CMymfc30bView construction/destruction

CMymfc30bView::CMymfc30bView(): m_sizeTotal(800, 1050), // 8-by-10.5 inches
                                                        //  when printed
    m_rectTracker(50, 50, 250, 250),
    m_dragOffset(0, 0),
    m_rectTrackerEnter(50, 50, 250, 250)

{
    // TODO: add construction code here

}
```

Listing 5.

```
void CMymfc30bView::OnDraw(CDC* pDC)
{
    CDib& dib = GetDocument()->m_dib;
    m_tracker.m_rect = m_rectTracker;
    pDC->LPtoDP(m_tracker.m_rect);  // tracker wants device coords
    m_tracker.Draw(pDC);
    dib.UsePalette(pDC);
    dib.Draw(pDC, m_rectTracker.TopLeft(), m_rectTracker.Size());
}
```

```
void CMymfc30bView::OnDraw(CDC* pDC)
{
    CDib& dib = GetDocument()->m_dib;
    m_tracker.m_rect = m_rectTracker;
    pDC->LPtoDP(m_tracker.m_rect);    // tracker wants device coords
    m_tracker.Draw(pDC);
    dib.UsePalette(pDC);
    dib.Draw(pDC, m_rectTracker.TopLeft(), m_rectTracker.Size());
}
```

Listing 6.

```
void CMymfc30bView::OnInitialUpdate()
{
    m_dropTarget.Register(this); // IMPORTANT
    SetScrollSizes(MM_TEXT, m_sizeTotal);
    m_tracker.m_nStyle = CRectTracker::solidLine | CRectTracker::resizeOutside;
    CScrollView::OnInitialUpdate();
}
```

```
void CMymfc30bView::OnInitialUpdate()
{
    m_dropTarget.Register(this); // IMPORTANT
    SetScrollSizes(MM_TEXT, m_sizeTotal);
    m_tracker.m_nStyle = CRectTracker::solidLine | CRectTracker::resizeOutside;
    CScrollView::OnInitialUpdate();
}
```

Listing 7.

```
BOOL CMymfc30bView::DoPasteDib(COleDataObject *pDataObject)
{
    // update command UI should keep us out of here if not CF_DIB
    if (!pDataObject->IsDataAvailable(CF_DIB))
    {
      TRACE("CF_DIB format is unavailable\n");
      return FALSE;
    }
    CMymfc30bDoc* pDoc = GetDocument();
    // seems to be MOVEABLE memory, so we must use GlobalLock!
    //  (hDib != lpDib) GetGlobalData copies the memory, so we can
    //  hang onto it until we delete the CDib
    HGLOBAL hDib = pDataObject->GetGlobalData(CF_DIB);
    ASSERT(hDib != NULL);
    LPVOID lpDib = ::GlobalLock(hDib);
    ASSERT(lpDib != NULL);
```

```
    pDoc->m_dib.AttachMemory(lpDib, TRUE, hDib);
    GetDocument()->SetModifiedFlag();
    pDoc->UpdateAllViews(NULL);
    return TRUE;
}
```

```
BOOL CMymfc30bView::DoPasteDib(COleDataObject *pDataObject)
{
    // update command UI should keep us out of here if not CF_DIB
    if (!pDataObject->IsDataAvailable(CF_DIB)) {
      TRACE("CF_DIB format is unavailable\n");
      return FALSE;
    }
    CMymfc30bDoc* pDoc = GetDocument();
    // seems to be MOVEABLE memory, so we must use GlobalLock!
    //  (hDib != lpDib) GetGlobalData copies the memory, so we can
    //  hang onto it until we delete the CDib
    HGLOBAL hDib = pDataObject->GetGlobalData(CF_DIB);
    ASSERT(hDib != NULL);
    LPVOID lpDib = ::GlobalLock(hDib);
    ASSERT(lpDib != NULL);
    pDoc->m_dib.AttachMemory(lpDib, TRUE, hDib);
    GetDocument()->SetModifiedFlag();
    pDoc->UpdateAllViews(NULL);
    return TRUE;
}
```

Listing 8.

```
COleDataSource* CMymfc30bView::SaveDib()
{
      CDib& dib = GetDocument()->m_dib;
      if (dib.GetSizeImage() > 0) {
      COleDataSource* pSource = new COleDataSource();
      int nHeaderSize =  dib.GetSizeHeader();
      int nImageSize = dib.GetSizeImage();
      HGLOBAL hHeader = ::GlobalAlloc(GMEM_SHARE,
          nHeaderSize + nImageSize);
      LPVOID pHeader = ::GlobalLock(hHeader);
      ASSERT(pHeader != NULL);
      LPVOID pImage = (LPBYTE) pHeader + nHeaderSize;
      memcpy(pHeader, dib.m_lpBMIH, nHeaderSize);
      memcpy(pImage, dib.m_lpImage, nImageSize);
      // receiver is supposed to free the global memory
      ::GlobalUnlock(hHeader);
      pSource->CacheGlobalData(CF_DIB, hHeader);
      return pSource;
    }
    return NULL;
}
```

```
COleDataSource* CMymfc30bView::SaveDib()
{
    CDib& dib = GetDocument()->m_dib;
    if (dib.GetSizeImage() > 0) {
      COleDataSource* pSource = new COleDataSource();
      int nHeaderSize =  dib.GetSizeHeader();
      int nImageSize = dib.GetSizeImage();
      HGLOBAL hHeader = ::GlobalAlloc(GMEM_SHARE,
          nHeaderSize + nImageSize);
      LPVOID pHeader = ::GlobalLock(hHeader);
      ASSERT(pHeader != NULL);
      LPVOID pImage = (LPBYTE) pHeader + nHeaderSize;
      memcpy(pHeader, dib.m_lpBMIH, nHeaderSize);
      memcpy(pImage, dib.m_lpImage, nImageSize);
      // receiver is supposed to free the global memory
      ::GlobalUnlock(hHeader);
      pSource->CacheGlobalData(CF_DIB, hHeader);
      return pSource;
    }
    return NULL;
}
```

Listing 9.

```
void CMymfc30bView::MoveTrackRect(CPoint point)
{
    CClientDC dc(this);
    OnPrepareDC(&dc);
    dc.DrawFocusRect(m_rectTracker);
    dc.LPtoDP(m_rectTracker);
    CSize sizeTrack = m_rectTracker.Size();
    CPoint newTopleft = point - m_dragOffset;  // still device
    m_rectTracker = CRect(newTopleft, sizeTrack);
    m_tracker.m_rect = m_rectTracker;
    dc.DPtoLP(m_rectTracker);
    dc.DrawFocusRect(m_rectTracker);
}
```

```
void CMymfc30bView::MoveTrackRect(CPoint point)
{
    CClientDC dc(this);
    OnPrepareDC(&dc);
    dc.DrawFocusRect(m_rectTracker);
    dc.LPtoDP(m_rectTracker);
    CSize sizeTrack = m_rectTracker.Size();
    CPoint newTopleft = point - m_dragOffset;  // still device
    m_rectTracker = CRect(newTopleft, sizeTrack);
    m_tracker.m_rect = m_rectTracker;
    dc.DPtoLP(m_rectTracker);
    dc.DrawFocusRect(m_rectTracker);
}
```

Listing 10.

```
DROPEFFECT CMymfc30bView::OnDragEnter(COleDataObject* pDataObject, DWORD
dwKeyState, CPoint point)
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE("Entering CMymfc30bView::OnDragEnter, point = (%d, %d)\n",
        point.x, point.y);
```

```
    m_rectTrackerEnter = m_rectTracker; // Save original coordinates
                                        //  for cursor leaving
                                        //  rectangle
    CClientDC dc(this);
    OnPrepareDC(&dc);
    dc.DrawFocusRect(m_rectTracker); // will be erased in OnDragOver
    return OnDragOver(pDataObject, dwKeyState, point);
}
```

```
DROPEFFECT CMymfc30bView::OnDragEnter(COleDataObject* pDataObject,
                                      DWORD dwKeyState, CPoint point)
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE("Entering CMymfc30bView::OnDragEnter, point = (%d, %d)\n",
        point.x, point.y);

    m_rectTrackerEnter = m_rectTracker; // Save original coordinates
                                        //  for cursor leaving
                                        //  rectangle
    CClientDC dc(this);
    OnPrepareDC(&dc);
    dc.DrawFocusRect(m_rectTracker); // will be erased in OnDragOver
    return OnDragOver(pDataObject, dwKeyState, point);
}
```

Listing 11.

```
void CMymfc30bView::OnDragLeave()
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE("Entering CMymfc30bView::OnDragLeave\n");
    CClientDC dc(this);
    OnPrepareDC(&dc);
    dc.DrawFocusRect(m_rectTracker);
    m_rectTracker = m_rectTrackerEnter; // Forget it ever happened
}
```

```
void CMymfc30bView::OnDragLeave()
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE("Entering CMymfc30bView::OnDragLeave\n");
    CClientDC dc(this);
    OnPrepareDC(&dc);
    dc.DrawFocusRect(m_rectTracker);
    m_rectTracker = m_rectTrackerEnter; // Forget it ever happened
}
```

Listing 12.

```
DROPEFFECT CMymfc30bView::OnDragOver(COleDataObject* pDataObject, DWORD
dwKeyState, CPoint point)
{
   // TODO: Add your specialized code here and/or call the base class
   if (!pDataObject->IsDataAvailable(CF_DIB))
   { return DROPEFFECT_NONE; }
    MoveTrackRect(point);
    if((dwKeyState & MK_CONTROL) == MK_CONTROL)
    { return DROPEFFECT_COPY; }
```

```
    // Check for force move
    if ((dwKeyState & MK_ALT) == MK_ALT)
    { return DROPEFFECT_MOVE; }
    // default -- recommended action is move
    return DROPEFFECT_MOVE;
}
```

```
DROPEFFECT CMymfc30bView::OnDragOver(COleDataObject* pDataObject,
                                     DWORD dwKeyState, CPoint point)
{
    // TODO: Add your specialized code here and/or call the base class
    if (!pDataObject->IsDataAvailable(CF_DIB)) {
        return DROPEFFECT_NONE;
    }
    MoveTrackRect(point);
    if((dwKeyState & MK_CONTROL) == MK_CONTROL) {
        return DROPEFFECT_COPY;
    }
    // Check for force move
    if ((dwKeyState & MK_ALT) == MK_ALT) {
        return DROPEFFECT_MOVE;
    }
    // default -- recommended action is move
    return DROPEFFECT_MOVE;
}
```

Listing 13.

```
BOOL CMymfc30bView::OnDrop(COleDataObject* pDataObject, DROPEFFECT dropEffect,
CPoint point)
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE("Entering CMymfc30bView::OnDrop -- dropEffect = %d\n", dropEffect);
    BOOL bRet;
    CMymfc30bDoc* pDoc = GetDocument();
    MoveTrackRect(point);
    if(pDoc->m_bDragHere)
    {
        pDoc->m_bDragHere = FALSE;
        bRet = TRUE;
    }
    else
    { bRet = DoPasteDib(pDataObject); }
    return bRet;
}
```

```
BOOL CMymfc30bView::OnDrop(COleDataObject* pDataObject,
                           DROPEFFECT dropEffect, CPoint point)
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE("Entering CMymfc30bView::OnDrop -- dropEffect = %d\n", dropEffect);
    BOOL bRet;
    CMymfc30bDoc* pDoc = GetDocument();
    MoveTrackRect(point);
    if(pDoc->m_bDragHere) {
        pDoc->m_bDragHere = FALSE;
        bRet = TRUE;
    }
    else {
        bRet = DoPasteDib(pDataObject);
    }
    return bRet;
}
```

Listing 14.

```
void CMymfc30bView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
    // TODO: Add your specialized code here and/or call the base class
    // custom MM_LOENGLISH with positive y down
    if(pDC->IsPrinting()) {
        int nHsize = pDC->GetDeviceCaps(HORZSIZE) * 1000 / 254;
        int nVsize = pDC->GetDeviceCaps(VERTSIZE) * 1000 / 254;
        pDC->SetMapMode(MM_ANISOTROPIC);
        pDC->SetWindowExt(nHsize, nVsize);
        pDC->SetViewportExt(pDC->GetDeviceCaps(HORZRES),
                            pDC->GetDeviceCaps(VERTRES));
    }
    else
    { CScrollView::OnPrepareDC(pDC, pInfo); }
}
```

```
void CMymfc30bView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
    // TODO: Add your specialized code here and/or call the base class
// custom MM_LOENGLISH with positive y down
    if(pDC->IsPrinting()) {
        int nHsize = pDC->GetDeviceCaps(HORZSIZE) * 1000 / 254;
        int nVsize = pDC->GetDeviceCaps(VERTSIZE) * 1000 / 254;
        pDC->SetMapMode(MM_ANISOTROPIC);
        pDC->SetWindowExt(nHsize, nVsize);
        pDC->SetViewportExt(pDC->GetDeviceCaps(HORZRES),
                            pDC->GetDeviceCaps(VERTRES));
    }
    else {
        CScrollView::OnPrepareDC(pDC, pInfo);
    }
}
```

Listing 15.

```
void CMymfc30bView::OnLButtonDown(UINT nFlags, CPoint point)
{
     // TODO: Add your message handler code here and/or call default
     CMymfc30bDoc* pDoc = GetDocument();
     if(m_tracker.HitTest(point) == CRectTracker::hitMiddle)
```

```
      {
        COleDataSource* pSource = SaveDib();
        if(pSource)
        {
            // DoDragDrop returns only after drop is complete
            CClientDC dc(this);
            OnPrepareDC(&dc);
            CPoint topleft = m_rectTracker.TopLeft();
            dc.LPtoDP(&topleft);
            // 'point' here is not the same as the point parameter in
            //  OnDragEnter, so we use this one to compute the offset
            m_dragOffset = point - topleft;  // device coordinates
            pDoc->m_bDragHere = TRUE;
            DROPEFFECT dropEffect = pSource->DoDragDrop(
                DROPEFFECT_MOVE|DROPEFFECT_COPY, CRect(0, 0, 0, 0));
            TRACE("after DoDragDrop -- dropEffect = %ld\n", dropEffect);
            if (dropEffect == DROPEFFECT_MOVE && pDoc->m_bDragHere)
            { pDoc->OnEditClearAll(); }
            pDoc->m_bDragHere = FALSE;
            delete pSource;
        }
    }
    else
    {
        if(m_tracker.Track(this, point, FALSE, NULL))
        {
            CClientDC dc(this);
            OnPrepareDC(&dc);
            // should have some way to prevent it going out of bounds
            m_rectTracker = m_tracker.m_rect;
            dc.DPtoLP(m_rectTracker); // Update logical coords
        }
    }
    Invalidate();
}
```

```
void CMymfc30bView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    CMymfc30bDoc* pDoc = GetDocument();
    if(m_tracker.HitTest(point) == CRectTracker::hitMiddle) {
        COleDataSource* pSource = SaveDib();
        if(pSource) {
            // DoDragDrop returns only after drop is complete
            CClientDC dc(this);
            OnPrepareDC(&dc);
            CPoint topleft = m_rectTracker.TopLeft();
            dc.LPtoDP(&topleft);
            // 'point' here is not the same as the point parameter in
            //  OnDragEnter, so we use this one to compute the offset
            m_dragOffset = point - topleft;  // device coordinates
            pDoc->m_bDragHere = TRUE;
            DROPEFFECT dropEffect = pSource->DoDragDrop(
                DROPEFFECT_MOVE|DROPEFFECT_COPY, CRect(0, 0, 0, 0));
            TRACE("after DoDragDrop -- dropEffect = %ld\n", dropEffect);
            if (dropEffect == DROPEFFECT_MOVE && pDoc->m_bDragHere) {
                pDoc->OnEditClearAll();
            }
            pDoc->m_bDragHere = FALSE;
            delete pSource;
        }
    }
    else {
        if(m_tracker.Track(this, point, FALSE, NULL)) {
            CClientDC dc(this);
            OnPrepareDC(&dc);
            // should have some way to prevent it going out of bounds
            m_rectTracker = m_tracker.m_rect;
            dc.DPtoLP(m_rectTracker); // Update logical coords
        }
    }
    Invalidate();
}
```

Listing 16.

```
BOOL CMymfc30bView::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
{
      // TODO: Add your message handler code here and/or call default
      if(m_tracker.SetCursor(pWnd, nHitTest))
      { return TRUE; }
      else
      { return CScrollView::OnSetCursor(pWnd, nHitTest, message); }
}
```

```
BOOL CMymfc30bView::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
{
    // TODO: Add your message handler code here and/or call default
    if(m_tracker.SetCursor(pWnd, nHitTest)) {
        return TRUE;
    }
    else {
        return CScrollView::OnSetCursor(pWnd, nHitTest, message);
    }
}
```

Listing 17.

```
void CMymfc30bView::OnEditCopy()
{
      // TODO: Add your command handler code here
      COleDataSource* pSource = SaveDib();
      if(pSource)
      { pSource->SetClipboard(); } // OLE deletes data source

}
```

```
void CMymfc30bView::OnEditCopy()
{
    // TODO: Add your command handler code here
    COleDataSource* pSource = SaveDib();
    if(pSource) {
        pSource->SetClipboard(); // OLE deletes data source
    }
}
```

Listing 18.

```
void CMymfc30bView::OnUpdateEditCopy(CCmdUI* pCmdUI)
{
      // TODO: Add your command update UI handler code here
      // serves Copy, Cut, and Copy To
    CDib& dib = GetDocument()->m_dib;
    pCmdUI->Enable(dib.GetSizeImage() > 0L);
}
```

```
void CMymfc30bView::OnUpdateEditCopy(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // serves Copy, Cut, and Copy To
    CDib& dib = GetDocument()->m_dib;
    pCmdUI->Enable(dib.GetSizeImage() > 0L);
}
```

Listing 19.

```
void CMymfc30bView::OnEditCopyto()
{
    // TODO: Add your command handler code here
    CDib& dib = GetDocument()->m_dib;
    CFileDialog dlg(FALSE, "bmp", "*.bmp");
    if(dlg.DoModal() != IDOK) return;

    BeginWaitCursor();
    dib.CopyToMapFile(dlg.GetPathName());
    EndWaitCursor();
}
```

```
void CMymfc30bView::OnEditCopyto()
{
    // TODO: Add your command handler code here
    CDib& dib = GetDocument()->m_dib;
    CFileDialog dlg(FALSE, "bmp", "*.bmp");
    if(dlg.DoModal() != IDOK) return;

    BeginWaitCursor();
    dib.CopyToMapFile(dlg.GetPathName());
    EndWaitCursor();
}
```

Listing 20.

```
void CMymfc30bView::OnEditCut()
{
    // TODO: Add your command handler code here
    OnEditCopy();
    GetDocument()->OnEditClearAll();
}
```

```
void CMymfc30bView::OnEditCut()
{
    // TODO: Add your command handler code here
    OnEditCopy();
    GetDocument()->OnEditClearAll();
}
```

Listing 21.

```
void CMymfc30bView::OnEditPaste()
{
    // TODO: Add your command handler code here
    CMymfc30bDoc* pDoc = GetDocument();
    COleDataObject dataObject;
    VERIFY(dataObject.AttachClipboard());
    DoPasteDib(&dataObject);
    pDoc->SetModifiedFlag();
    pDoc->UpdateAllViews(NULL);
}
```

```
void CMymfc30bView::OnEditPaste()
{
    // TODO: Add your command handler code here
    CMymfc30bDoc* pDoc = GetDocument();
    COleDataObject dataObject;
    VERIFY(dataObject.AttachClipboard());
    DoPasteDib(&dataObject);
    pDoc->SetModifiedFlag();
    pDoc->UpdateAllViews(NULL);
}
```

Listing 22.

```
void CMymfc30bView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    COleDataObject dataObject;
```

```
    BOOL bAvail = dataObject.AttachClipboard() &&
dataObject.IsDataAvailable(CF_DIB);
    pCmdUI->Enable(bAvail);
}
```

```
void CMymfc30bView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    COleDataObject dataObject;
    BOOL bAvail = dataObject.AttachClipboard() && dataObject.IsDataAvailable(CF_DIB);
    pCmdUI->Enable(bAvail);
}
```

Listing 23.

```
void CMymfc30bView::OnEditPastefrom()
{
    // TODO: Add your command handler code here
    CMymfc30bDoc* pDoc = GetDocument();
    CFileDialog dlg(TRUE, "bmp", "*.bmp");
    if(dlg.DoModal() != IDOK)
       return;
    if(pDoc->m_dib.AttachMapFile(dlg.GetPathName(), TRUE))
    { // share
          CClientDC dc(this);
        pDoc->m_dib.SetSystemPalette(&dc);
        pDoc->SetModifiedFlag();
        pDoc->UpdateAllViews(NULL);
    }
}
```

```
void CMymfc30bView::OnEditPastefrom()
{
    // TODO: Add your command handler code here
    CMymfc30bDoc* pDoc = GetDocument();
    CFileDialog dlg(TRUE, "bmp", "*.bmp");
    if(dlg.DoModal() != IDOK) return;
    if(pDoc->m_dib.AttachMapFile(dlg.GetPathName(), TRUE)) { // share
        CClientDC dc(this);
        pDoc->m_dib.SetSystemPalette(&dc);
        pDoc->SetModifiedFlag();
        pDoc->UpdateAllViews(NULL);
    }
}
```

Listing 24.

Add the `#include` statement in **Mymfc30bView.cpp** and **Mymfc30bDoc.cpp**.

```
#include "cdib.h"
```

```
#include "stdafx.h"
#include "mymfc30b.h"

#include "cdib.h"

#include "mymfc30bDoc.h"
```

Listing 25.

```
#include "stdafx.h"
#include "mymfc30b.h"

#include "cdib.h"

#include "mymfc30bDoc.h"
```

Listing 26.

Add `DeleteContents()` message handler to the `CMymfc30bDoc` class.

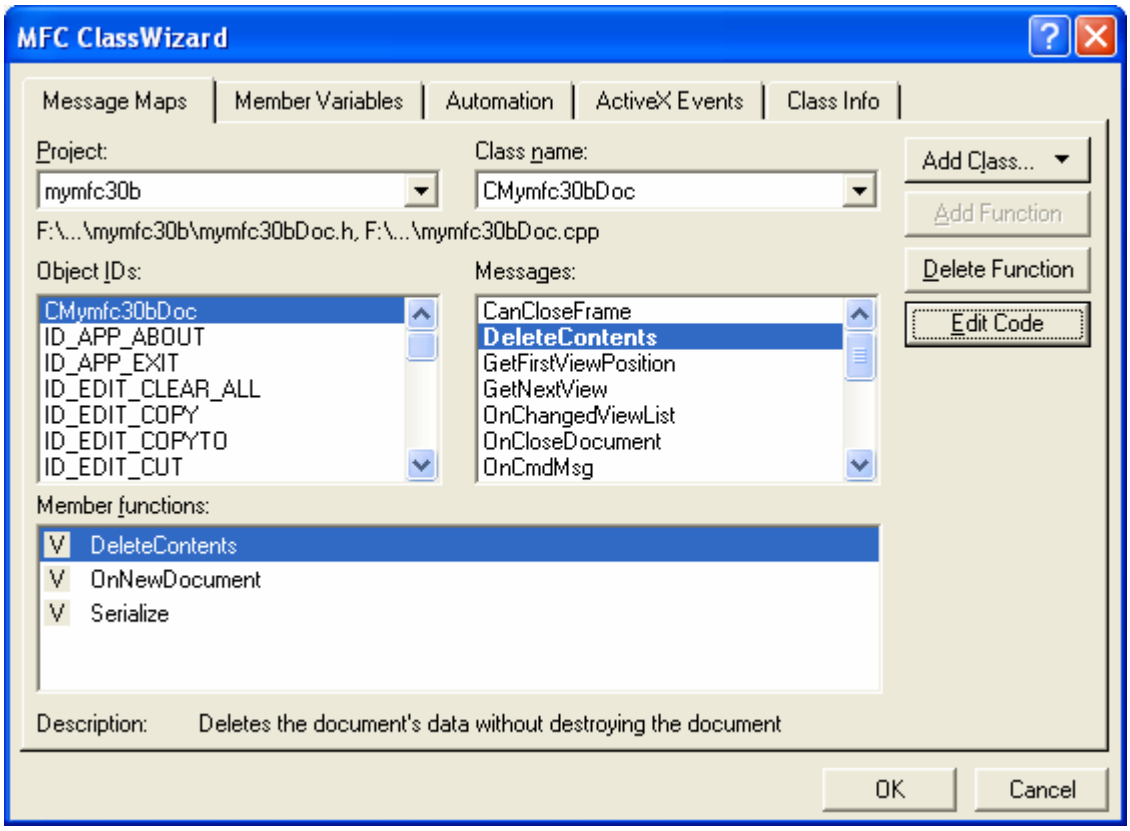

Figure 23: Adding `DeleteContents()` message handler to `CMymfc30bDoc` class.

Then, add command and update command for `ID_EDIT_CLEAR_ALL` to the `CMymfc30bDoc` class.

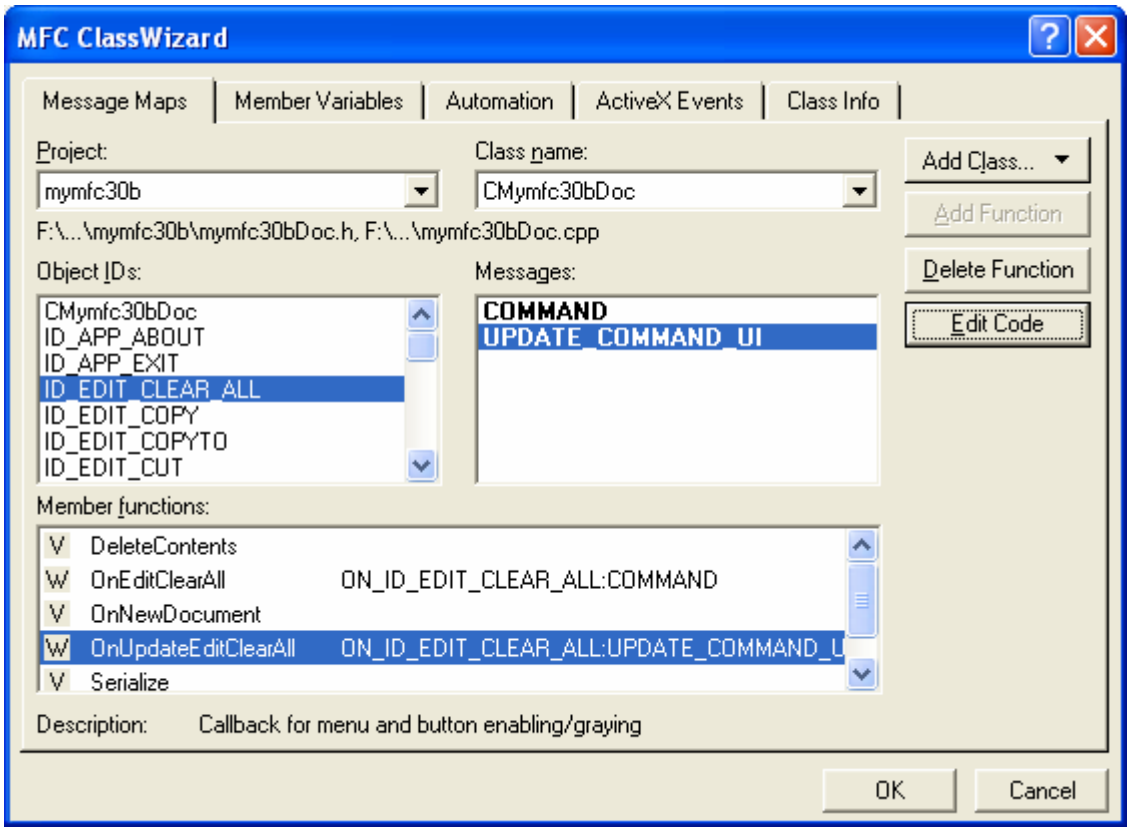

Figure 24: Adding command and update command for `ID_EDIT_CLEAR_ALL`.

Change the `protected` to `public` for the generated message map functions in **mymfc30bDoc.h** (or you can use `friend` keyword).

```
// Generated message map functions
public:
    //{{AFX_MSG(CMymfc30bDoc)
    afx_msg void OnEditClearAll();
    afx_msg void OnUpdateEditClearAll(CCmdUI* pCmdUI);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

Listing 27.

Add the following `public` member variables to `CMymfc30bDoc` class.

```
public:
    CDib m_dib;
    BOOL m_bDragHere; // for drags between two wins linked to this doc
```

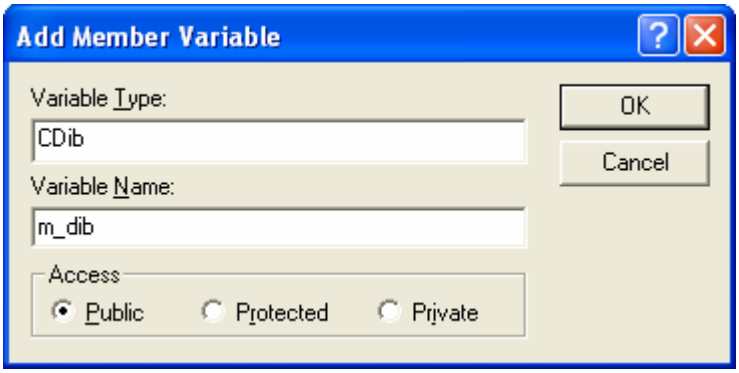

Figure 25: Adding `m_dib` member variable.

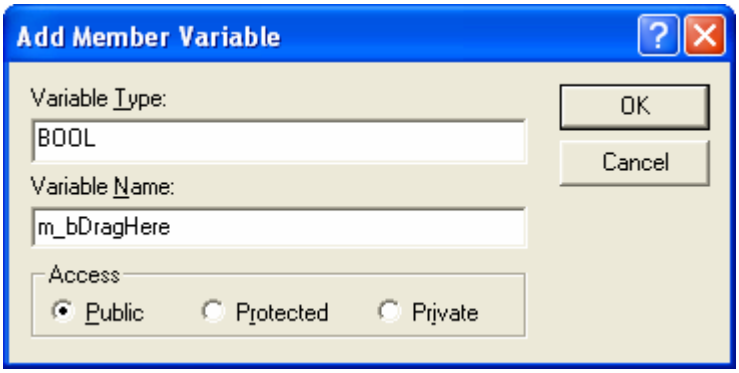

Figure 26: Adding `m_bDragHere` member variable.

Add/edit the codes to the implementation file, **mymfc30bDoc.cpp** as shown below.

```
CMymfc30bDoc::CMymfc30bDoc()
{
    m_bDragHere = FALSE;

}
```

```
CMymfc30bDoc::CMymfc30bDoc()
{
    // TODO: add one-time construction code here
    m_bDragHere = FALSE;
}
```

Listing 28.

```
void CMymfc30bDoc::Serialize(CArchive& ar)
{
      m_dib.Serialize(ar);
}
```

```
// CMymfc30bDoc serialization

void CMymfc30bDoc::Serialize(CArchive& ar)
{
    m_dib.Serialize(ar);
}
```

Listing 29.

```
void CMymfc30bDoc::DeleteContents()
{
      // TODO: Add your specialized code here and/or call the base class
      m_dib.Empty();
}
```

```
void CMymfc30bDoc::DeleteContents()
{
    // TODO: Add your specialized code here and/or call the base class
    m_dib.Empty();
}
```

Listing 30.

```
void CMymfc30bDoc::OnEditClearAll()
{
    // TODO: Add your command handler code here
    DeleteContents();
    UpdateAllViews(NULL);
    SetModifiedFlag();
}
```

```
void CMymfc30bDoc::OnEditClearAll()
{
    // TODO: Add your command handler code here
    DeleteContents();
    UpdateAllViews(NULL);
    SetModifiedFlag();
}
```

Listing 31.

```
void CMymfc30bDoc::OnUpdateEditClearAll(CCmdUI* pCmdUI)
{
      // TODO: Add your command update UI handler code here
      pCmdUI->Enable(m_dib.GetSizeImage() > 0);
}
```

```
void CMymfc30bDoc::OnUpdateEditClearAll(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(m_dib.GetSizeImage() > 0);
}
```

Listing 32.

Finally add the following line in your **StdAfx.h** file for automation support.

```
      #include <afxole.h>

#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxole.h>

//{{AFX_INSERT_LOCATION}}
```

Listing 33.

Then, add the following call at the start of the application's (**mymfc30b.cpp**) `InitInstance()` function.

```
      AfxOleInit();
```

```
// CMymfc30bApp initialization

BOOL CMymfc30bApp::InitInstance()
{
    AfxOleInit();
    // Standard initialization
    // If you are not using these
```

Listing 34.

Build and run MYMFC30B. First of all, select the **Paste From** menu. Select any bitmap sample, for example, under the Windows system directory (these bitmaps normally used for Windows wallpaper).

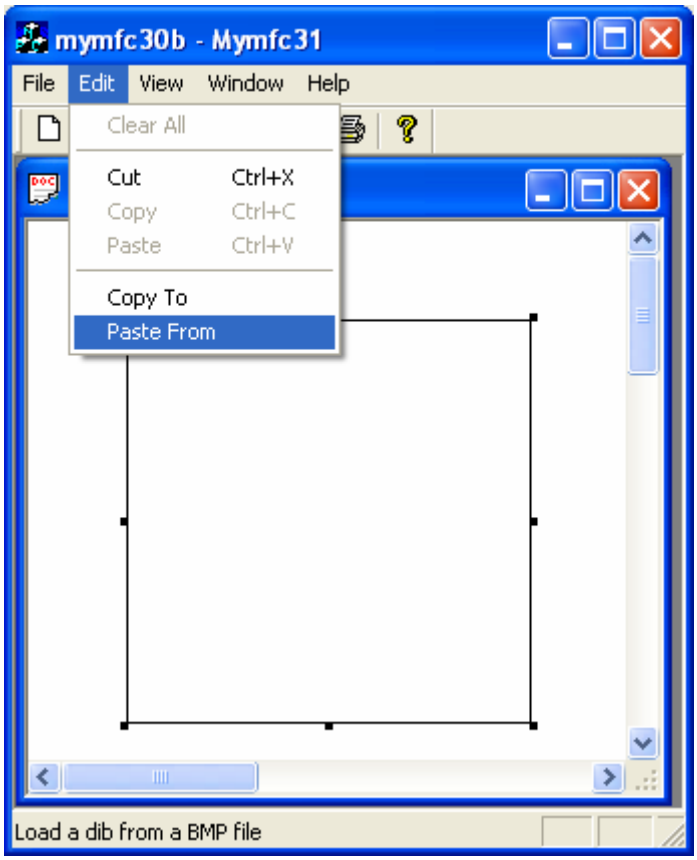

Figure 27: MYMFC30B output - testing the **Paste From** menu.

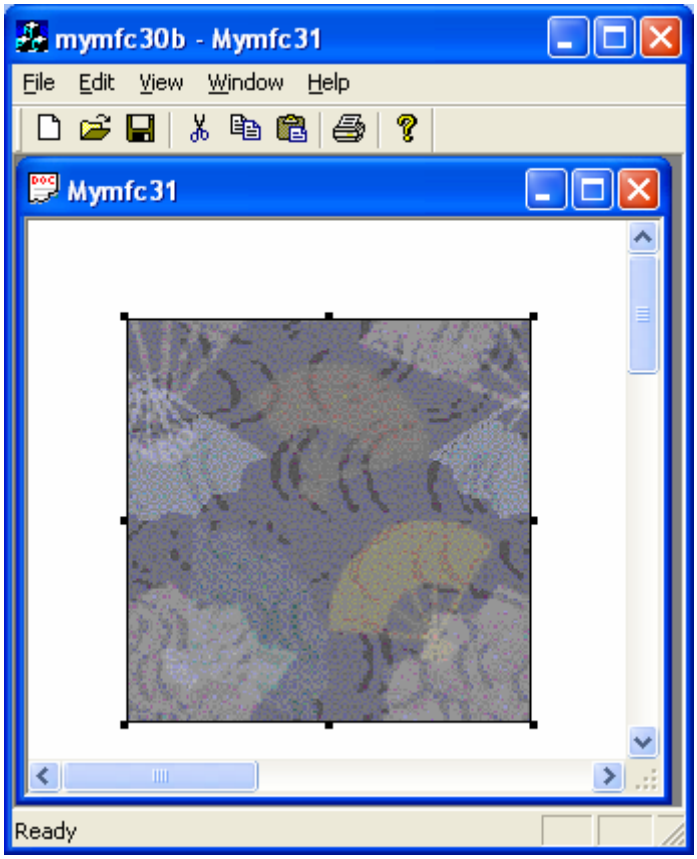

Figure 28: MYMFC30B output – displaying bitmap.

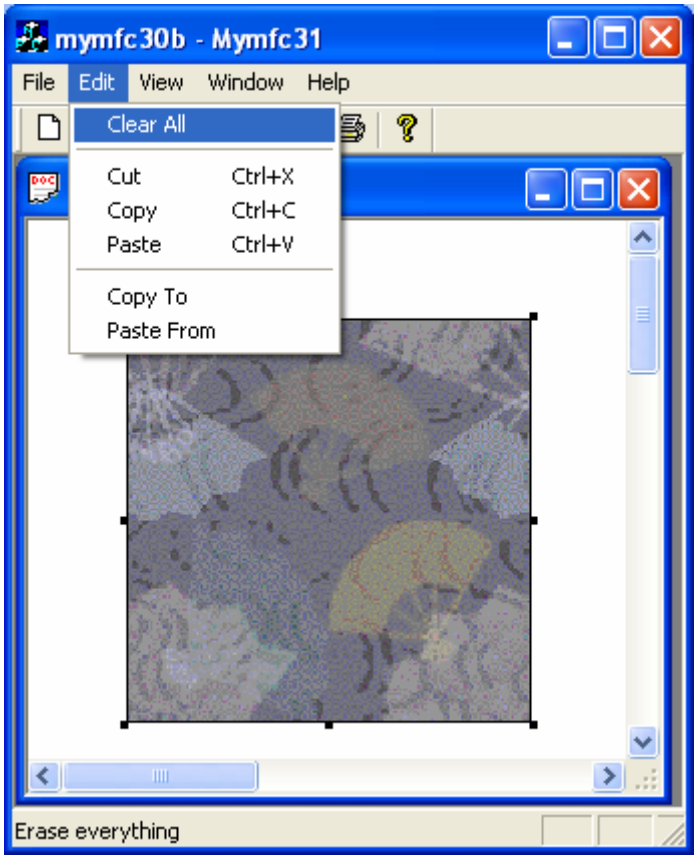

Figure 29: MYMFC30B output - testing the **Clear All** menu.

Try dragging and dropping the bitmap in the client area. Also try resizing the bitmap.

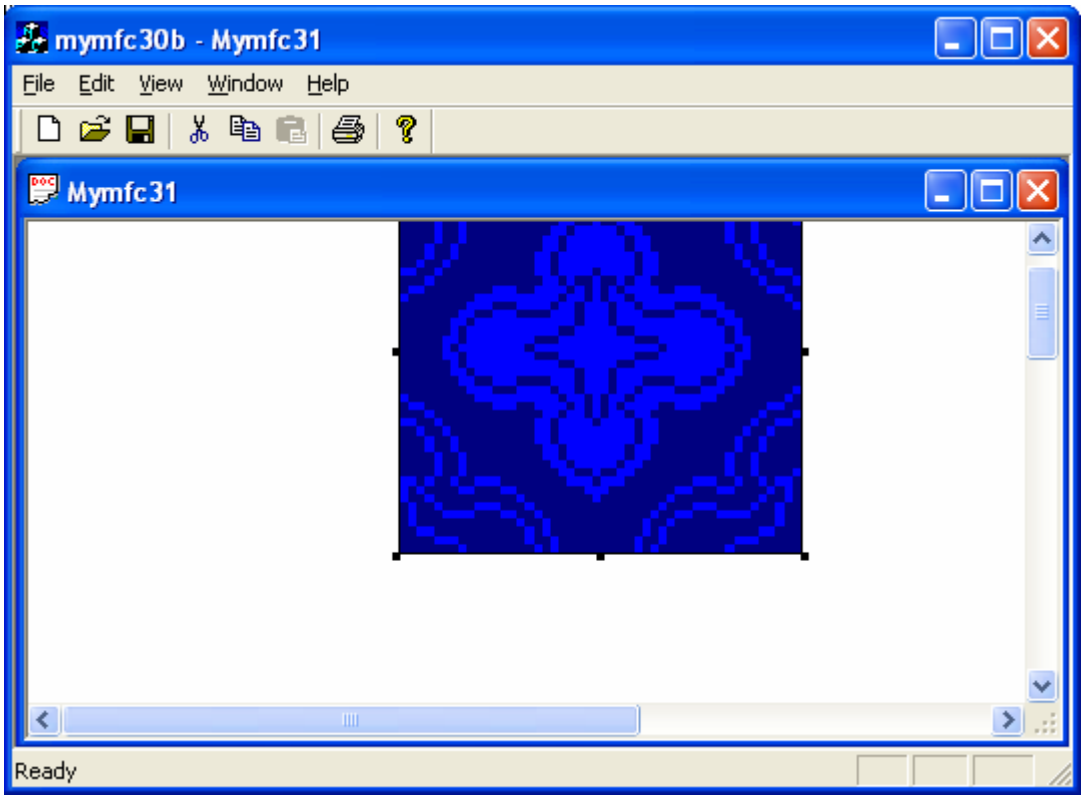

Figure 30: MYMFC30B - drag-and-drop functionality in the same client area.

Next, display bitmap as previously done, open another window by clicking the **File New** menu or the **New** icon. Arrange the windows side-by-side for viewing convenience. Drag the bitmap from one window and drop it in another window. The following Figure shows how the bitmap has been dragged from the left window and then dropped to the right window.

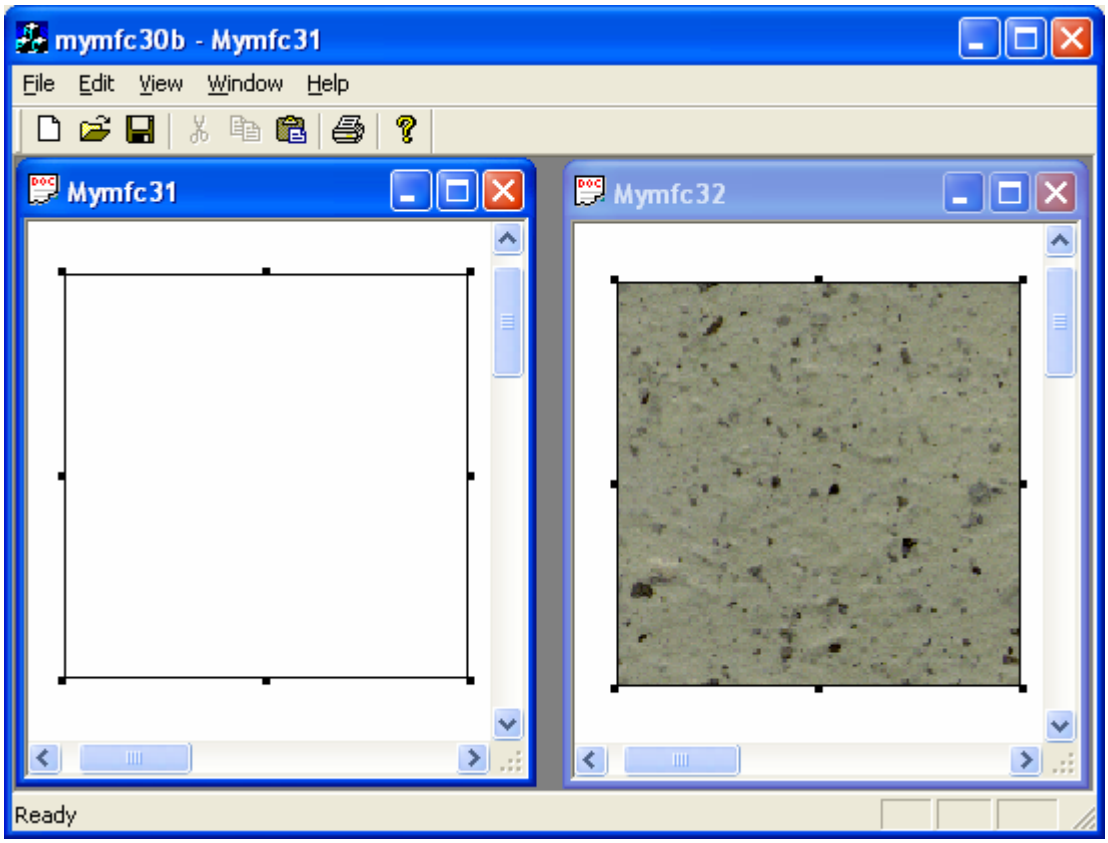

Figure 31: MYMFC30B - drag-and-drop functionality in the different client area.

Finally, launch two MYMFC30B programs. Open a sample bitmap in the first program and then drag and drop it in another MYMFC30B program as shown below.

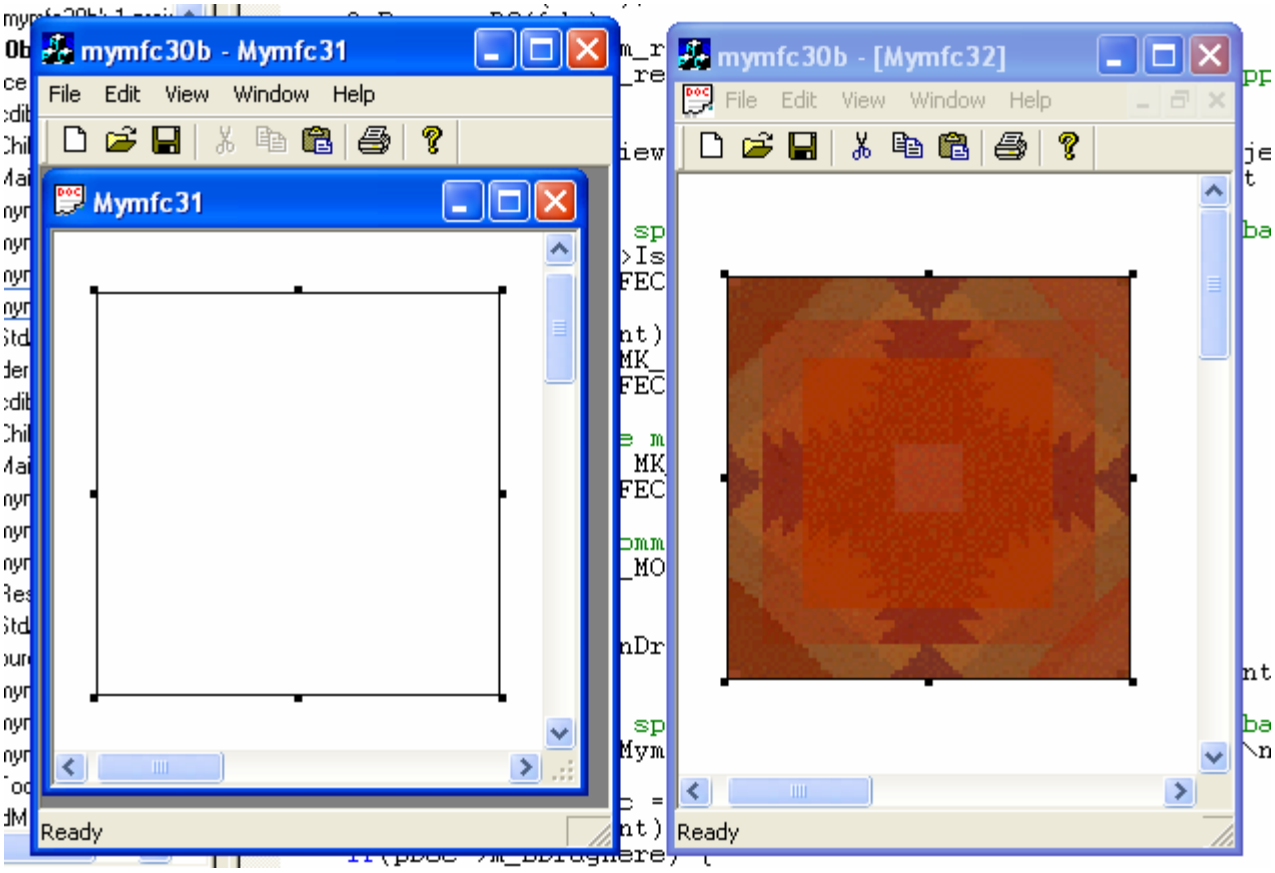

Figure 32: MYMFC30B - drag-and-drop functionality in different client area and different MYMFC30B program.

The Story

The CMymfc30bDoc Class

This class is just like the MYMFC30A version except for an added flag data member, `m_bDragHere`. This flag is `TRUE` when a drag-and-drop operation is in progress for this document. The flag is in the document and not in the view because it is possible to have multiple views attached to the same document. It doesn't make sense to drag a DIB from one view to another when both views reflect the document's `m_dib` member.

The CMymfc30bView Class

To start with, this class has three additional data members and a constructor that initializes all the data members, as shown here:

```
CRect m_rectTrackerEnter; // original logical coordinates
COleDropTarget m_dropTarget;
CSize m_dragOffset; // device coordinates

CMymfc30bView::CMymfc30bView(): m_sizeTotal(800, 1050), // 8-by-10.5 inches
                                                        //  when printed
    m_rectTracker(50, 50, 250, 250),
    m_dragOffset(0, 0),
    m_rectTrackerEnter(50, 50, 250, 250)
{
```

```
}
```

The `OnInitialUpdate()` function needs one additional line to register the drop target:

```
m_dropTarget.Register(this);
```

Following are the drag-and-drop virtual override functions. Note that `OnDrop()` replaces the DIB only if the document's `m_bDragHere` flag is `TRUE`, so if the user drops the DIB in the same window or in another window connected to the same document, nothing happens.

```
DROPEFFECT CMymfc30bView::OnDragEnter(COleDataObject* pDataObject,
    DWORD dwKeyState, CPoint point)
{
    TRACE("Entering CMymfc30bView::OnDragEnter, point = (%d, %d)\n",
        point.x, point.y);
    m_rectTrackerEnter = m_rectTracker; // save original coordinates
                                        //  for cursor leaving
                                        //  rectangle
    CClientDC dc(this);
    OnPrepareDC(&dc);
    dc.DrawFocusRect(m_rectTracker); // will be erased in OnDragOver
    return OnDragOver(pDataObject, dwKeyState, point);
}

DROPEFFECT CMymfc30bView::OnDragOver(COleDataObject* pDataObject,
    DWORD dwKeyState, CPoint point)
{
    if (!pDataObject->IsDataAvailable(CF_DIB))
    { return DROPEFFECT_NONE; }
    MoveTrackRect(point);
    if ((dwKeyState & MK_CONTROL) == MK_CONTROL)
    { return DROPEFFECT_COPY; }
    // Check for force move
    if ((dwKeyState & MK_ALT) == MK_ALT)
    { return DROPEFFECT_MOVE; }
    // default -- recommended action is move
    return DROPEFFECT_MOVE;
}

void CMymfc30bView::OnDragLeave()
{
    TRACE("Entering CMymfc30bView::OnDragLeave\n");
    CClientDC dc(this);
    OnPrepareDC(&dc);
    dc.DrawFocusRect(m_rectTracker);
    // Forget it ever happened
    m_rectTracker = m_rectTrackerEnter;
}

BOOL CMymfc30bView::OnDrop(COleDataObject* pDataObject,
    DROPEFFECT dropEffect, CPoint point)
{
    TRACE("Entering CMymfc30bView::OnDrop -- dropEffect = %d\n", dropEffect);
    BOOL bRet;
    CMymfc30bDoc* pDoc = GetDocument();
    MoveTrackRect(point);
    if (pDoc->m_bDragHere)
```

```
    {
        pDoc->m_bDragHere = FALSE;
        bRet = TRUE;
    }
    else
    { bRet = DoPasteDib(pDataObject); }
    return bRet;
}
```

The handler for the `WM_LBUTTONDOWN` message needs substantial overhaul. It must call `DoDragDrop()` if the cursor is inside the rectangle and `Track()` if it is on the rectangle border. The revised code is shown here:

```
void CMymfc30bView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CMymfc30bDoc* pDoc = GetDocument();
    if (m_tracker.HitTest(point) == CRectTracker::hitMiddle)
    {
        COleDataSource* pSource = SaveDib();
        if (pSource)
        {
            // DoDragDrop returns only after drop is complete
            CClientDC dc(this);
            OnPrepareDC(&dc);
            CPoint topleft = m_rectTracker.TopLeft();
            dc.LPtoDP(&topleft);
            // 'point' here is not the same as the point parameter in
            //   OnDragEnter, so we use this one to compute the offset
            m_dragOffset = point - topleft;  // device coordinates
            pDoc->m_bDragHere = TRUE;
            DROPEFFECT dropEffect = pSource->DoDragDrop(
                DROPEFFECT_MOVE | DROPEFFECT_COPY, CRect(0, 0, 0, 0));
            TRACE("after DoDragDrop -- dropEffect = %ld\n", dropEffect);
            if (dropEffect == DROPEFFECT_MOVE && pDoc->m_bDragHere)
            {
                pDoc>OnEditClearAll();
            }
            pDoc->m_bDragHere = FALSE;
            delete pSource;
        }
    }
    else
    {
        if (m_tracker.Track(this, point, FALSE, NULL))
        {
            CClientDC dc(this);
            OnPrepareDC(&dc);
            // should have some way to prevent it going out of bounds
            m_rectTracker = m_tracker.m_rect;
            // update logical coordinates
            dc.DPtoLP(m_rectTracker);
        }
    }
    Invalidate();
}
```

Finally, the new `MoveTrackRect()` helper function, shown here, moves the tracker's focus rectangle each time the `OnDragOver()` function is called. This job was done by `CRectTracker::Track` in the MYMFC30A example.

```
void CMymfc30bView::MoveTrackRect(CPoint point)
{
    CClientDC dc(this);
    OnPrepareDC(&dc);
    dc.DrawFocusRect(m_rectTracker);
    dc.LPtoDP(m_rectTracker);
    CSize sizeTrack = m_rectTracker.Size();
    CPoint newTopleft = point - m_dragOffset;  // still device
    m_rectTracker = CRect(newTopleft, sizeTrack);
    m_tracker.m_rect = m_rectTracker;
    dc.DPtoLP(m_rectTracker);
    dc.DrawFocusRect(m_rectTracker);
}
```

Windows Applications and Drag and Drop: `Dobjview`

I tested MYMFC30B with the Microsoft Office 2003 suite. I tried both drag-and-drop and clipboard transfers, with the results shown in the following table.

MYMFC30B	Word	Excel	PowerPoint	FrontPage
Sends clipboard data to	x	x (no palettes)	x	x
Accepts clipboard data from	x		x	x
Sends drag-drop data to	x		x	x
Accepts drag-drop data from	x			x

Table 4.

When I started to investigate why these programs were so uncooperative, I discovered a useful OLE utility called **Dobjview** (**DataObject** viewer).

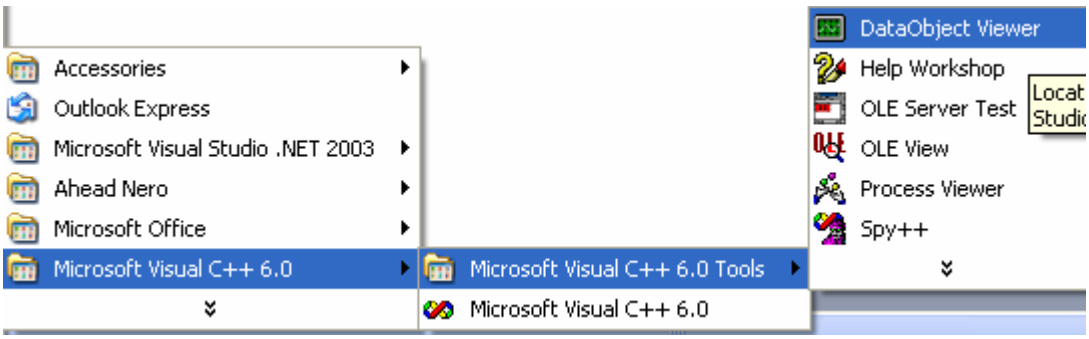

Figure 33: Invoking **DataObject Viewer**.

I could use Dobjview to examine a data object on the clipboard, and I could drag objects to the Dobjview window. Here's what I got when I dragged a picture from Microsoft Excel.

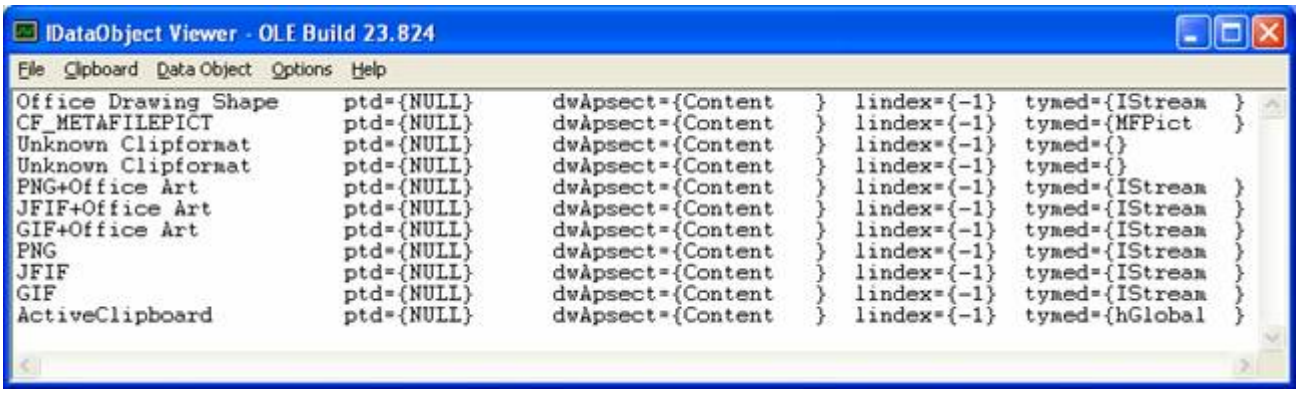

Figure 34: Testing MYMFC30B's drag and drop functionality among Microsoft Office applications.

No `CF_DIB` format is present. If you want pictures from Excel, you must enhance MYMFC30B to process metafiles. Another alternative is to rewrite the program with compound document support as described in Module 27. The OLE libraries contain code to display bitmaps and metafiles.

Conclusion

As you can see, MFC makes clipboard and drag-and-drop data transfer pretty easy. While you can always write all the code necessary to implement the interfaces (`IDataObject`, `IDropTarget`, and `IDropSource`), using MFC's implementations is much more convenient. While we've looked only at clipboard and drag and drop transfers through `IDataObject` in this Module, everything you learn about the `IDataObject` interface will carry forward to your study of compound documents in the next Module.

----------End part 2----------

Further reading and digging:

1. MSDN MFC 6.0 class library online documentation - used throughout this Tutorial.
2. MSDN MFC 7.0 class library online documentation - used in .Net framework and also backward compatible with 6.0 class library
3. MSDN Library
4. DCOM at MSDN.
5. COM+ at MSDN.
6. COM at MSDN.
7. Windows data type.
8. Win32 programming Tutorial.
9. The best of C/C++, MFC, Windows and other related books.
10. Unicode and Multibyte character set: Story and program examples.