Uniform Data Transfer: Clipboard Transfer and OLE Drag and Drop Part 1

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:

Intro

ActiveX technology includes a powerful mechanism for transferring data within and among Microsoft Windows-based applications. The COM `IDataObject` interface is the key element of what is known as **Uniform Data Transfer**. As you'll see, Uniform Data Transfer (UDT) gives you all sorts of options for the formatting and storage of your transferred data, going well beyond standard clipboard transfers.
Microsoft Foundation Class support is available for Uniform Data Transfer, but MFC's support for UDT is not so high-level as to obscure what's going on at the COM interface level. One of the useful applications of UDT is **OLE Drag and Drop**. Many developers want to use drag-and-drop capabilities in their applications, and drag-and-drop support means that programs now have a standard for information interchange. The MFC library supports drag-and-drop operations, and that, together with clipboard transfer, is the main focus of this Module.

The `IDataObject` Interface

The `IDataObject` interface is used for clipboard transfers and drag-and-drop operations, but it's also used in **compound documents**, **ActiveX Controls**, and **custom OLE** features. In his book Inside OLE, 2d ed. (Microsoft Press, 1995) Kraig Brockschmidt says, "Think of objects as little piles of stuff." The `IDataObject` interface helps you move those piles around, no matter what kind of stuff they contain.
If you were programming at the Win32 level, you would write C++ code that supported the `IDataObject` interface. Your program would then construct data objects of this class, and you would manipulate those objects with the `IDataObject` member functions. In this Module you'll see how to accomplish the same results by using MFC's implementation of `IDataObject`. Let's start by taking a quick look at why the OLE clipboard is an improvement on the regular Windows clipboard.

How `IDataObject` Improves on Standard Clipboard Support

There has never been much MFC support for the Windows Clipboard. If you've written programs for the clipboard already, you've used Win32 clipboard functions such as `OpenClipboard()`, `CloseClipboard()`, `GetClipboardData()`, and `SetClipboardData()`. One program copies a single data element of a specified format to the clipboard, and another program selects the data by format code and pastes it. Standard clipboard formats include global memory (specified by an `HGLOBAL` variable) and various GDI objects, such as bitmaps and metafiles (specified by their handles). Global memory can contain text as well as custom formats.
The `IDataObject` interface picks up where the Windows Clipboard leaves off. To make a long story short, you transfer a single `IDataObject` pointer to or from the clipboard instead of transferring a series of discrete formats. The underlying data object can contain a whole array of formats. Those formats can carry information about target devices, such as printer characteristics, and they can specify the data's aspect or view. The standard aspect is content. Other aspects include an icon for the data and a thumbnail picture.
Note that the `IDataObject` interface specifies the storage medium of a data object format. Conventional clipboard transfer relies exclusively on global memory. The `IDataObject` interface permits the transmission of a disk filename or a structured storage pointer instead. Thus, if you want to transfer a very large block of data that's already in a disk file, you don't have to waste time copying it to and from a memory block.
In case you were wondering, `IDataObject` pointers are compatible with programs that use existing clipboard transfer methods. The format codes are the same. Windows takes care of the conversion to and from the data object. Of course, if an OLE-aware program puts an `IStorage` pointer in a data object and puts the object on the clipboard; older, non-OLE-aware programs are unable to read that format.

The FORMATETC and STGMEDIUM Structures

Before you're ready for the `IDataObject` member functions, you need to examine two important COM structures that are used as parameter types: the `FORMATETC` structure and the `STGMEDIUM` structure.

FORMATETC

The `FORMATETC` structure is often used instead of a clipboard format to represent data format information. However, unlike the clipboard format, the `FORMATETC` structure includes information about a target device, the aspect or view of the data, and a storage medium indicator.

```
typedef struct tagFORMATETC
{
    CLIPFORMAT      cfFormat;
    DVTARGETDEVICE  *ptd;
    DWORD           dwAspect;
    LONG            lindex;
    DWORD           tymed;
}FORMATETC, *LPFORMATETC;
```

Here are the members of the `FORMATETC` structure.

Type	Name	Description
CLIPFORMAT	cfFormat	Structure that contains clipboard formats, such as **standard interchange formats** for example, `CF_TEXT`, which is a text format, and `CF_DIB`, which is an image compression format, **custom formats** such as rich text format, and **OLE formats** used to create linked or embedded objects.
DVTARGETDEVICE*	ptd	Structure that contains information about the target device for the data, including the device driver name (can be `NULL`).
DWORD	dwAspect	A `DVASPECT` enumeration constant (`DVASPECT_CONTENT`, `DVASPECT _THUMBNAIL`, and so on).
LONG	lindex	Usually `-1`.
DWORD	tymed	Specifies type of media used to transfer the object's data (`TYMED_HGLOBAL`, `TYMED_FILE`, `TYMED_ISTORAGE`, and so on).

Table 1.

An individual data object accommodates a collection of FORMATETC elements, and the IDataObject interface provides a way to enumerate them. A useful macro for filling in a FORMATETC structure appears below.

```
#define SETFORMATETC(fe, cf, asp, td, med, li)   \
    ((fe).cfFormat=cf, \
    (fe).dwAspect=asp, \
    (fe).ptd=td, \
    (fe).tymed=med, \
    (fe).lindex=li)
```

STGMEDIUM

The other important structure for IDataObject members is the STGMEDIUM structure.

```
typedef struct tagSTGMEDIUM
{
    DWORD tymed;
    [switch_type(DWORD), switch_is((DWORD) tymed)]
    union {
        [case(TYMED_GDI)]       HBITMAP        hBitmap;
        [case(TYMED_MFPICT)]    HMETAFILEPICT  hMetaFilePict;
        [case(TYMED_ENHMF)]     HENHMETAFILE   hEnhMetaFile;
        [case(TYMED_HGLOBAL)]   HGLOBAL        hGlobal;
        [case(TYMED_FILE)]      LPWSTR         lpszFileName;
        [case(TYMED_ISTREAM)]   IStream        *pstm;
        [case(TYMED_ISTORAGE)]  IStorage       *pstg;
        [default];
    };
    [unique] IUnknown *pUnkForRelease;
}STGMEDIUM;
typedef STGMEDIUM *LPSTGMEDIUM;
```

The STGMEDIUM structure is a global memory handle used for operations involving data transfer. Here are the members.

Type	Name	Description
DWORD	tymed	Storage medium value used in marshaling and unmarshaling routines.
HBITMAP	hBitmap	Bitmap handle*.
HMETAFILEPICT	hMetaFilePict	Metafile handle*.
HENHMETAFILE	hEnhMetaFile	Enhanced metafile handle*.
HGLOBAL	hGlobal	Global memory handle*.
LPOLESTR	lpszFileName	Disk filename (double-byte)*.
ISTREAM*	pstm	IStream interface pointer*.
ISTORAGE*	pstg	IStorage interface pointer*.
IUNKNOWN	pUnkForRelease	Used by clients to call Release() for formats with interface pointers.

Table 2.

* This member is part of a union, including handles, strings, and interface pointers used by the receiving process to access the transferred data.
As you can see, the STGMEDIUM structure specifies where data is stored. The tymed variable determines which union member is valid.

The IDataObject Interface Member Functions

This interface has nine member functions.

IDataObject Methods	Description
`GetData()`	Renders the data described in a `FORMATETC` structure and transfers it through the `STGMEDIUM` structure.
`GetDataHere()`	Renders the data described in a `FORMATETC` structure and transfers it through the `STGMEDIUM` structure allocated by the caller.
`QueryGetData()`	Determines whether the data object is capable of rendering the data described in the `FORMATETC` structure.
`GetCanonicalFormatEtc()`	Provides a potentially different but logically equivalent `FORMATETC` structure.
`SetData()`	Provides the source data object with data described by a `FORMATETC` structure and an `STGMEDIUM` structure.
`EnumFormatEtc()`	Creates and returns a pointer to an object to enumerate the `FORMATETC` supported by the data object.
`DAdvise()`	Creates a connection between a data object and an advise sink so the advise sink can receive notifications of changes in the data object.
`DUnadvise()`	Destroys a notification previously set up with the `DAdvise()` method.
`EnumDAdvise()`	Creates and returns a pointer to an object to enumerate the current advisory connections.

Table 3.

Following are the functions that are important for this Module.

```
HRESULT EnumFormatEtc(DWORD dwDirection, IEnumFORMATETC ppEnum);
```

If you have an `IDataObject` pointer for a data object, you can use `EnumFormatEtc()` to enumerate all the formats that it supports. This is an ugly API that the MFC library insulates you from. You'll learn how this happens when you examine the `COleDataObject` class.

```
HRESULT GetData(FORMATETC* pFEIn, STGMEDIUM* pSTM);
```

`GetData()` is the most important function in the interface. Somewhere, up in the sky, is a data object, and you have an `IDataObject` pointer to it. You specify, in a `FORMATETC` variable, the exact format you want to use when you retrieve the data, and you prepare an empty `STGMEDIUM` variable to accept the results. If the data object has the format you want, `GetData()` fills in the `STGMEDIUM` structure. Otherwise, you get an error return value.

```
HRESULT QueryGetData(FORMATETC* pFE);
```

You call `QueryGetData()` if you're not sure whether the data object can deliver data in the format specified in the `FORMATETC` structure. The return value says, "Yes, I can" (`S_OK`) or "No, I can't" (an error code). Calling this function is definitely more efficient than allocating a `STGMEDIUM` variable and calling `GetData()`.

```
HRESULT SetData(FORMATETC* pFEIn, STGMEDIUM* pSTM, BOOL fRelease);
```

Data objects rarely support `SetData()`. Data objects are normally loaded with formats in their own server module; clients retrieve data by calling `GetData()`. With `SetData()`, you'd be transferring data in the other direction, like pumping water from your house back to the water company.

Other IDataObject Member Functions—Advisory Connections

The interface contains other important functions that let you implement an advisory connection. When the program using a data object needs to be notified whether the object's data changes, the program can pass an `IAdviseSink` pointer to the object by calling the `IDataObject::DAdvise` function. The object then calls various `IAdviseSink` member functions, which the client program implements. You won't need advisory connections for drag-and-drop operations, but you will need them when you get to embedding in Module 27.

MFC Uniform Data Transfer Support

The MFC library does a lot to make data object programming easier. As you study the MFC data object classes, you'll start to see a pattern in MFC COM support. At the component end, the MFC library provides a base class that implements one or more OLE interfaces. The interface member functions call virtual functions that you override in your derived class. At the client end, the MFC library provides a class that wraps an interface pointer. You call simple member functions that use the interface pointer to make COM calls.
The terminology needs some clarification here. The data object that's been described is the actual C++ object that you construct, and that's the way Brockschmidt uses the term. In the MFC documentation, a data object is what the client program sees through an `IDataObject` pointer. A data source is the object you construct in a component program.

The COleDataSource Class

When you want to use a data source, you construct an object of class `COleDataSource`, which implements the `IDataObject` interface (without advisory connection support). This class builds and manages a collection of data formats stored in a cache in memory. A data source is a regular COM object that keeps a reference count. Usually, you construct and fill a data source, and then you pass it to the clipboard or drag and drop it in another location, never to worry about it again. If you decide not to pass off a data source, you can invoke the destructor, which cleans up all its formats.
Following are some of the more useful member functions of the `COleDataSource` class.

```
void CacheData(CLIPFORMAT cfFormat, STGMEDIUM* lpStgMedium, FORMATETC*
lpFormatEtc = NULL);
```

This function inserts an element in the data object's cache for data transfer. The `lpStgMedium` parameter points to the data, and the `lpFormatEtc` parameter describes the data. If, for example, the `STGMEDIUM` structure specifies a disk filename, that filename gets stored inside the data object. If `lpFormatEtc` is set to NULL, the function fills in a `FORMATETC` structure with default values. It's safer, though, if you create your `FORMATETC` variable with the tymed member set.

```
void CacheGlobalData(CLIPFORMAT cfFormat, HGLOBAL hGlobal, FORMATETC*
lpFormatEtc = NULL);
```

You call this specialized version of `CacheData()` to pass data in global memory (identified by an `HGLOBAL` variable). The data source object is considered the owner of that global memory block, so you should not free it after you cache it. You can usually omit the `lpFormatEtc` parameter. The `CacheGlobalData()` function does not make a copy of the data.

```
DROPEFFECT DoDragDrop(DWORD dwEffects = DROPEFFECT_COPY|DROPEFFECT_MOVE|
DROPEFFECT_LINK, LPCRECT lpRectStartDrag = NULL, COleDropSource* pDropSource
= NULL);
```

You call this function for drag-and-drop operations on a data source. You'll see it used in the MYMFC30B example.

```
void SetClipboard(void);
```

The `SetClipboard()` function, which you'll see in the MYMFC30A example, calls the `OleSetClipboard()` function to put a data source on the Windows Clipboard. The clipboard is responsible for deleting the data source and thus for freeing the global memory associated with the formats in the cache. When you construct a `COleDataSource` object and call `SetClipboard()`, COM calls `AddRef()` on the object.

The COleDataObject Class

This class is on the destination side of a data object transfer. Its base class is `CCmdTarget`, and it has a public member `m_lpDataObject` that holds an `IDataObject` pointer. That member must be set before you can effectively use the object. The class destructor only calls `Release()` on the `IDataObject` pointer.
Following are a few of the more useful `COleDataObject` member functions.

```
BOOL AttachClipboard(void);
```

As Brockschmidt points out, OLE clipboard processing is internally complex. From your point of view, however, it's straightforward, as long as you use the `COleDataObject` member functions. You first construct an "empty" `COleDataObject` object, and then you call `AttachClipboard()`, which calls the global `OleGetClipboard()` function. Now the `m_lpDataObject` data member points back to the source data object (or so it appears), and you can access its formats.
If you call the `GetData()` member function to get a format, you must remember that the clipboard owns the format and you cannot alter its contents. If the format consists of an `HGLOBAL` pointer, you must not free that memory and you cannot hang on to the pointer. If you need to have long-term access to the data in global memory, consider calling `GetGlobalData()` instead.
If a non-COM-aware program copies data onto the clipboard, the `AttachClipboard()` function still works because COM invents a data object that contains formats corresponding to the regular Windows data on the clipboard.

```
void BeginEnumFormats(void); BOOL GetNextFormat(FORMATETC* lpFormatEtc);
```

These two functions allow you to iterate through the formats that the data object contains. You call `BeginEnumFormats()` first, and then you call `GetNextFormat()` in a loop until it returns `FALSE`.

```
BOOL GetData(CLIPFORMAT cfFormat, STGMEDIUM* lpStgMedium FORMATETC*
lpFormatEtc = NULL);
```

This function calls `IDataObject::GetData` and not much more. The function returns `TRUE` if the data source contains the format you asked for. You generally need to supply the `lpFormatEtc` parameter.

```
HGLOBAL GetGlobalData(CLIPFORMAT cfFormat, FORMATETC* lpFormatEtc = NULL);
```

Use the `GetGlobalData()` function if you know your requested format is compatible with global memory. This function makes a copy of the selected format's memory block, and it gives you an `HGLOBAL` handle that you must free later. You can often omit the `lpFormatEtc` parameter.

```
BOOL IsDataAvailable(CLIPFORMAT cfFormat, FORMATETC* lpFormatEtc = NULL);
```

The `IsDataAvailable()` function tests whether the data object contains a given format.

MFC Data Object Clipboard Transfer

Now that you've seen the `COleDataObject` and `COleDataSource` classes, you'll have an easy time doing clipboard data object transfers. But why not just do clipboard transfers the old way with `GetClipboardData()` and `SetClipboardData()`? You could for most common formats, but if you write functions that process data objects, you can use those same functions for drag and drop.
Figure 1 shows the relationship between the clipboard and the `COleDataSource` and `COleDataObject` classes.

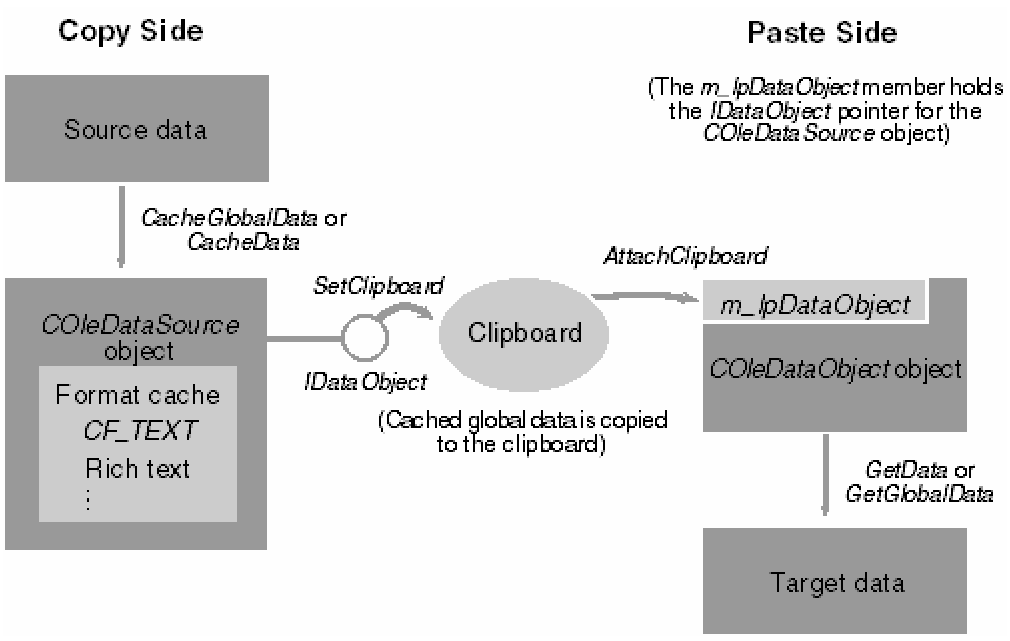


Figure 1: MFC OLE clipboard processing.

You construct a `COleDataSource` object on the copy side, and then you fill its cache with formats. When you call `SetClipboard()`, the formats are copied to the clipboard. On the paste side, you call `AttachClipboard()` to attach an `IDataObject` pointer to a `COleDataObject` object, after which you can retrieve individual formats.

Suppose you have a document-view application whose document has a `CString` data member `m_strText`. You want to use view class command handler functions that copy to and paste from the clipboard. Before you write those functions, write two helper functions. The first, `SaveText()`, creates a data source object from the contents of `m_strText`. The function constructs a `COleDataSource` object, and then it copies the string contents to global memory. Last it calls `CacheGlobalData()` to store the `HGLOBAL` handle in the data source object. Here is the `SaveText()` code:

```
COleDataSource* CMyView::SaveText()
{
    CEx26fDoc* pDoc = GetDocument();
    if (!pDoc->m_strtext.IsEmpty())
    {
        COleDataSource* pSource = new COleDataSource();
        int nTextSize = GetDocument()->m_strText.GetLength() + 1;
        HGLOBAL hText = ::GlobalAlloc(GMEM_SHARE, nTextSize);
        LPSTR pText = (LPSTR)::GlobalLock(hText);
        ASSERT(pText);
        strcpy(pText, GetDocument()->m_strText);
        ::GlobalUnlock(hText);
        pSource->CacheGlobalData(CF_TEXT, hText);
        return pSource;
    }
    return NULL;
}
```

The second helper function, `DoPasteText()`, fills in `m_strText` from a data object specified as a parameter. We're using `COleDataObject::GetData` here instead of `GetGlobalData()` because `GetGlobalData()` makes a copy of the global memory block. That extra copy operation is unnecessary because we're copying the text to the `CString` object. We don't free the original memory block because the data object owns it. Here is the `DoPasteText()` code:

```
    // Memory is MOVEABLE, so we must use GlobalLock!
    SETFORMATETC(fmt, CF_TEXT, DVASPECT_CONTENT, NULL, TYMED_HGLOBAL, -1);
    VERIFY(pDataObject->GetData(CF_TEXT, &stg, &fmt));
    HGLOBAL hText = stg.hGlobal;
    GetDocument()->m_strText = (LPSTR)::GlobalLock(hText);
    ::GlobalUnlock(hText);
    return TRUE;
}
```

Here are the two command handler functions:

```
void CMyView::OnEditCopy()
{
    COleDataSource* pSource = SaveText();
    if (pSource)
    { pSource->SetClipboard(); }
}

void CMyView::OnEditPaste()
{
    COleDataObject dataObject;
    VERIFY(dataObject.AttachClipboard());
    DoPasteText(&dataObject);
    // dataObject released
}
```

The MFC CRectTracker Class

The CRectTracker class is useful in both OLE and non-OLE programs. It allows the user to move and resize a rectangular object in a view window. There are two important data members: the m_nStyle member determines the border, resize handle, and other characteristics; and the m_rect member holds the device coordinates for the rectangle.
The important member functions follow.

```
void Draw(CDC* pDC) const;
```

The Draw() function draws the tracker, including border and resize handles, but it does not draw anything inside the rectangle. That's your job.

```
BOOL Track(CWnd* pWnd, CPoint point, BOOL bAllowInvert = FALSE, CWnd*
pWndClipTo = NULL);
```

You call this function in a WM_LBUTTONDOWN handler. If the cursor is on the rectangle border, the user can resize the tracker by holding down the mouse button; if the cursor is inside the rectangle, the user can move the tracker. If the cursor is outside the rectangle, Track() returns FALSE immediately; otherwise, Track returns TRUE only when the user releases the mouse button. That means Track works a little like CDialog::DoModal. It contains its own message dispatch logic.

```
int HitTest(CPoint point) const;
```

Call HitTest() if you need to distinguish between mouse button hits inside and on the tracker rectangle. The function returns immediately with the hit status in the return value.

```
BOOL SetCursor(CWnd* pWnd, UINT nHitTest) const;
```

Call this function in your view's WM_SETCURSOR handler to ensure that the cursor changes during tracking. If SetCursor() returns FALSE, call the base class OnSetCursor() function; if SetCursor() returns TRUE, you return TRUE.

```
CRectTracker Rectangle Coordinate Conversion
```

You must deal with the fact that the `CRectTracker::m_rect` member stores device coordinates. If you are using a scrolling view or have otherwise changed the mapping mode or viewport origin, you must do coordinate conversion. Here's a strategy:

1. Define a `CRectTracker` data member in your view class. Use the name `m_tracker`.
2. Define a separate data member in your view class to hold the rectangle in logical coordinates. Use the name `m_rectTracker`.
3. In your view's `OnDraw()` function, set `m_rect` to the updated device coordinates, and then draw the tracker. This adjusts for any scrolling since the last `OnDraw()`. Some sample code appears below.

```
m_tracker.m_rect = m_rectTracker;
pDC->LPtoDP(m_tracker.m_rect); // tracker requires device
                               //  coordinates
m_tracker.Draw(pDC);
```

4. In your mouse button down message handler, call `Track()`, set `m_rectTracker` to the updated logical coordinates, and call `Invalidate()`, as shown here:

```
if (m_tracker.Track(this, point, FALSE, NULL))
{
    CClientDC dc(this);
    OnPrepareDC(&dc);
    m_rectTracker = m_tracker.m_rect;
    dc.DPtoLP(m_rectTracker);
    Invalidate();
}
```

The MYMFC30A Example: A Data Object Clipboard

This example uses the `CDib` class from Module 21 (MYMFC26C). Here you'll be able to move and resize the DIB image with a tracker rectangle, and you'll be able to copy and paste the DIB to and from the clipboard using a COM data object. The example also includes functions for reading DIBs from and writing DIBs to BMP files.
If you create such an example from scratch, use AppWizard to create MDI, without any **ActiveX** or **Automation** options and then add the following line in your **StdAfx.h** file:

```
#include <afxole.h>
```

Add the following call at the start of the application's `InitInstance()` function:

```
AfxOleInit();
```

To prepare MYMFC30A, open the Mymfc30a.dsw workspace and then build the project. Run the application, and paste a bitmap into the rectangle by choosing **Paste From** on the **Edit** menu. You'll see an MDI application similar to the one shown in Figure 2.

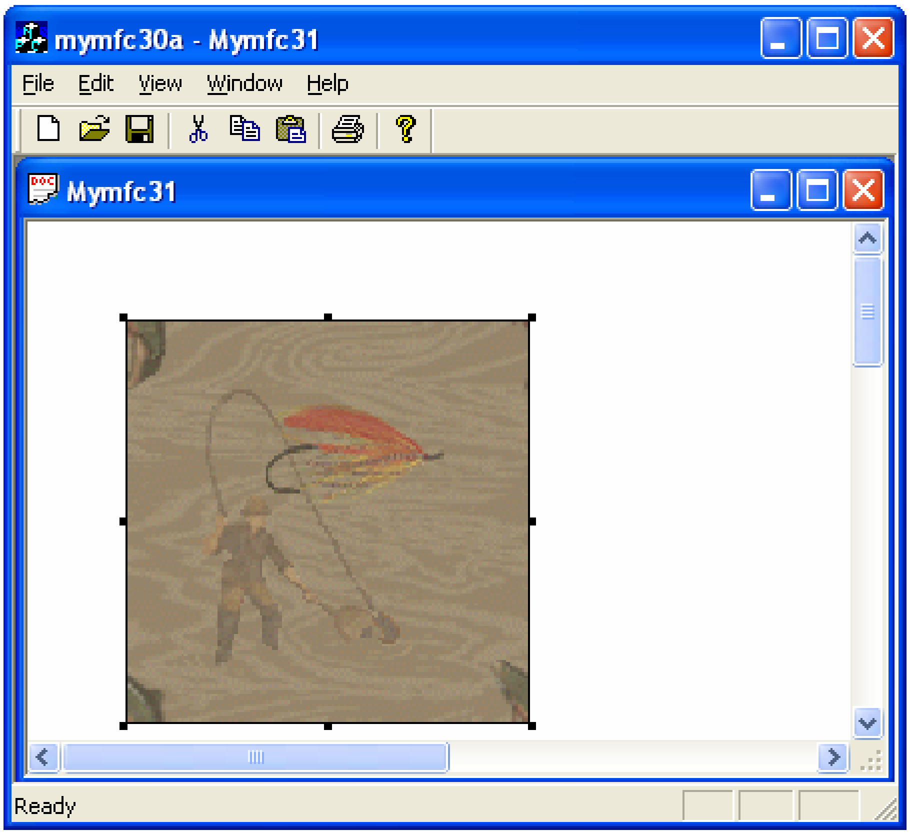
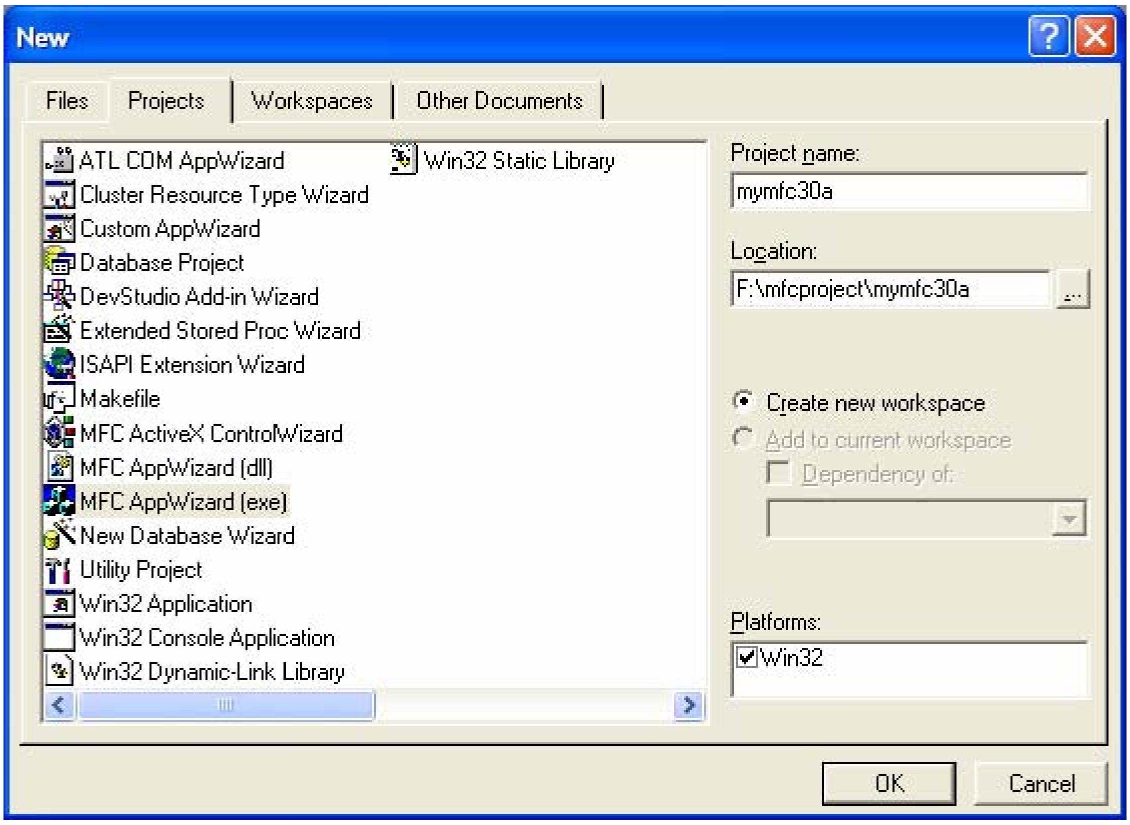


Figure 2: The MYMFC30A program in operation.

The MYMFC30A Steps From Scratch

The following are the steps to build MYMFC30A program from scratch. This is MDI without Automation and ActiveX support. The View base class is `CScrollView`.

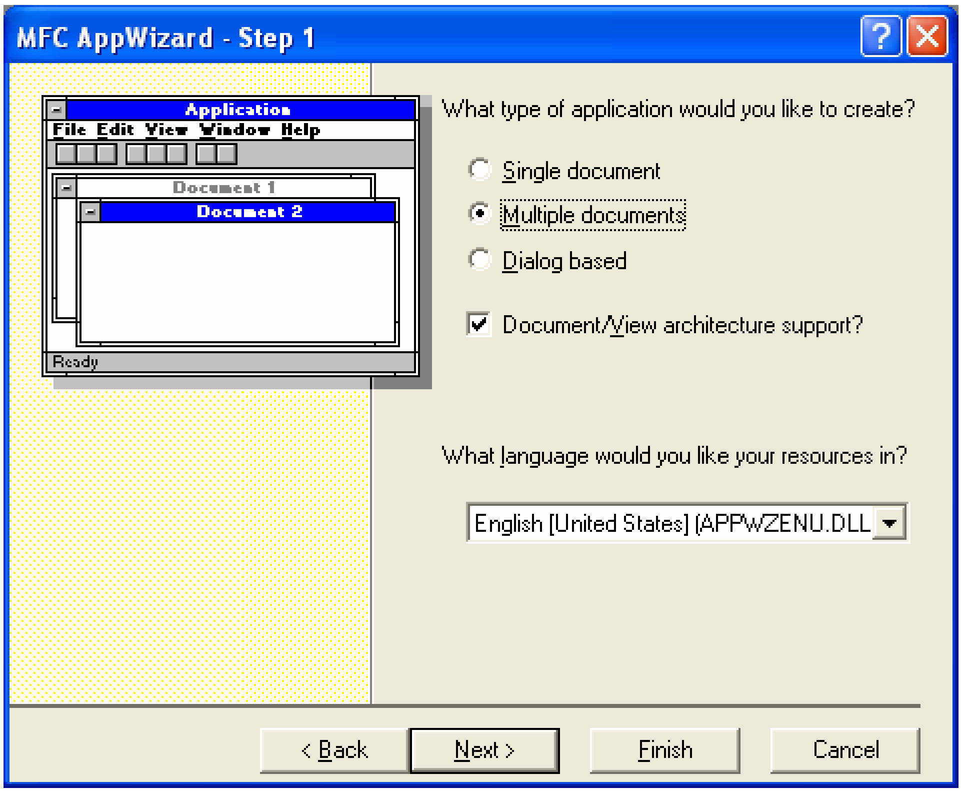


Project name: mymfc30a

Location: F:\mfcproject\mymfc30a

Platforms: ✓ Win32

Figure 3: MYMFC30A – Visual C++ new project dialog.



What type of application would you like to create?

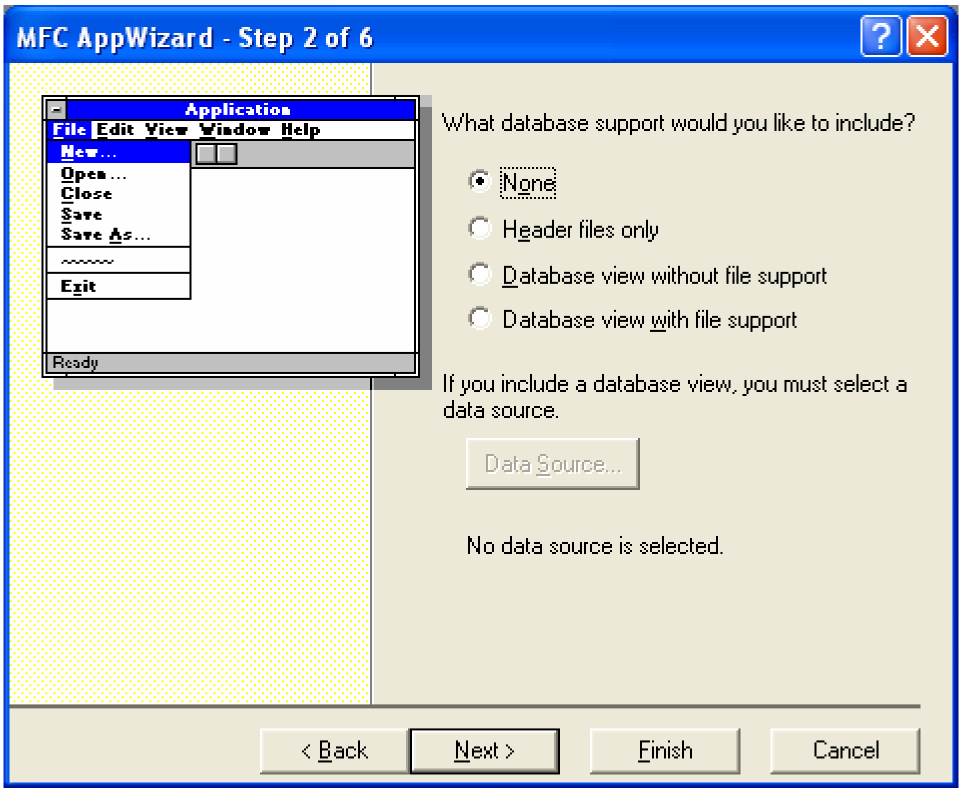
Single document [✓ Multiple documents] Dialog based

✓ Document/View architecture support?

What language would you like your resources in?

English [United States] (APPWZENU.DLL

Figure 4: MYMFC30A – AppWizard step 1 of 6, select **Multiple documents** option.



What database support would you like to include?

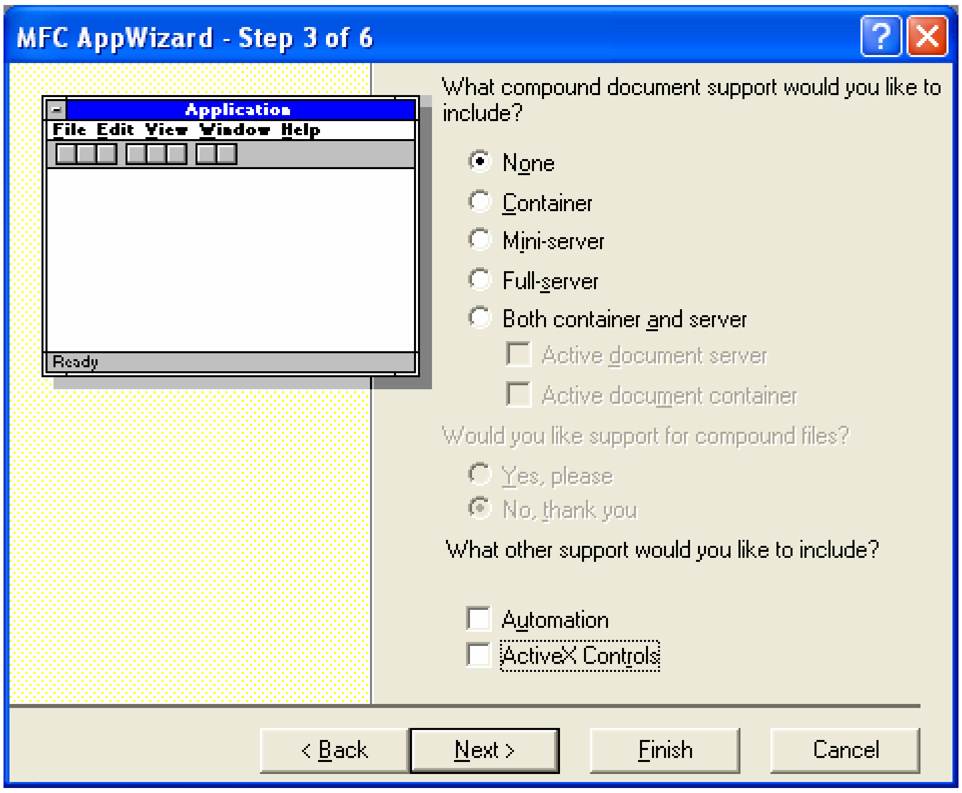
- [✓ None]
- Header files only
- Database view without file support
- Database view with file support

If you include a database view, you must select a data source.

No data source is selected.

Figure 5: MYMFC30A – AppWizard step 2 of 6.

For step 3 of 6, don’t forget to deselect the **Automation** and **ActiveX Controls** check boxes.



What compound document support would you like to include?

- [✓ None]
- Container
- Mini-server
- Full-server
- Both container and server
  - ☐ Active document server
  - ☐ Active document container

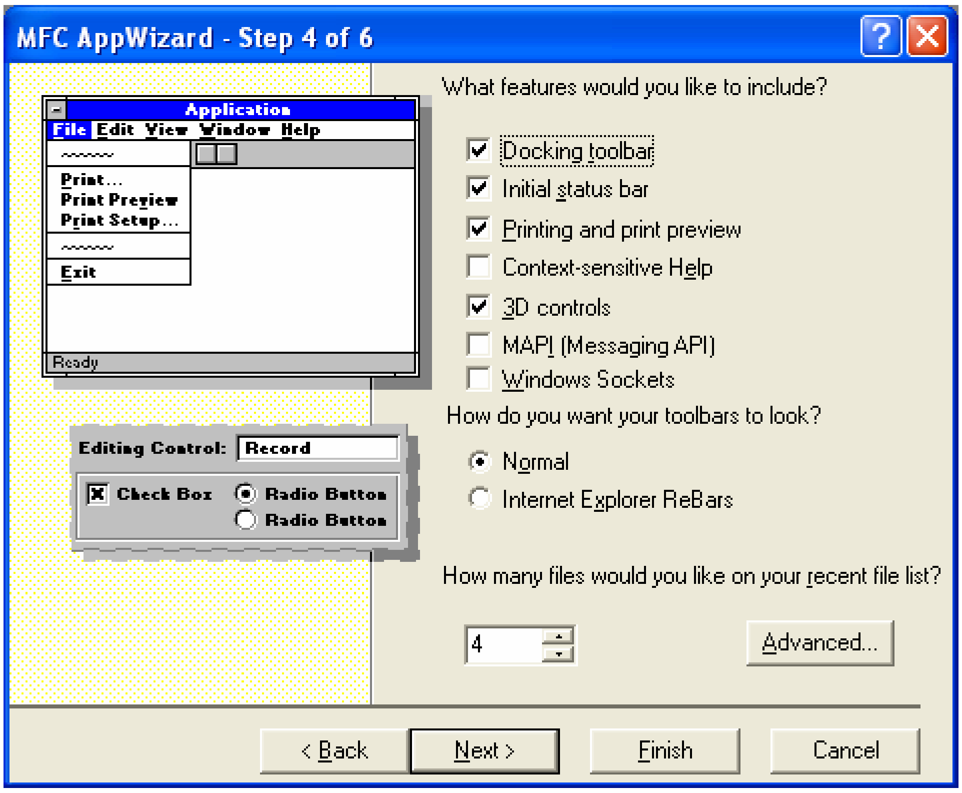
Would you like support for compound files?

- Yes, please
- [✓ No, thank you]

What other support would you like to include?

- ☐ Automation
- ☐ ActiveX Controls

Figure 6: MYMFC30A – AppWizard step 3 of 6.



What features would you like to include?

[✓] Docking toolbar
[✓] Initial status bar
[✓] Printing and print preview
[ ] Context-sensitive Help
[✓] 3D controls
[ ] MAPI (Messaging API)
[ ] Windows Sockets

How do you want your toolbars to look?

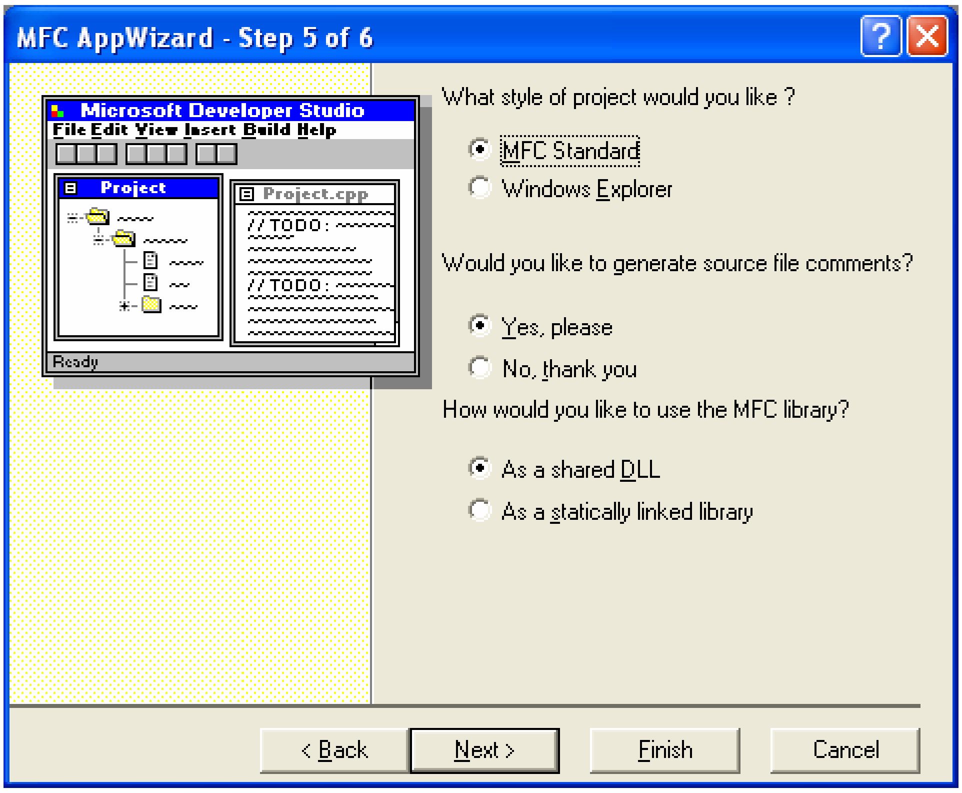
[✓ Normal]
Internet Explorer ReBars

How many files would you like on your recent file list? 4

Advanced...

< Back | Next > | Finish | Cancel

Figure 7: MYMFC30A – AppWizard step 4 of 6.


MFC AppWizard - Step 5 of 6
Microsoft Developer Studio
File Edit View Insert Build Help
Project
Project.cpp
// TODO:
// TODO:
Ready

What style of project would you like ?

[✓ MFC Standard]
Windows Explorer

Would you like to generate source file comments?

[✓ Yes, please]
No, thank you

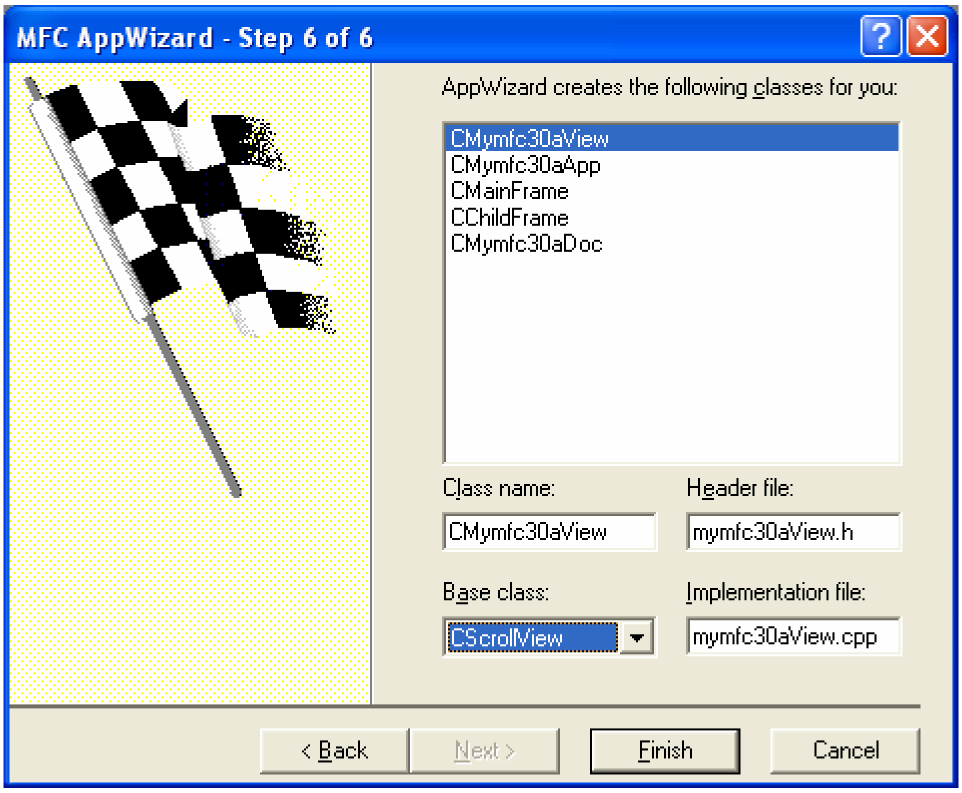
How would you like to use the MFC library?

[✓ As a shared DLL]
As a statically linked library

< Back | Next > | Finish | Cancel

Figure 8: MYMFC30A – AppWizard step 5 of 6.

Change the `CView` to `CScrollView` base class for `CMymfc30aView` class.

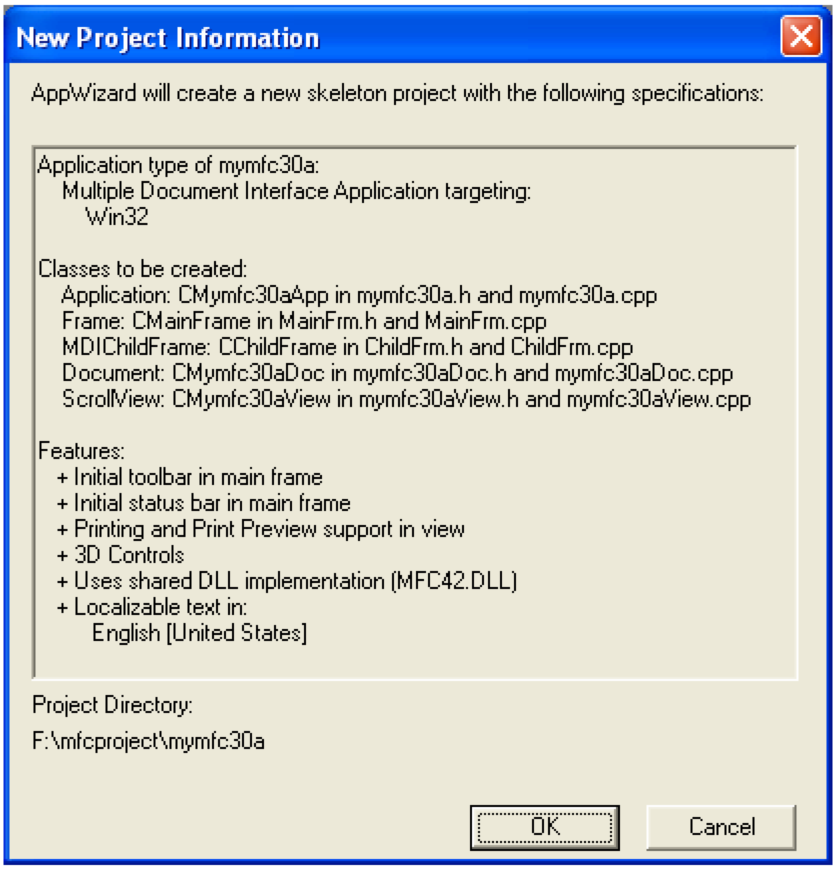


AppWizard creates the following classes for you:

CMymfc30aView
CMymfc30aApp
CMainFrame
CChildFrame
CMymfc30aDoc

Class name: CMymfc30aView
Header file: mymfc30aView.h
Base class: CScrollView
Implementation file: mymfc30aView.cpp

Figure 9: MYMFC30A – AppWizard step 6 of 6.



AppWizard will create a new skeleton project with the following specifications:

```
Application type of mymfc30a:
    Multiple Document Interface Application targeting:
        Win32

Classes to be created:
    Application: CMymfc30aApp in mymfc30a.h and mymfc30a.cpp
    Frame: CMainFrame in MainFrm.h and MainFrm.cpp
    MDIChildFrame: CChildFrame in ChildFrm.h and ChildFrm.cpp
    Document: CMymfc30aDoc in mymfc30aDoc.h and mymfc30aDoc.cpp
    ScrollView: CMymfc30aView in mymfc30aView.h and mymfc30aView.cpp

Features:
   + Initial toolbar in main frame
   + Initial status bar in main frame
   + Printing and Print Preview support in view
   + 3D Controls
   + Uses shared DLL implementation (MFC42.DLL)
   + Localizable text in:
        English [United States]
```

Project Directory:
F:\mfcproject\mymfc30a

Figure 10: MYMFC30A project summary.

Firstly, add the `CDib` class (device independent bitmap).

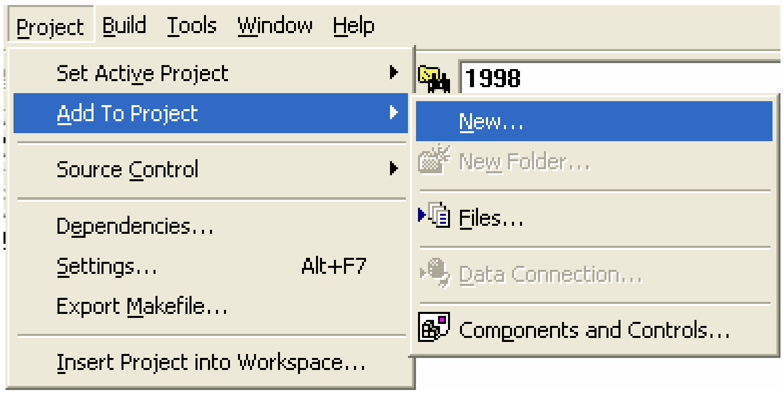
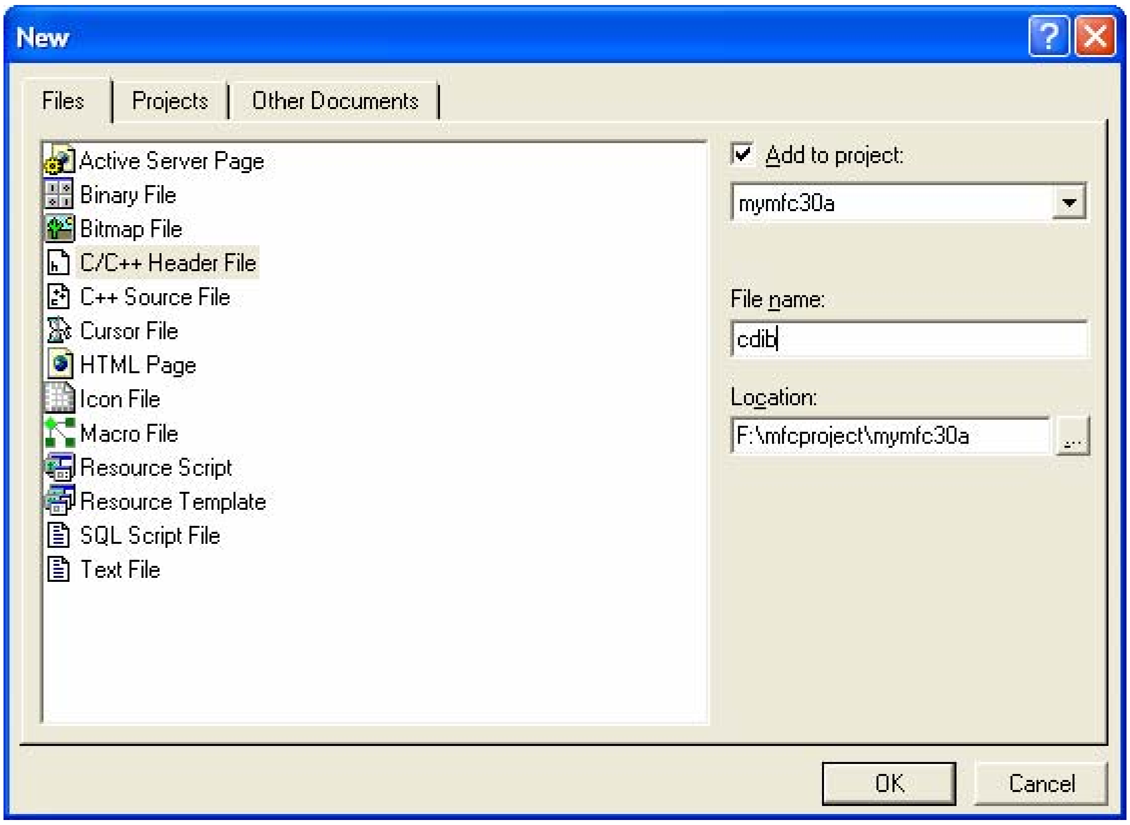


Figure 11: Adding new class to project.



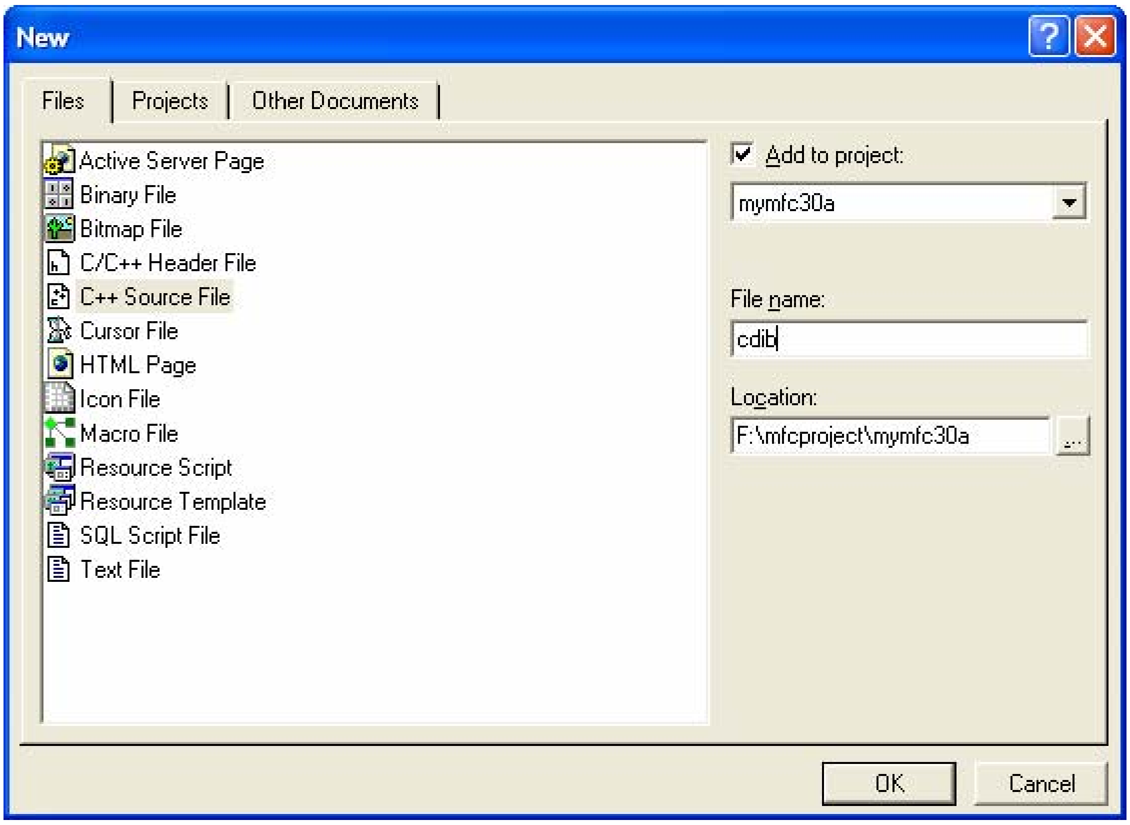
Add to project: [✓] mymfc30a

File name: cdib

Location: F:\mfcproject\mymfc30a

Figure 12: Adding an empty **cdib.h**, a header file to project.

Copy and paste the cdib.h file content. Repeat the same step but select the C++ Source File for cdib.cpp. Here, we do not use the ClassWizard to add new class, so you won't find the `CDib` class in ClassWizard, though the ClassWizard database file (**CLW**) can be rebuilt to include the class.



☑ Add to project: mymfc30a

File name: cdib

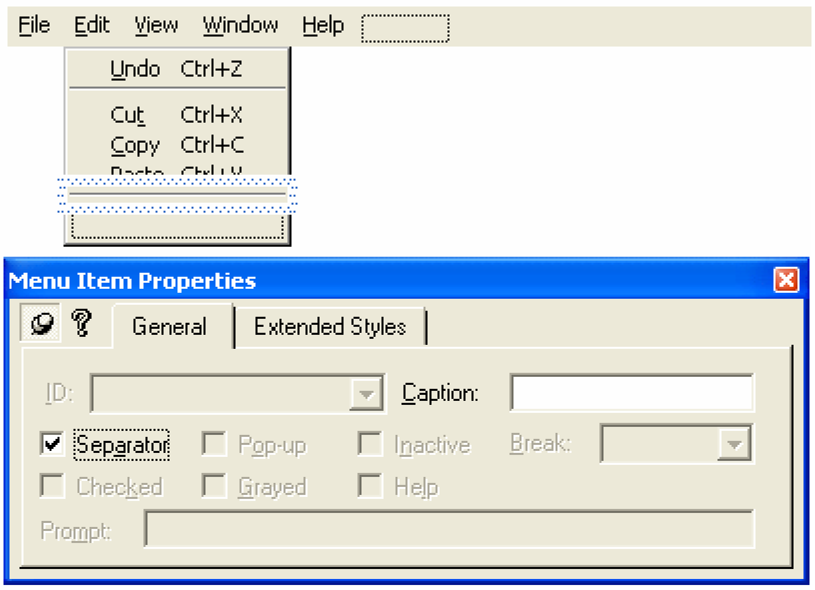
Location: F:\mfcproject\mymfc30a

Figure 13: Adding **cdib.cpp**, a source file to project.

Add menu items under the **Edit** menu of the `IDR_MYMFC3TYPE`. Use the information in the following Table.

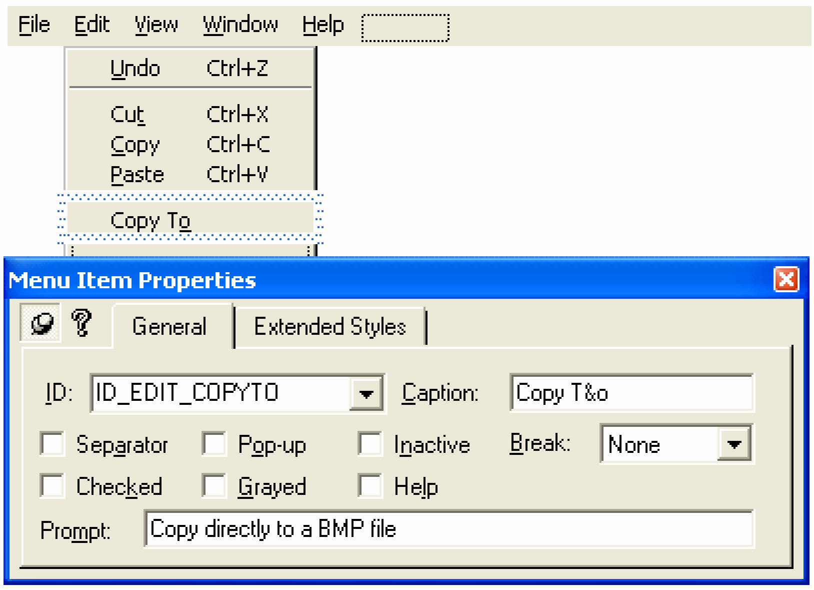
ID	Menu caption	Prompt
Separator	-	-
`ID_EDIT_COPYTO`	Copy T&o	Copy directly to a BMP file
`ID_EDIT_PASTEFROM`	P&aste From	Load a dib from a BMP file
`ID_EDIT_CLEAR_ALL`	Clear A&ll	-

Table 4.



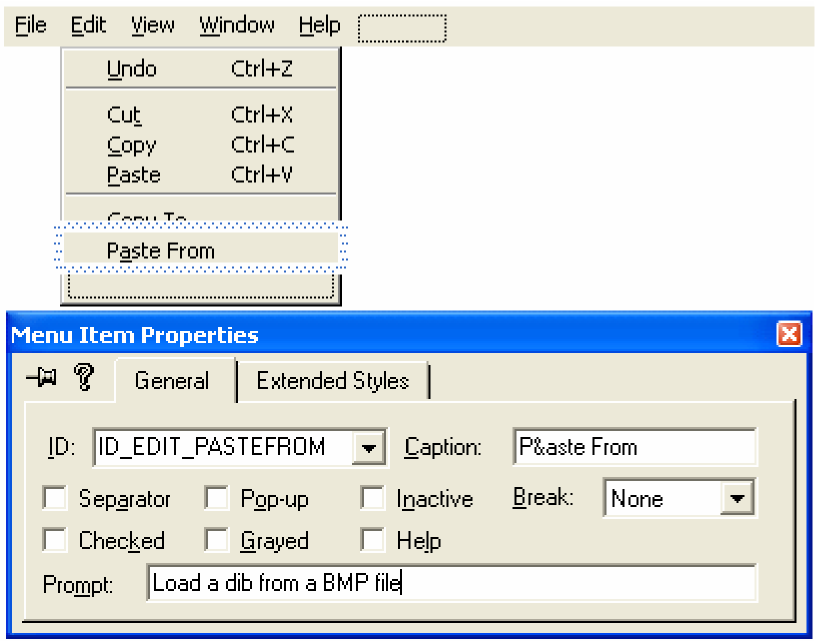
☑ Separator

Figure 14: Adding separator.



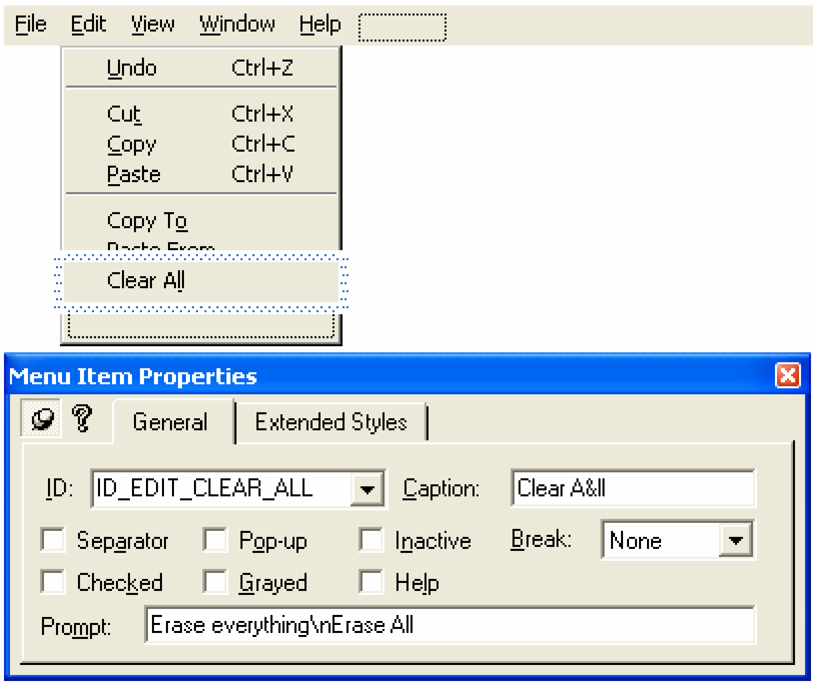
ID: ID_EDIT_COPYTO
Caption: Copy T&o
Separator [ ] Pop-up [ ] Inactive [ ] Break: None
Checked [ ] Grayed [ ] Help [ ]
Prompt: Copy directly to a BMP file

Figure 15: Adding **Copy To** menu item.



ID: ID_EDIT_PASTEFROM
Caption: P&aste From
Separator [ ] Pop-up [ ] Inactive [ ] Break: None
Checked [ ] Grayed [ ] Help [ ]
Prompt: Load a dib from a BMP file

Figure 16: Adding **Paste From** menu item.



ID: ID_EDIT_CLEAR_ALL
Caption: Clear A&ll
Separator [ ] Pop-up [ ] Inactive [ ] Break: None
Checked [ ] Grayed [ ] Help [ ]
Prompt: Erase everything\nErase All

Figure 17: Adding **Clear All** menu item.

Our completed menu items look something like the following.

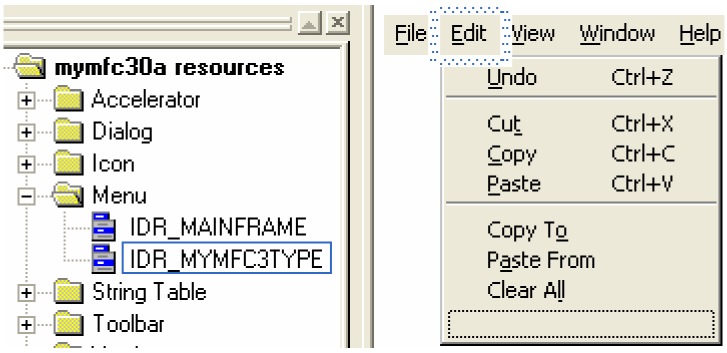
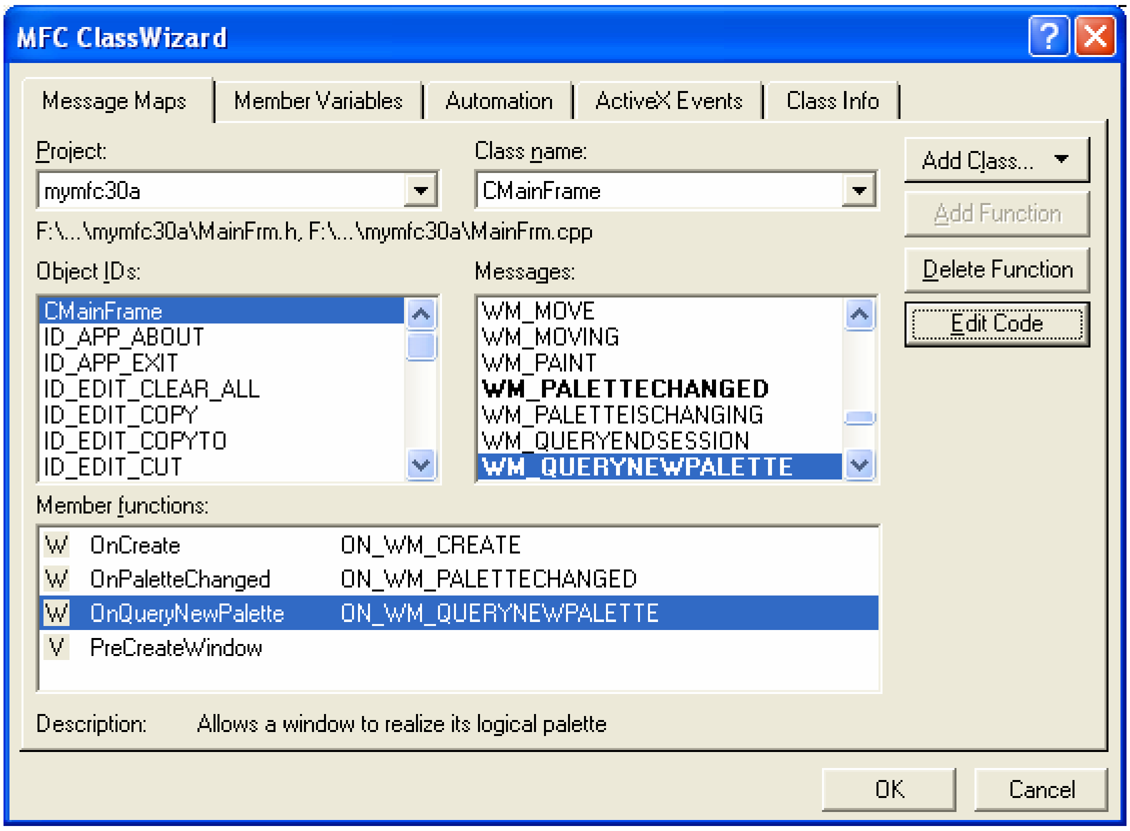


Figure 18: A completed menu items.

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



Project: mymfc30a

Class name: CMainFrame

F:\...\mymfc30a\MainFrm.h, F:\...\mymfc30a\MainFrm.cpp

Object IDs:

```
CMainFrame
ID_APP_ABOUT
ID_APP_EXIT
ID_EDIT_CLEAR_ALL
ID_EDIT_COPY
ID_EDIT_COPYTO
ID_EDIT_CUT
```

Messages:

```
WM_MOVE
WM_MOVING
WM_PAINT
WM_PALETTECHANGED
WM_PALETTEISCHANGING
WM_QUERYENDSESSION
WM_QUERYNEWPALETTE
```

Member functions:

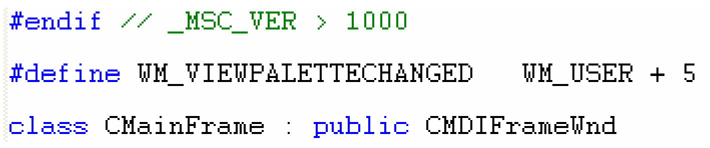
```
W  OnCreate             ON_WM_CREATE
W  OnPaletteChanged     ON_WM_PALETTECHANGED
W  OnQueryNewPalette    ON_WM_QUERYNEWPALETTE
V  PreCreateWindow
```

Description: Allows a window to realize its logical palette

Figure 19: Adding Windows message handlers to `CMainFrame` class.

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

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

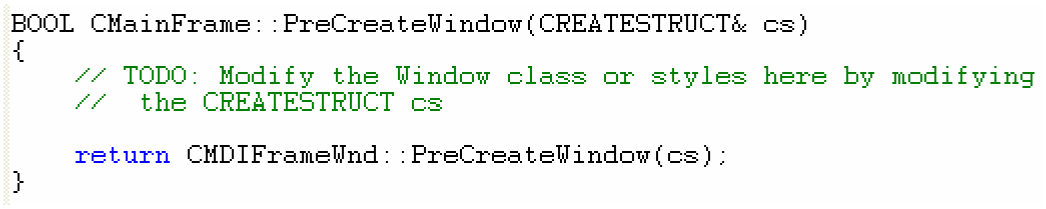


```
#endif // _MSC_VER > 1000

#define WM_VIEWPALETTECHANGED   WM_USER + 5

class CMainFrame : public CMDIFrameWnd
```

Listing 1.

In **MainFrm.cpp**, modify and/or add the following codes.

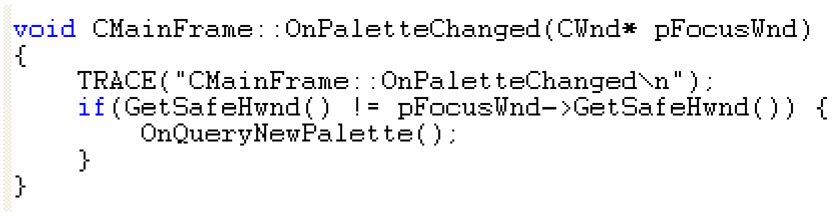
```
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs

        return CMDIFrameWnd::PreCreateWindow(cs);
}
```



```
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CMDIFrameWnd::PreCreateWindow(cs);
}
```

Listing 2.

```
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();
	}
}
```

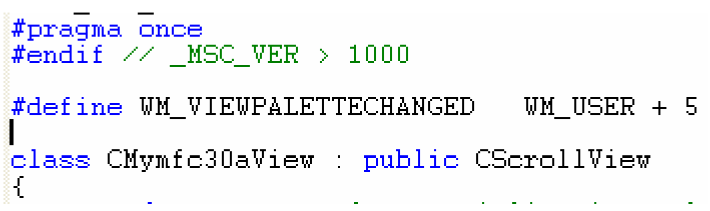


```
void CMainFrame::OnPaletteChanged(CWnd* pFocusWnd)
{
    TRACE("CMainFrame::OnPaletteChanged\n");
    if(GetSafeHwnd() != pFocusWnd->GetSafeHwnd()) {
        OnQueryNewPalette();
    }
}
```

Listing 3.

Add in the following `#define` directive in **mymfc30aView.h** file.

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

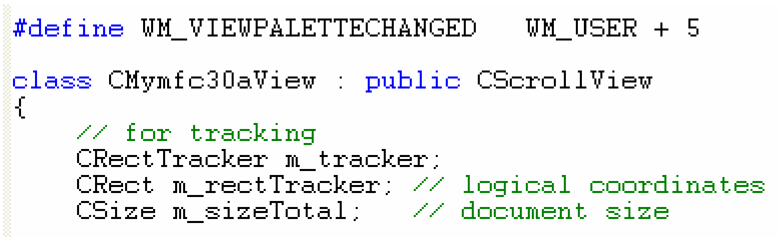


```
#pragma once
#endif // _MSC_VER > 1000

#define WM_VIEWPALETTECHANGED    WM_USER + 5

class CMymfc30aView : public CScrollView
{
```

Listing 4.

Then, manually or using ClassView, add member variables as shown below. These variables are `private` by default.

```
    // for tracking
    CRectTracker m_tracker;
    CRect m_rectTracker; // logical coordinates
    CSize m_sizeTotal;   // document size
```

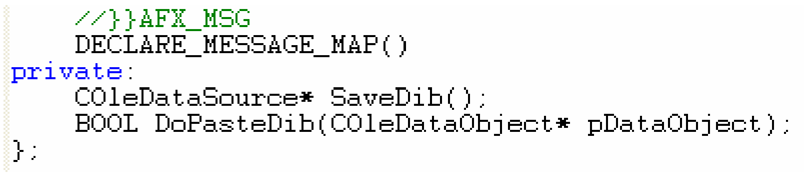


```
#define WM_VIEWPALETTECHANGED    WM_USER + 5

class CMymfc30aView : public CScrollView
{
    // for tracking
    CRectTracker m_tracker;
    CRect m_rectTracker; // logical coordinates
    CSize m_sizeTotal;   // document size
```

Listing 5.

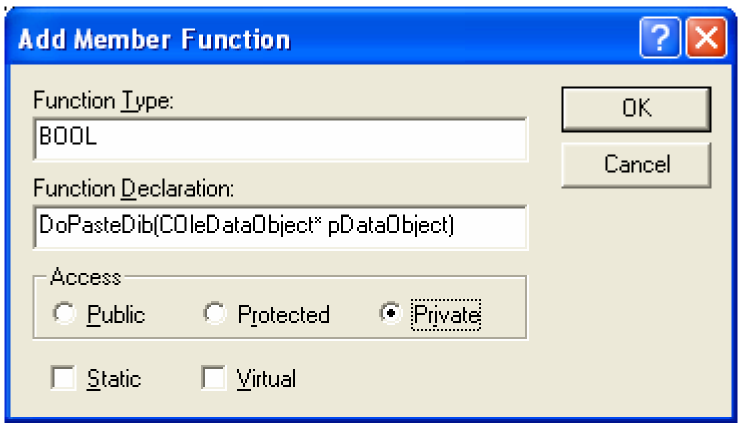
Use ClassView to add private member functions as shown here to `CMymfc30aView` class.

```
private:
    BOOL DoPasteDib(COleDataObject* pDataObject);
    COleDataSource* SaveDib();
```



```
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    COleDataSource* SaveDib();
    BOOL DoPasteDib(COleDataObject* pDataObject);
};
```

Listing 6.



Add Member Function

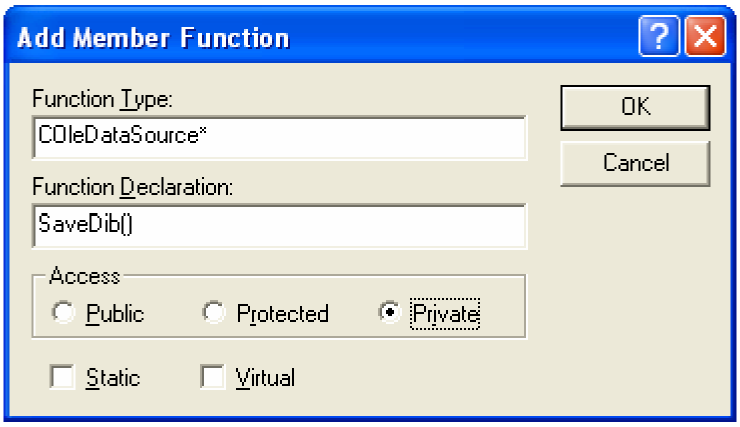
Function Type:
BOOL

Function Declaration:
DoPasteDib(COleDataObject* pDataObject)

Access
Public Protected [✓ Private]

☐ Static ☐ Virtual

Figure 1: Adding member function, `DoPasteDib()` to `CMymfc30aView` class.

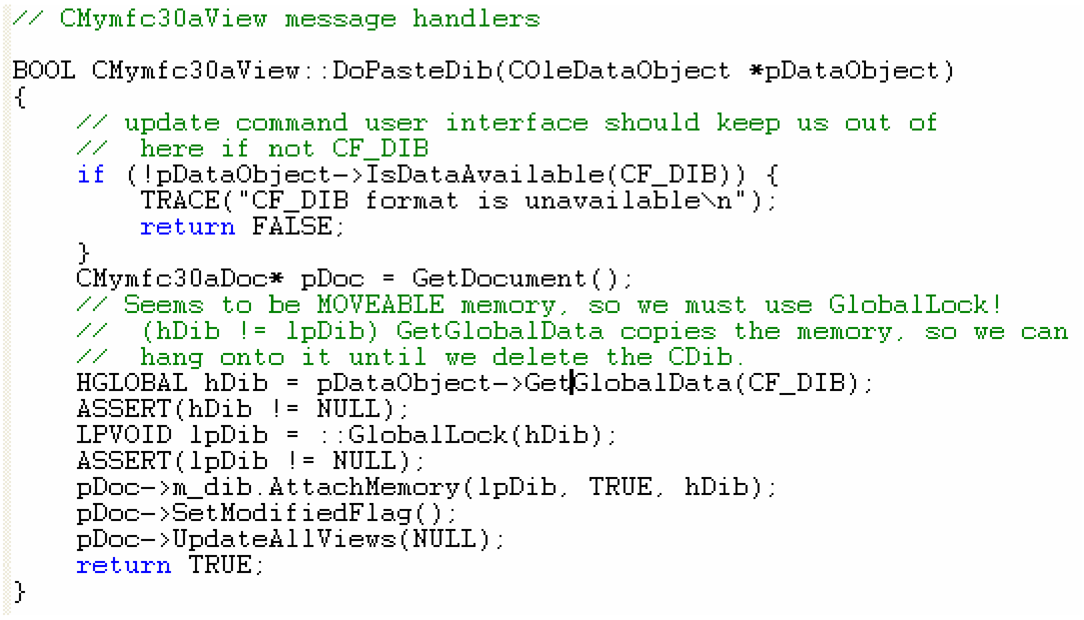


Function Type: COleDataSource*
Function Declaration: SaveDib()
Access: Public / Protected / [✓ Private]
Static / Virtual

Figure 20: Adding member function, `SaveDib()` to `CMymfc30aView` class.

Then, add the following codes (**mymfc30aView.cpp**).

```
//////////////////////////////////////////////////////////////
// helper functions used for clipboard and drag-drop

BOOL CMymfc30aView::DoPasteDib(COleDataObject* pDataObject)
{
    // update command user interface should keep us out of
    //  here if not CF_DIB
    if (!pDataObject->IsDataAvailable(CF_DIB)) {
        TRACE("CF_DIB format is unavailable\n");
        return FALSE;
    }
    CMymfc30aDoc* 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);
    pDoc->SetModifiedFlag();
    pDoc->UpdateAllViews(NULL);
    return TRUE;
}
```



```
// CMymfc30aView message handlers

BOOL CMymfc30aView::DoPasteDib(COleDataObject *pDataObject)
{
    // update command user interface should keep us out of
    //  here if not CF_DIB
    if (!pDataObject->IsDataAvailable(CF_DIB)) {
        TRACE("CF_DIB format is unavailable\n");
        return FALSE;
    }
    CMymfc30aDoc* 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);
    pDoc->SetModifiedFlag();
    pDoc->UpdateAllViews(NULL);
    return TRUE;
}
```

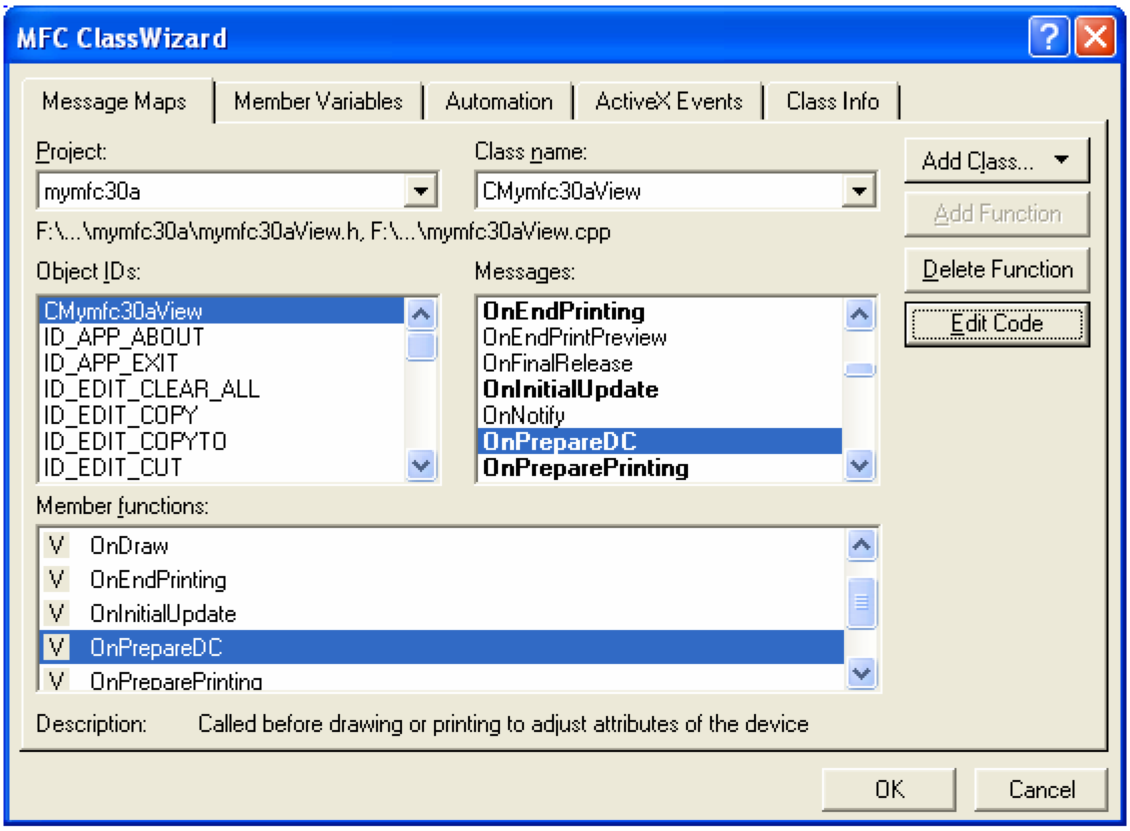
Listing 7.

```
COleDataSource* CMymfc30aView::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* CMymfc30aView::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 8.

Using ClassWizard, add `OnPrepareDC()`, a virtual function for `CMymfc30aView` class.



Project: mymfc30a

Class name: CMymfc30aView

F:\...\mymfc30a\mymfc30aView.h, F:\...\mymfc30aView.cpp

Object IDs: CMymfc30aView, ID_APP_ABOUT, ID_APP_EXIT, ID_EDIT_CLEAR_ALL, ID_EDIT_COPY, ID_EDIT_COPYTO, ID_EDIT_CUT

Messages: OnEndPrinting, OnEndPrintPreview, OnFinalRelease, OnInitialUpdate, OnNotify, OnPrepareDC, OnPreparePrinting

Member functions:

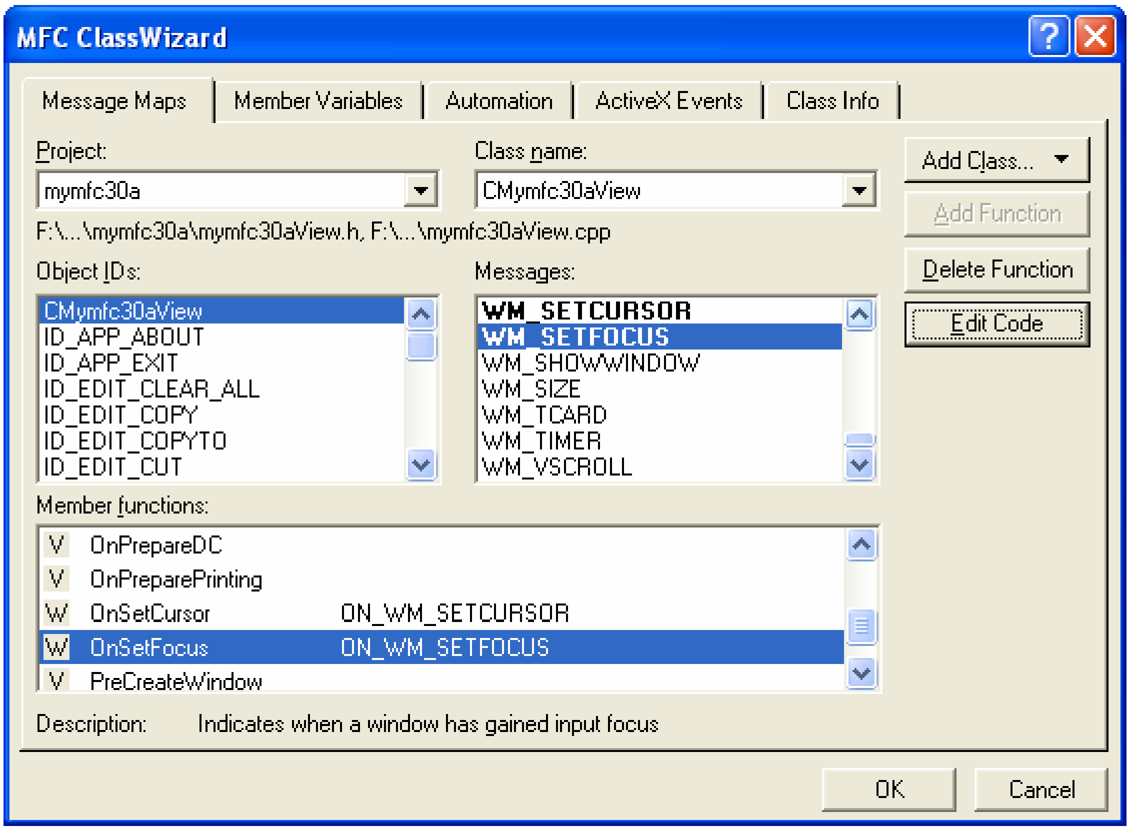
V OnDraw
V OnEndPrinting
V OnInitialUpdate
V OnPrepareDC
V OnPreparePrinting

Description: Called before drawing or printing to adjust attributes of the device

Figure 21: Adding/overriding `OnPrepareDC()` virtual function.

Add window messages to the `CMymfc30aView` class as shown below.

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



Project: mymfc30a

Class name: CMymfc30aView

F:\...\mymfc30a\mymfc30aView.h, F:\...\mymfc30aView.cpp

Object IDs: CMymfc30aView, ID_APP_ABOUT, ID_APP_EXIT, ID_EDIT_CLEAR_ALL, ID_EDIT_COPY, ID_EDIT_COPYTO, ID_EDIT_CUT

Messages: WM_SETCURSOR, WM_SETFOCUS, WM_SHOWWINDOW, WM_SIZE, WM_TCARD, WM_TIMER, WM_VSCROLL

Member functions:

```
V  OnPrepareDC
V  OnPreparePrinting
W  OnSetCursor      ON_WM_SETCURSOR
W  OnSetFocus       ON_WM_SETFOCUS
V  PreCreateWindow
```

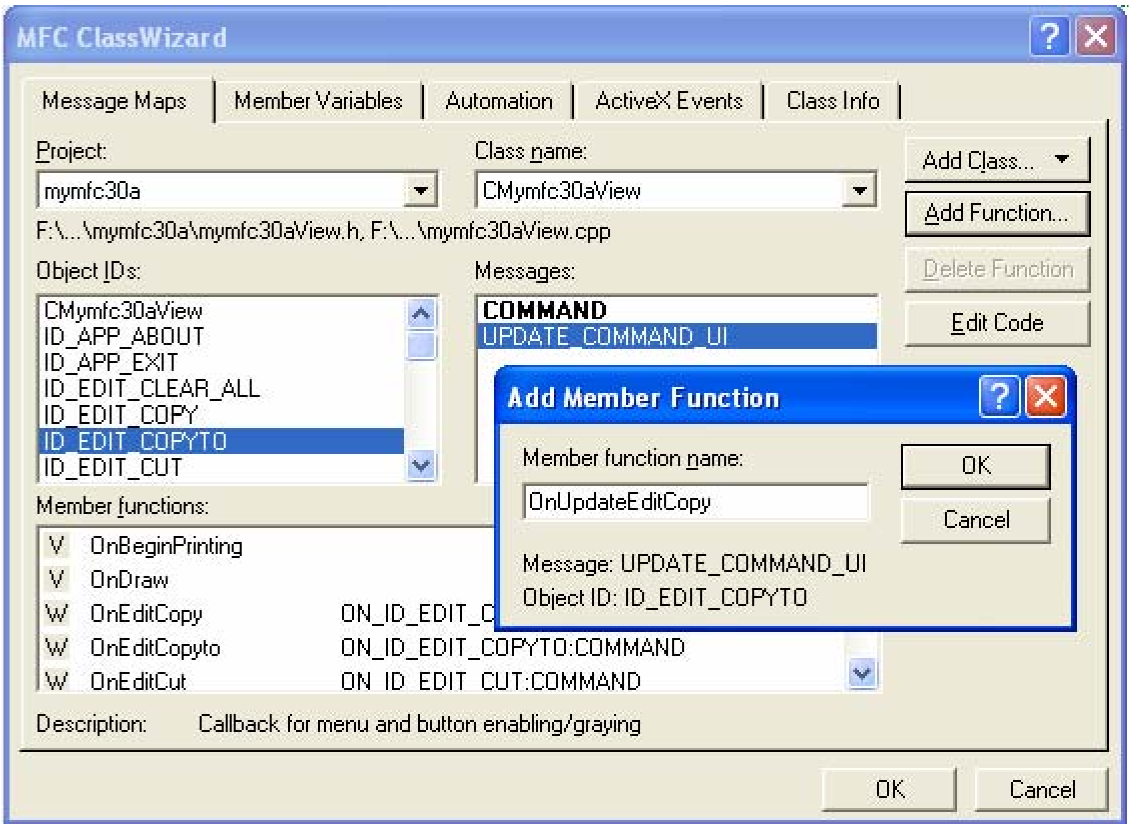
Description: Indicates when a window has gained input focus

Figure 22: Adding Window messages to `CMymfc30aView` 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 5.


MFC ClassWizard

Message Maps | Member Variables | Automation | ActiveX Events | Class Info

Project: mymfc30a

Class name: CMymfc30aView

F:\...\mymfc30a\mymfc30aView.h, F:\...\mymfc30aView.cpp

Object IDs: CMymfc30aView, ID_APP_ABOUT, ID_APP_EXIT, ID_EDIT_CLEAR_ALL, ID_EDIT_COPY, ID_EDIT_COPYTO, ID_EDIT_CUT

Messages: COMMAND, UPDATE_COMMAND_UI

Member functions:

```
V   OnBeginPrinting
V   OnDraw
W   OnEditCopy         ON_ID_EDIT_C
W   OnEditCopyto       ON_ID_EDIT_COPYTO:COMMAND
W   OnEditCut          ON_ID_EDIT_CUT:COMMAND
```

Description: Callback for menu and button enabling/graying

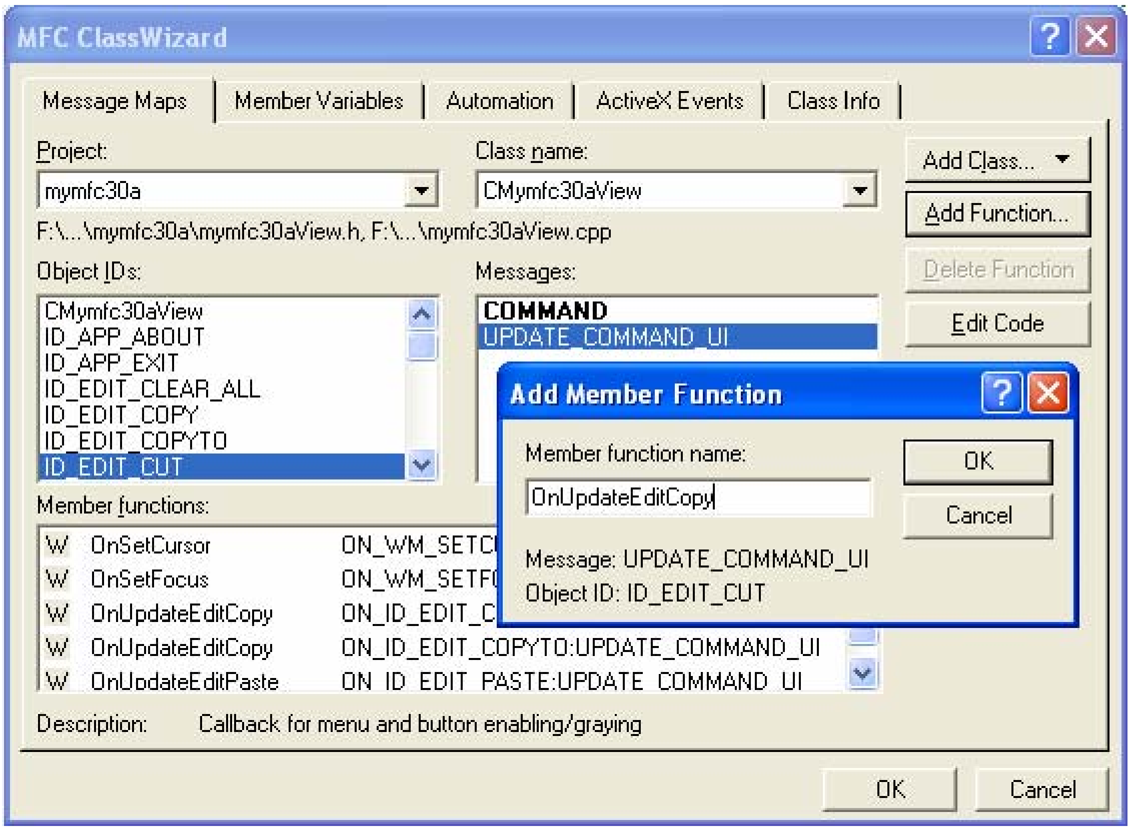
Add Member Function

Member function name: OnUpdateEditCopy

Message: UPDATE_COMMAND_UI

Object ID: ID_EDIT_COPYTO

Figure 23: Adding Command and Update Commands to `CMymfc30aView` class.


MFC ClassWizard

Message Maps | Member Variables | Automation | ActiveX Events | Class Info

Project: mymfc30a

Class name: CMymfc30aView

F:\...\mymfc30a\mymfc30aView.h, F:\...\mymfc30aView.cpp

Object IDs: CMymfc30aView, ID_APP_ABOUT, ID_APP_EXIT, ID_EDIT_CLEAR_ALL, ID_EDIT_COPY, ID_EDIT_COPYTO, ID_EDIT_CUT

Messages: COMMAND, UPDATE_COMMAND_UI

Member functions:

```
W   OnSetCursor        ON_WM_SETC
W   OnSetFocus         ON_WM_SETF
W   OnUpdateEditCopy   ON_ID_EDIT_C
W   OnUpdateEditCopy   ON_ID_EDIT_COPYTO:UPDATE_COMMAND_UI
W   OnUpdateEditPaste  ON_ID_EDIT_PASTE:UPDATE_COMMAND_UI
```

Description: Callback for menu and button enabling/graying

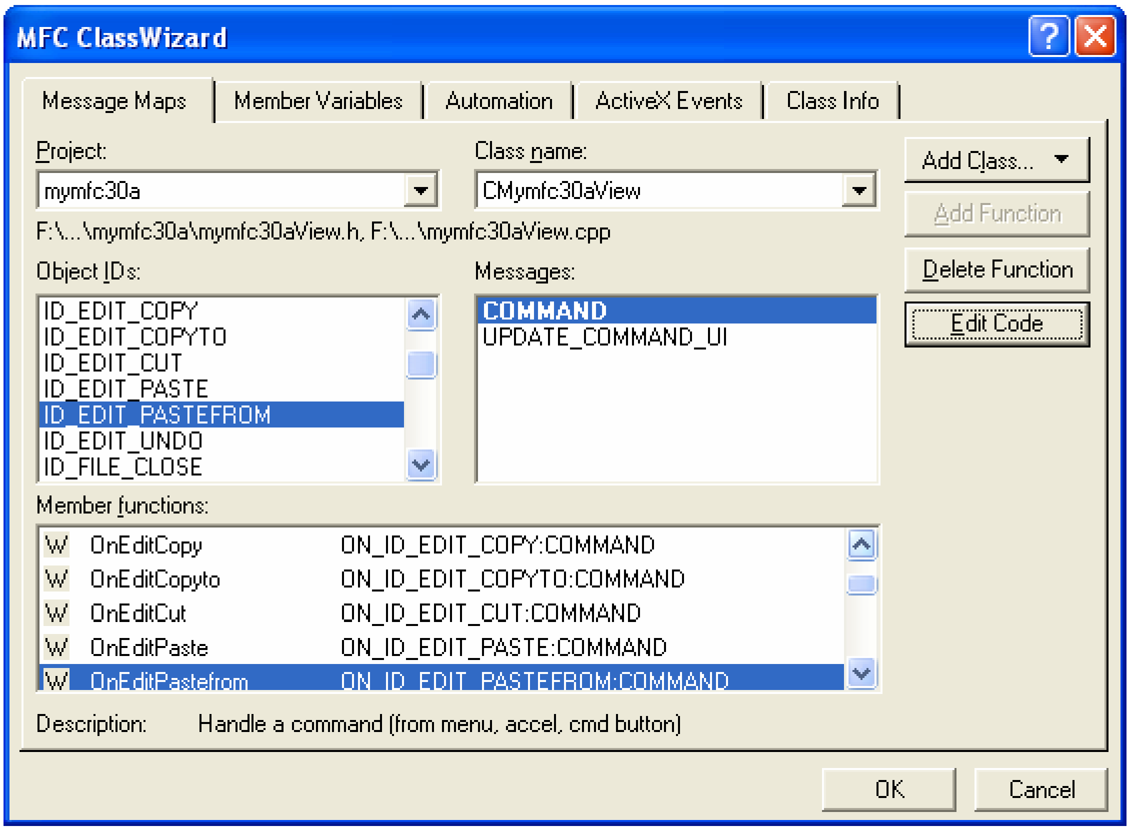
Add Member Function

Member function name: OnUpdateEditCopy

Message: UPDATE_COMMAND_UI

Object ID: ID_EDIT_CUT

Figure 24: Adding Command and Update Commands to `CMymfc30aView` class.



Project: mymfc30a

Class name: CMymfc30aView

F:\...\mymfc30a\mymfc30aView.h, F:\...\mymfc30aView.cpp

Object IDs:

```
ID_EDIT_COPY
ID_EDIT_COPYTO
ID_EDIT_CUT
ID_EDIT_PASTE
ID_EDIT_PASTEFROM
ID_EDIT_UNDO
ID_FILE_CLOSE
```

Messages:

```
COMMAND
UPDATE_COMMAND_UI
```

Member functions:

```
W   OnEditCopy        ON_ID_EDIT_COPY:COMMAND
W   OnEditCopyto      ON_ID_EDIT_COPYTO:COMMAND
W   OnEditCut         ON_ID_EDIT_CUT:COMMAND
W   OnEditPaste       ON_ID_EDIT_PASTE:COMMAND
W   OnEditPastefrom   ON_ID_EDIT_PASTEFROM:COMMAND
```

Description: Handle a command (from menu, accel, cmd button)

Figure 25: A completed commands and command updates addition.

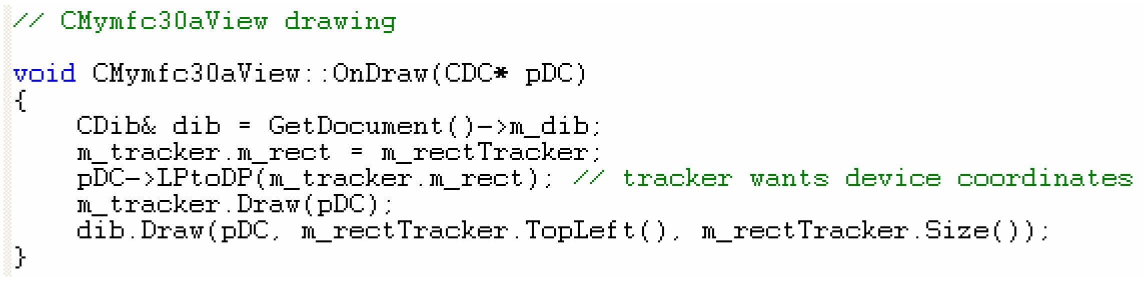
Add codes to **mymfc30aView.cpp**.

```
CMymfc30aView::CMymfc30aView(): m_sizeTotal(800, 1050), // 8 by 10.5 inches
                                                    //  when printed
    m_rectTracker(50, 50, 250, 250)
{
}
```

```
// CMymfc30aView construction/destruction

CMymfc30aView::CMymfc30aView() : m_sizeTotal(800, 1050), // 8 by 10.5 inches
                                                    //  when printed
    m_rectTracker(50, 50, 250, 250)

{
    // TODO: add construction code here

}
```

Listing 9.

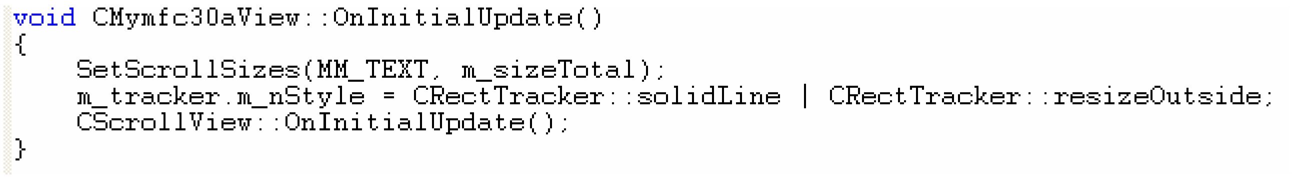
```
void CMymfc30aView::OnDraw(CDC* pDC)
{
   CDib& dib = GetDocument()->m_dib;
    m_tracker.m_rect = m_rectTracker;
    pDC->LPtoDP(m_tracker.m_rect); // tracker wants device coordinates
    m_tracker.Draw(pDC);
    dib.Draw(pDC, m_rectTracker.TopLeft(), m_rectTracker.Size());
}
```



```
// CMymfc30aView drawing

void CMymfc30aView::OnDraw(CDC* pDC)
{
    CDib& dib = GetDocument()->m_dib;
    m_tracker.m_rect = m_rectTracker;
    pDC->LPtoDP(m_tracker.m_rect); // tracker wants device coordinates
    m_tracker.Draw(pDC);
    dib.Draw(pDC, m_rectTracker.TopLeft(), m_rectTracker.Size());
}
```

Listing 10.

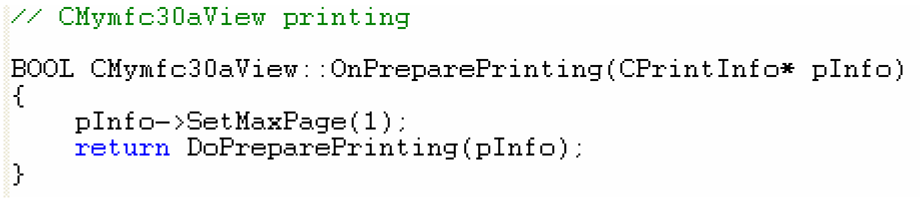
```
void CMymfc30aView::OnInitialUpdate()
{
    SetScrollSizes(MM_TEXT, m_sizeTotal);
    m_tracker.m_nStyle = CRectTracker::solidLine |
CRectTracker::resizeOutside;
    CScrollView::OnInitialUpdate();
}
```



```
void CMymfc30aView::OnInitialUpdate()
{
    SetScrollSizes(MM_TEXT, m_sizeTotal);
    m_tracker.m_nStyle = CRectTracker::solidLine | CRectTracker::resizeOutside;
    CScrollView::OnInitialUpdate();
}
```

Listing 11.

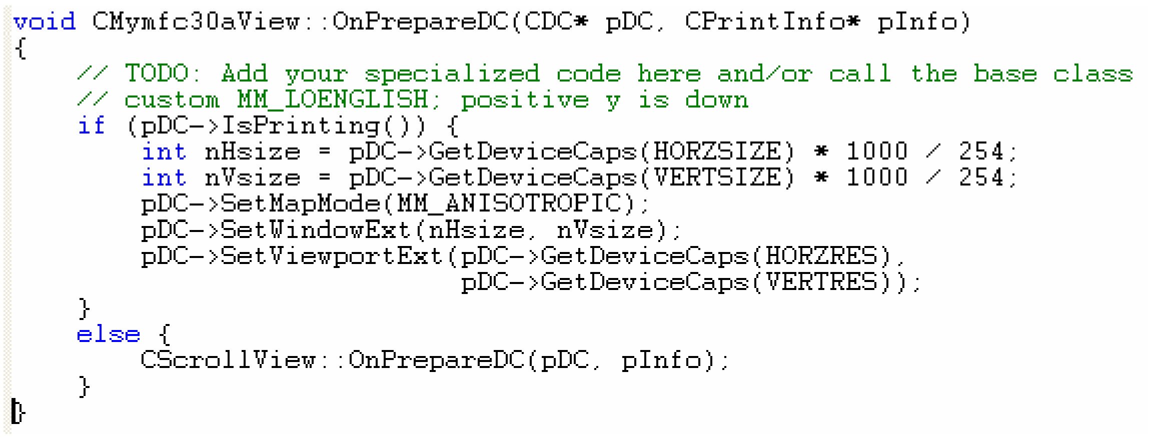
```
BOOL CMymfc30aView::OnPreparePrinting(CPrintInfo* pInfo)
{
    pInfo->SetMaxPage(1);
    return DoPreparePrinting(pInfo);
}
```



```
// CMymfc30aView printing

BOOL CMymfc30aView::OnPreparePrinting(CPrintInfo* pInfo)
{
    pInfo->SetMaxPage(1);
    return DoPreparePrinting(pInfo);
}
```

Listing 12.

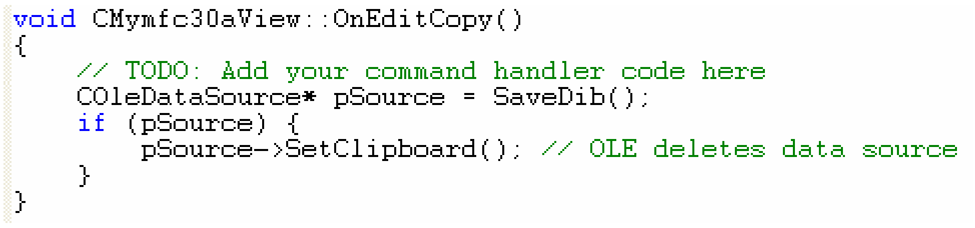
```
void CMymfc30aView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
    // custom MM_LOENGLISH; positive y is down
    if (pDC->IsPrinting())
    {
        int nHsize = pDC->GetDeviceCaps(HORZSIZE) * 1000 / 254;
        int nVsize = pDC->GetDeviceCaps(VERTSIZE) * 1000 / 254;
        pDC->SetMapMode(MM_ANISOTROPIC);
        pDC->SetWindowExt(nHsize, nVsize);
        pDC->SetViewportExt(pDC->GetDeviceCaps(HORZRES), pDC-
>GetDeviceCaps(VERTRES));
    }
    else {
        CScrollView::OnPrepareDC(pDC, pInfo);
    }
}
```



```
void CMymfc30aView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
    // TODO: Add your specialized code here and/or call the base class
    // custom MM_LOENGLISH; positive y is 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 13.

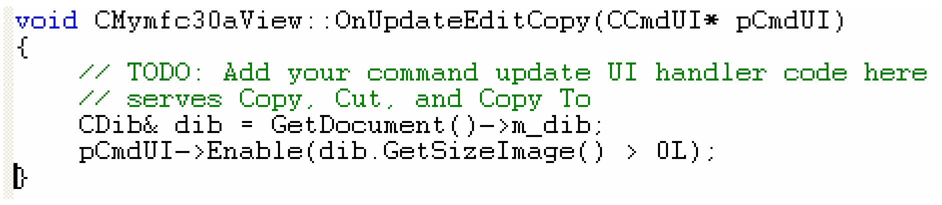
```
void CMymfc30aView::OnEditCopy()
{
    COleDataSource* pSource = SaveDib();
    if (pSource)
    {
        pSource->SetClipboard(); // OLE deletes data source
    }
}
```



```
void CMymfc30aView::OnEditCopy()
{
    // TODO: Add your command handler code here
    COleDataSource* pSource = SaveDib();
    if (pSource) {
        pSource->SetClipboard(); // OLE deletes data source
    }
}
```

Listing 14.

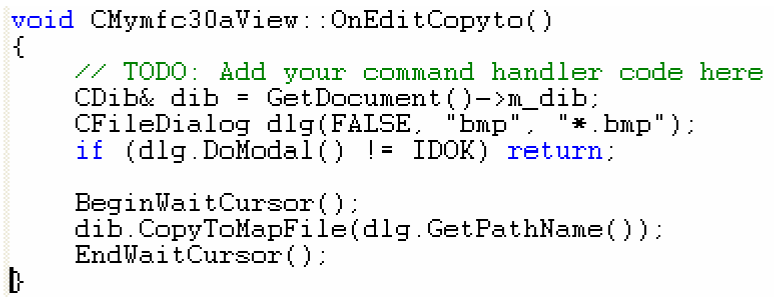
```
void CMymfc30aView::OnUpdateEditCopy(CCmdUI* pCmdUI)
{
    // serves Copy, Cut, and Copy To
    CDib& dib = GetDocument()->m_dib;
    pCmdUI->Enable(dib.GetSizeImage() > 0L);
}
```



```
void CMymfc30aView::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 15.

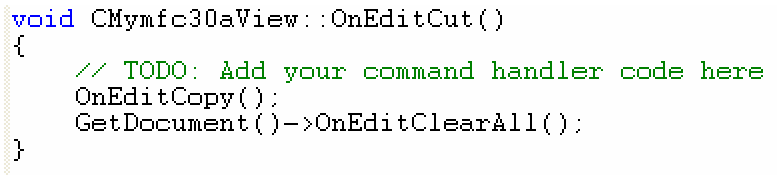
```
void CMymfc30aView::OnEditCopyto()
{
    CDib& dib = GetDocument()->m_dib;
    CFileDialog dlg(FALSE, "bmp", "*.bmp");
    if (dlg.DoModal() != IDOK) return;

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



```
void CMymfc30aView::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 16.

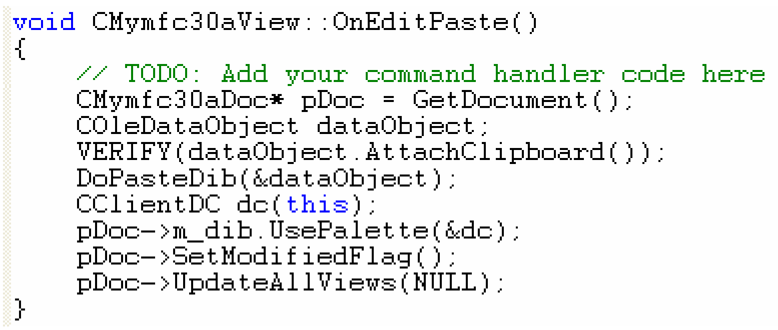
```
void CMymfc30aView::OnEditCut()
{
    OnEditCopy();
    GetDocument()->OnEditClearAll();
}
```



```
void CMymfc30aView::OnEditCut()
{
    // TODO: Add your command handler code here
    OnEditCopy();
    GetDocument()->OnEditClearAll();
}
```

Listing 17.

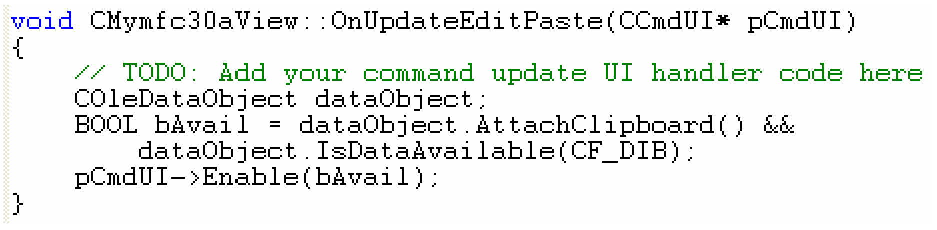
```
void CMymfc30aView::OnEditPaste()
{
    CMymfc30aDoc* pDoc = GetDocument();
    COleDataObject dataObject;
    VERIFY(dataObject.AttachClipboard());
    DoPasteDib(&dataObject);
    CClientDC dc(this);
    pDoc->m_dib.UsePalette(&dc);
    pDoc->SetModifiedFlag();
    pDoc->UpdateAllViews(NULL);
}
```



```
void CMymfc30aView::OnEditPaste()
{
    // TODO: Add your command handler code here
    CMymfc30aDoc* pDoc = GetDocument();
    COleDataObject dataObject;
    VERIFY(dataObject.AttachClipboard());
    DoPasteDib(&dataObject);
    CClientDC dc(this);
    pDoc->m_dib.UsePalette(&dc);
    pDoc->SetModifiedFlag();
    pDoc->UpdateAllViews(NULL);
}
```

Listing 18.

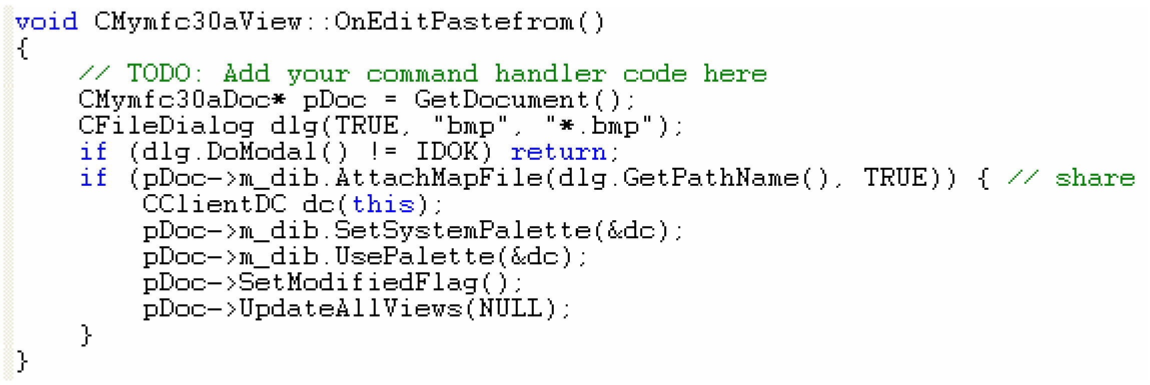
```
void CMymfc30aView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
    COleDataObject dataObject;
    BOOL bAvail = dataObject.AttachClipboard() &&
        dataObject.IsDataAvailable(CF_DIB);
    pCmdUI->Enable(bAvail);
}
```



```
void CMymfc30aView::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 19.

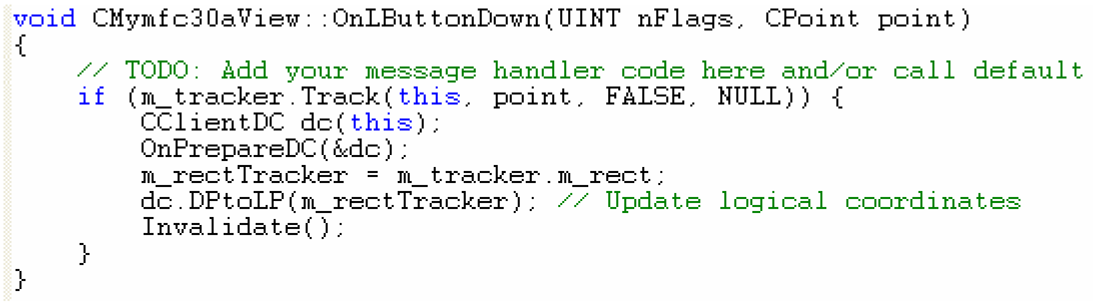
```
void CMymfc30aView::OnEditPastefrom()
{
    CMymfc30aDoc* pDoc = GetDocument();
    CFileDialog dlg(TRUE, "bmp", "*.bmp");
    if (dlg.DoModal() != IDOK) return;
    if (pDoc->m_dib.AttachMapFile(dlg.GetPathName(), TRUE))
    { // share
          CClientDC dc(this);
          pDoc->m_dib.SetSystemPalette(&dc);
          pDoc->m_dib.UsePalette(&dc);
        pDoc->SetModifiedFlag();
        pDoc->UpdateAllViews(NULL);
    }
}
```



```
void CMymfc30aView::OnEditPastefrom()
{
    // TODO: Add your command handler code here
    CMymfc30aDoc* 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->m_dib.UsePalette(&dc);
        pDoc->SetModifiedFlag();
        pDoc->UpdateAllViews(NULL);
    }
}
```

Listing 20.

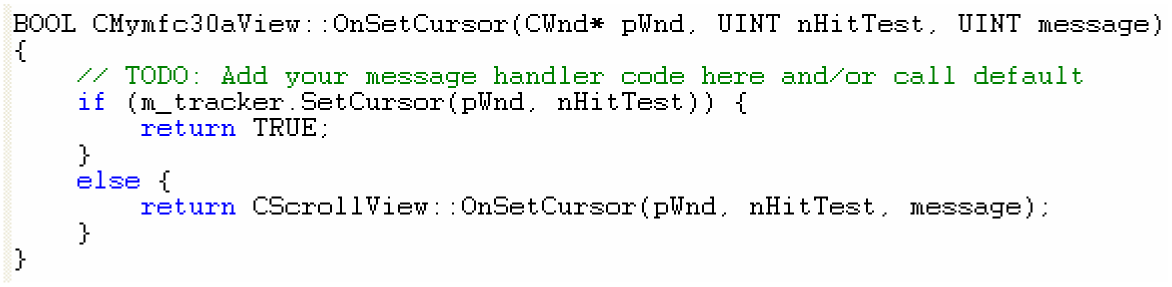
```
void CMymfc30aView::OnLButtonDown(UINT nFlags, CPoint point)
{
    if (m_tracker.Track(this, point, FALSE, NULL)) {
        CClientDC dc(this);
        OnPrepareDC(&dc);
        m_rectTracker = m_tracker.m_rect;
        dc.DPtoLP(m_rectTracker); // Update logical coordinates
        Invalidate();
    }
}
```



```
void CMymfc30aView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    if (m_tracker.Track(this, point, FALSE, NULL)) {
        CClientDC dc(this);
        OnPrepareDC(&dc);
        m_rectTracker = m_tracker.m_rect;
        dc.DPtoLP(m_rectTracker); // Update logical coordinates
        Invalidate();
    }
}
```

Listing 21.

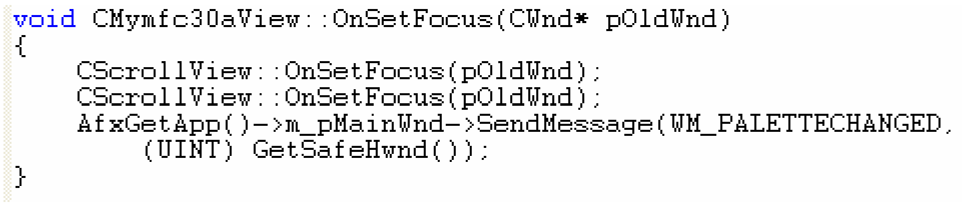
```
BOOL CMymfc30aView::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
{
    if (m_tracker.SetCursor(pWnd, nHitTest))
    { return TRUE; }
    else
    { return CScrollView::OnSetCursor(pWnd, nHitTest, message); }
}
```



```
BOOL CMymfc30aView::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 22.

```
void CMymfc30aView::OnSetFocus(CWnd* pOldWnd)
{
      CScrollView::OnSetFocus(pOldWnd);
      AfxGetApp()->m_pMainWnd->SendMessage(WM_PALETTECHANGED, (UINT)
GetSafeHwnd());
}
```

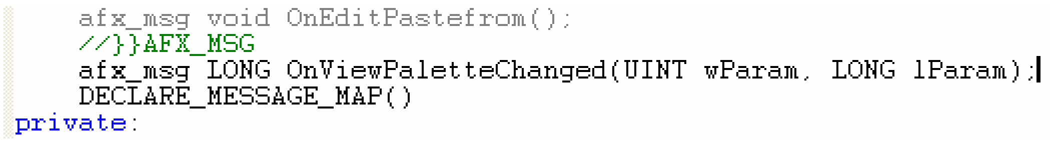


```
void CMymfc30aView::OnSetFocus(CWnd* pOldWnd)
{
    CScrollView::OnSetFocus(pOldWnd);
    CScrollView::OnSetFocus(pOldWnd);
    AfxGetApp()->m_pMainWnd->SendMessage(WM_PALETTECHANGED,
        (UINT) GetSafeHwnd());
}
```

Listing 23.

Manually add the following message map function in **mymfc30aView.h**

```
afx_msg LONG OnViewPaletteChanged(UINT wParam, LONG lParam);
```

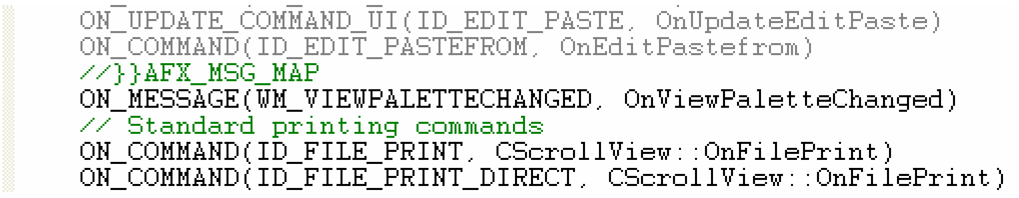


```
    afx_msg void OnEditPastefrom();
    //}}AFX_MSG
    afx_msg LONG OnViewPaletteChanged(UINT wParam, LONG lParam);
    DECLARE_MESSAGE_MAP()
private:
```

Listing 24.

And in **mymfc30aView.cpp**, add the following code.

```
ON_MESSAGE(WM_VIEWPALETTECHANGED, OnViewPaletteChanged)
```

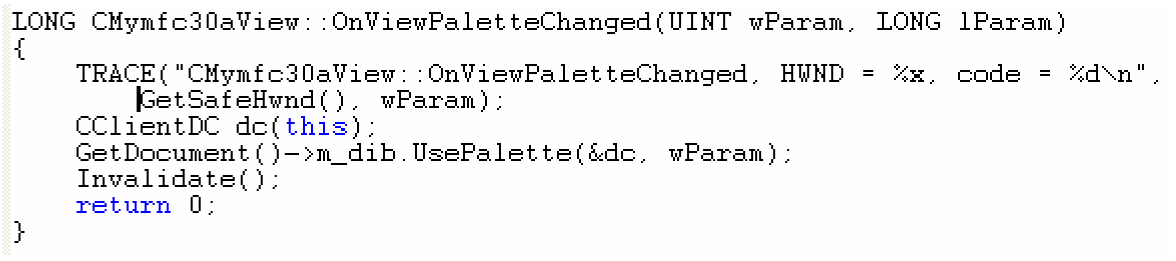


```
    ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, OnUpdateEditPaste)
    ON_COMMAND(ID_EDIT_PASTEFROM, OnEditPastefrom)
    //}}AFX_MSG_MAP
    ON_MESSAGE(WM_VIEWPALETTECHANGED, OnViewPaletteChanged)
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CScrollView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CScrollView::OnFilePrint)
```

Listing 25.

Finally add the code in **mymfc30aView.cpp**

```
LONG CMymfc30aView::OnViewPaletteChanged(UINT wParam, LONG lParam)
{
      TRACE("CMymfc30aView::OnViewPaletteChanged, HWND = %x, code = %d\n",
GetSafeHwnd(), wParam);
      CClientDC dc(this);
      GetDocument()->m_dib.UsePalette(&dc, wParam);
      Invalidate();
      return 0;
}
```

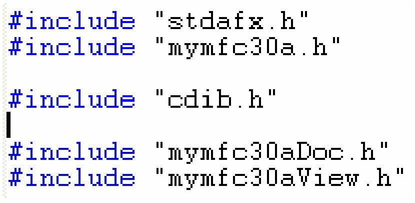


```
LONG CMymfc30aView::OnViewPaletteChanged(UINT wParam, LONG lParam)
{
    TRACE("CMymfc30aView::OnViewPaletteChanged, HWND = %x, code = %d\n",
        GetSafeHwnd(), wParam);
    CClientDC dc(this);
    GetDocument()->m_dib.UsePalette(&dc, wParam);
    Invalidate();
    return 0;
}
```

Listing 26.

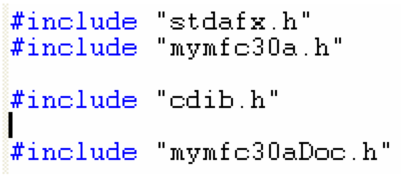
Add the `#include` statement in **mymfc30aView.cpp** and **mymfc30aDoc.cpp**.

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



```
#include "stdafx.h"
#include "mymfc30a.h"

#include "cdib.h"

#include "mymfc30aDoc.h"
#include "mymfc30aView.h"
```

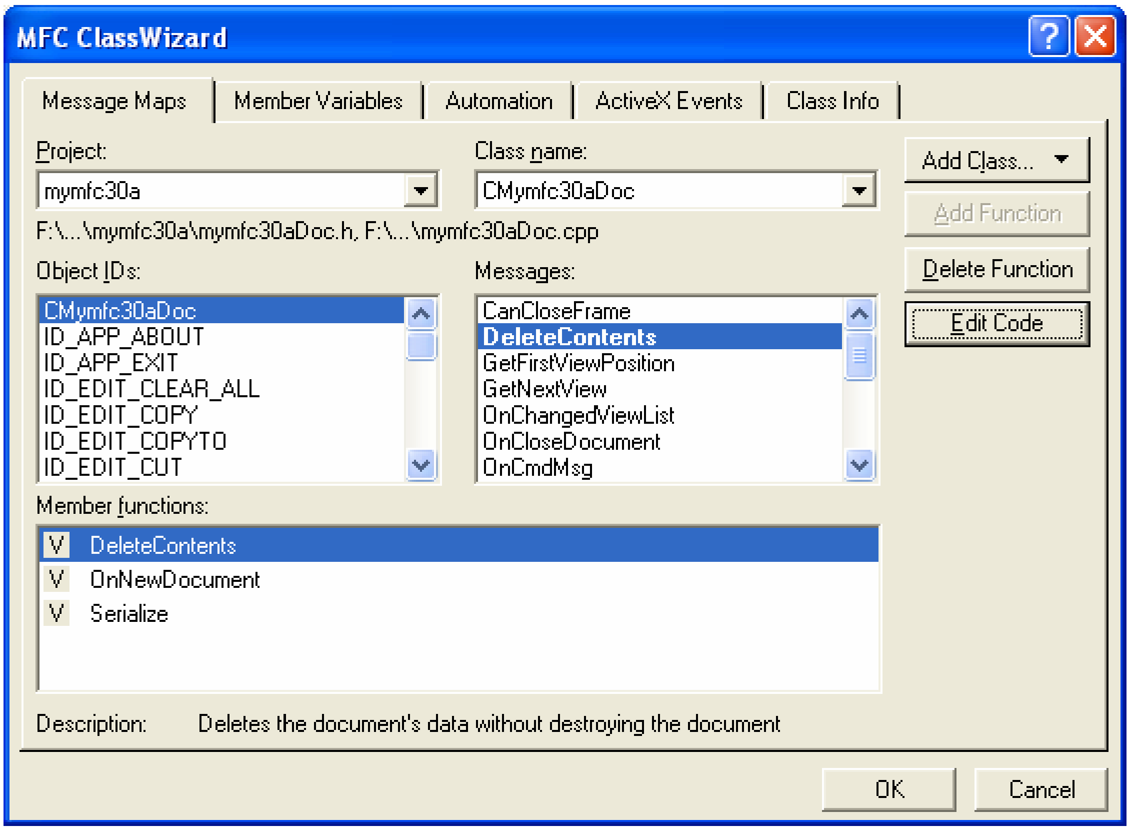
Listing 27.



```
#include "stdafx.h"
#include "mymfc30a.h"

#include "cdib.h"

#include "mymfc30aDoc.h"
```

Listing 28.

Using ClassWizard add the `DeleteContents()` virtual function to `CMymfc30aDoc` class.



Project: mymfc30a

Class name: CMymfc30aDoc

F:\...\mymfc30a\mymfc30aDoc.h, F:\...\mymfc30aDoc.cpp

Object IDs:
CMymfc30aDoc
ID_APP_ABOUT
ID_APP_EXIT
ID_EDIT_CLEAR_ALL
ID_EDIT_COPY
ID_EDIT_COPYTO
ID_EDIT_CUT

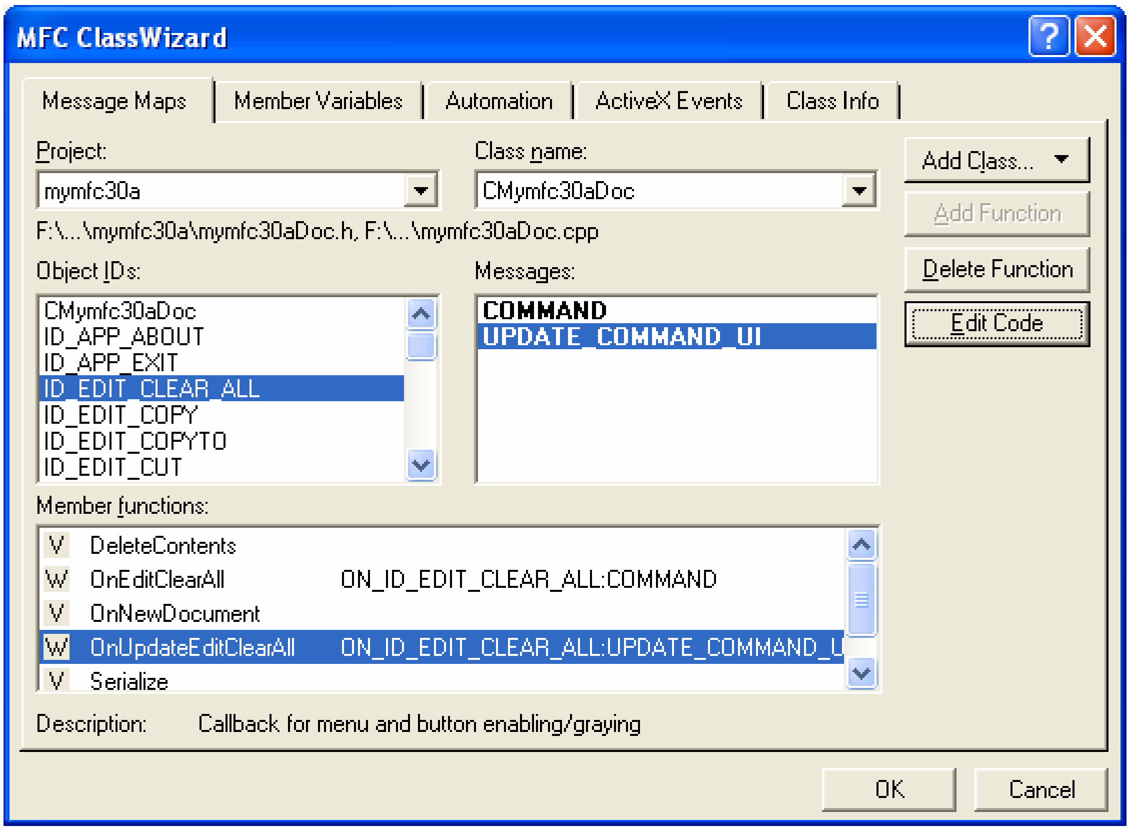
Messages:
CanCloseFrame
DeleteContents
GetFirstViewPosition
GetNextView
OnChangedViewList
OnCloseDocument
OnCmdMsg

Member functions:
V DeleteContents
V OnNewDocument
V Serialize

Description: Deletes the document's data without destroying the document

Figure 26: Adding `DeleteContents()` to `CMymfc30aDoc` class.

Then add command and update command for `ID_EDIT_CLEAR_ALL` f the `CMymfc30aDoc` class.



Project: mymfc30a

Class name: CMymfc30aDoc

F:\...\mymfc30a\mymfc30aDoc.h, F:\...\mymfc30aDoc.cpp

Object IDs:
CMymfc30aDoc
ID_APP_ABOUT
ID_APP_EXIT
ID_EDIT_CLEAR_ALL
ID_EDIT_COPY
ID_EDIT_COPYTO
ID_EDIT_CUT

Messages:
COMMAND
UPDATE_COMMAND_UI

Member functions:

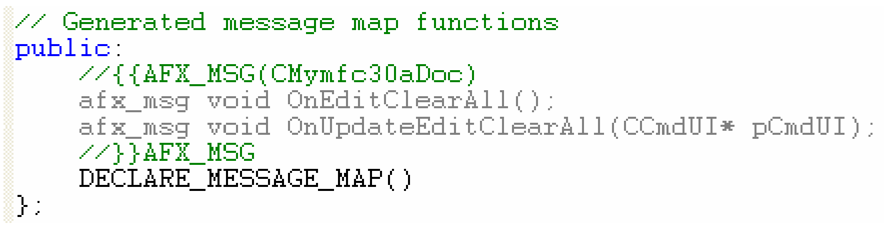
V	DeleteContents	
W	OnEditClearAll	ON_ID_EDIT_CLEAR_ALL:COMMAND
V	OnNewDocument	
W	OnUpdateEditClearAll	ON_ID_EDIT_CLEAR_ALL:UPDATE_COMMAND_U
V	Serialize	

Description: Callback for menu and button enabling/graying

Figure 27: Adding command and update command for `ID_EDIT_CLEAR_ALL`.

Change the `protected` to `public` for the generated message map functions in **mymfc30aDoc.h** (or you can use `friend` keyword).

```
// Generated message map functions
public:
      //{{AFX_MSG(CMymfc30aDoc)
      afx_msg void OnEditClearAll();
      afx_msg void OnUpdateEditClearAll(CCmdUI* pCmdUI);
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
```

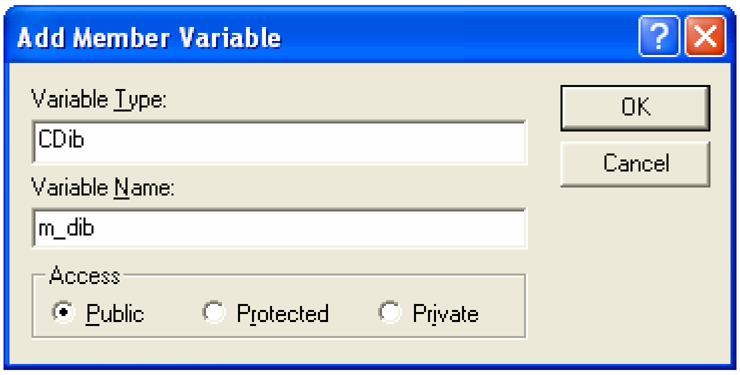


```
// Generated message map functions
public:
    //{{AFX_MSG(CMymfc30aDoc)
    afx_msg void OnEditClearAll();
    afx_msg void OnUpdateEditClearAll(CCmdUI* pCmdUI);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

Listing 29.

Add member variable to `CMymfc30aDoc` class.

```
public:
    CDib m_dib;
```



Add Member Variable

Variable Type: CDib

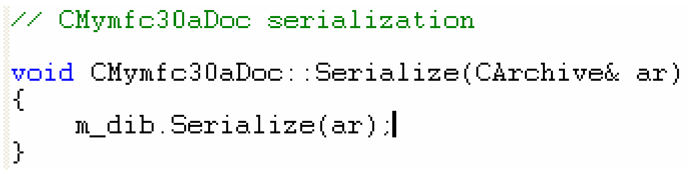
Variable Name: m_dib

Access: [✓ Public] Protected Private

Figure 28: Adding member variable, `m_dib` to `CMymfc30aDoc` class.

Finally add the codes.

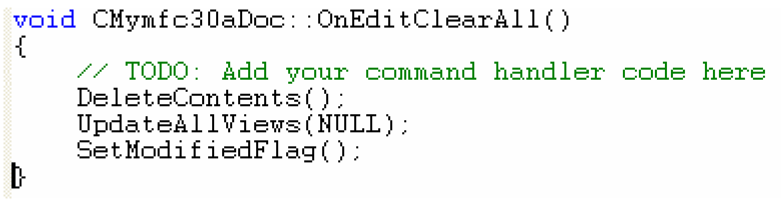
```
void CMymfc30aDoc::Serialize(CArchive& ar)
{
   m_dib.Serialize(ar);
}
```



```
// CMymfc30aDoc serialization

void CMymfc30aDoc::Serialize(CArchive& ar)
{
    m_dib.Serialize(ar);
}
```

Listing 30.

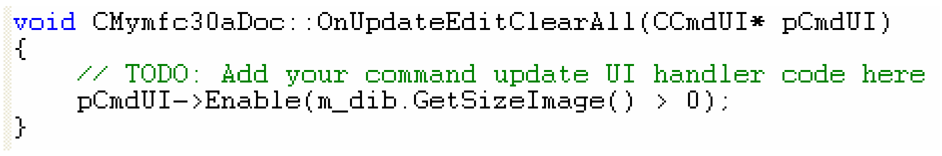
```
void CMymfc30aDoc::OnEditClearAll()
{
    DeleteContents();
    UpdateAllViews(NULL);
    SetModifiedFlag();
}
```



```
void CMymfc30aDoc::OnEditClearAll()
{
    // TODO: Add your command handler code here
    DeleteContents();
    UpdateAllViews(NULL);
    SetModifiedFlag();
}
```

Listing 31.

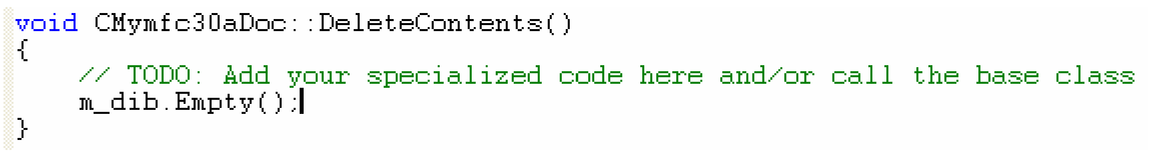
```
void CMymfc30aDoc::OnUpdateEditClearAll(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_dib.GetSizeImage() > 0);
}
```



```
void CMymfc30aDoc::OnUpdateEditClearAll(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(m_dib.GetSizeImage() > 0);
}
```

Listing 32.

```
void CMymfc30aDoc::DeleteContents()
{
   m_dib.Empty();
}
```

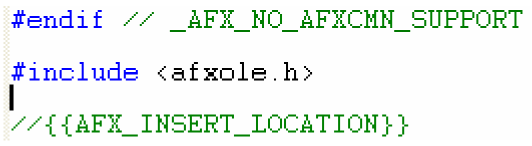


```
void CMymfc30aDoc::DeleteContents()
{
    // TODO: Add your specialized code here and/or call the base class
    m_dib.Empty();
}
```

Listing 33.

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

```
#include <afxole.h>
```

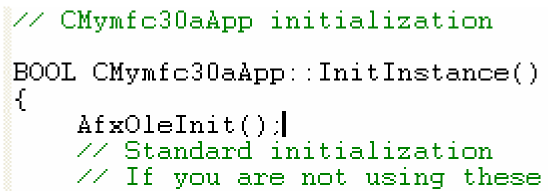


```
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxole.h>

//{{AFX_INSERT_LOCATION}}
```

Listing 34.

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

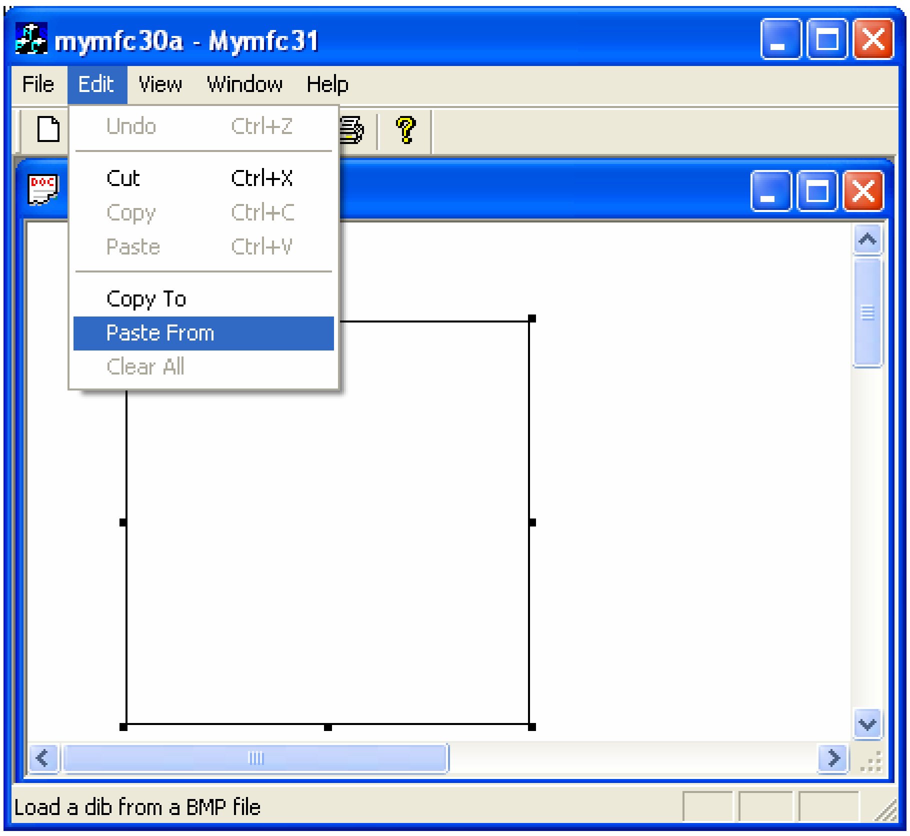
```
AfxOleInit();
```



```
// CMymfc30aApp initialization

BOOL CMymfc30aApp::InitInstance()
{
    AfxOleInit();
    // Standard initialization
    // If you are not using these
```

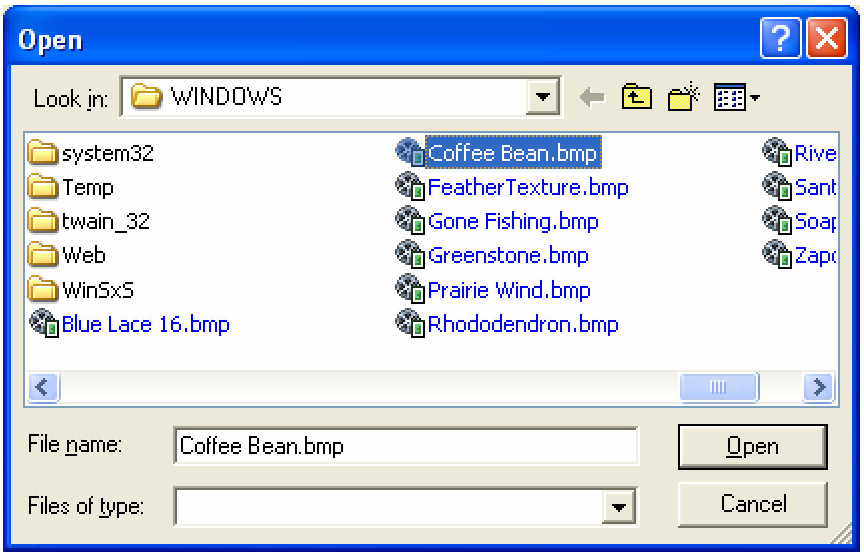
Listing 35.

Build and run MYMFC30A program. Use the **Paste From** menu to paste a BMP file.



Load a dib from a BMP file

Figure 29: MYMFC30A in action. Select the **Paste From** menu.



File name: Coffee Bean.bmp

Files of type:

Figure 30: Browsing BMP file under Windows directory.

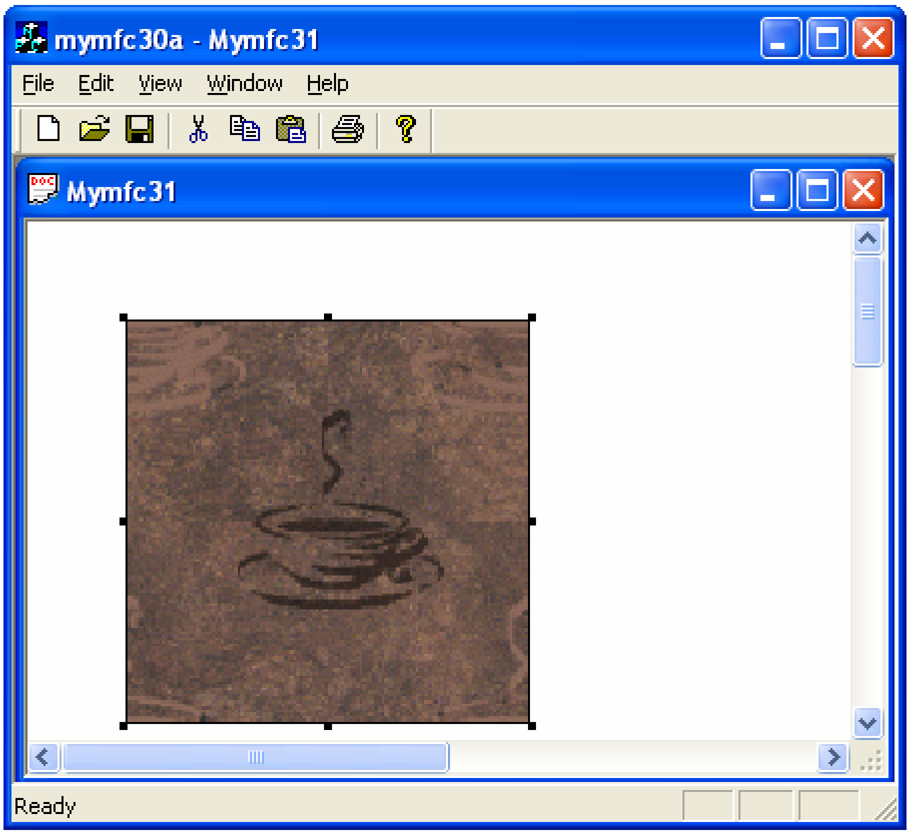


Figure 31: The pasted BMP file in MYMFC30A.

Next, select the **Copy To** menu.

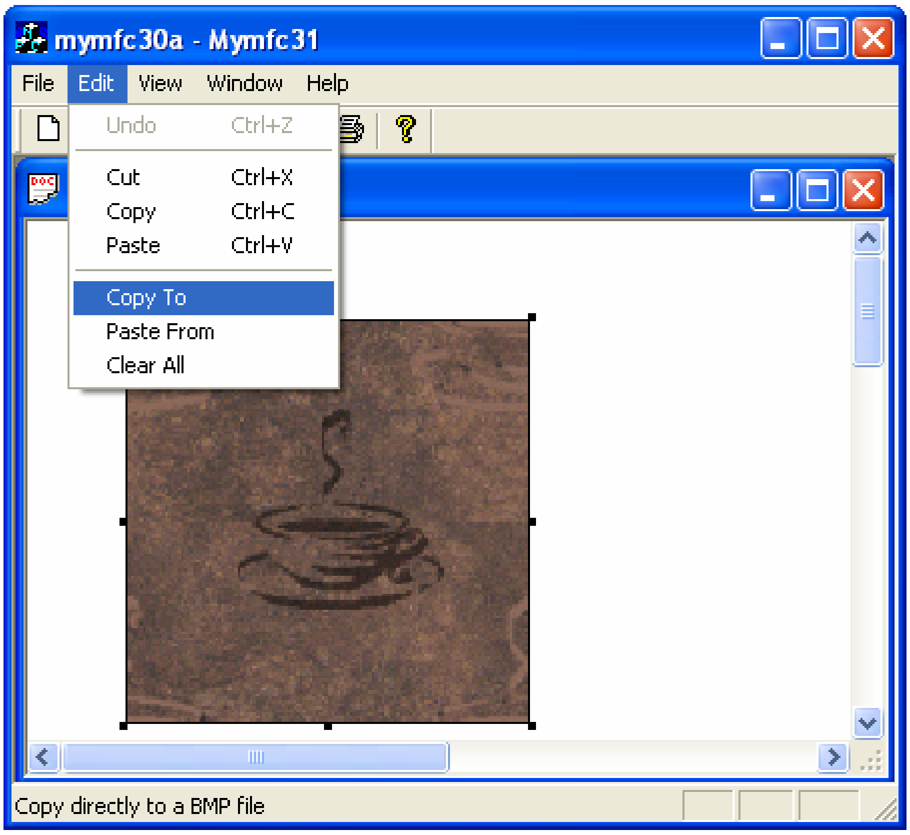


Figure 32: Testing the **Copy To** menu.

Find appropriate directory and put the BMP new filename. Click the **Save** button.

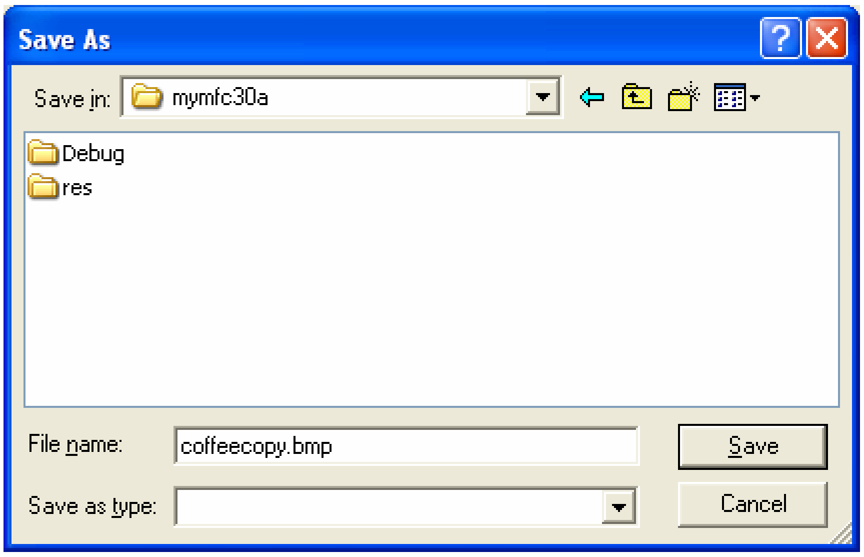


Figure 33: Copying the BMP file to a new file (save as).

Test the **Clear All** menu. This will clear the current BMP.

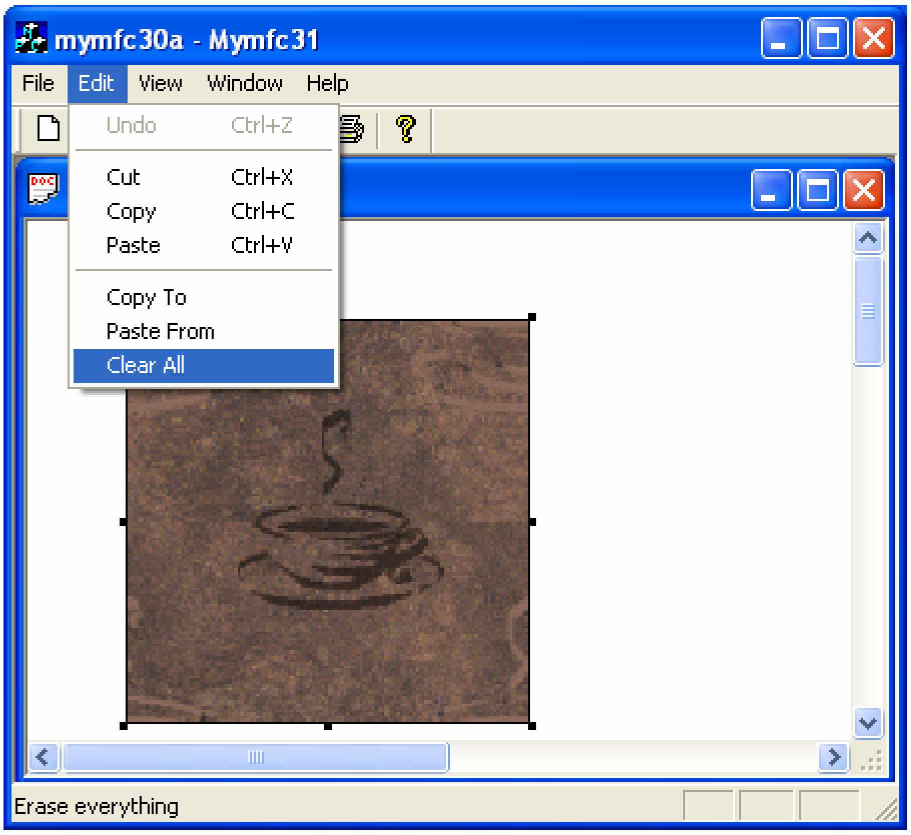


Figure 34: Testing the **Clear All** menu.

The Story

The CMainFrame Class

This class contains the handlers `OnQueryNewPalette()` and `OnPaletteChanged()` for the `WM_QUERYNEWPALETTE` and `WM_PALETTECHANGED` messages, respectively. These handlers send a **user-defined** `WM_VIEWPALETTECHANGED` message to all the views, and then the handler calls `CDib::UsePalette` to realize the palette. The value of `wParam` tells the view whether it should realize the palette in background or foreground mode.

The `CMymfc30aDoc` Class

This class is pretty straightforward. It contains an embedded `CDib` object, `m_dib`, plus a **Clear All** command handler. The overridden `DeleteContents()` member function calls the `CDib::Empty` function.

The `CMymfc30aView` Class

This class contains the clipboard function command handlers, the tracking code, the DIB drawing code, and the palette message handler. Listing 36 shows the header and implementation files with manually entered code in orange.

MYMFC30AVIEW.H

```
// mymfc30aView.h : interface of the CMymfc30aView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MYMFC30AVIEW_H__856698BD_A201_40D4_8CD7_4A9F20DA6994__INCLUDED_)
#define AFX_MYMFC30AVIEW_H__856698BD_A201_40D4_8CD7_4A9F20DA6994__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define WM_VIEWPALETTECHANGED     WM_USER + 5

class CMymfc30aView : public CScrollView
{
    // for tracking
    CRectTracker m_tracker;
    CRect m_rectTracker; // logical coordinates
    CSize m_sizeTotal;   // document size

protected: // create from serialization only
       CMymfc30aView();
       DECLARE_DYNCREATE(CMymfc30aView)

// Attributes
public:
       CMymfc30aDoc* GetDocument();

// Operations
public:

// Overrides
       // ClassWizard generated virtual function overrides
       //{{AFX_VIRTUAL(CMymfc30aView)
       public:
       virtual void OnDraw(CDC* pDC);  // overridden to draw this view
       virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
       virtual void OnPrepareDC(CDC* pDC, CPrintInfo* pInfo = NULL);
       protected:
       virtual void OnInitialUpdate(); // called first time after construct
       virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
       virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
       virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
       //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMymfc30aView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CMymfc30aView)
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message);
        afx_msg void OnSetFocus(CWnd* pOldWnd);
        afx_msg void OnEditCopy();
        afx_msg void OnUpdateEditCopy(CCmdUI* pCmdUI);
        afx_msg void OnEditCopyto();
        afx_msg void OnEditCut();
        afx_msg void OnEditPaste();
        afx_msg void OnUpdateEditPaste(CCmdUI* pCmdUI);
        afx_msg void OnEditPastefrom();
        //}}AFX_MSG
        afx_msg LONG OnViewPaletteChanged(UINT wParam, LONG lParam);
        DECLARE_MESSAGE_MAP()
private:
        COleDataSource* SaveDib();
        BOOL DoPasteDib(COleDataObject* pDataObject);
};

#ifndef _DEBUG  // debug version in mymfc30aView.cpp
inline CMymfc30aDoc* CMymfc30aView::GetDocument()
   { return (CMymfc30aDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_MYMFC30AVIEW_H__856698BD_A201_40D4_8CD7_4A9F20DA6994__INCLUDED_)
```

MYMFC30AVIEW.CPP

```
// mymfc30aView.cpp : implementation of the CMymfc30aView class
//

#include "stdafx.h"
#include "mymfc30a.h"

#include "cdib.h"

#include "mymfc30aDoc.h"
#include "mymfc30aView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMymfc30aView

IMPLEMENT_DYNCREATE(CMymfc30aView, CScrollView)
```

```
BEGIN_MESSAGE_MAP(CMymfc30aView, CScrollView)
       //{{AFX_MSG_MAP(CMymfc30aView)
       ON_WM_LBUTTONDOWN()
       ON_WM_SETCURSOR()
       ON_WM_SETFOCUS()
       ON_COMMAND(ID_EDIT_COPY, OnEditCopy)
       ON_UPDATE_COMMAND_UI(ID_EDIT_COPY, OnUpdateEditCopy)
       ON_COMMAND(ID_EDIT_COPYTO, OnEditCopyto)
       ON_COMMAND(ID_EDIT_CUT, OnEditCut)
       ON_COMMAND(ID_EDIT_PASTE, OnEditPaste)
       ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, OnUpdateEditPaste)
       ON_COMMAND(ID_EDIT_PASTEFROM, OnEditPastefrom)
       //}}AFX_MSG_MAP
       ON_MESSAGE(WM_VIEWPALETTECHANGED, OnViewPaletteChanged)
       // Standard printing commands
       ON_COMMAND(ID_FILE_PRINT, CScrollView::OnFilePrint)
       ON_COMMAND(ID_FILE_PRINT_DIRECT, CScrollView::OnFilePrint)
       ON_COMMAND(ID_FILE_PRINT_PREVIEW, CScrollView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMymfc30aView construction/destruction

CMymfc30aView::CMymfc30aView() : m_sizeTotal(800, 1050), // 8 by 10.5 inches
                                                   //  when printed
    m_rectTracker(50, 50, 250, 250)

{
       // TODO: add construction code here

}

CMymfc30aView::~CMymfc30aView()
{
}

BOOL CMymfc30aView::PreCreateWindow(CREATESTRUCT& cs)
{
       // TODO: Modify the Window class or styles here by modifying
       //  the CREATESTRUCT cs

       return CScrollView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMymfc30aView drawing

void CMymfc30aView::OnDraw(CDC* pDC)
{
    CDib& dib = GetDocument()->m_dib;
    m_tracker.m_rect = m_rectTracker;
    pDC->LPtoDP(m_tracker.m_rect); // tracker wants device coordinates
    m_tracker.Draw(pDC);
    dib.Draw(pDC, m_rectTracker.TopLeft(), m_rectTracker.Size());
}

void CMymfc30aView::OnInitialUpdate()
{
    SetScrollSizes(MM_TEXT, m_sizeTotal);
    m_tracker.m_nStyle = CRectTracker::solidLine | CRectTracker::resizeOutside;
    CScrollView::OnInitialUpdate();
}

/////////////////////////////////////////////////////////////////////////////
// CMymfc30aView printing

BOOL CMymfc30aView::OnPreparePrinting(CPrintInfo* pInfo)
```

```
{
    pInfo->SetMaxPage(1);
    return DoPreparePrinting(pInfo);
}

void CMymfc30aView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
       // TODO: add extra initialization before printing
}

void CMymfc30aView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
       // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CMymfc30aView diagnostics

#ifdef _DEBUG
void CMymfc30aView::AssertValid() const
{
       CScrollView::AssertValid();
}

void CMymfc30aView::Dump(CDumpContext& dc) const
{
       CScrollView::Dump(dc);
}

CMymfc30aDoc* CMymfc30aView::GetDocument() // non-debug version is inline
{
       ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMymfc30aDoc)));
       return (CMymfc30aDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMymfc30aView message handlers

BOOL CMymfc30aView::DoPasteDib(COleDataObject *pDataObject)
{
    // update command user interface should keep us out of
    //  here if not CF_DIB
    if (!pDataObject->IsDataAvailable(CF_DIB)) {
        TRACE("CF_DIB format is unavailable\n");
        return FALSE;
    }
    CMymfc30aDoc* 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);
    pDoc->SetModifiedFlag();
    pDoc->UpdateAllViews(NULL);
    return TRUE;
}

COleDataSource* CMymfc30aView::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;
}

void CMymfc30aView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
    // TODO: Add your specialized code here and/or call the base class
     // custom MM_LOENGLISH; positive y is 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 CMymfc30aView::OnLButtonDown(UINT nFlags, CPoint point)
{
       // TODO: Add your message handler code here and/or call default
       if (m_tracker.Track(this, point, FALSE, NULL)) {
         CClientDC dc(this);
         OnPrepareDC(&dc);
         m_rectTracker = m_tracker.m_rect;
         dc.DPtoLP(m_rectTracker); // Update logical coordinates
         Invalidate();
     }
}

BOOL CMymfc30aView::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);
     }
}

void CMymfc30aView::OnSetFocus(CWnd* pOldWnd)
{
       CScrollView::OnSetFocus(pOldWnd);
       CScrollView::OnSetFocus(pOldWnd);
       AfxGetApp()->m_pMainWnd->SendMessage(WM_PALETTECHANGED, (UINT)
GetSafeHwnd());
}

void CMymfc30aView::OnEditCopy()
{
       // TODO: Add your command handler code here
       COleDataSource* pSource = SaveDib();
     if (pSource) {
        pSource->SetClipboard(); // OLE deletes data source
    }
}

void CMymfc30aView::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 CMymfc30aView::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 CMymfc30aView::OnEditCut()
{
    // TODO: Add your command handler code here
    OnEditCopy();
    GetDocument()->OnEditClearAll();
}

void CMymfc30aView::OnEditPaste()
{
    // TODO: Add your command handler code here
    CMymfc30aDoc* pDoc = GetDocument();
    COleDataObject dataObject;
    VERIFY(dataObject.AttachClipboard());
    DoPasteDib(&dataObject);
    CClientDC dc(this);
    pDoc->m_dib.UsePalette(&dc);
    pDoc->SetModifiedFlag();
    pDoc->UpdateAllViews(NULL);
}

void CMymfc30aView::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 CMymfc30aView::OnEditPastefrom()
{
    // TODO: Add your command handler code here
    // You can try changing to other extensions...
    CMymfc30aDoc* 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->m_dib.UsePalette(&dc);
        pDoc->SetModifiedFlag();
        pDoc->UpdateAllViews(NULL);
    }
}
```

```
LONG CMymfc30aView::OnViewPaletteChanged(UINT wParam, LONG lParam)
{
       TRACE("CMymfc30aView::OnViewPaletteChanged, HWND = %x, code = %d\n",
              GetSafeHwnd(), wParam);
       CClientDC dc(this);
       GetDocument()->m_dib.UsePalette(&dc, wParam);
       Invalidate();
       return 0;
}
```

Listing 36: The `CMymfc30aView` class listing.

Several interesting things happen in the view class. In the `DoPasteDib()` helper, we can call `GetGlobalData()` because we can attach the returned `HGLOBAL` variable to the document's `Cdib` object. If we called `GetData()`, we would have to copy the memory block ourselves. The **Paste From** and **Copy To** command handlers rely on the memory-mapped file support in the `Cdib` class. The `OnPrepareDC()` function creates a special printer-mapping mode that is just like `MM_LOENGLISH` except that positive y is down. One pixel on the display corresponds to 0.01 inch on the printer.

Continue on next module...part 2...

-------------------End part I-----------------

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.