OLE Embedded Components and Containers part 2

Program examples compiled using Visual C++ 6.0 compiler on Windows XP Pro machine with Service Pack 2. The Excel version is Excel 2003/Office 11. 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 IOleObject and OLE.

Index:

An MDI Embedded Component?

The EX32A example is an **SDI mini-server**. Each time a **controller** creates an EX32A object, a new EX32A process is started. You might expect an MDI mini-server process to support multiple component objects, each with its own document, but this is not the case. When you ask AppWizard to generate an MDI mini-server, it generates an SDI program, as in EX32A. It's theoretically possible to have a single process support multiple embedded objects in different windows, but you can't easily create such a program with the MFC library.

In-Place Component Sizing Strategy

If you look at the EX32A output, you'll observe that the **metafile image** does not always match the image in the **in-place** frame window. We had hoped to create another example in which the two images matched. We were unsuccessful, however, when we tried to use the Microsoft Office 97 applications as containers. Each one did something a little different and unpredictable. A complicating factor is the containers' different **zooming abilities**.
When AppWizard generates a component program, it gives you an overridden `OnGetExtent()` function in your server item class. This function returns a hard-coded size of (3000, 3000). You can certainly change this value to suit your needs, but be careful if you change it dynamically. We tried maintaining our own document data member for the component's extent, but that messed us up when the container's zoom factor changed. We thought containers would make more use of another component item virtual function, `OnSetExtent()`, but they don't.
You'll be safest if you simply make your component extents fixed and assume that the container will do the right thing. Keep in mind that when the container application prints its document, it prints the component metafiles. The metafiles are more important than the in-place views.
If you control both container and component programs, however, you have more flexibility. You can build up a modular document processing system with its own sizing protocol. You can even use other OLE interfaces.

Container-Component Interactions

Analyzing the component and the container separately won't help you to understand fully how they work. You must watch them working together to understand their interactions. Let's reveal the complexity one step at a time.
Consider first that you have a container EXE and a component EXE, and the container must manage the component by means of OLE interfaces.
Look back to the space simulation example in Module 23. The client program called `CoGetClassObject()` and `IClassFactory::CreateInstance` to load the spaceship component and to create a spaceship object, and then it called `QueryInterface()` to get `IMotion` and `IVisual` pointers. An embedding container program works the same way that the space simulation client works. It starts the component program based on the component's class ID, and the component program constructs an object. Only the interfaces are different.
Figure 1 shows a container program looking at a component. You've already seen all the interfaces except one, `IOleObject`.

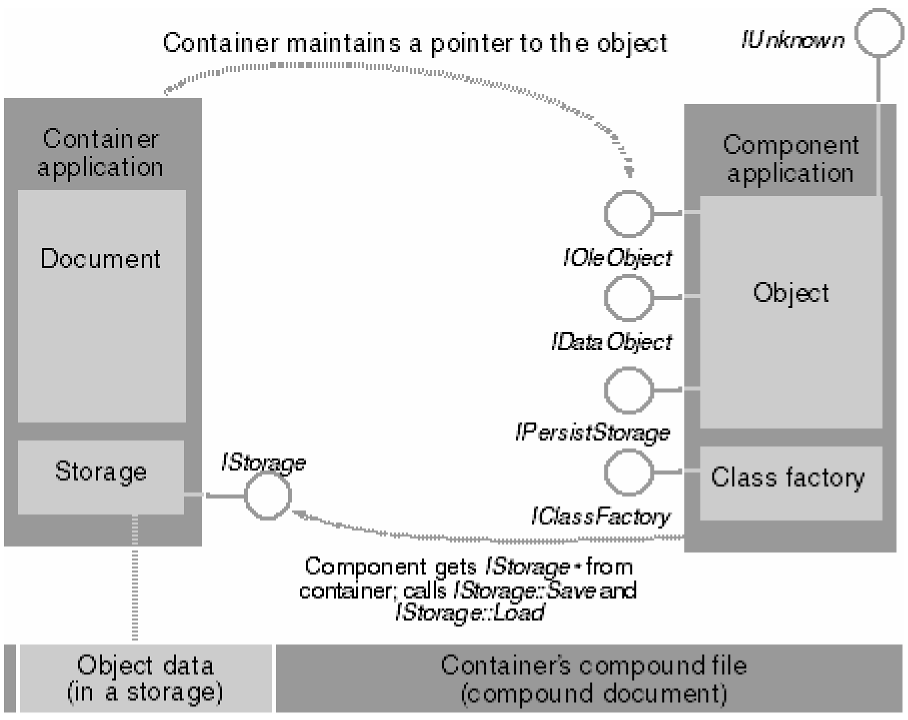

Container maintains a pointer to the object
IUnknown
Container application
Document
Storage
Component application
Object
Class factory
IOleObject
IDataObject
IPersistStorage
IClassFactory
IStorage
Component gets IStorage* from container; calls IStorage::Save and IStorage::Load
Object data (in a storage)
Container's compound file (compound document)

Figure 1: A container program's view of the component.

Using the Component's IOleObject Interface

Loading a component is not the same as activating it. Loading merely starts a process, which then sits waiting for further instructions. If the container gets an `IOleObject` pointer to the component object, it can call the `DoVerb()` member function with a verb parameter such as `OLEIVERB_SHOW`. The component should then show its main window and act like a Windows-based program. If you look at the `IOleObject::DoVerb` description, you'll see an `IOleClientSite*` parameter. We'll consider client sites shortly, but for now you can simply set the parameter to NULL and most components will work okay.
Another important `IOleObject` function, `Close()`, is useful at this stage. As you might expect, the container calls `Close()` when it wants to terminate the component program. If the component process is currently servicing one embedded object (as is the case with MFC components), the process exits.

Loading and Saving the Component's Native Data: Compound Documents

Figure 1 demonstrates that the container manages a storage through an `IStorage` pointer and that the component implements `IPersistStorage`. That means that the component can load and save its native data when the container calls the `Load()` and `Save()` functions of `IPersistStorage`. You've seen the `IStorage` and `IPersistStorage` interfaces used in Module 26, but this time the container is going to save the component's class ID in the storage. The container can read the class ID from the storage and use it to start the component program prior to calling `IPersistStorage::Load`.
Actually, the storage is very important to the embedded object. Just as a virus needs to live in a cell, an embedded object needs to live in a storage. The storage must always be available because the object is constantly loading and saving itself and reading and writing temporary data.
A compound document appears at the bottom of Figure 1. The container manages the whole file, but the embedded components are responsible for the storages inside it. There's one main storage for each embedded object, and the container doesn't know or care what's inside those storages.

Clipboard Data Transfers

If you've run any OLE container programs, including Microsoft Excel, you've noticed that you can copy and paste whole embedded objects. There's a special data object format, `CF_EMBEDDEDOBJECT`, for embedded objects. If you put an `IDataObject` pointer on the clipboard and that data object contains the `CF_EMBEDDEDOBJECT` format (and the companion `CF_OBJECTDESCRIPTOR` format), another program can load the proper component program and reconstruct the object.
There's actually less here than meets the eye. The only thing inside the `CF_EMBEDDEDOBJECT` format is an `IStorage` pointer. The clipboard copy program verifies that `IPersistStorage::Save` has been called to save the embedded object's data in the storage, and then it passes off the `IStorage` pointer in a data object. The clipboard paste program gets the class ID from the source storage, loads the component program, and then calls `IPersistStorage::Load` to load the data from the source storage.
The data objects for the clipboard are generated as needed by the container program. The component's `IDataObject` interface isn't used for transferring the objects' native data.

Getting the Component's Metafile

You already know that a component program is supposed to draw in a metafile and that a container is supposed to play it. But how does the component deliver the metafile? That's what the `IDataObject` interface, shown in Figure 1, is for. The container calls `IDataObject::GetData`, asking for a `CF_METAFILEPICT` format. But wait a minute. The container is supposed to get the metafile even if the component program isn't running. So now you're ready for the next complexity level.

The Role of the In-Process Handler

If the component program is running, it's in a separate process. Sometimes it's not running at all. In either case, the OLE32 DLL is linked into the container's process. This DLL is known as the object handler. It's possible for an EXE component to have its own custom handler DLL, but most components use the "default" OLE32 DLL.
Figure 2 shows the new picture. The handler communicates with the component over the **RPC link**, marshaling all interface function calls. But the handler does more than act as the component's proxy for marshaling; it maintains a

cache that contains the component object's metafile. The handler saves and loads the cache to and from storage, and it can fill the cache by calling the component's `IDataObject::GetData` function.
When the container wants to draw the metafile, it doesn't do the drawing itself; instead, it asks the handler to draw the metafile by calling the handler's `IViewObject2::Draw` function. The handler tries to satisfy as many container requests as it can without bothering the component program. If the handler needs to call a component function, it takes care of loading the component program if it is not already loaded.
The `IViewObject2` interface is an example of OLE's design evolution. Someone decided to add a new function, in this case, `GetExtent()`, to the `IViewObject` interface. `IViewObject2` is derived from `IViewObject` and contains the new function. All new components should implement the new interface and should return an `IViewObject2` pointer when `QueryInterface()` is called for either `IID_IViewObject` or `IID_IViewObject2`. This is easy with the MFC library because you write two interface map entries that link to the same nested class.

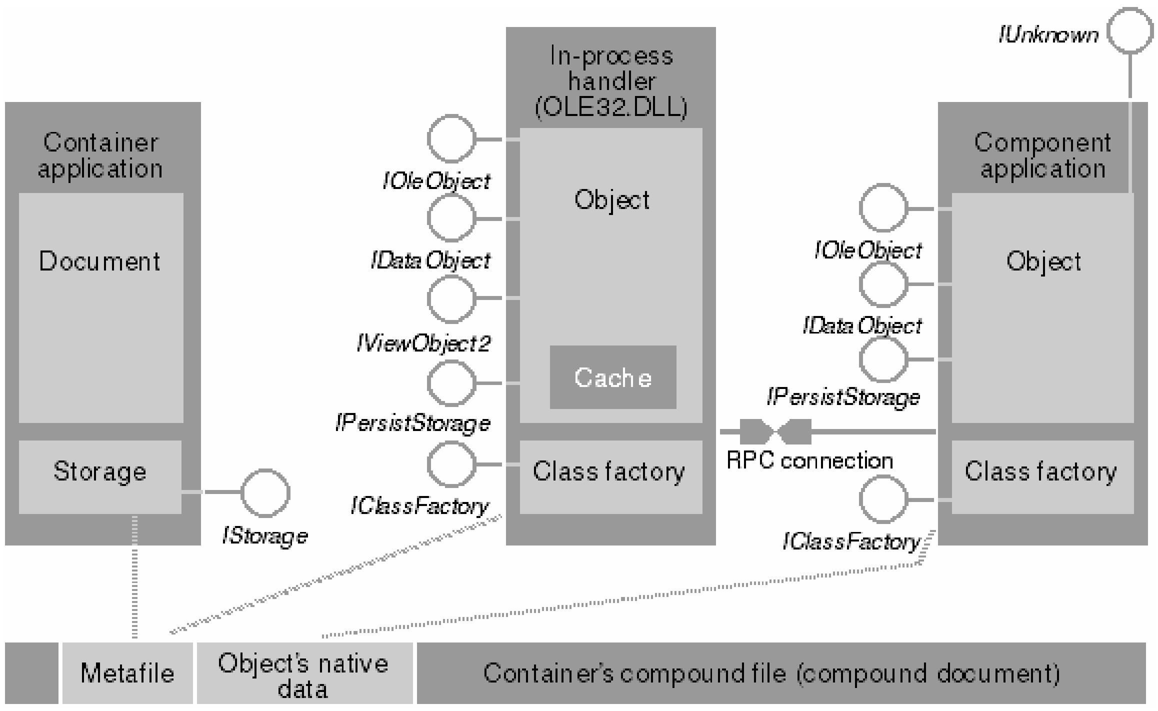


Figure 2: The in-process handler and the component.

Figure 2 shows both object data and metafile data in the object's storage. When the container calls the handler's `IPersistStorage::Save` function, the handler writes the cache (containing the metafile) to the storage and then calls the component's `IPersistStorage::Save` function, which writes the object's native data to the same storage. The reverse happens when the object is loaded.

Component States

Now that you know what a handler is, you're ready for a description of the four states that an embedded object can assume.

State	Description
Passive	The object exists only in a storage.
Loaded	The object handler is running and has a metafile in its cache, but the EXE component program is not running.
Running	The EXE component program is loaded and running, but the window is not visible to the user.
Active	The EXE component's window is visible to the user.

Table 1.

The Container Interfaces

Now for the container side of the conversation. Look at Figure 3. The container consists of a document and one or more sites. The `IOleContainer` interface has functions for iterating over the sites, but we won't worry about iterating over the client sites here. The important interface is `IOleClientSite`. Each site is an object that the component accesses through an `IOleClientSite` pointer. When the container creates an embedded object, it calls `IOleObject::SetClientSite` to establish one of the two connections from component to container. The site maintains an `IOleObject` pointer to its component object.
One important `IOleClientSite` function is `SaveObject()`. When the component decides it's time to save itself to its storage, it doesn't do so directly; instead, it asks the site to do the job by calling `IOleClientSite::SaveObject`. "Why the indirection?" you ask. The handler needs to save the metafile to the storage, that's why. The `SaveObject()` function calls `IPersistStorage::Save` at the handler level, so the handler can do its job before calling the component's `Save()` function.
Another important `IOleClientSite` function is `OnShowWindow()`. The component program calls this function when it starts running and when it stops running. The client is supposed to display a hatched pattern in the embedded object's rectangle when the component program is running or active.

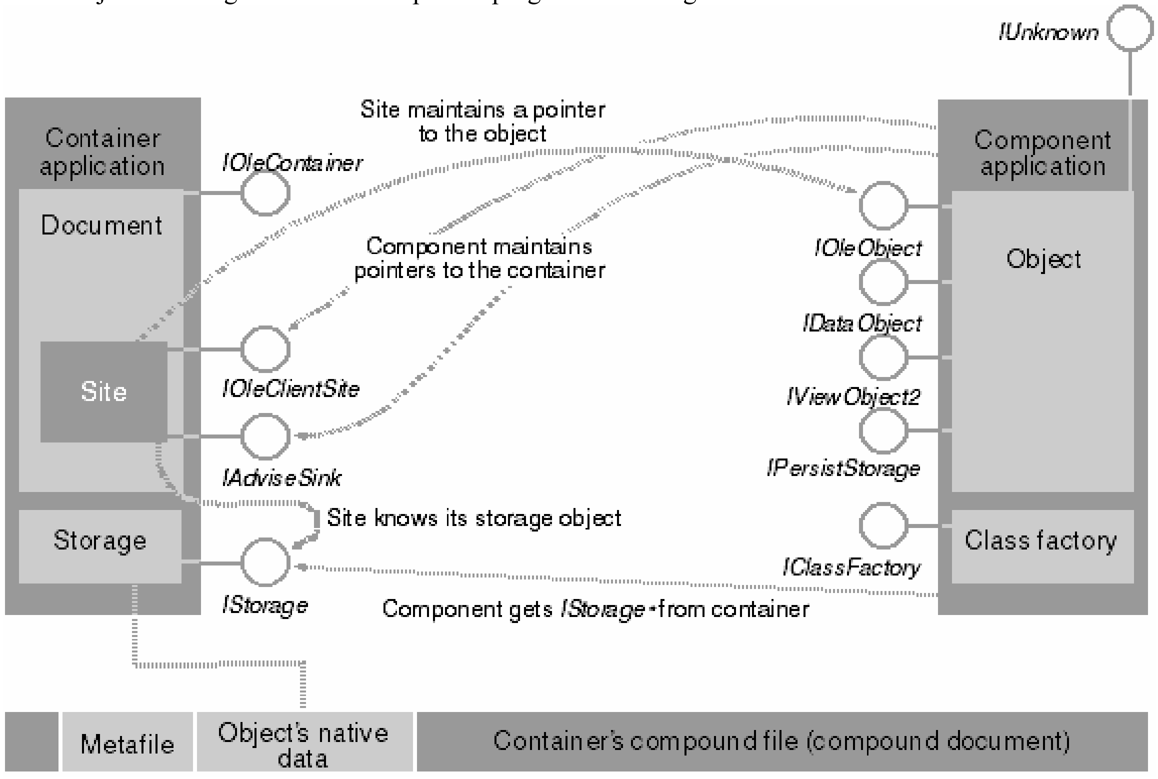


Figure 3: The interaction between the container and the component.

The Advisory Connection

Figure 3 shows another interface attached to the site, `IAdviseSink`. This is the container's end of the second component connection. Why have another connection? The `IOleClientSite` connection goes directly from the component to the container, but the `IAdviseSink` connection is routed through the handler. After the site has created the embedded object, it calls `IViewObject2::SetAdvise`, passing its `IAdviseSink` pointer. Meanwhile, the handler has gone ahead and established two advisory connections to the component. When the embedded object is created, the handler calls `IOleObject::Advise` and then calls `IDataObject::DAdvise` to notify the advise sink of changes in the data object. When the component's data changes, it notifies the handler through the `IDataObject` advisory connection. When the user saves the component's data or closes the program, the component notifies the handler through the `IOleObject` advisory connection. Figure 4 shows these connections.
When the handler gets the notification that the component's data has changed (the component calls `IAdviseSink::OnDataChange`), it can notify the container by calling `IAdviseSink::OnViewChange`. The container responds by calling `IViewObject2::Draw` in the handler. If the component program is not running, the handler draws its metafile from the cache. If the component program is running, the handler calls the

component's `IDataObject::GetData` function to get the latest metafile, which it draws. The `OnClose()` and `OnSave()` notifications are passed in a similar manner.

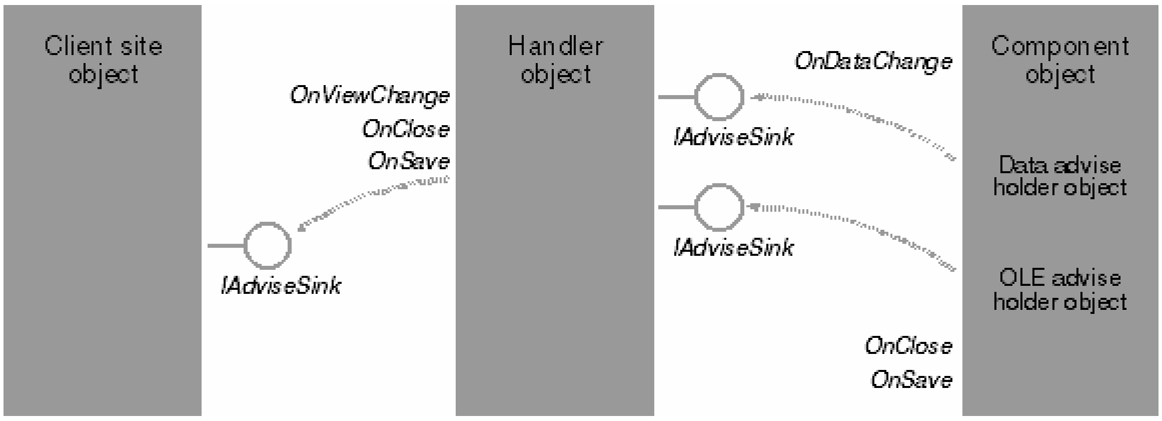


Figure 4: Advisory connection details.

A Metafile for the Clipboard

As you've just learned, the container doesn't deal with the metafile directly when it wants to draw the embedded object; instead, it calls `IViewObject2::Draw`. In one case, however, the container needs direct access to the metafile. When the container copies an embedded object to the clipboard, it must copy a metafile in addition to the embedded object and the object descriptor. That's what the handler's `IDataObject` interface is for. The container calls `IDataObject::GetData`, requesting a metafile format, and it copies that format into the clipboard's data object.

An Interface Summary

Following is a summary of the important OLE interfaces we'll be using in the remaining examples in this module. The function lists are by no means complete, nor are the parameter lists.

The `IOleObject` Interface

Embedded components implement this interface. The client site maintains an `IOleObject` pointer to an embedded object. The `IOleObject` interface is the principal means by which an embedded object provides basic functionality to, and communicates with, its container.

When to Implement

An object application must implement this interface, along with at least `IDataObject` and `IPersistStorage`, for each type of embedded object that it supports. Although this interface contains 21 methods, only three are nontrivial to implement and must be fully implemented: `DoVerb()`, `SetHostNames()`, and `Close()`. Six of the methods provide optional functionality, which, if not desired, can be implemented to return `E_NOTIMPL`: `SetExtent()`, `InitFromData()`, `GetClipboardData()`, `SetColorScheme()`, `SetMoniker()`, and `GetMoniker()`. The latter two methods are useful mainly for enabling links to embedded objects.

When to Use

Call the methods of this interface to enable a container to communicate with an embedded object. A container must call `DoVerb()` to activate an embedded object, `SetHostNames()` to communicate the names of the container application and container document, and `Close()` to move an object from a running to a loaded state. Calls to all other methods are optional. Methods available for `IOleObject` is listed below.

IUnknown Methods	Description
`QueryInterface()`	Returns pointers to supported interfaces.

`AddRef()`	Increments reference count.
`Release()`	Decrements reference count.

Table 2.

IOleObject Methods	Description
`SetClientSite()`	Informs object of its client site in container.
`GetClientSite()`	Retrieves object's client site.
`SetHostNames()`	Communicates names of container application and container document.
`Close()`	Moves object from running to loaded state.
`SetMoniker()`	Informs object of its moniker.
`GetMoniker()`	Retrieves object's moniker.
`InitFromData()`	Initializes embedded object from selected data.
`GetClipboardData()`	Retrieves a data transfer object from the Clipboard.
`DoVerb()`	Invokes object to perform one of its enumerated actions ("verbs").
`EnumVerbs()`	Enumerates actions ("verbs") for an object.
`Update()`	Updates an object.
`IsUpToDate()`	Checks if object is up to date.
`GetUserClassID()`	Returns an object's class identifier.
`GetUserType()`	Retrieves object's user-type name.
`SetExtent()`	Sets extent of object's display area.
`GetExtent()`	Retrieves extent of object's display area.
`Advise()`	Establishes advisory connection with object.
`Unadvise()`	Destroys advisory connection with object.
`EnumAdvise()`	Enumerates object's advisory connections.
`GetMiscStatus()`	Retrieves status of object.
`SetColorScheme()`	Recommends color scheme to object application.

Table 3.

Some of the important methods are explained below.

```
HRESULT Advise(IAdviseSink* AdvSink, DWORD* pdwConnection);
```

The handler calls this function to establish one of the two advisory connections from the component to the handler. The component usually implements `Advise()` with an OLE advise holder object, which can manage multiple advisory connections.

```
HRESULT Close(DWORD dwSaveOption);
```

The container calls `Close()` to terminate the component application but to leave the object in the loaded state. Containers call this function when the user clicks outside an in-place-active component's window. Components that support in-place activation should clean up and terminate.

```
HRESULT DoVerb(LONG iVerb, …, IOleClientSite* pActiveSite, …);
```

Components support numeric verbs as defined in the Registry. A sound component might support a "Play" verb, for example. Embedded components should support the `OLEIVERB_SHOW` verb, which instructs the object to show itself for editing or viewing. If the component supports in-place activation, this verb starts the **Visual Editing** process; otherwise, it starts the component program in a window separate from that of its container. The `OLEIVERB_OPEN` verb causes an in-place-activation-capable component to start in a separate window.

```
HRESULT GetExtent(DWORD dwDrawAspect, SIZEL* pSizel);
```

The component returns the object extent in HIMETRIC dimensions. The container uses these dimensions to size the rectangle for the component's metafile. Sometimes the container uses the extents that are included in the component's metafile picture.

```
HRESULT SetClientSite(IOleClientSite* pClientSite);
```

The container calls SetClientSite() to enable the component to store a pointer back to the site in the container.

```
HRESULT SetExtent(DWORD dwDrawAspect, SIZEL* pSizel);
```

Some containers call this function to impose extents on the component.

```
HRESULT SetHostNames(LPCOLESTR szContainerApp, PCOLESTR szContainerObj);
```

The container calls SetHostNames() so that the component can display the container program's name in its window caption.

```
HRESULT Unadvise(DWORD* dwConnection);
```

This function terminates the advisory connection set up by Advise().

The IViewObject2 Interface

Embedded component handlers implement this interface. Handlers are a type of COM component for dealing with certain client-side aspects of linking and embedding. The default handler (the one provided by Microsoft) lives in a DLL named "**OLE32.DLL**." The container calls its functions, but the component program itself doesn't implement them. An IViewObject2 interface cannot be marshaled across a process boundary because it's associated with a device context.
The IViewObject2 interface is an extension to the IViewObject interface which returns the size of the drawing for a given view of an object. You can prevent the object from being run if it isn't already running by calling this method instead of IOleObject::GetExtent. Like the IViewObject interface, IViewObject2 cannot be marshaled to another process. This is because device contexts are only effective in the context of one process. The OLE-provided default implementation provides the size of the object in the cache.

When to Implement

Object handlers and in-process servers that manage their own presentations implement IViewObject2 for use by compound document containers.

When to Use

A container application or object handler calls the GetExtent() method in the IViewObject2 interface to get the object's size from its cache. Methods available for IViewObject2 are listed below.

IUnknown Methods	Description
QueryInterface()	Returns pointers to supported interfaces.
AddRef()	Increments reference count.
Release()	Decrements reference count.

Table 4.

IViewObject Methods	Description
`Draw()`	Draws a representation of the object onto a device context.
`GetColorSet()`	Returns the logical palette the object uses for drawing.
`Freeze()`	Freezes the drawn representation of an object so it will not change until a subsequent `Unfreeze()`.
`Unfreeze()`	Unfreezes the drawn representation of an object.
`SetAdvise()`	Sets up a connection between the view object and an advise sink so that the advise sink can receive notifications of changes in the view object.
`GetAdvise()`	Returns the information on the most recent `SetAdvise()`.

Table 5.

IViewObject2 Method	Description
`GetExtent()`	Returns the size of the view object from the cache.

Table 6.

Important methods used in this module are:

```
HRESULT Draw(DWORD dwAspect, …, const LPRECTL lprcBounds, …);
```

The container calls this function to draw the component's metafile in a specified rectangle.

```
HRESULT SetAdvise(DWORD dwAspect, …, IAdviseSink* pAdvSink);
```

The container calls `SetAdvise()` to set up the advisory connection to the handler, which in turn sets up the advisory connection to the component.

The IOleClientSite Interface

Containers implement this interface and there is one client site object per component object. The `IOleClientSite` interface is the primary means by which an embedded object obtains information about the location and extent of its display site, its moniker, its user interface, and other resources provided by its container. An object server calls `IOleClientSite` to request services from the container. A container must provide one instance of `IOleClientSite` for every compound-document object it contains. Methods available in `IOleClientSite` are listed below.

IUnknown Methods	Description
`QueryInterface()`	Returns pointers to supported interfaces.
`AddRef()`	Increments reference count.
`Release()`	Decrements reference count.

Table 7.

IOleClientSite Methods	Description
`SaveObject()`	Saves embedded object.
`GetMoniker()`	Requests object's moniker.
`GetContainer()`	Requests pointer to object's container.
`ShowObject()`	Asks container to display object.
`OnShowWindow()`	Notifies container when object becomes visible or invisible.

`RequestNewObjectLayout()`	Asks container to resize display site.

Table 8.

And the related methods used in this module are:

```
HRESULT GetContainer(IOleContainer** ppContainer);
```

The `GetContainer()` function retrieves a pointer to the container object (document), which can be used to enumerate the container's sites.

```
HRESULT OnShowWindow(BOOL fShow);
```

The component program calls this function when it switches between the running and the loaded (or active) state. When the object is in the loaded state, the container should display a hatched pattern on the embedded object's rectangle.

```
HRESULT SaveObject(void);
```

The component program calls `SaveObject()` when it wants to be saved to its storage. The container calls `IPersistStorage::Save`.

The IAdviseSink Interface

Containers implement this interface. Embedded object handlers call its functions in response to component notifications. The `IAdviseSink` interface enables containers and other objects to receive notifications of data changes, view changes, and compound-document changes occurring in objects of interest. Container applications, for example, require such notifications to keep cached presentations of their linked and embedded objects up-to-date. Calls to `IAdviseSink` methods are asynchronous, so the call is sent and then the next instruction is executed without waiting for the call's return.
For an advisory connection to exist, the object that is to receive notifications must implement `IAdviseSink`, and the objects in which it is interested must implement `IOleObject::Advise` and `IDataObject::DAdvise`. In-process objects and handlers may also implement `IViewObject::SetAdvise`. Objects implementing `IOleObject` must support all reasonable advisory methods. To simplify advisory notifications, OLE supplies implementations of the `IDataAdviseHolder` and `IOleAdviseHolder`, which keep track of advisory connections and send notifications to the proper sinks through pointers to their `IAdviseSink` interfaces. `IViewObject` (and its advisory methods) is implemented in the default handler.
As shown in the following table, an object that has implemented an advise sink registers its interest in receiving certain types of notifications by calling the appropriate method:

Call This Method	**To Register for These Notifications**
`IOleObject::Advise`	When a document is saved, closed, or renamed.
`IDataObject::DAdvise`	When a document's data changes.
`IViewObject::SetAdvise`	When a document's presentation changes.

Table 9.

When an event occurs that applies to a registered notification type, the object application calls the appropriate `IAdviseSink` method. For example, when an embedded object closes, it calls the `IAdviseSink::OnClose` method to notify its container. These notifications are asynchronous, occurring after the events that trigger them.

When to Implement

Objects, such as container applications and compound documents, implement `IAdviseSink` to receive notification of changes in data, presentation, name, or state of their linked and embedded objects. Implementers register for one or more types of notification, depending on their needs.
Notifications of changes to an embedded object originate in the server and flow to the container by way of the object handler. If the object is a linked object, the OLE link object intercepts the notifications from the object handler and notifies the container directly. All containers, the object handler, and the OLE link object register for OLE notifications. The typical container also registers for view notifications. Data notifications are usually sent to the OLE link object and object handler.

When to Use

Servers call the methods of `IAdviseSink` to notify objects with which they have an advisory connection of changes in an object's data, view, name, or state. OLE does not permit synchronous calls in the implementation of asynchronous methods, so you cannot make synchronous calls within any of the `IAdviseSink` interface's methods. For example, an implementation of `IAdviseSink::OnDataChange` cannot contain a call to `IDataObject::GetData`. Methods available in `IAdviseSink` are listed below.

IUnknown Methods	Description
`QueryInterface()`	Returns pointers to supported interfaces.
`AddRef()`	Increments reference count.
`Release()`	Decrements reference count.

Table 10.

IAdviseSink Methods	Description
`OnDataChange()`	Advises that data has changed.
`OnViewChange()`	Advises that view of object has changed.
`OnRename()`	Advises that name of object has changed.
`OnSave()`	Advises that object has been saved to disk.
`OnClose()`	Advises that object has been closed.

Table 11.

Important methods used in this module are:

```
void OnClose(void);
```

Component programs call this function when they are being terminated.

```
void OnViewChange(DWORD dwAspect, …);
```

The handler calls `OnViewChange()` when the metafile has changed. Because the component program must have been running for this notification to have been sent, the handler can call the component's `IDataObject::GetData` function to get the latest metafile for its cache. The container can then draw this metafile by calling `IViewObject2::Draw`.

OLE Helper Functions

A number of global OLE functions encapsulate a sequence of OLE interface calls. Following are some that we'll use in the EX32B example:

```
HRESULT OleCreate(REFCLSID rclsid, REFIID riid, …, IOleClientSite*
pClientSite, IStorage* pStg, void** ppvObj);
```

The `OleCreate()` function first executes the COM creation sequence using the specified class ID. This loads the component program. Then the function calls `QueryInterface()` for an `IPersistStorage` pointer, which it uses to call `InitNew()`, passing the `pStg` parameter. It also calls `QueryInterface()` to get an `IOleObject` pointer, which it uses to call `SetClientSite()` using the `pClientSite` parameter. Finally it calls `QueryInterface()` for the interface specified by **riid**, which is usually `IID_IOleObject`.

```
HRESULT OleCreateFromData(IDataObject* pSrcDataObj, REFIID riid, …,
IOleClientSite* pClientSite, IStorage* pStg, void** ppvObj);
```

The `OleCreateFromData()` function creates an embedded object from a data object. In the EX32B example, the incoming data object has the `CF_EMBEDDEDOBJECT` format with an `IStorage` pointer. The function then loads the component program based on the class `ID` in the storage, and then it calls `IPersistStorage::Load` to make the component load the object's native data. Along the way, it calls `IOleObject::SetClientSite`.

```
HRESULT OleDraw(IUnknown* pUnk, DWORD dwAspect, HDC hdcDraw, LPCRECT
lprcBounds);
```

This function calls `QueryInterface()` on `pUnk` to get an `IViewObject` pointer, and then it calls `IViewObject::Draw`, passing the `lprcBounds` parameter.

```
HRESULT OleLoad(IStorage* pStg, REFIID riid, IOleClientSite* pClientSite,
void** ppvObj);
```

The `OleLoad()` function first executes the COM creation sequence by using the class `ID` in the specified storage. Then it calls `IOleObject::SetClientSite` and `IPersistStorage::Load`. Finally, it calls `QueryInterface()` for the interface specified by riid, which is usually `IID_IOleObject`.

```
HRESULT OleSave(IPersistStorage* pPS, IStorage* pStg, …);
```

This function calls `IPersistStorage::GetClassID` to get the object's class `ID`, and then it writes that class `ID` in the storage specified by `pStg`. Finally it calls `IPersistStorage::Save`.

An OLE Embedding Container Application

Now that we've got a working mini-server that supports embedding (EX32A), we'll write a container program to run it. We're not going to use the MFC container support, however, because you need to see what's happening at the OLE interface level. We will use the MFC document-view architecture and the MFC interface maps, and we'll also use the MFC data object classes.

MFC Support for OLE Containers

If you did use AppWizard to build an MFC OLE container application, you'd get a class derived from `COleDocument` and a class derived from `COleClientItem`. These MFC base classes implement a number of important OLE container interfaces for embedding and in-place activation. The idea is that you have one `COleClientItem` object for each embedded object in a single container document. Each `COleClientItem` object defines a site, which is where the component object lives in the window.
The `COleDocument` class maintains a list of client items, but it's up to you to specify how to select an item and how to synchronize the metafile's position with the in-place frame position. AppWizard generates a basic container application with no support for linking, clipboard processing, or drag and drop. If you want those features, you might be better off looking at the MFC DRAWCLI and OCLIENT samples.
We will use one MFC OLE class in the container, `COleInsertDialog`. This class wraps the `OleUIInsertObject` function, which invokes the standard **Insert Object** dialog box. This **Insert Object** dialog enables the user to select from a list of registered component programs.

Some Container Limitations

Because our container application is designed for learning, we'll make some simplifications to reduce the bulk of the code. First of all, this container won't support in-place activation; it allows the user to edit embedded objects only in a separate window. Also, the container supports only one embedded item per document, and that means there's no linking support. The container uses a structured storage file to hold the document's embedded item, but it handles the storage directly, bypassing the framework's serialization system. Clipboard support is provided; drag-and-drop support is not. Outside these limitations, however, it's a pretty good container!

Container Features

So, what does the container actually do? Here's a list of features:

- As an MFC MDI application, it handles multiple documents.
- Displays the component's metafile in a sizeable, moveable tracker rectangle in the view window.
- Maintains a temporary storage for each embedded object.
- Implements the **Insert Object** menu option, which allows the user to select a registered component. The selected component program starts in its own window.
- Allows embedded objects to be copied (and cut) to the clipboard and pasted. These objects can be transferred to and from other containers such as Microsoft Word and Microsoft Excel.
- Allows an embedded object to be deleted.
- Tracks the component program's loaded-running transitions and hatches the tracker rectangle when the component is running or active.
- Redraws the embedded object's metafile on receipt of component change notifications.
- Saves the object in its temporary storage when the component updates the object or exits.
- Copies the embedded object's temporary storage to and from named storage files in response to **Copy To** and **Paste From** commands on the **Edit** menu.

The EX32B Example: An Embedding Container

Now we can move on to the working program. It's a good time to open the **Ex32b.dsw** workspace and build the EX32B project. If you choose **Insert Object** from the **Edit** menu and select **Ex32a Document**, the EX32A component will start. If you change the component's data, the **container** and the **component** will look something like this.

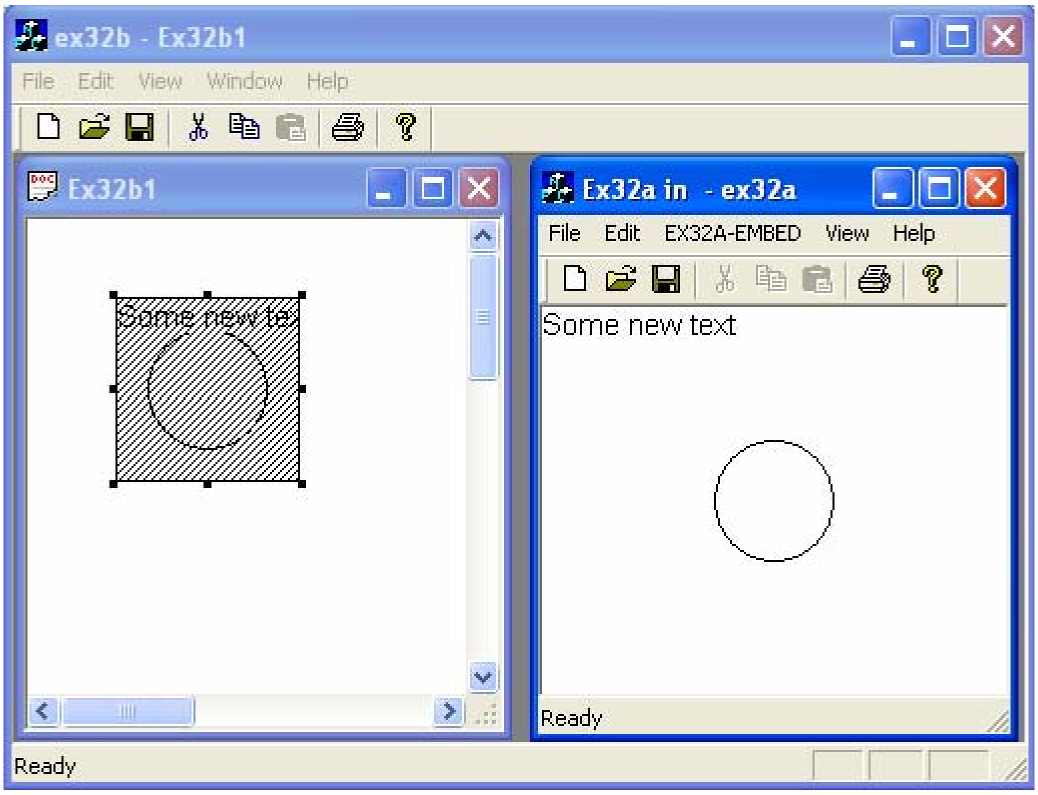
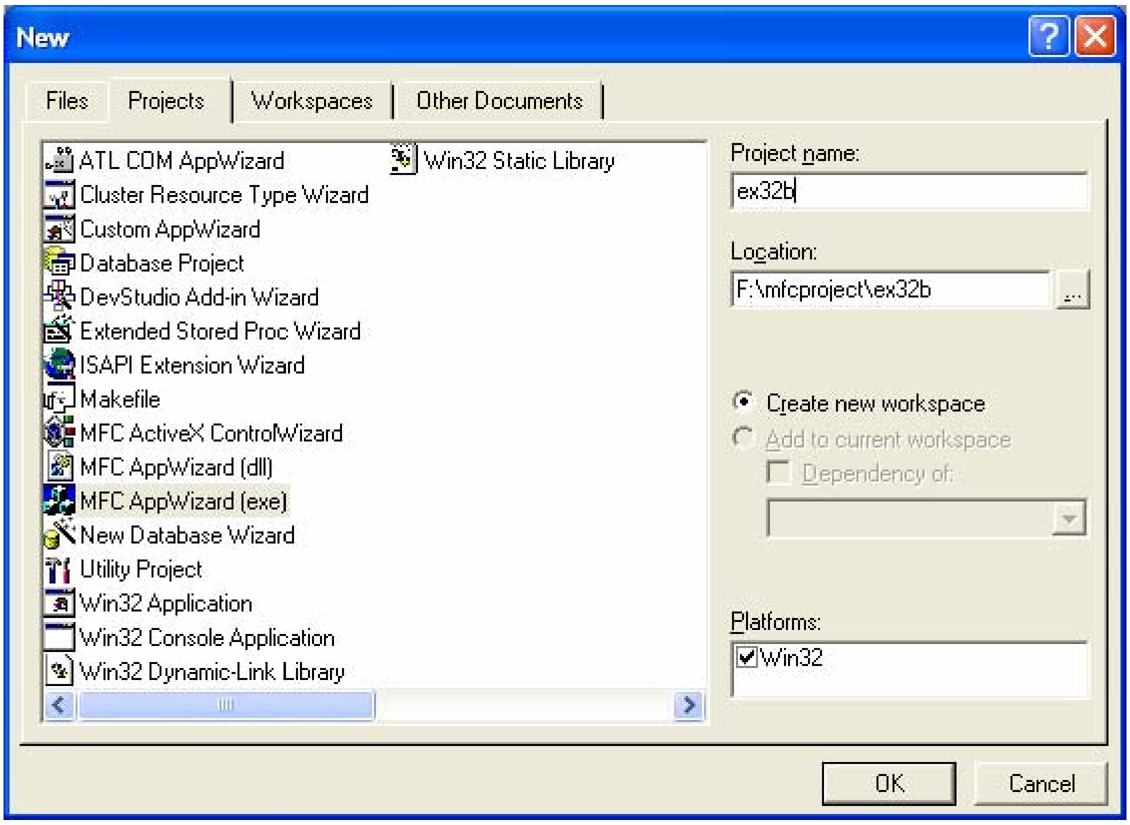


Figure 5: EX32B in action.

The EX32B From Scratch

This is an MDI application without **Automation** and **ActiveX Controls** support. The View's base class is `CScrollView`.



Project name: ex32b

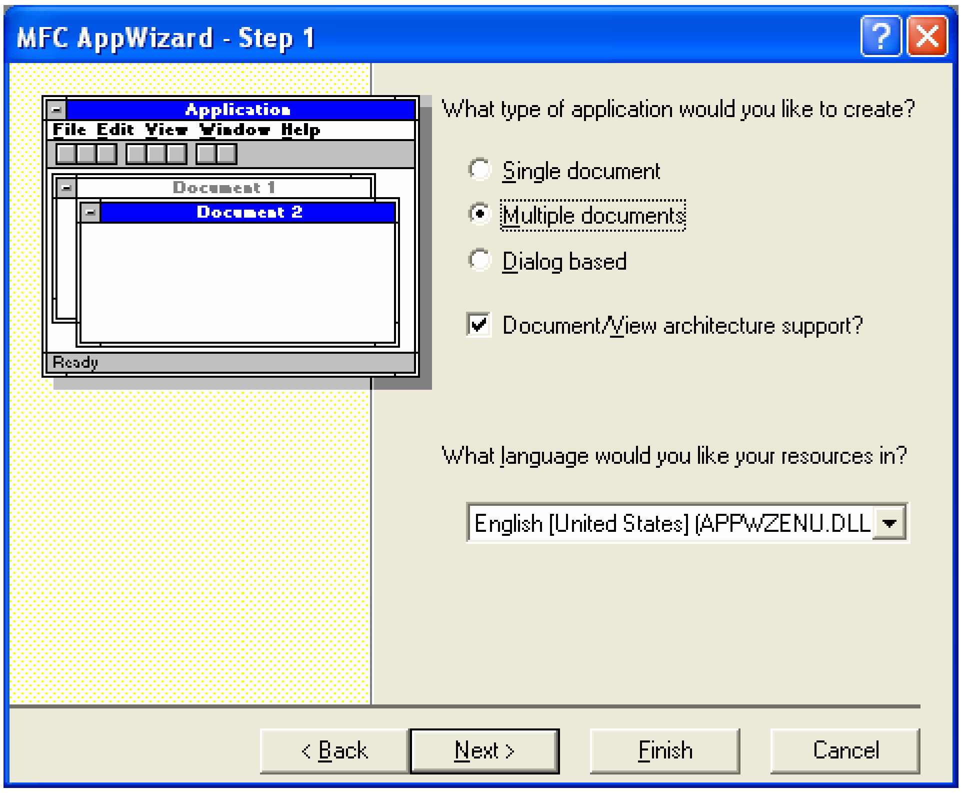
Location: F:\mfcproject\ex32b

Create new workspace

Platforms: ✓ Win32

Figure 6: EX32B – Visual C++ new project dialog.

Select **Multiple documents** option.


MFC AppWizard - Step 1

What type of application would you like to create?

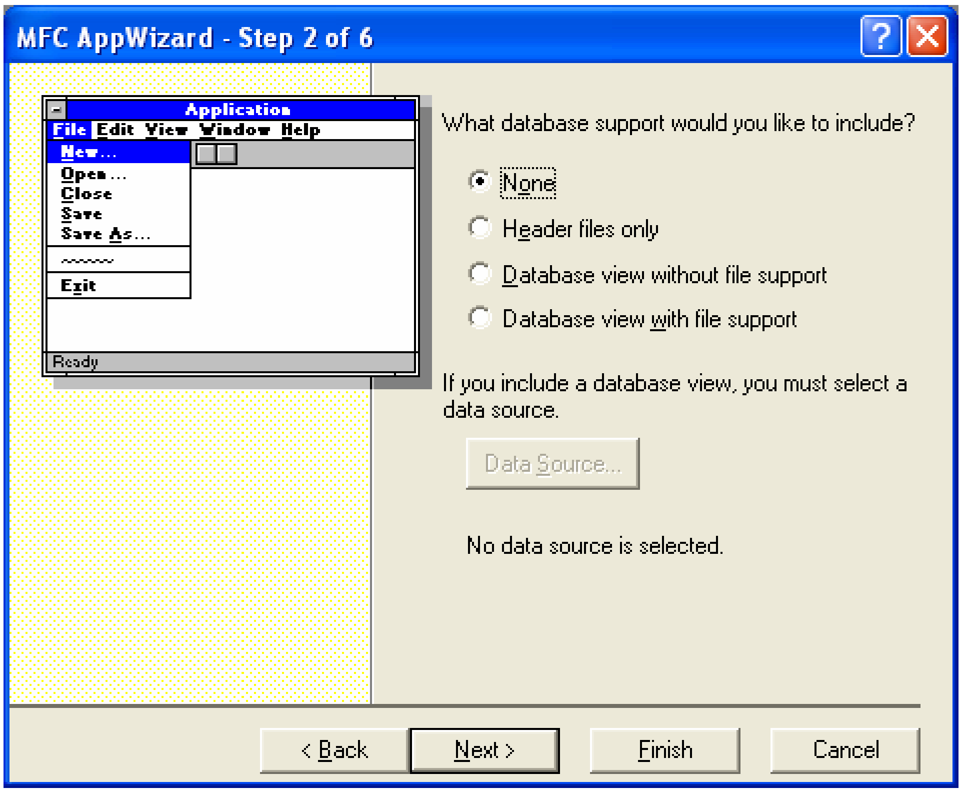
- ( ) Single document
- [✓ Multiple documents]
- ( ) Dialog based

[✓] Document/View architecture support?

What language would you like your resources in?

English [United States] (APPWZENU.DLL

Figure 7: EX32B – AppWizard step 1 of 6.


MFC AppWizard - Step 2 of 6

What database support would you like to include?

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

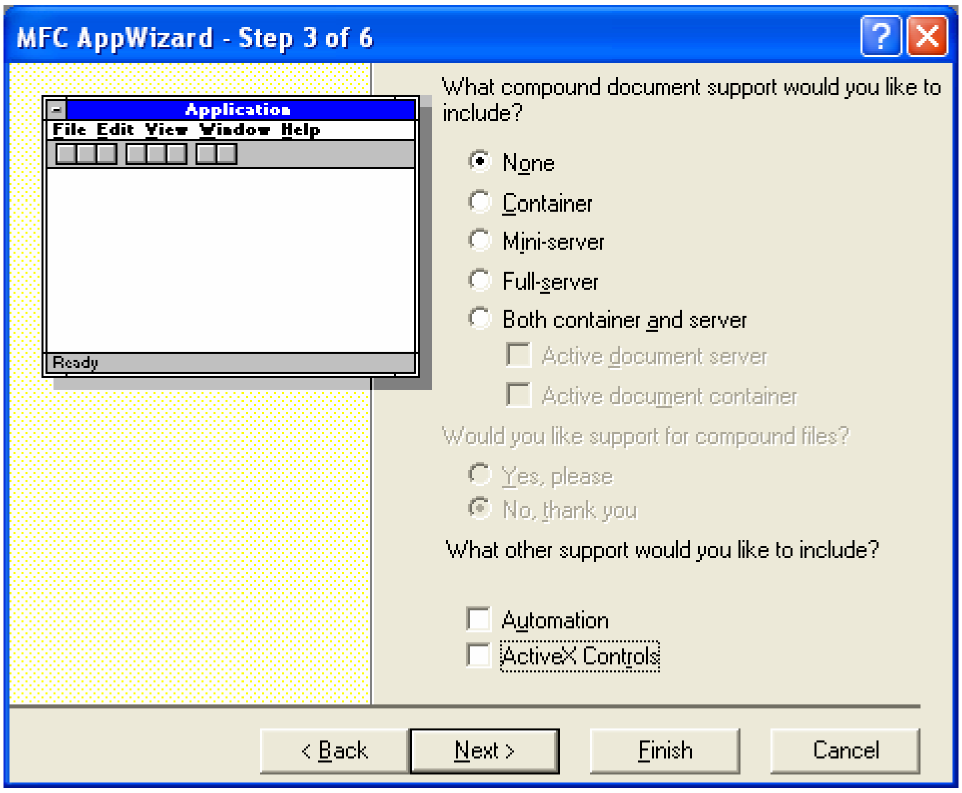
If you include a database view, you must select a data source.

Data Source...

No data source is selected.

Figure 8: EX32B – AppWizard step 2 of 6.

Deselect the **Automation** and **ActiveX Controls** options.



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

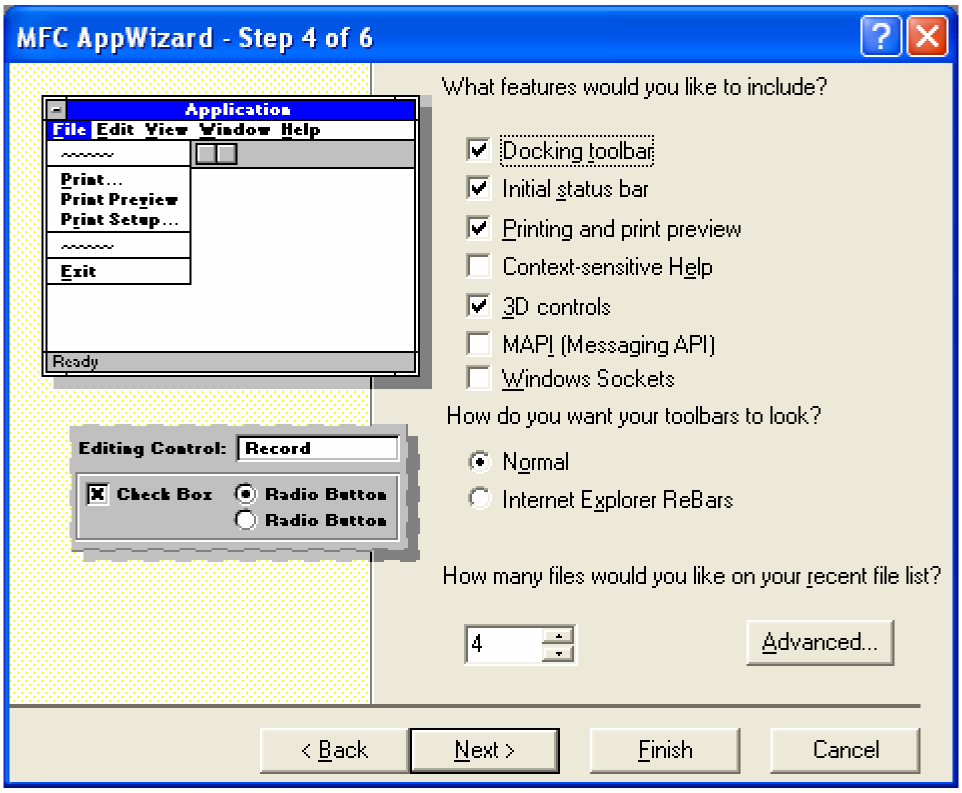
Would you like support for compound files?

Yes, please
[✓ No, thank you]

What other support would you like to include?

☐ Automation
☐ ActiveX Controls

Figure 9: EX32B – 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

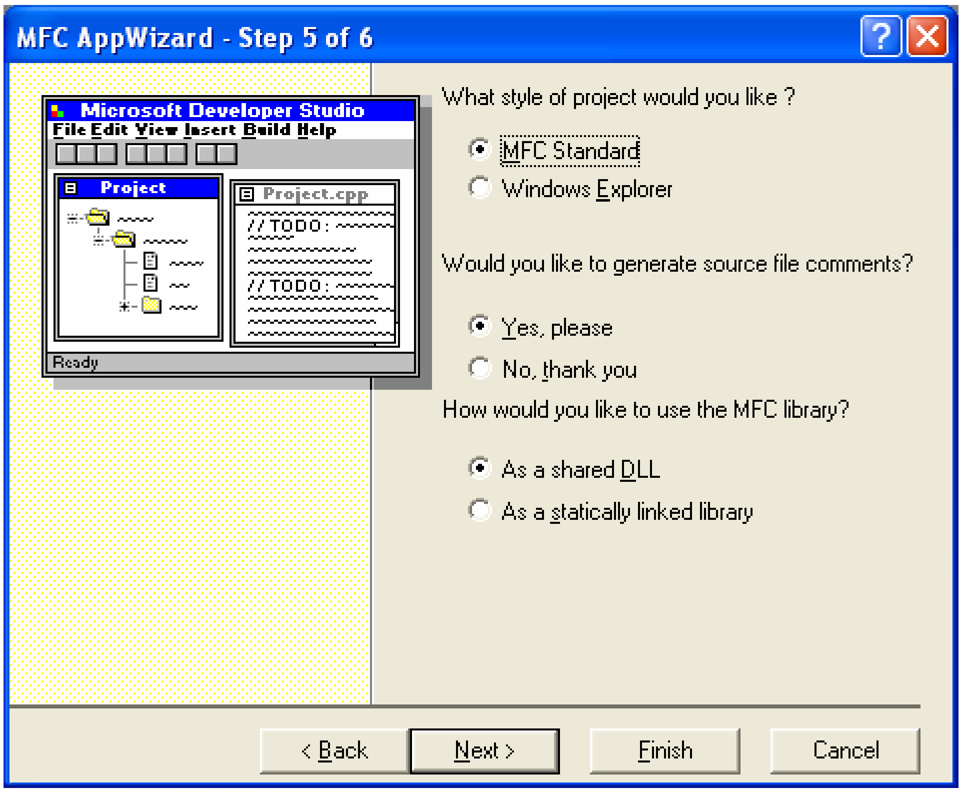
How do you want your toolbars to look?

[✓ Normal]
Internet Explorer ReBars

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

4

Figure 10: EX32B – 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
< Back
Next >
Finish
Cancel

What style of project would you like ?

[✓ MFC Standard] Windows Explorer

Would you like to generate source file comments?

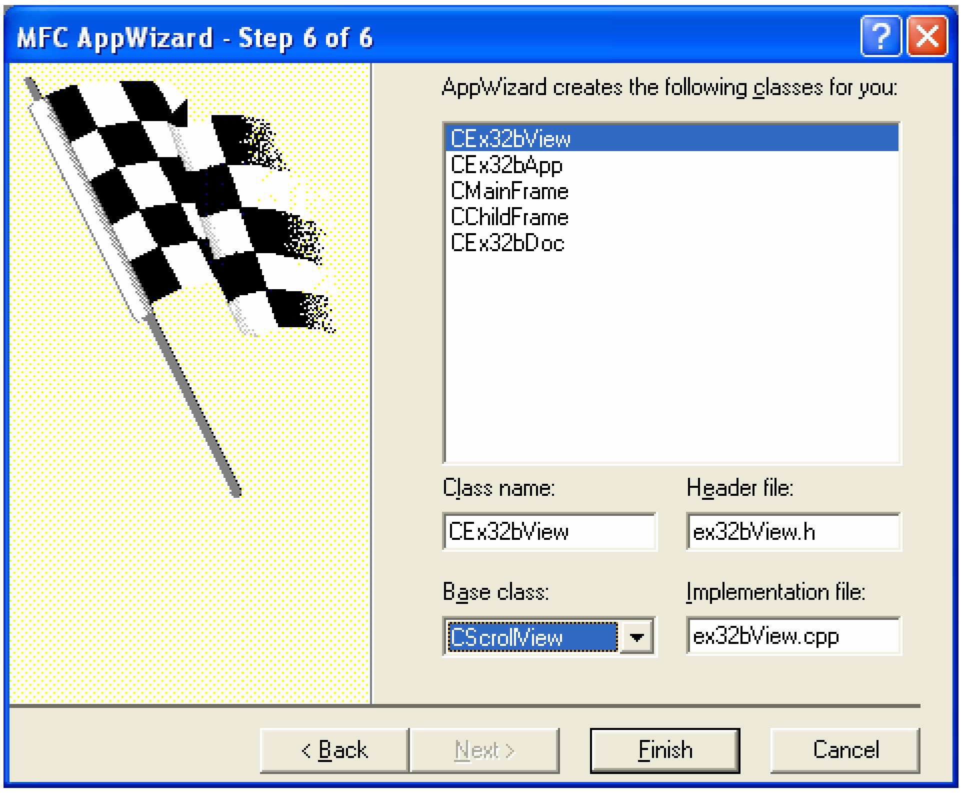
[✓ Yes, please] No, thank you

How would you like to use the MFC library?

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

Figure 11: EX32B – AppWizard step 5 of 6.

Change the view base class to `CScrollView`.



AppWizard creates the following classes for you:

CEx32bView
CEx32bApp
CMainFrame
CChildFrame
CEx32bDoc

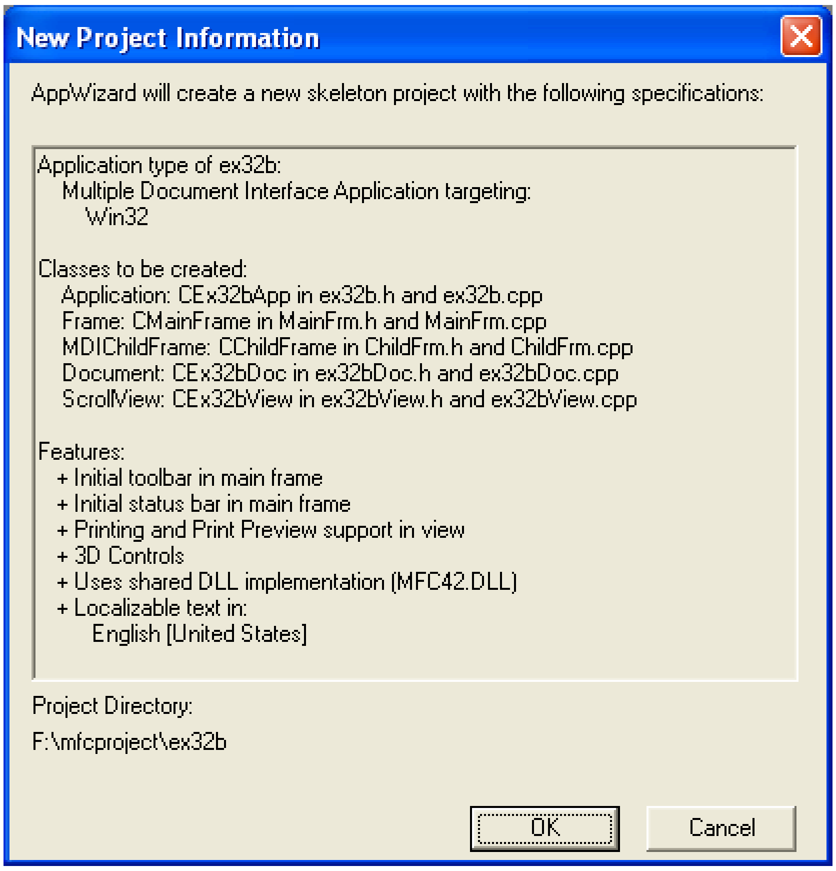
Class name: CEx32bView

Header file: ex32bView.h

Base class: CScrollView

Implementation file: ex32bView.cpp

Figure 12: EX32B – AppWizard step 6 of 6.


New Project Information

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

```
Application type of ex32b:
    Multiple Document Interface Application targeting:
        Win32

Classes to be created:
    Application: CEx32bApp in ex32b.h and ex32b.cpp
    Frame: CMainFrame in MainFrm.h and MainFrm.cpp
    MDIChildFrame: CChildFrame in ChildFrm.h and ChildFrm.cpp
    Document: CEx32bDoc in ex32bDoc.h and ex32bDoc.cpp
    ScrollView: CEx32bView in ex32bView.h and ex32bView.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:

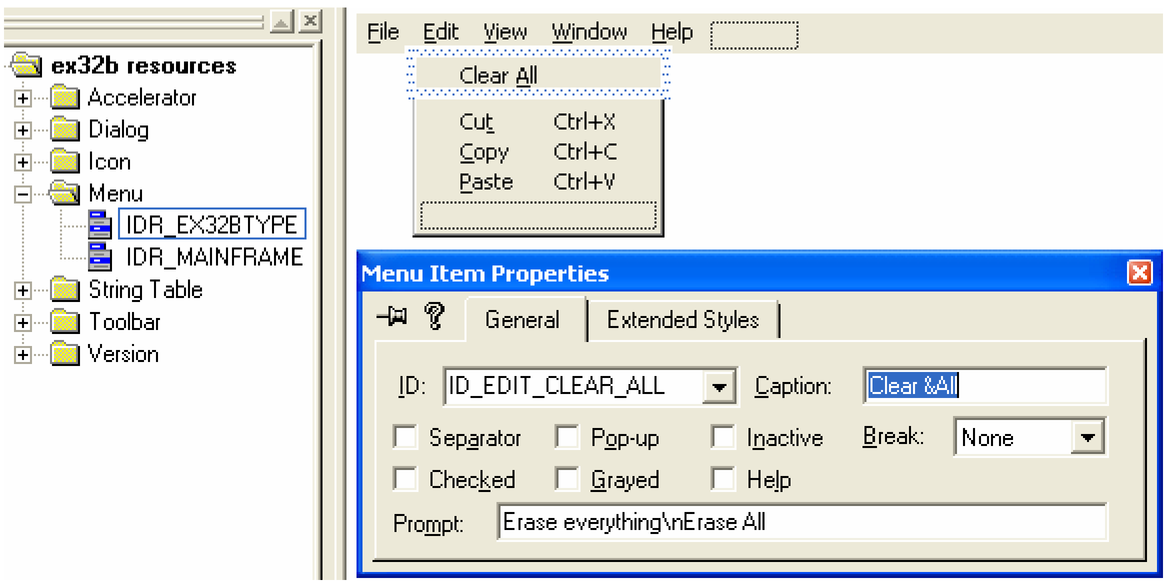
F:\mfcproject\ex32b

Figure 13: EX32B project summary.

Add the following menu items under the existing **Edit** menu.

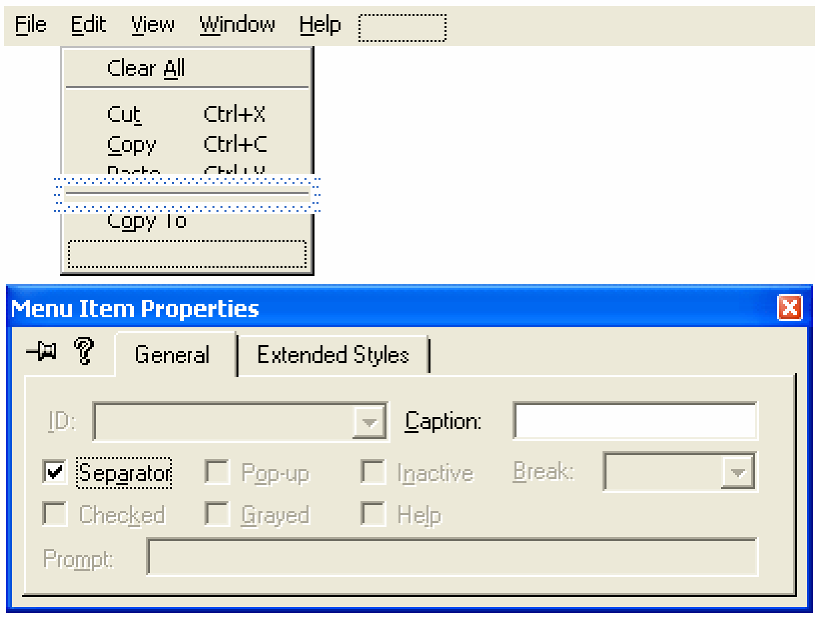
ID	**Caption**
ID_EDIT_CLEAR_ALL	Clear All (replacing the **Undo**)
ID_EDIT_COPYTO	Copy To
ID_EDIT_PASTEFROM	Paste From
ID_EDIT_INSERTOBJECT	Insert Object

Table 12.



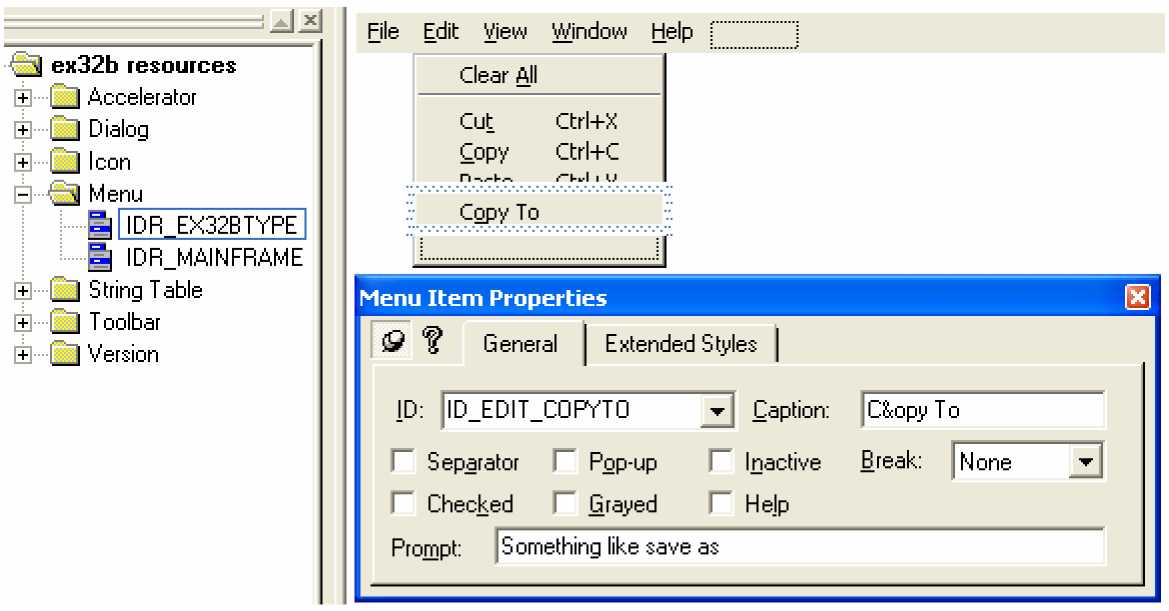
ID: ID_EDIT_CLEAR_ALL
Caption: Clear &All
Separator, Pop-up, Inactive, Checked, Grayed, Help: unchecked
Break: None
Prompt: Erase everything\nErase All

Figure 14: Replacing **Undo** with **Clear All** menu item.



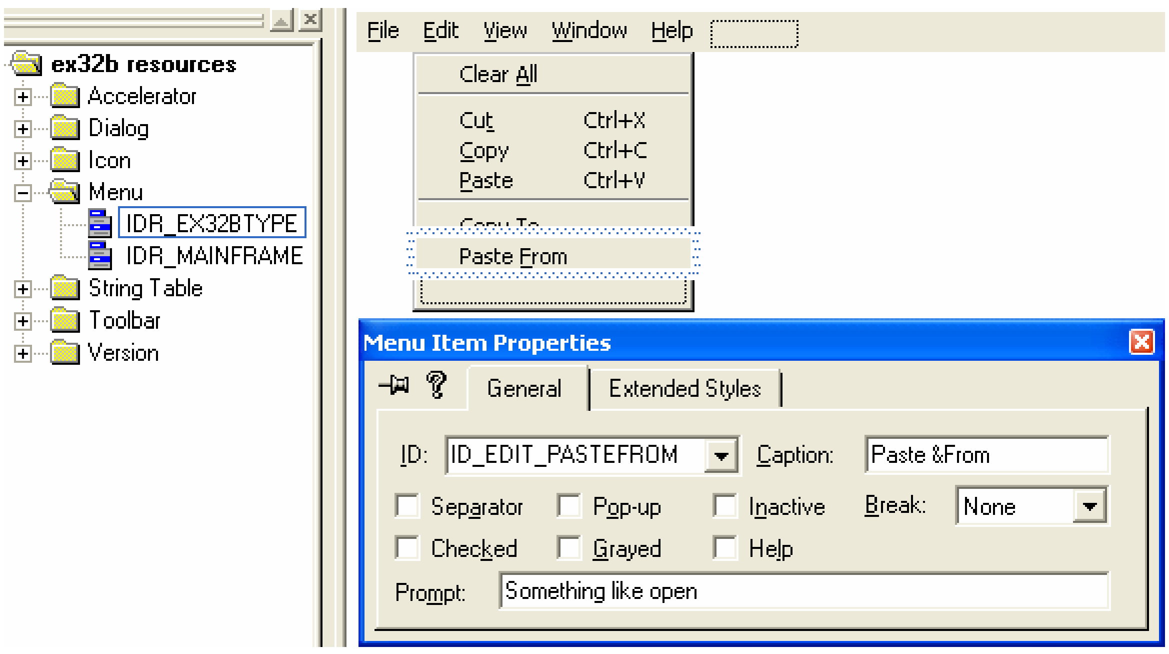
ID:
Caption:
[✓ Separator] Pop-up Inactive
Break:
Checked Grayed Help
Prompt:

Figure 15: Adding **Separator**.



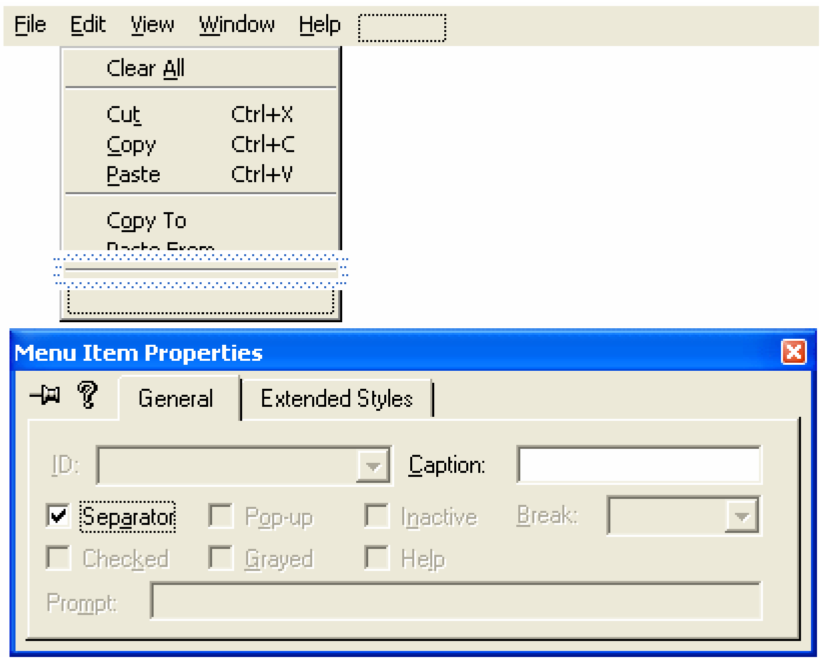
ID: ID_EDIT_COPYTO
Caption: C&opy To
Separator ☐ Pop-up ☐ Inactive ☐ Break: None
Checked ☐ Grayed ☐ Help ☐
Prompt: Something like save as

Figure 16: Adding **Copy To** menu item.



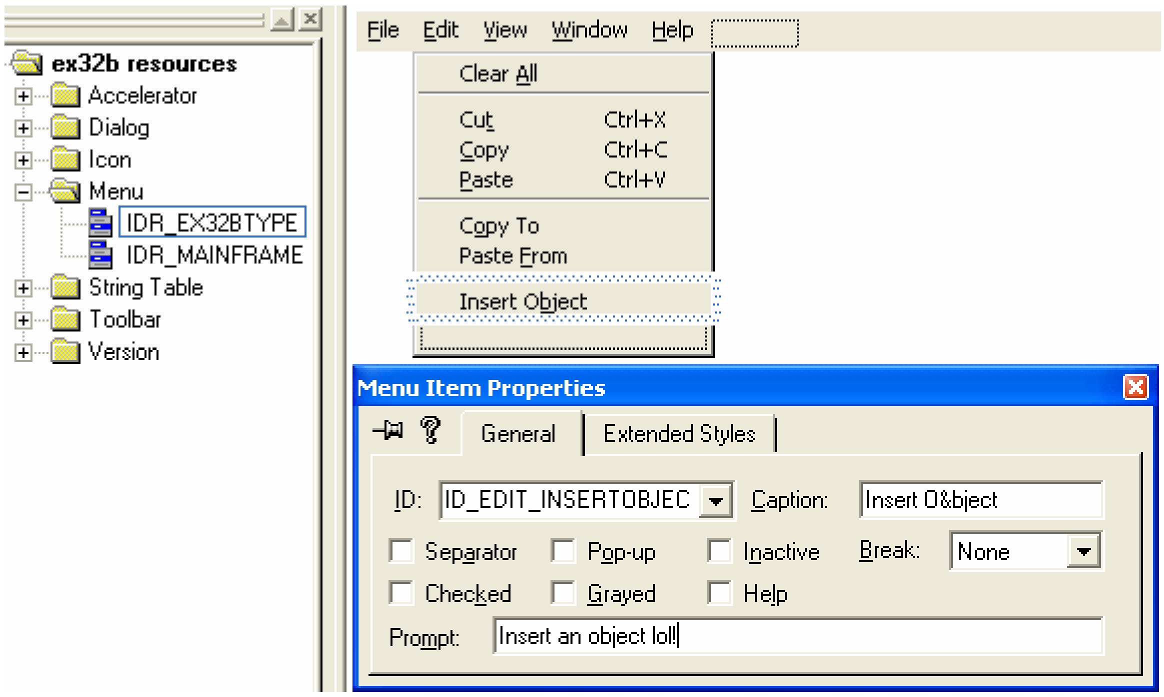
ID: ID_EDIT_PASTEFROM
Caption: Paste &From
Separator ☐ Pop-up ☐ Inactive ☐ Break: None
Checked ☐ Grayed ☐ Help ☐
Prompt: Something like open

Figure 17: Adding **Paste From** menu item.



ID:
Caption:
[✓] Separator
[ ] Pop-up
[ ] Inactive
Break:
[ ] Checked
[ ] Grayed
[ ] Help
Prompt:

Figure 18: Adding **Separator**.



ID: ID_EDIT_INSERTOBJEC
Caption: Insert O&bject
[ ] Separator
[ ] Pop-up
[ ] Inactive
Break: None
[ ] Checked
[ ] Grayed
[ ] Help
Prompt: Insert an object lol!

Figure 19: Adding **Insert Object** menu item.

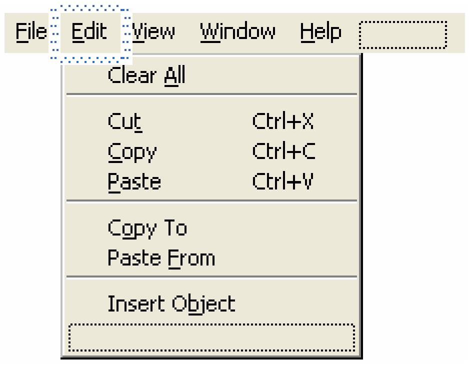
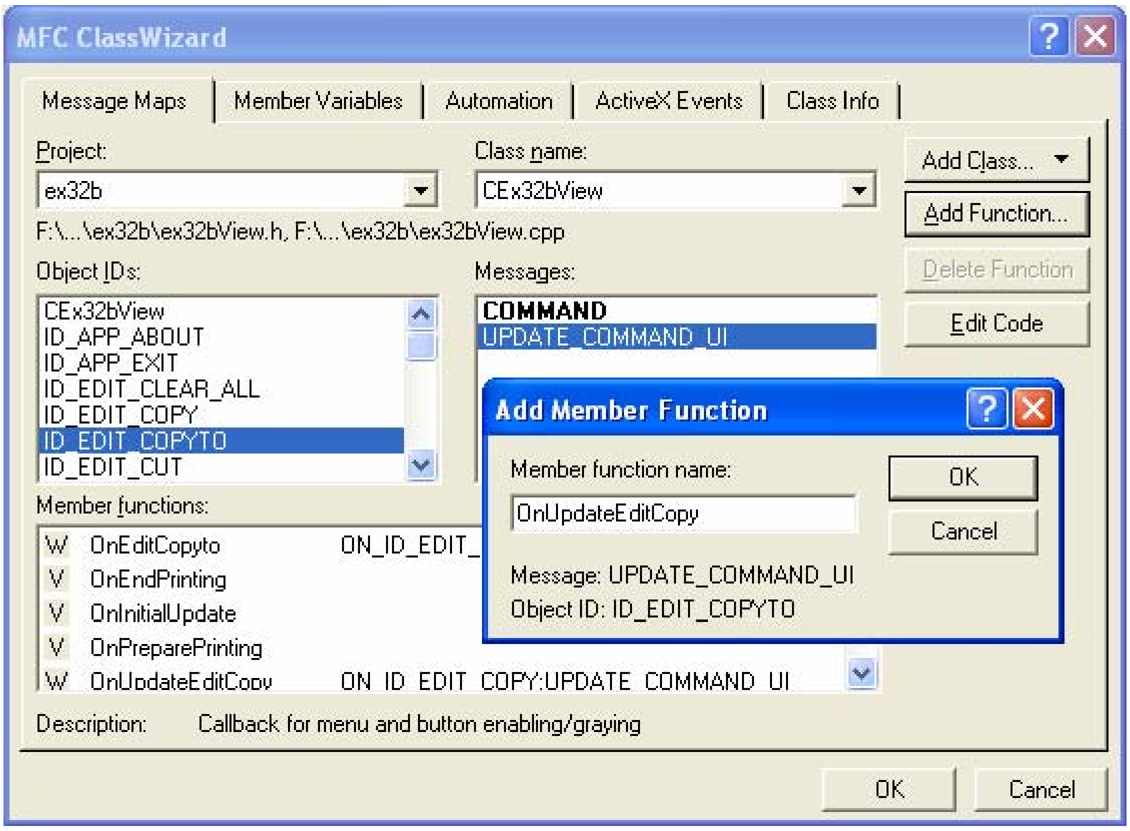


Figure 20: A completed EX32B menu items.

Use ClassWizard to add the following message handlers to `CEx32bView` class. Take note that `ID_EDIT_COPY`, `ID_EDIT_CUT` and `ID_EDIT_COPYTO` having same **update command** handler.

ID	Type	Handler
ID_EDIT_COPY	COMMAND	OnEditCopy()
ID_EDIT_COPY	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_COPYTO	COMMAND	OnEditCopyto()
ID_EDIT_COPYTO	UPDATE COMMAND	OnUpdateEditCopy()
ID_EDIT_INSERTOBJECT	COMMAND	OnEditInsertobject()
ID_EDIT_INSERTOBJECT	UPDATE COMMAND	OnUpdateEditInsertobject()
ID_EDIT_PASTEFROM	COMMAND	OnEditPasteFrom()

Table 13.



Project: ex32b

Class name: CEx32bView

F:\...\ex32b\ex32bView.h, F:\...\ex32b\ex32bView.cpp

Object IDs:

```
CEx32bView
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  OnEditCopyto          ON_ID_EDIT_
V  OnEndPrinting
V  OnInitialUpdate
V  OnPreparePrinting
W  OnUpdateEditCopy      ON_ID_EDIT_COPY:UPDATE_COMMAND_UI
```

Description: Callback for menu and button enabling/graying

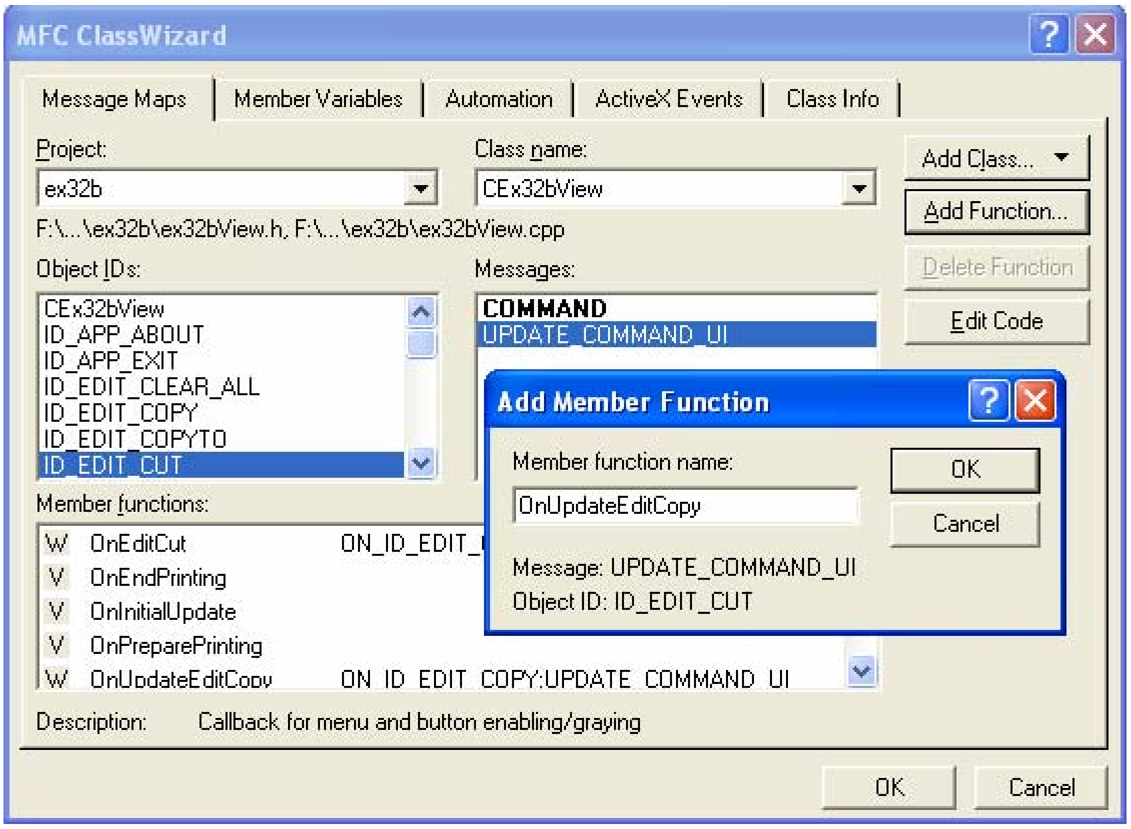
Add Member Function

Member function name: OnUpdateEditCopy

Message: UPDATE_COMMAND_UI

Object ID: ID_EDIT_COPYTO

Figure 21: Adding update command handler for `ID_EDIT_COPYTO`.



Project: ex32b

Class name: CEx32bView

F:\...\ex32b\ex32bView.h, F:\...\ex32b\ex32bView.cpp

Object IDs:

```
CEx32bView
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  OnEditCut             ON_ID_EDIT_
V  OnEndPrinting
V  OnInitialUpdate
V  OnPreparePrinting
W  OnUpdateEditCopy      ON_ID_EDIT_COPY:UPDATE_COMMAND_UI
```

Description: Callback for menu and button enabling/graying

Add Member Function

Member function name: OnUpdateEditCopy

Message: UPDATE_COMMAND_UI

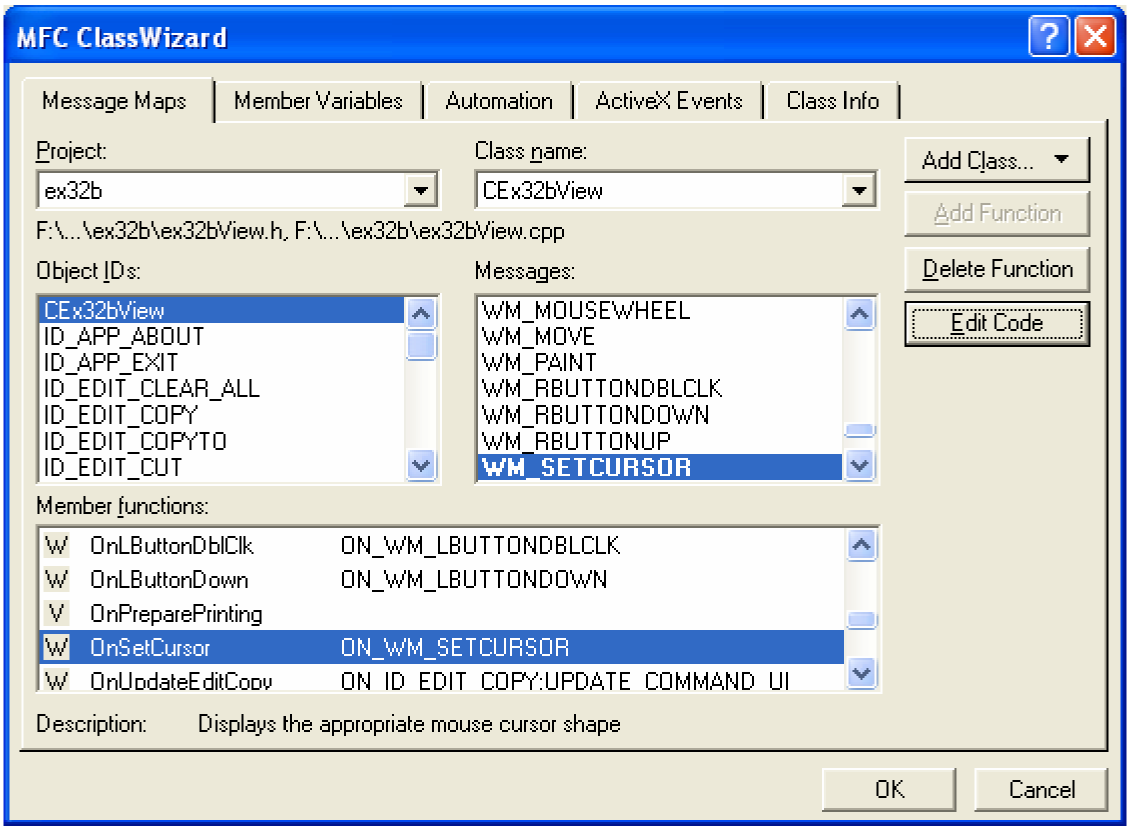
Object ID: ID_EDIT_CUT

Figure 22: Adding update command handler for `ID_EDIT_CUT`.

Add the following Window message handlers to `CEx32bView` class.

Message
WM_LBUTTONDBLCLK
WM_LBUTTONDOWN
WM_SETCURSOR

Table 14.



Project: ex32b

Class name: CEx32bView

F:\...\ex32b\ex32bView.h, F:\...\ex32b\ex32bView.cpp

Object IDs:

```
CEx32bView
ID_APP_ABOUT
ID_APP_EXIT
ID_EDIT_CLEAR_ALL
ID_EDIT_COPY
ID_EDIT_COPYTO
ID_EDIT_CUT
```

Messages:

```
WM_MOUSEWHEEL
WM_MOVE
WM_PAINT
WM_RBUTTONDBLCLK
WM_RBUTTONDOWN
WM_RBUTTONUP
WM_SETCURSOR
```

Member functions:

```
W  OnLButtonDblClk     ON_WM_LBUTTONDBLCLK
W  OnLButtonDown       ON_WM_LBUTTONDOWN
V  OnPreparePrinting
W  OnSetCursor         ON_WM_SETCURSOR
W  OnUpdateEditCopy    ON_ID_EDIT_COPY:UPDATE_COMMAND_UI
```

Description: Displays the appropriate mouse cursor shape

Figure 23: Adding Window message handlers to `CEx32bView` class.

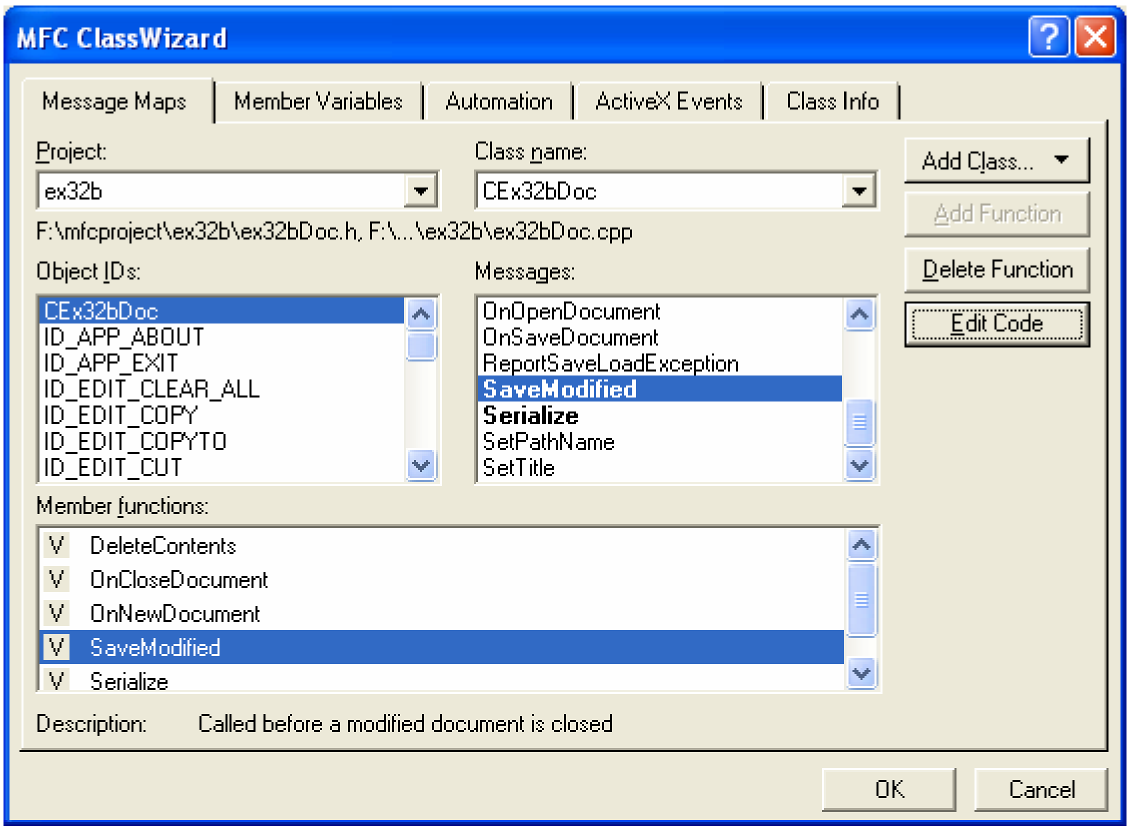
You can verify the added handlers in **ex32bView.h** as shown below.

```
BEGIN_MESSAGE_MAP(CEx32bView, CScrollView)
    //{{AFX_MSG_MAP(CEx32bView)
    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_INSERTOBJECT, OnEditInsertobject)
    ON_UPDATE_COMMAND_UI(ID_EDIT_INSERTOBJECT, OnUpdateEditInsertobject)
    ON_COMMAND(ID_EDIT_PASTE, OnEditPaste)
    ON_WM_LBUTTONDBLCLK()
    ON_WM_LBUTTONDOWN()
    ON_WM_SETCURSOR()
    ON_COMMAND(ID_EDIT_PASTEFROM, OnEditPastefrom)
    ON_UPDATE_COMMAND_UI(ID_EDIT_COPYTO, OnUpdateEditCopy)
    ON_UPDATE_COMMAND_UI(ID_EDIT_CUT, OnUpdateEditCopy)
    ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, OnUpdateEditPaste)
    //}}AFX_MSG_MAP
    // 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()
```

Listing 1.

Add/override the following message handlers to `CEx32bDoc` class.

```
DeleteContents()
OnCloseDocument()
SaveModified()
```



MFC ClassWizard

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

Project: ex32b

Class name: CEx32bDoc

F:\mfcproject\ex32b\ex32bDoc.h, F:\...\ex32b\ex32bDoc.cpp

Object IDs:
- CEx32bDoc
- ID_APP_ABOUT
- ID_APP_EXIT
- ID_EDIT_CLEAR_ALL
- ID_EDIT_COPY
- ID_EDIT_COPYTO
- ID_EDIT_CUT

Messages:
- OnOpenDocument
- OnSaveDocument
- ReportSaveLoadException
- **SaveModified**
- **Serialize**
- SetPathName
- SetTitle

Member functions:

V	DeleteContents
V	OnCloseDocument
V	OnNewDocument
V	SaveModified
V	Serialize

Description: Called before a modified document is closed

Add Class... | Add Function | Delete Function | Edit Code | OK | Cancel

Figure 24: Adding message handlers to `CEx32bDoc` class.

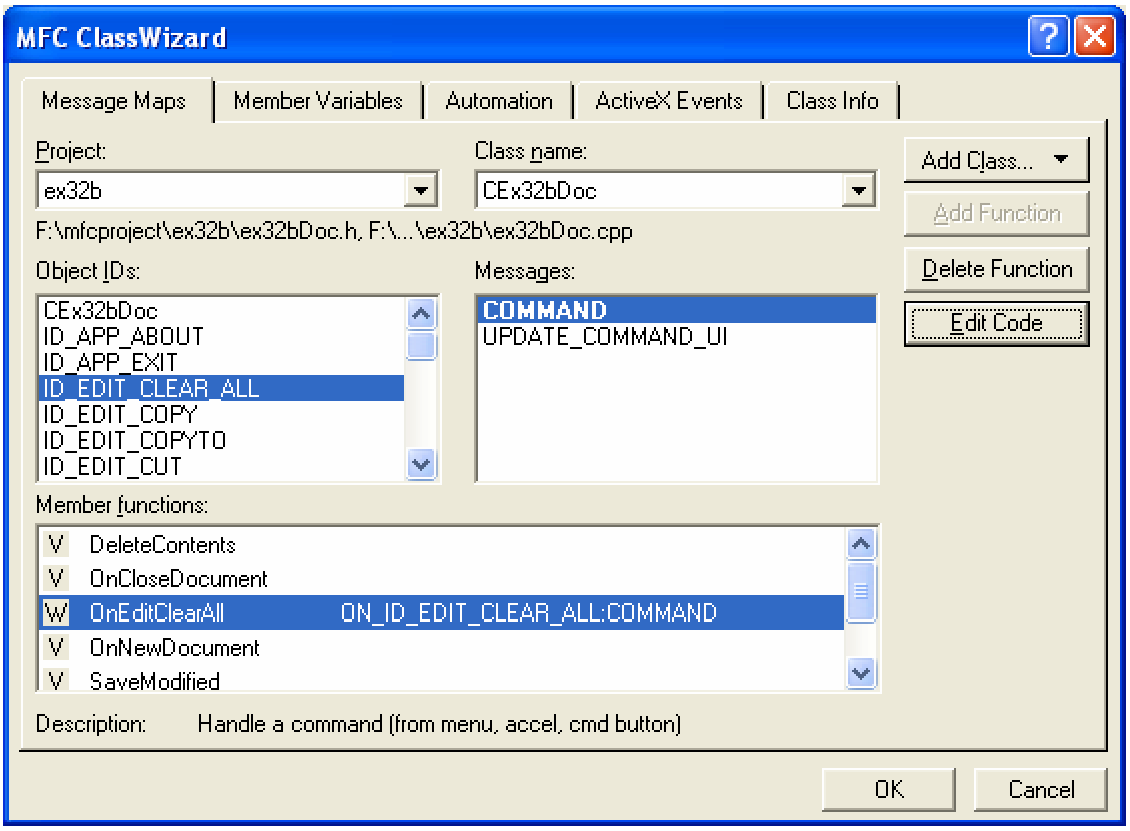
```
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CEx32bDoc)
    public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    virtual void DeleteContents();
    virtual void OnCloseDocument();
    protected:
    virtual BOOL SaveModified();
    //}}AFX_VIRTUAL
```

Listing 2.

Add the following message map for the **Clear All** menu to the `CEx32bDoc` class.

ID		Handler
ID_EDIT_CLEAR_ALL	COMMAND	OnEditClearAll

Table 15.



Project: ex32b

Class name: CEx32bDoc

F:\mfcproject\ex32b\ex32bDoc.h, F:\...\ex32b\ex32bDoc.cpp

Object IDs:

```
CEx32bDoc
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
V  OnCloseDocument
W  OnEditClearAll        ON_ID_EDIT_CLEAR_ALL:COMMAND
V  OnNewDocument
V  SaveModified
```

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

Figure 25: Adding message map for the **Clear All** menu to the `CEx32bDoc` class.

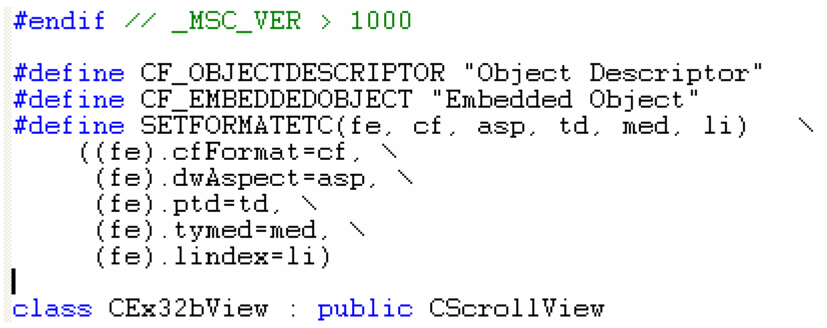
The ClassWizard added code is shown below.

```
// Generated message map functions
protected:
	//{{AFX_MSG(CEx32bDoc)
	afx_msg void OnEditClearAll();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

Listing 3.

Add the following `#define` directives in **ex32bView.h**.

```
#define CF_OBJECTDESCRIPTOR   "Object Descriptor"
#define CF_EMBEDDEDOBJECT "Embedded Object"
#define SETFORMATETC(fe, cf, asp, td, med, li)    \
      ((fe).cfFormat=cf, \
       (fe).dwAspect=asp, \
       (fe).ptd=td, \
       (fe).tymed=med, \
       (fe).lindex=li)
```

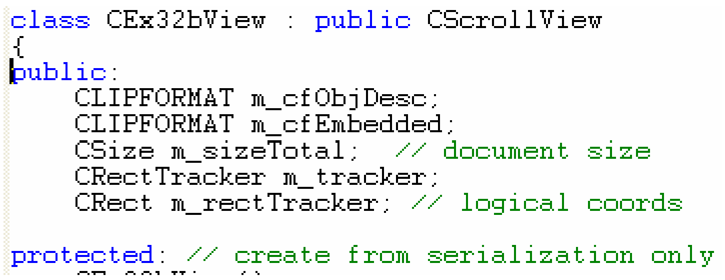


```
#endif // _MSC_VER > 1000

#define CF_OBJECTDESCRIPTOR "Object Descriptor"
#define CF_EMBEDDEDOBJECT "Embedded Object"
#define SETFORMATETC(fe, cf, asp, td, med, li)   \
    ((fe).cfFormat=cf, \
     (fe).dwAspect=asp, \
     (fe).ptd=td, \
     (fe).tymed=med, \
     (fe).lindex=li)

class CEx32bView : public CScrollView
```

Listing 4.

Then, add the following public member variables.

```
public:
      CLIPFORMAT m_cfObjDesc;
      CLIPFORMAT m_cfEmbedded;
      CSize m_sizeTotal;  // document size
      CRectTracker m_tracker;
      CRect m_rectTracker; // logical coords
```



```
class CEx32bView : public CScrollView
{
public:
    CLIPFORMAT m_cfObjDesc;
    CLIPFORMAT m_cfEmbedded;
    CSize m_sizeTotal;  // document size
    CRectTracker m_tracker;
    CRect m_rectTracker; // logical coords

protected: // create from serialization only
```

Listing 5.

Use ClassView, add the following private member functions to `CEx32bView` class.

```
private:
      void GetSize();
      void SetNames();
      void SetViewAdvise();
      BOOL MakeMetafilePict(COleDataSource* pSource);
      COleDataSource* SaveObject();
      BOOL DoPasteObject(COleDataObject* pDataObject);
      BOOL DoPasteObjectDescriptor(COleDataObject* pDataObject);
```

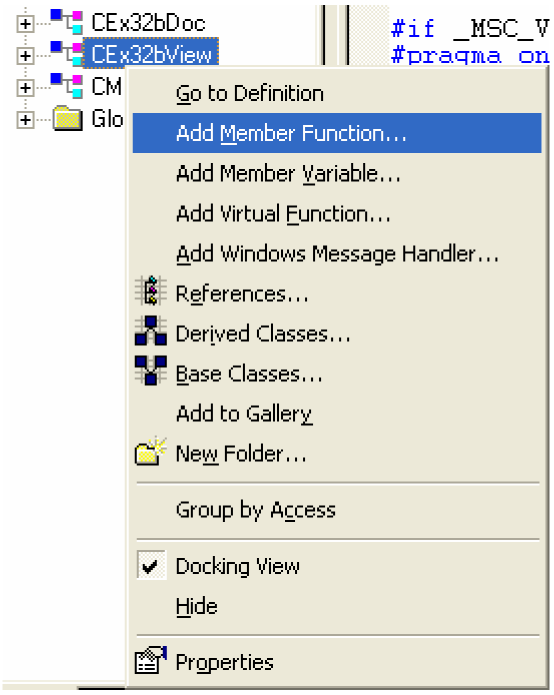
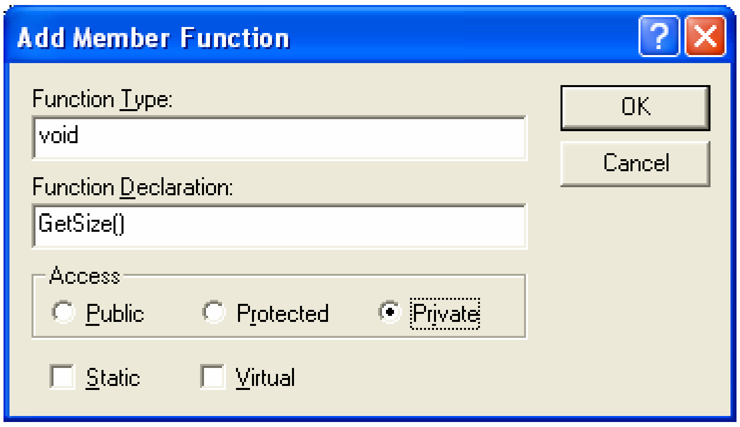


Figure 26: Add new member function using ClassView.



Function Type: void

Function Declaration: GetSize()

Access: Public Protected [✓ Private]

Static Virtual

Figure 27: Adding `CEx32bView`'s `GetSize()` member function.

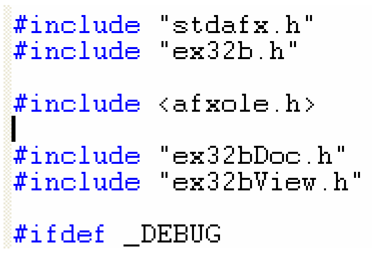
You can verify the added member functions in **ex32bView.h** file.

```
	DECLARE_MESSAGE_MAP()
private:
	BOOL DoPasteObjectDescriptor(COleDataObject* pDataObject);
	BOOL DoPasteObject(COleDataObject* pDataObject);
	COleDataSource* SaveObject();
	BOOL MakeMetafilePict(COleDataSource* pSource);
	void SetViewAdvise();
	void SetNames();
	void GetSize();
};
```

Listing 6.

Add the `#include` directive in **ex32bView.cpp**.

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

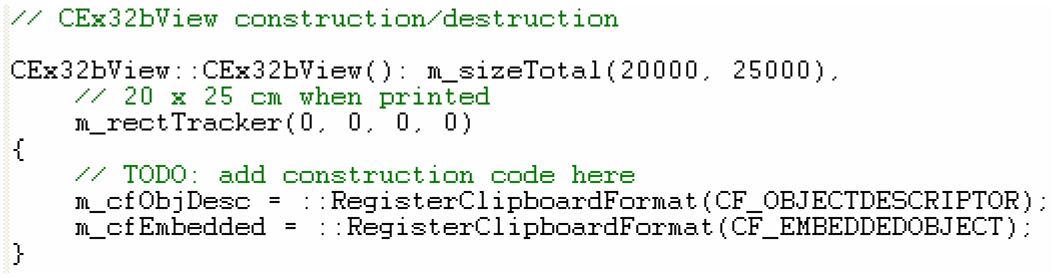


```
#include "stdafx.h"
#include "ex32b.h"

#include <afxole.h>

#include "ex32bDoc.h"
#include "ex32bView.h"

#ifdef _DEBUG
```

Listing 7.

Modify the constructor as shown below.

```
CEx28bView::CEx28bView(): m_sizeTotal(20000, 25000),
    // 20 x 25 cm when printed
      m_rectTracker(0, 0, 0, 0)
{
      m_cfObjDesc = ::RegisterClipboardFormat(CF_OBJECTDESCRIPTOR);
      m_cfEmbedded = ::RegisterClipboardFormat(CF_EMBEDDEDOBJECT);
}
```

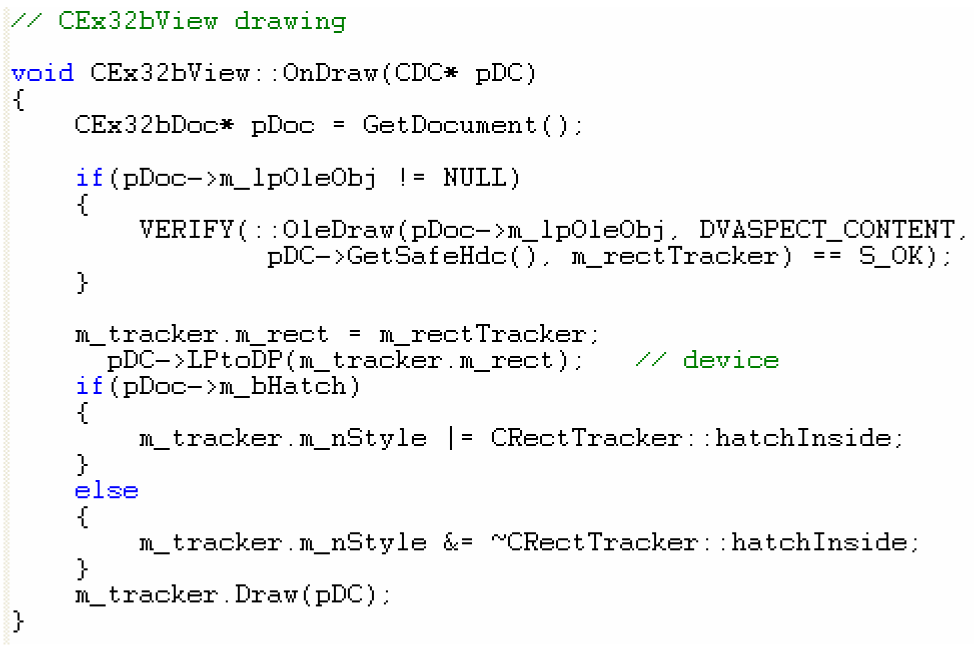


```
// CEx32bView construction/destruction

CEx32bView::CEx32bView(): m_sizeTotal(20000, 25000),
    // 20 x 25 cm when printed
    m_rectTracker(0, 0, 0, 0)
{
    // TODO: add construction code here
    m_cfObjDesc = ::RegisterClipboardFormat(CF_OBJECTDESCRIPTOR);
    m_cfEmbedded = ::RegisterClipboardFormat(CF_EMBEDDEDOBJECT);
}
```

Listing 8.

Modify the `OnDraw()` as shown below.

```
void CEx32bView::OnDraw(CDC* pDC)
{
      CEx32bDoc* pDoc = GetDocument();

      if(pDoc->m_lpOleObj != NULL)
      {
            VERIFY(::OleDraw(pDoc->m_lpOleObj, DVASPECT_CONTENT,
                        pDC->GetSafeHdc(), m_rectTracker) == S_OK);
      }

      m_tracker.m_rect = m_rectTracker;
      pDC->LPtoDP(m_tracker.m_rect);   // device
      if(pDoc->m_bHatch)
      {
            m_tracker.m_nStyle |= CRectTracker::hatchInside;
      }
      else
      {
            m_tracker.m_nStyle &= ~CRectTracker::hatchInside;
      }
      m_tracker.Draw(pDC);
}
```

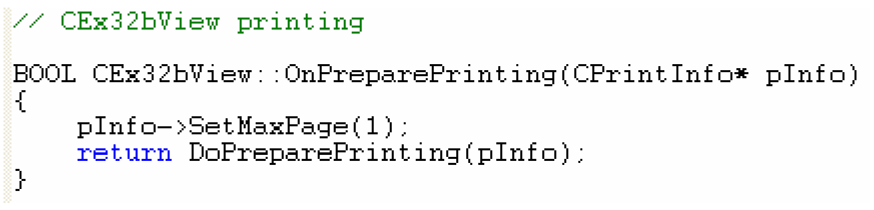


```
// CEx32bView drawing

void CEx32bView::OnDraw(CDC* pDC)
{
    CEx32bDoc* pDoc = GetDocument();

    if(pDoc->m_lpOleObj != NULL)
    {
        VERIFY(::OleDraw(pDoc->m_lpOleObj, DVASPECT_CONTENT,
                pDC->GetSafeHdc(), m_rectTracker) == S_OK);
    }

    m_tracker.m_rect = m_rectTracker;
      pDC->LPtoDP(m_tracker.m_rect);   // device
    if(pDoc->m_bHatch)
    {
        m_tracker.m_nStyle |= CRectTracker::hatchInside;
    }
    else
    {
        m_tracker.m_nStyle &= ~CRectTracker::hatchInside;
    }
    m_tracker.Draw(pDC);
}
```

Listing 9.

Edit `OnPreparePrinting()` as shown below.

```
BOOL CEx32bView::OnPreparePrinting(CPrintInfo* pInfo)
{
      pInfo->SetMaxPage(1);
      return DoPreparePrinting(pInfo);
}
```

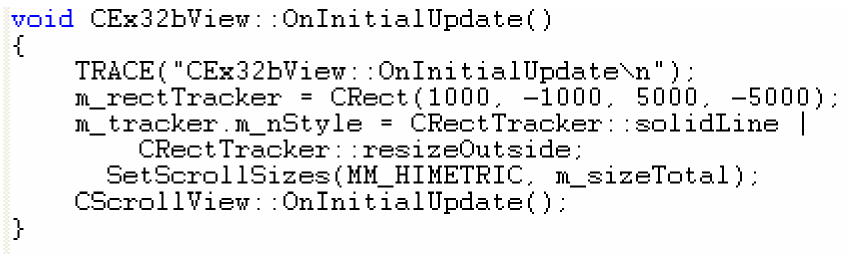


```
// CEx32bView printing

BOOL CEx32bView::OnPreparePrinting(CPrintInfo* pInfo)
{
    pInfo->SetMaxPage(1);
    return DoPreparePrinting(pInfo);
}
```

Listing 10.

Edit `OnInitialUpdate()` as shown below.

```
void CEx32bView::OnInitialUpdate()
{
      TRACE("CEx32bView::OnInitialUpdate\n");
      m_rectTracker = CRect(1000, -1000, 5000, -5000);
      m_tracker.m_nStyle = CRectTracker::solidLine |
            CRectTracker::resizeOutside;
      SetScrollSizes(MM_HIMETRIC, m_sizeTotal);
      CScrollView::OnInitialUpdate();
}
```

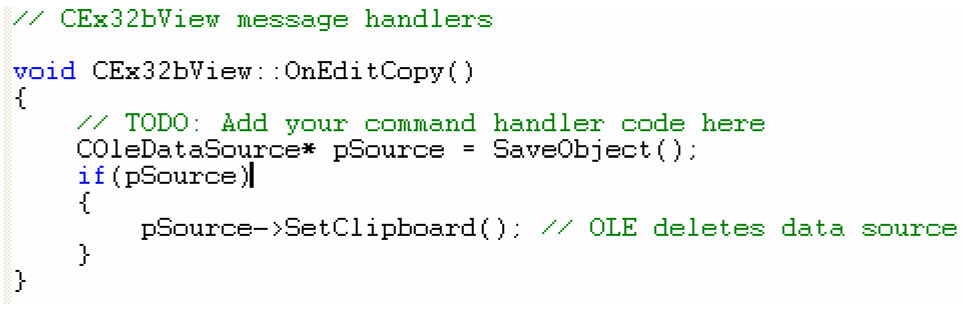


```
void CEx32bView::OnInitialUpdate()
{
    TRACE("CEx32bView::OnInitialUpdate\n");
    m_rectTracker = CRect(1000, -1000, 5000, -5000);
    m_tracker.m_nStyle = CRectTracker::solidLine |
        CRectTracker::resizeOutside;
      SetScrollSizes(MM_HIMETRIC, m_sizeTotal);
    CScrollView::OnInitialUpdate();
}
```

Listing 11.

Edit other message handlers as shown in the following codes.

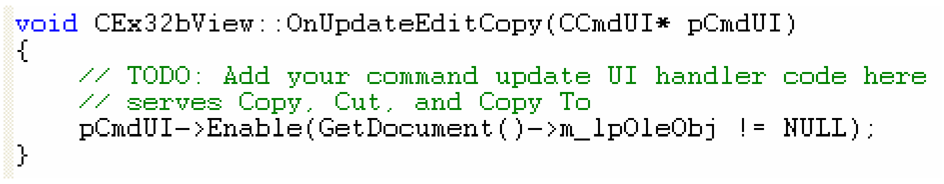
```
void CEx32bView::OnEditCopy()
{
      // TODO: Add your command handler code here
      COleDataSource* pSource = SaveObject();
      if(pSource)
    {
      pSource->SetClipboard(); // OLE deletes data source
    }
}
```



```
// CEx32bView message handlers

void CEx32bView::OnEditCopy()
{
    // TODO: Add your command handler code here
    COleDataSource* pSource = SaveObject();
    if(pSource)
    {
        pSource->SetClipboard(); // OLE deletes data source
    }
}
```

Listing 12.

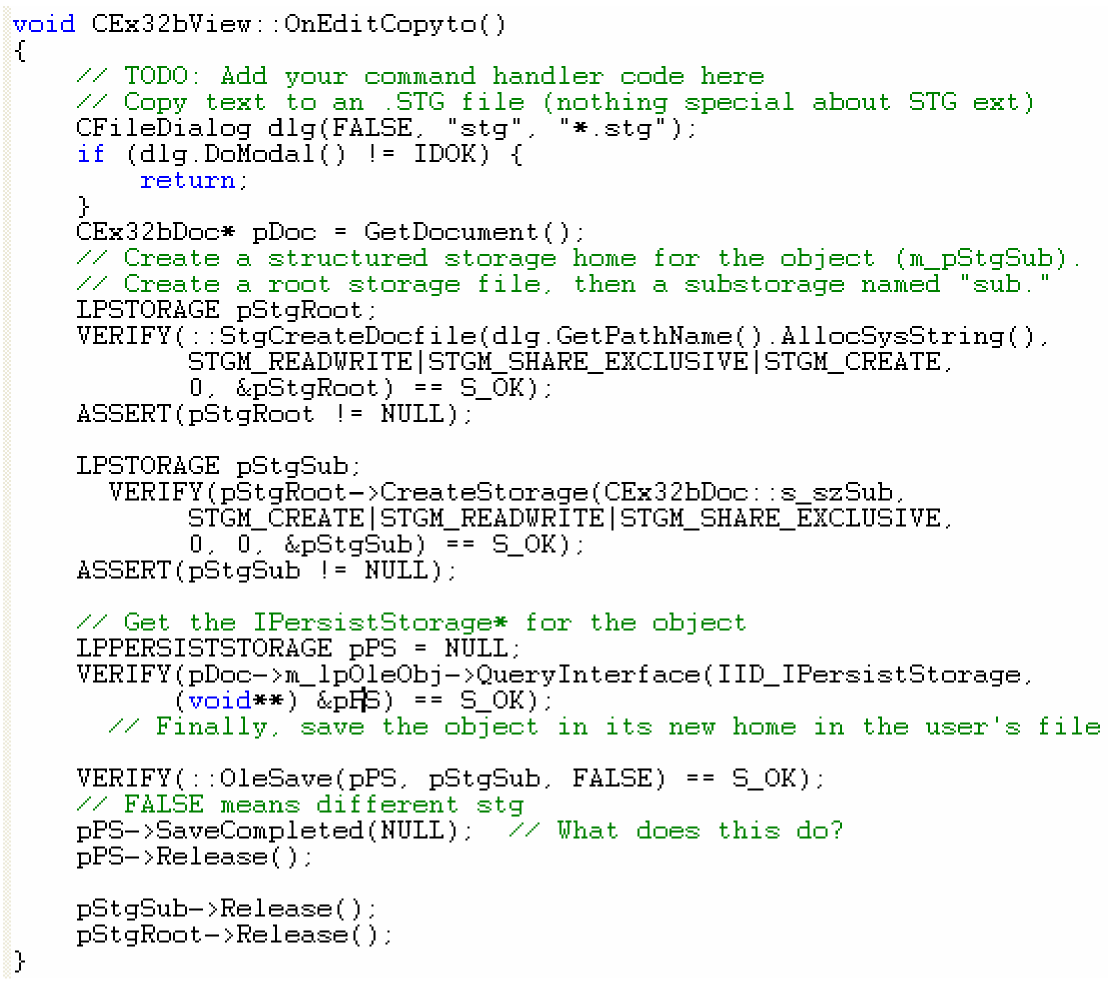
```
void CEx32bView::OnUpdateEditCopy(CCmdUI* pCmdUI)
{
      // TODO: Add your command update UI handler code here
      // serves Copy, Cut, and Copy To
    pCmdUI->Enable(GetDocument()->m_lpOleObj != NULL);
}
```



```
void CEx32bView::OnUpdateEditCopy(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // serves Copy, Cut, and Copy To
    pCmdUI->Enable(GetDocument()->m_lpOleObj != NULL);
}
```

Listing 13.

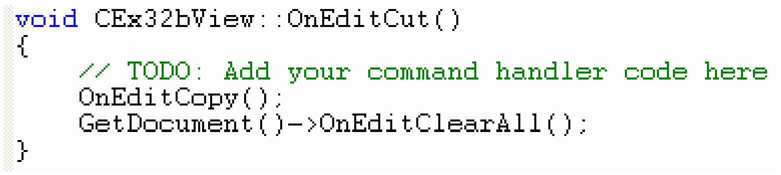
```
void CEx32bView::OnEditCopyto()
{
    // TODO: Add your command handler code here
    // Copy text to an .STG file (nothing special about STG ext)
    CFileDialog dlg(FALSE, "stg", "*.stg");
    if (dlg.DoModal() != IDOK) {
        return;
    }
      CEx32bDoc* pDoc = GetDocument();
      // Create a structured storage home for the object (m_pStgSub).
      // Create a root storage file, then a substorage named "sub."
      LPSTORAGE pStgRoot;
      VERIFY(::StgCreateDocfile(dlg.GetPathName().AllocSysString(),
               STGM_READWRITE|STGM_SHARE_EXCLUSIVE|STGM_CREATE,
               0, &pStgRoot) == S_OK);
      ASSERT(pStgRoot != NULL);

      LPSTORAGE pStgSub;
      VERIFY(pStgRoot->CreateStorage(CEx32bDoc::s_szSub,
         STGM_CREATE|STGM_READWRITE|STGM_SHARE_EXCLUSIVE,
         0, 0, &pStgSub) == S_OK);
      ASSERT(pStgSub != NULL);

      // Get the IPersistStorage* for the object
      LPPERSISTSTORAGE pPS = NULL;
      VERIFY(pDoc->m_lpOleObj->QueryInterface(IID_IPersistStorage,
              (void**) &pPS) == S_OK);
      // Finally, save the object in its new home in the user's file

      VERIFY(::OleSave(pPS, pStgSub, FALSE) == S_OK);
      // FALSE means different stg
      pPS->SaveCompleted(NULL);  // What does this do?
      pPS->Release();

      pStgSub->Release();
      pStgRoot->Release();
}
```



```
void CEx32bView::OnEditCopyto()
{
    // TODO: Add your command handler code here
    // Copy text to an .STG file (nothing special about STG ext)
    CFileDialog dlg(FALSE, "stg", "*.stg");
    if (dlg.DoModal() != IDOK) {
        return;
    }
    CEx32bDoc* pDoc = GetDocument();
    // Create a structured storage home for the object (m_pStgSub).
    // Create a root storage file, then a substorage named "sub."
    LPSTORAGE pStgRoot;
    VERIFY(::StgCreateDocfile(dlg.GetPathName().AllocSysString(),
           STGM_READWRITE|STGM_SHARE_EXCLUSIVE|STGM_CREATE,
           0, &pStgRoot) == S_OK);
    ASSERT(pStgRoot != NULL);

    LPSTORAGE pStgSub;
      VERIFY(pStgRoot->CreateStorage(CEx32bDoc::s_szSub,
           STGM_CREATE|STGM_READWRITE|STGM_SHARE_EXCLUSIVE,
           0, 0, &pStgSub) == S_OK);
    ASSERT(pStgSub != NULL);

    // Get the IPersistStorage* for the object
    LPPERSISTSTORAGE pPS = NULL;
    VERIFY(pDoc->m_lpOleObj->QueryInterface(IID_IPersistStorage,
          (void**) &pPS) == S_OK);
      // Finally, save the object in its new home in the user's file

    VERIFY(::OleSave(pPS, pStgSub, FALSE) == S_OK);
    // FALSE means different stg
    pPS->SaveCompleted(NULL);  // What does this do?
    pPS->Release();

    pStgSub->Release();
    pStgRoot->Release();
}
```

Listing 14.

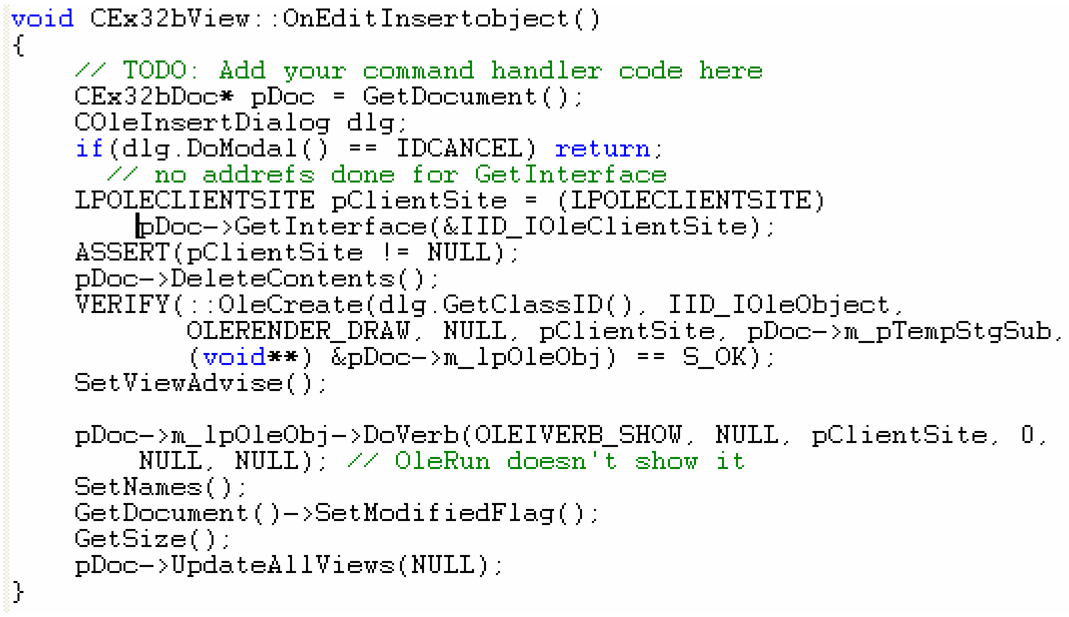
```
void CEx32bView::OnEditCut()
{
    // TODO: Add your command handler code here
    OnEditCopy();
    GetDocument()->OnEditClearAll();
}
```



```
void CEx32bView::OnEditCut()
{
    // TODO: Add your command handler code here
    OnEditCopy();
    GetDocument()->OnEditClearAll();
}
```

Listing 15.

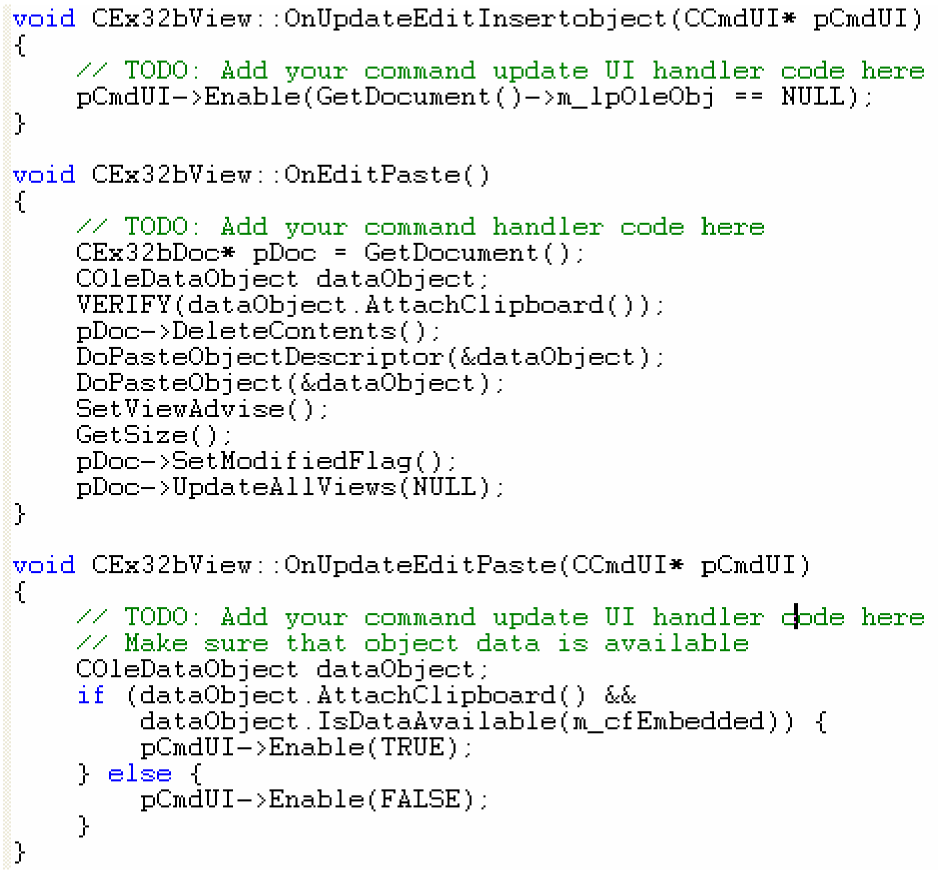
```
void CEx32bView::OnEditInsertobject()
{
      // TODO: Add your command handler code here
      CEx32bDoc* pDoc = GetDocument();
      COleInsertDialog dlg;
      if(dlg.DoModal() == IDCANCEL) return;
      // no addrefs done for GetInterface
      LPOLECLIENTSITE pClientSite = (LPOLECLIENTSITE)pDoc-
>GetInterface(&IID_IOleClientSite);
      ASSERT(pClientSite != NULL);
      pDoc->DeleteContents();
      VERIFY(::OleCreate(dlg.GetClassID(), IID_IOleObject,
               OLERENDER_DRAW, NULL, pClientSite, pDoc->m_pTempStgSub,
      (void**) &pDoc->m_lpOleObj) == S_OK);
      SetViewAdvise();

      pDoc->m_lpOleObj->DoVerb(OLEIVERB_SHOW, NULL, pClientSite, 0,
            NULL, NULL); // OleRun doesn't show it
      SetNames();
      GetDocument()->SetModifiedFlag();
      GetSize();
      pDoc->UpdateAllViews(NULL);
}
```



```
void CEx32bView::OnEditInsertobject()
{
	// TODO: Add your command handler code here
	CEx32bDoc* pDoc = GetDocument();
	COleInsertDialog dlg;
	if(dlg.DoModal() == IDCANCEL) return;
	  // no addrefs done for GetInterface
	LPOLECLIENTSITE pClientSite = (LPOLECLIENTSITE)
	    pDoc->GetInterface(&IID_IOleClientSite);
	ASSERT(pClientSite != NULL);
	pDoc->DeleteContents();
	VERIFY(::OleCreate(dlg.GetClassID(), IID_IOleObject,
	       OLERENDER_DRAW, NULL, pClientSite, pDoc->m_pTempStgSub,
	       (void**) &pDoc->m_lpOleObj) == S_OK);
	SetViewAdvise();

	pDoc->m_lpOleObj->DoVerb(OLEIVERB_SHOW, NULL, pClientSite, 0,
	    NULL, NULL); // OleRun doesn't show it
	SetNames();
	GetDocument()->SetModifiedFlag();
	GetSize();
	pDoc->UpdateAllViews(NULL);
}
```

Listing 16.

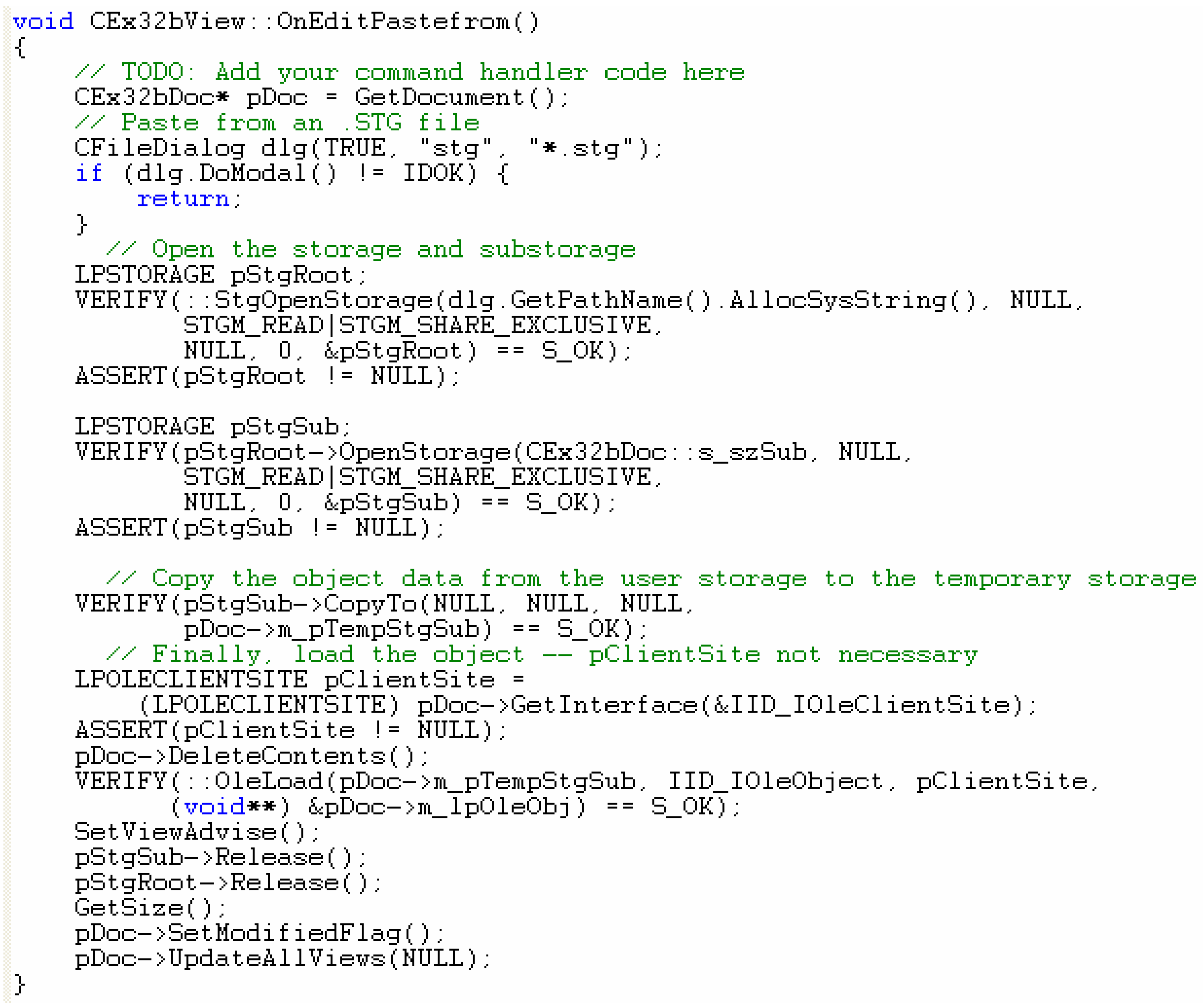
```
void CEx32bView::OnUpdateEditInsertobject(CCmdUI* pCmdUI)
{
	// TODO: Add your command update UI handler code here
	pCmdUI->Enable(GetDocument()->m_lpOleObj == NULL);
}

void CEx32bView::OnEditPaste()
{
	// TODO: Add your command handler code here
	CEx32bDoc* pDoc = GetDocument();
	COleDataObject dataObject;
	VERIFY(dataObject.AttachClipboard());
	pDoc->DeleteContents();
	DoPasteObjectDescriptor(&dataObject);
	DoPasteObject(&dataObject);
	SetViewAdvise();
	GetSize();
	pDoc->SetModifiedFlag();
	pDoc->UpdateAllViews(NULL);
}

void CEx32bView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
	// TODO: Add your command update UI handler code here
	// Make sure that object data is available
	COleDataObject dataObject;
	if (dataObject.AttachClipboard() &&
dataObject.IsDataAvailable(m_cfEmbedded))
	 { pCmdUI->Enable(TRUE); }
    else
    { pCmdUI->Enable(FALSE); }
}
```



```
void CEx32bView::OnUpdateEditInsertobject(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(GetDocument()->m_lpOleObj == NULL);
}

void CEx32bView::OnEditPaste()
{
    // TODO: Add your command handler code here
    CEx32bDoc* pDoc = GetDocument();
    COleDataObject dataObject;
    VERIFY(dataObject.AttachClipboard());
    pDoc->DeleteContents();
    DoPasteObjectDescriptor(&dataObject);
    DoPasteObject(&dataObject);
    SetViewAdvise();
    GetSize();
    pDoc->SetModifiedFlag();
    pDoc->UpdateAllViews(NULL);
}

void CEx32bView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // Make sure that object data is available
    COleDataObject dataObject;
    if (dataObject.AttachClipboard() &&
        dataObject.IsDataAvailable(m_cfEmbedded)) {
        pCmdUI->Enable(TRUE);
    } else {
        pCmdUI->Enable(FALSE);
    }
}
```

Listing 17.

```
void CEx32bView::OnEditPastefrom()
{
    // TODO: Add your command handler code here
    CEx32bDoc* pDoc = GetDocument();
    // Paste from an .STG file
    CFileDialog dlg(TRUE, "stg", "*.stg");
    if (dlg.DoModal() != IDOK)
    { return; }
    // Open the storage and substorage
      LPSTORAGE pStgRoot;
      VERIFY(::StgOpenStorage(dlg.GetPathName().AllocSysString(), NULL,
               STGM_READ|STGM_SHARE_EXCLUSIVE,
               NULL, 0, &pStgRoot) == S_OK);
      ASSERT(pStgRoot != NULL);

      LPSTORAGE pStgSub;
      VERIFY(pStgRoot->OpenStorage(CEx32bDoc::s_szSub, NULL,
               STGM_READ|STGM_SHARE_EXCLUSIVE,
               NULL, 0, &pStgSub) == S_OK);
      ASSERT(pStgSub != NULL);

      // Copy the object data from the user storage to the temporary storage
      VERIFY(pStgSub->CopyTo(NULL, NULL, NULL,
               pDoc->m_pTempStgSub) == S_OK);
      // Finally, load the object -- pClientSite not necessary
      LPOLECLIENTSITE pClientSite =
            (LPOLECLIENTSITE) pDoc->GetInterface(&IID_IOleClientSite);
      ASSERT(pClientSite != NULL);
      pDoc->DeleteContents();
      VERIFY(::OleLoad(pDoc->m_pTempStgSub, IID_IOleObject, pClientSite,
              (void**) &pDoc->m_lpOleObj) == S_OK);
      SetViewAdvise();
      pStgSub->Release();
      pStgRoot->Release();
      GetSize();
    pDoc->SetModifiedFlag();
    pDoc->UpdateAllViews(NULL);
}
```

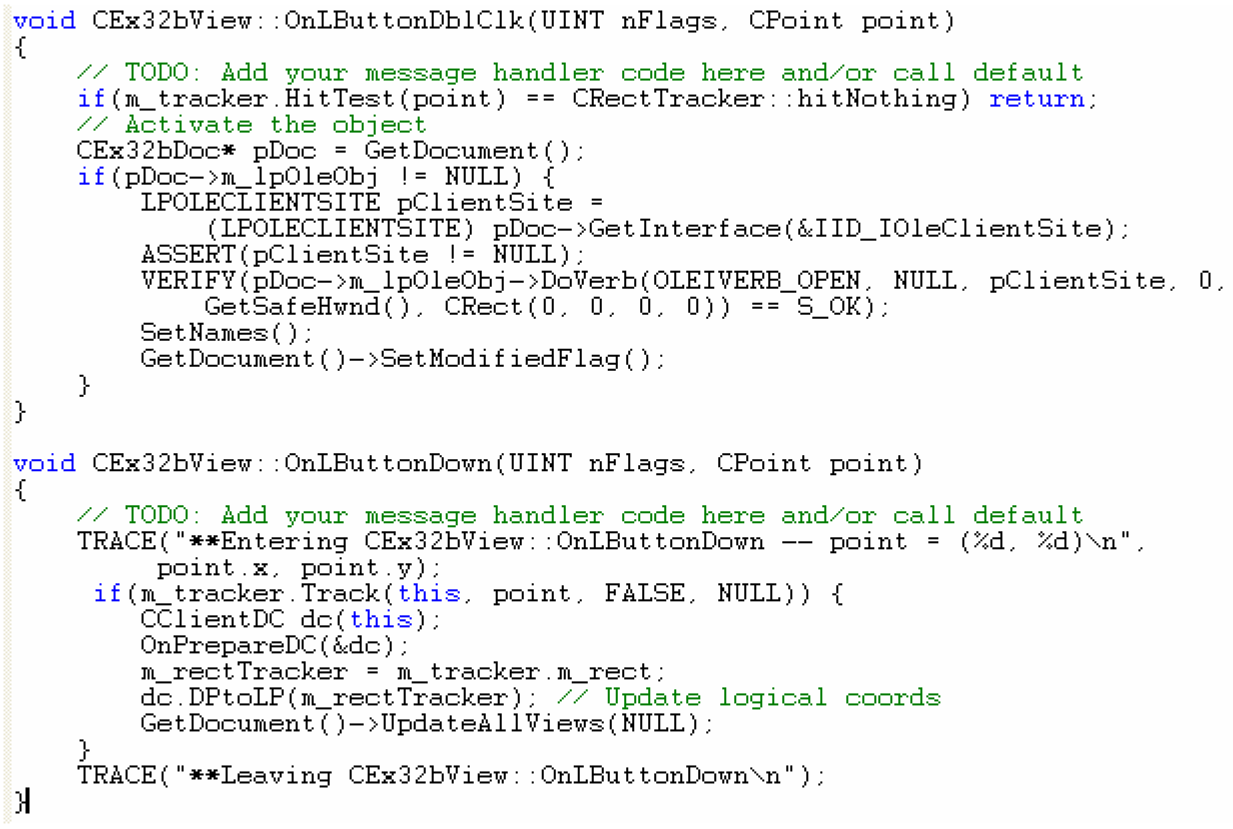


```
void CEx32bView::OnEditPastefrom()
{
    // TODO: Add your command handler code here
    CEx32bDoc* pDoc = GetDocument();
    // Paste from an .STG file
    CFileDialog dlg(TRUE, "stg", "*.stg");
    if (dlg.DoModal() != IDOK) {
        return;
    }
      // Open the storage and substorage
    LPSTORAGE pStgRoot;
    VERIFY(::StgOpenStorage(dlg.GetPathName().AllocSysString(), NULL,
           STGM_READ|STGM_SHARE_EXCLUSIVE,
           NULL, 0, &pStgRoot) == S_OK);
    ASSERT(pStgRoot != NULL);

    LPSTORAGE pStgSub;
    VERIFY(pStgRoot->OpenStorage(CEx32bDoc::s_szSub, NULL,
           STGM_READ|STGM_SHARE_EXCLUSIVE,
           NULL, 0, &pStgSub) == S_OK);
    ASSERT(pStgSub != NULL);

      // Copy the object data from the user storage to the temporary storage
    VERIFY(pStgSub->CopyTo(NULL, NULL, NULL,
           pDoc->m_pTempStgSub) == S_OK);
      // Finally, load the object -- pClientSite not necessary
    LPOLECLIENTSITE pClientSite =
        (LPOLECLIENTSITE) pDoc->GetInterface(&IID_IOleClientSite);
    ASSERT(pClientSite != NULL);
    pDoc->DeleteContents();
    VERIFY(::OleLoad(pDoc->m_pTempStgSub, IID_IOleObject, pClientSite,
          (void**) &pDoc->m_lpOleObj) == S_OK);
    SetViewAdvise();
    pStgSub->Release();
    pStgRoot->Release();
    GetSize();
    pDoc->SetModifiedFlag();
    pDoc->UpdateAllViews(NULL);
}
```

Listing 18.

```
void CEx32bView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    if(m_tracker.HitTest(point) == CRectTracker::hitNothing) return;
    // Activate the object
    CEx32bDoc* pDoc = GetDocument();
    if(pDoc->m_lpOleObj != NULL)
    {
      LPOLECLIENTSITE pClientSite =
            (LPOLECLIENTSITE) pDoc->GetInterface(&IID_IOleClientSite);
      ASSERT(pClientSite != NULL);
      VERIFY(pDoc->m_lpOleObj->DoVerb(OLEIVERB_OPEN, NULL, pClientSite, 0,
                    GetSafeHwnd(), CRect(0, 0, 0, 0)) == S_OK);
      SetNames();
```

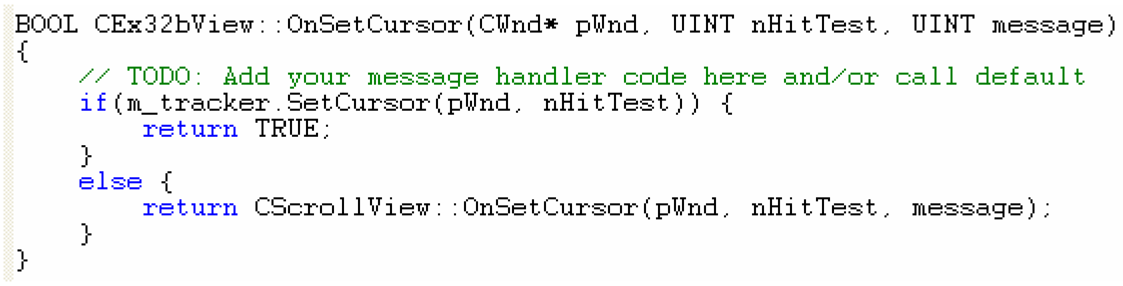
```
        GetDocument()->SetModifiedFlag();
    }
}

void CEx32bView::OnLButtonDown(UINT nFlags, CPoint point)
{
        // TODO: Add your message handler code here and/or call default
        TRACE("**Entering CEx32bView::OnLButtonDown -- point = (%d, %d)\n",
                point.x, point.y);
         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 coords
                GetDocument()->UpdateAllViews(NULL);
        }
        TRACE("**Leaving CEx32bView::OnLButtonDown\n");
}
```



```
void CEx32bView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    if(m_tracker.HitTest(point) == CRectTracker::hitNothing) return;
    // Activate the object
    CEx32bDoc* pDoc = GetDocument();
    if(pDoc->m_lpOleObj != NULL) {
        LPOLECLIENTSITE pClientSite =
            (LPOLECLIENTSITE) pDoc->GetInterface(&IID_IOleClientSite);
        ASSERT(pClientSite != NULL);
        VERIFY(pDoc->m_lpOleObj->DoVerb(OLEIVERB_OPEN, NULL, pClientSite, 0,
            GetSafeHwnd(), CRect(0, 0, 0, 0)) == S_OK);
        SetNames();
        GetDocument()->SetModifiedFlag();
    }
}

void CEx32bView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    TRACE("**Entering CEx32bView::OnLButtonDown -- point = (%d, %d)\n",
          point.x, point.y);
     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 coords
        GetDocument()->UpdateAllViews(NULL);
    }
    TRACE("**Leaving CEx32bView::OnLButtonDown\n");
}
```

Listing 19.

```
BOOL CEx32bView::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);
        }
}
```

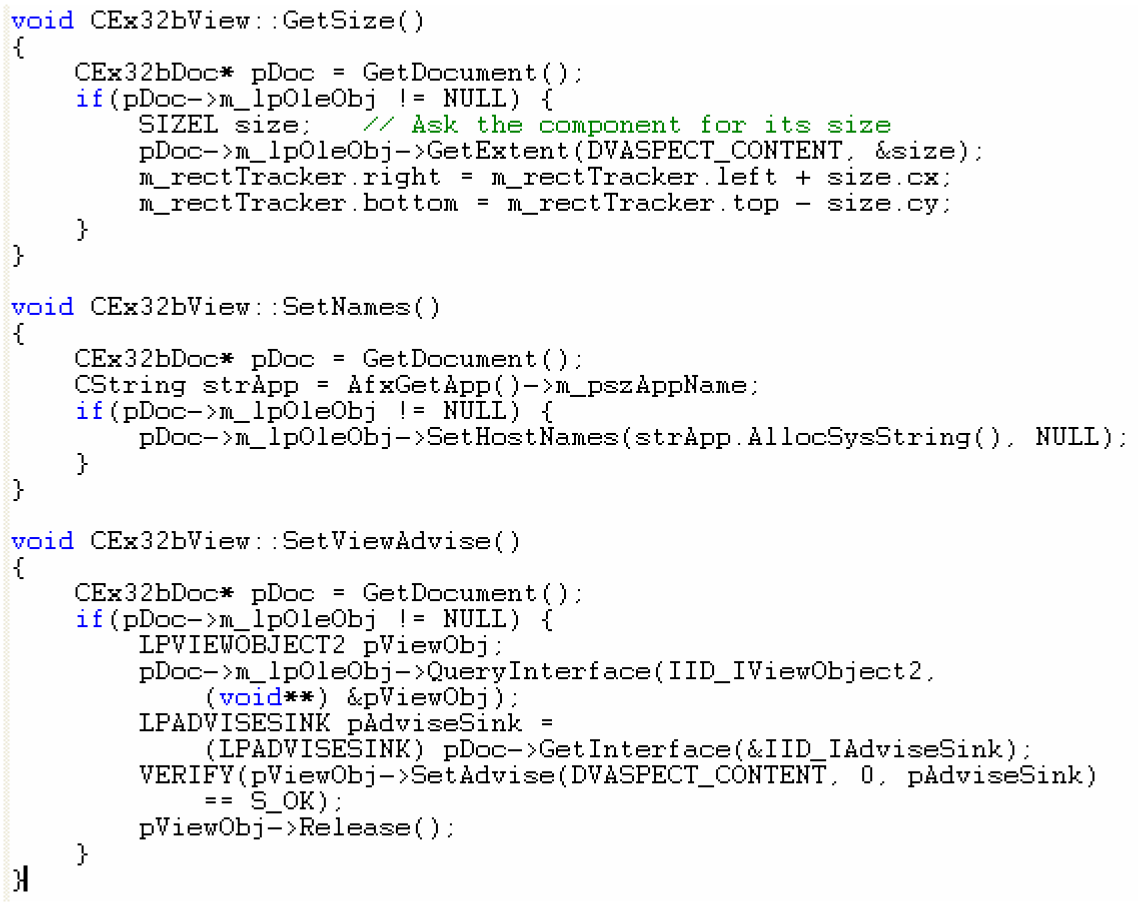


```
BOOL CEx32bView::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
{
    // TODO: Add your message handler code here and/or call default
    if(m_tracker.SetCursor(pWnd, nHitTest)) {
        return TRUE;
    }
    else {
        return CScrollView::OnSetCursor(pWnd, nHitTest, message);
    }
}
```

Listing 20.

Complete other message handlers' codes previously added.

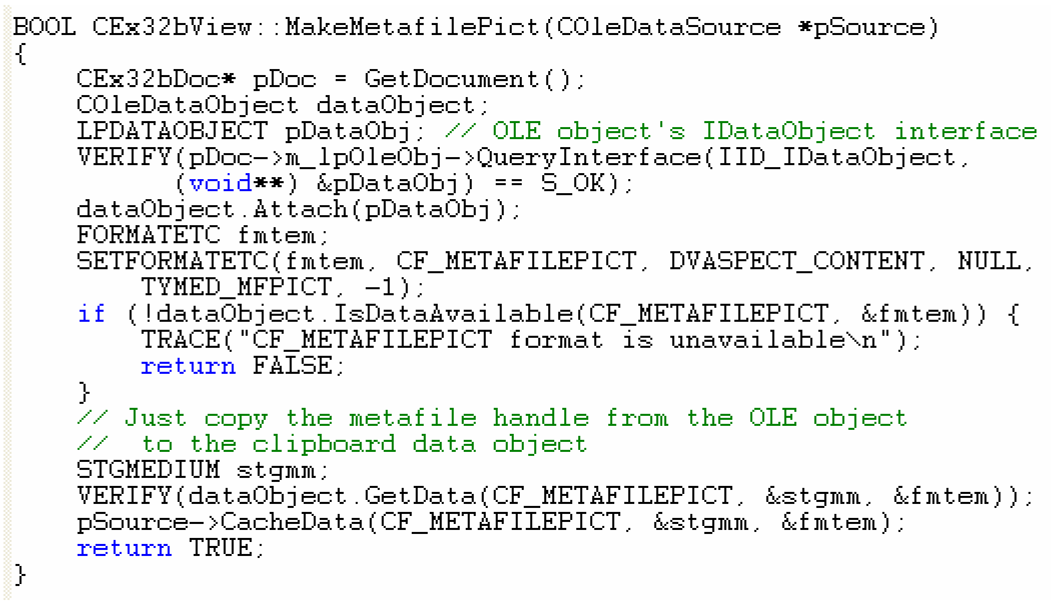
```
void CEx32bView::GetSize()
{
      CEx32bDoc* pDoc = GetDocument();
      if(pDoc->m_lpOleObj != NULL)
      {
            SIZEL size;   // Ask the component for its size
            pDoc->m_lpOleObj->GetExtent(DVASPECT_CONTENT, &size);
            m_rectTracker.right = m_rectTracker.left + size.cx;
            m_rectTracker.bottom = m_rectTracker.top - size.cy;
      }
}

void CEx32bView::SetNames()
{
      CEx32bDoc* pDoc = GetDocument();
      CString strApp = AfxGetApp()->m_pszAppName;
      if(pDoc->m_lpOleObj != NULL)
      { pDoc->m_lpOleObj->SetHostNames(strApp.AllocSysString(), NULL); }
}

void CEx32bView::SetViewAdvise()
{
      CEx32bDoc* pDoc = GetDocument();
      if(pDoc->m_lpOleObj != NULL)
      {
            LPVIEWOBJECT2 pViewObj;
            pDoc->m_lpOleObj->QueryInterface(IID_IViewObject2,
                  (void**) &pViewObj);
            LPADVISESINK pAdviseSink =
                  (LPADVISESINK) pDoc->GetInterface(&IID_IAdviseSink);
            VERIFY(pViewObj->SetAdvise(DVASPECT_CONTENT, 0, pAdviseSink)
                  == S_OK);
            pViewObj->Release();
      }
}
```



```cpp
void CEx32bView::GetSize()
{
    CEx32bDoc* pDoc = GetDocument();
    if(pDoc->m_lpOleObj != NULL) {
        SIZEL size;   // Ask the component for its size
        pDoc->m_lpOleObj->GetExtent(DVASPECT_CONTENT, &size);
        m_rectTracker.right = m_rectTracker.left + size.cx;
        m_rectTracker.bottom = m_rectTracker.top - size.cy;
    }
}

void CEx32bView::SetNames()
{
    CEx32bDoc* pDoc = GetDocument();
    CString strApp = AfxGetApp()->m_pszAppName;
    if(pDoc->m_lpOleObj != NULL) {
        pDoc->m_lpOleObj->SetHostNames(strApp.AllocSysString(), NULL);
    }
}

void CEx32bView::SetViewAdvise()
{
    CEx32bDoc* pDoc = GetDocument();
    if(pDoc->m_lpOleObj != NULL) {
        LPVIEWOBJECT2 pViewObj;
        pDoc->m_lpOleObj->QueryInterface(IID_IViewObject2,
            (void**) &pViewObj);
        LPADVISESINK pAdviseSink =
            (LPADVISESINK) pDoc->GetInterface(&IID_IAdviseSink);
        VERIFY(pViewObj->SetAdvise(DVASPECT_CONTENT, 0, pAdviseSink)
            == S_OK);
        pViewObj->Release();
    }
}
```

Listing 21.

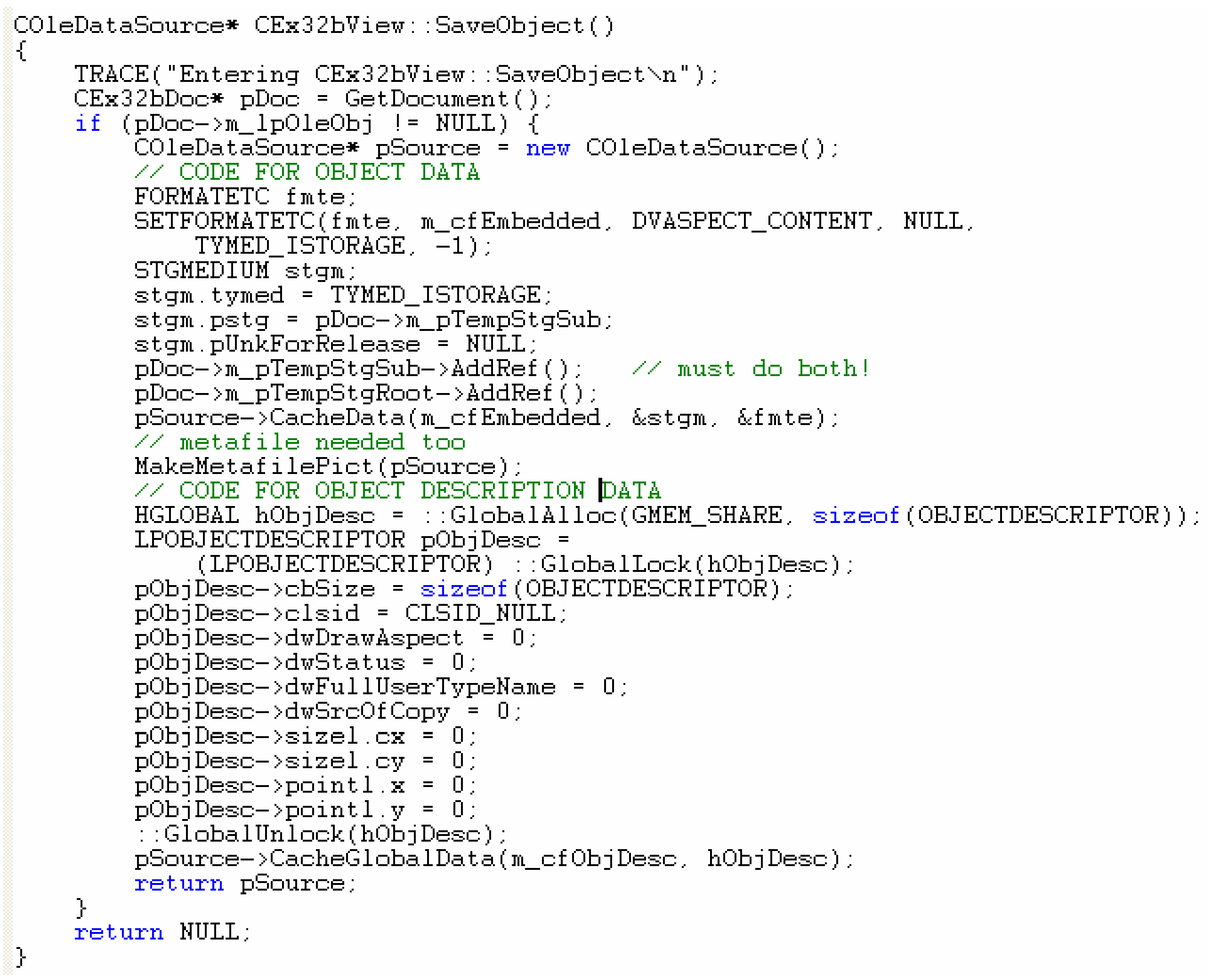
```cpp
BOOL CEx32bView::MakeMetafilePict(COleDataSource *pSource)
{
      CEx32bDoc* pDoc = GetDocument();
      COleDataObject dataObject;
      LPDATAOBJECT pDataObj; // OLE object's IDataObject interface
      VERIFY(pDoc->m_lpOleObj->QueryInterface(IID_IDataObject,
              (void**) &pDataObj) == S_OK);
      dataObject.Attach(pDataObj);
    FORMATETC fmtem;
    SETFORMATETC(fmtem, CF_METAFILEPICT, DVASPECT_CONTENT, NULL,
        TYMED_MFPICT, -1);
    if (!dataObject.IsDataAvailable(CF_METAFILEPICT, &fmtem))
    {
      TRACE("CF_METAFILEPICT format is unavailable\n");
      return FALSE;
    }
      // Just copy the metafile handle from the OLE object
      //  to the clipboard data object
      STGMEDIUM stgmm;
      VERIFY(dataObject.GetData(CF_METAFILEPICT, &stgmm, &fmtem));
      pSource->CacheData(CF_METAFILEPICT, &stgmm, &fmtem);
      return TRUE;
}
```



```
BOOL CEx32bView::MakeMetafilePict(COleDataSource *pSource)
{
    CEx32bDoc* pDoc = GetDocument();
    COleDataObject dataObject;
    LPDATAOBJECT pDataObj; // OLE object's IDataObject interface
    VERIFY(pDoc->m_lpOleObj->QueryInterface(IID_IDataObject,
          (void**) &pDataObj) == S_OK);
    dataObject.Attach(pDataObj);
    FORMATETC fmtem;
    SETFORMATETC(fmtem, CF_METAFILEPICT, DVASPECT_CONTENT, NULL,
        TYMED_MFPICT, -1);
    if (!dataObject.IsDataAvailable(CF_METAFILEPICT, &fmtem)) {
        TRACE("CF_METAFILEPICT format is unavailable\n");
        return FALSE;
    }
    // Just copy the metafile handle from the OLE object
    //  to the clipboard data object
    STGMEDIUM stgmm;
    VERIFY(dataObject.GetData(CF_METAFILEPICT, &stgmm, &fmtem));
    pSource->CacheData(CF_METAFILEPICT, &stgmm, &fmtem);
    return TRUE;
}
```

Listing 22.

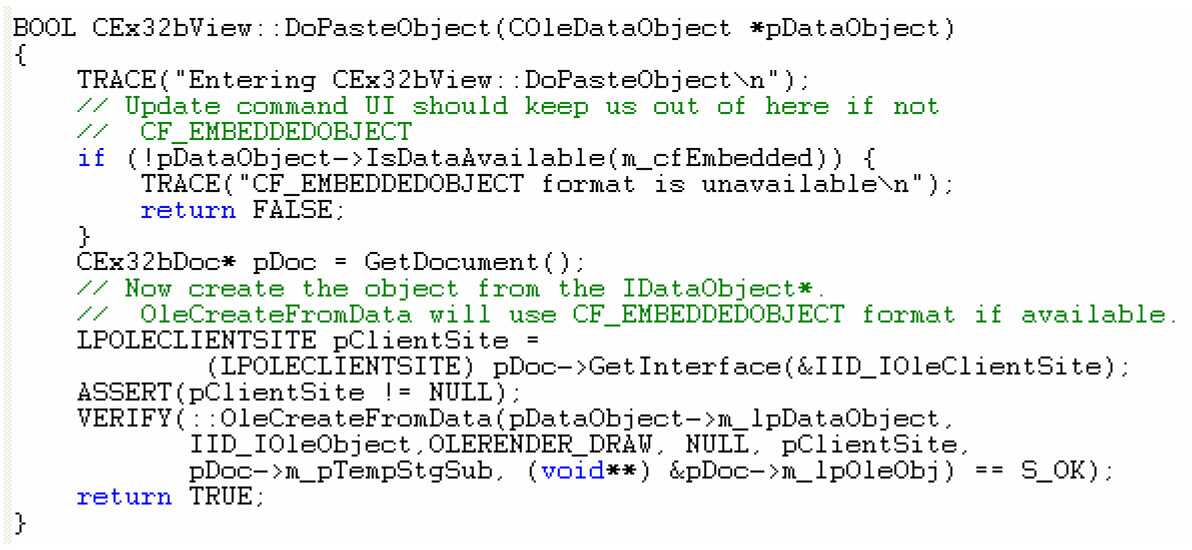
```
COleDataSource* CEx32bView::SaveObject()
{
      TRACE("Entering CEx32bView::SaveObject\n");
      CEx32bDoc* pDoc = GetDocument();
       if (pDoc->m_lpOleObj != NULL)
       {
        COleDataSource* pSource = new COleDataSource();
        // CODE FOR OBJECT DATA
        FORMATETC fmte;
        SETFORMATETC(fmte, m_cfEmbedded, DVASPECT_CONTENT, NULL,
            TYMED_ISTORAGE, -1);
        STGMEDIUM stgm;
        stgm.tymed = TYMED_ISTORAGE;
        stgm.pstg = pDoc->m_pTempStgSub;
        stgm.pUnkForRelease = NULL;
            pDoc->m_pTempStgSub->AddRef();   // must do both!
            pDoc->m_pTempStgRoot->AddRef();
            pSource->CacheData(m_cfEmbedded, &stgm, &fmte);
            // metafile needed too
            MakeMetafilePict(pSource);
            // CODE FOR OBJECT DESCRIPTION DATA
            HGLOBAL hObjDesc = ::GlobalAlloc(GMEM_SHARE,
sizeof(OBJECTDESCRIPTOR));
            LPOBJECTDESCRIPTOR pObjDesc =
                (LPOBJECTDESCRIPTOR) ::GlobalLock(hObjDesc);
            pObjDesc->cbSize = sizeof(OBJECTDESCRIPTOR);
            pObjDesc->clsid = CLSID_NULL;
            pObjDesc->dwDrawAspect = 0;
            pObjDesc->dwStatus = 0;
            pObjDesc->dwFullUserTypeName = 0;
            pObjDesc->dwSrcOfCopy = 0;
            pObjDesc->sizel.cx = 0;
            pObjDesc->sizel.cy = 0;
            pObjDesc->pointl.x = 0;
            pObjDesc->pointl.y = 0;
            ::GlobalUnlock(hObjDesc);
            pSource->CacheGlobalData(m_cfObjDesc, hObjDesc);
            return pSource;
      }
      return NULL;
}
```



```
COleDataSource* CEx32bView::SaveObject()
{
    TRACE("Entering CEx32bView::SaveObject\n");
    CEx32bDoc* pDoc = GetDocument();
    if (pDoc->m_lpOleObj != NULL) {
        COleDataSource* pSource = new COleDataSource();
        // CODE FOR OBJECT DATA
        FORMATETC fmte;
        SETFORMATETC(fmte, m_cfEmbedded, DVASPECT_CONTENT, NULL,
            TYMED_ISTORAGE, -1);
        STGMEDIUM stgm;
        stgm.tymed = TYMED_ISTORAGE;
        stgm.pstg = pDoc->m_pTempStgSub;
        stgm.pUnkForRelease = NULL;
        pDoc->m_pTempStgSub->AddRef();   // must do both!
        pDoc->m_pTempStgRoot->AddRef();
        pSource->CacheData(m_cfEmbedded, &stgm, &fmte);
        // metafile needed too
        MakeMetafilePict(pSource);
        // CODE FOR OBJECT DESCRIPTION DATA
        HGLOBAL hObjDesc = ::GlobalAlloc(GMEM_SHARE, sizeof(OBJECTDESCRIPTOR));
        LPOBJECTDESCRIPTOR pObjDesc =
            (LPOBJECTDESCRIPTOR) ::GlobalLock(hObjDesc);
        pObjDesc->cbSize = sizeof(OBJECTDESCRIPTOR);
        pObjDesc->clsid = CLSID_NULL;
        pObjDesc->dwDrawAspect = 0;
        pObjDesc->dwStatus = 0;
        pObjDesc->dwFullUserTypeName = 0;
        pObjDesc->dwSrcOfCopy = 0;
        pObjDesc->sizel.cx = 0;
        pObjDesc->sizel.cy = 0;
        pObjDesc->pointl.x = 0;
        pObjDesc->pointl.y = 0;
        ::GlobalUnlock(hObjDesc);
        pSource->CacheGlobalData(m_cfObjDesc, hObjDesc);
        return pSource;
    }
    return NULL;
}
```

Listing 23.

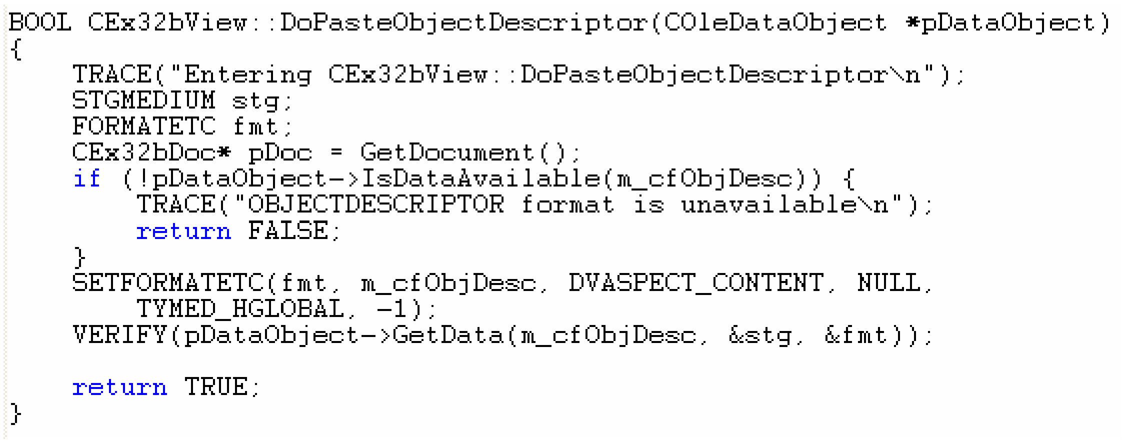
```
BOOL CEx32bView::DoPasteObject(COleDataObject *pDataObject)
{
    TRACE("Entering CEx32bView::DoPasteObject\n");
    // Update command UI should keep us out of here if not
    //  CF_EMBEDDEDOBJECT
    if (!pDataObject->IsDataAvailable(m_cfEmbedded))
    {
      TRACE("CF_EMBEDDEDOBJECT format is unavailable\n");
      return FALSE;
    }
      CEx32bDoc* pDoc = GetDocument();
      // Now create the object from the IDataObject*.
      //  OleCreateFromData will use CF_EMBEDDEDOBJECT format if available.
      LPOLECLIENTSITE pClientSite =
                  (LPOLECLIENTSITE) pDoc->GetInterface(&IID_IOleClientSite);
      ASSERT(pClientSite != NULL);
      VERIFY(::OleCreateFromData(pDataObject->m_lpDataObject,
               IID_IOleObject,OLERENDER_DRAW, NULL, pClientSite,
               pDoc->m_pTempStgSub, (void**) &pDoc->m_lpOleObj) == S_OK);
      return TRUE;
}
```



```
BOOL CEx32bView::DoPasteObject(COleDataObject *pDataObject)
{
    TRACE("Entering CEx32bView::DoPasteObject\n");
    // Update command UI should keep us out of here if not
    //  CF_EMBEDDEDOBJECT
    if (!pDataObject->IsDataAvailable(m_cfEmbedded)) {
        TRACE("CF_EMBEDDEDOBJECT format is unavailable\n");
        return FALSE;
    }
    CEx32bDoc* pDoc = GetDocument();
    // Now create the object from the IDataObject*.
    //  OleCreateFromData will use CF_EMBEDDEDOBJECT format if available.
    LPOLECLIENTSITE pClientSite =
            (LPOLECLIENTSITE) pDoc->GetInterface(&IID_IOleClientSite);
    ASSERT(pClientSite != NULL);
    VERIFY(::OleCreateFromData(pDataObject->m_lpDataObject,
           IID_IOleObject,OLERENDER_DRAW, NULL, pClientSite,
           pDoc->m_pTempStgSub, (void**) &pDoc->m_lpOleObj) == S_OK);
    return TRUE;
}
```

Listing 24.

```
BOOL CEx32bView::DoPasteObjectDescriptor(COleDataObject *pDataObject)
{
      TRACE("Entering CEx32bView::DoPasteObjectDescriptor\n");
      STGMEDIUM stg;
      FORMATETC fmt;
      CEx32bDoc* pDoc = GetDocument();
    if (!pDataObject->IsDataAvailable(m_cfObjDesc))
    {
      TRACE("OBJECTDESCRIPTOR format is unavailable\n");
      return FALSE;
    }
      SETFORMATETC(fmt, m_cfObjDesc, DVASPECT_CONTENT, NULL,
            TYMED_HGLOBAL, -1);
      VERIFY(pDataObject->GetData(m_cfObjDesc, &stg, &fmt));

      return TRUE;
}
```

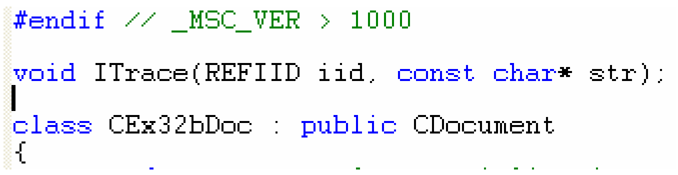


```
BOOL CEx32bView::DoPasteObjectDescriptor(COleDataObject *pDataObject)
{
    TRACE("Entering CEx32bView::DoPasteObjectDescriptor\n");
    STGMEDIUM stg;
    FORMATETC fmt;
    CEx32bDoc* pDoc = GetDocument();
    if (!pDataObject->IsDataAvailable(m_cfObjDesc)) {
        TRACE("OBJECTDESCRIPTOR format is unavailable\n");
        return FALSE;
    }
    SETFORMATETC(fmt, m_cfObjDesc, DVASPECT_CONTENT, NULL,
        TYMED_HGLOBAL, -1);
    VERIFY(pDataObject->GetData(m_cfObjDesc, &stg, &fmt));

    return TRUE;
}
```

Listing 25.

Add the following prototype in **ex32bDoc.h**.

```
void ITrace(REFIID iid, const char* str);
```

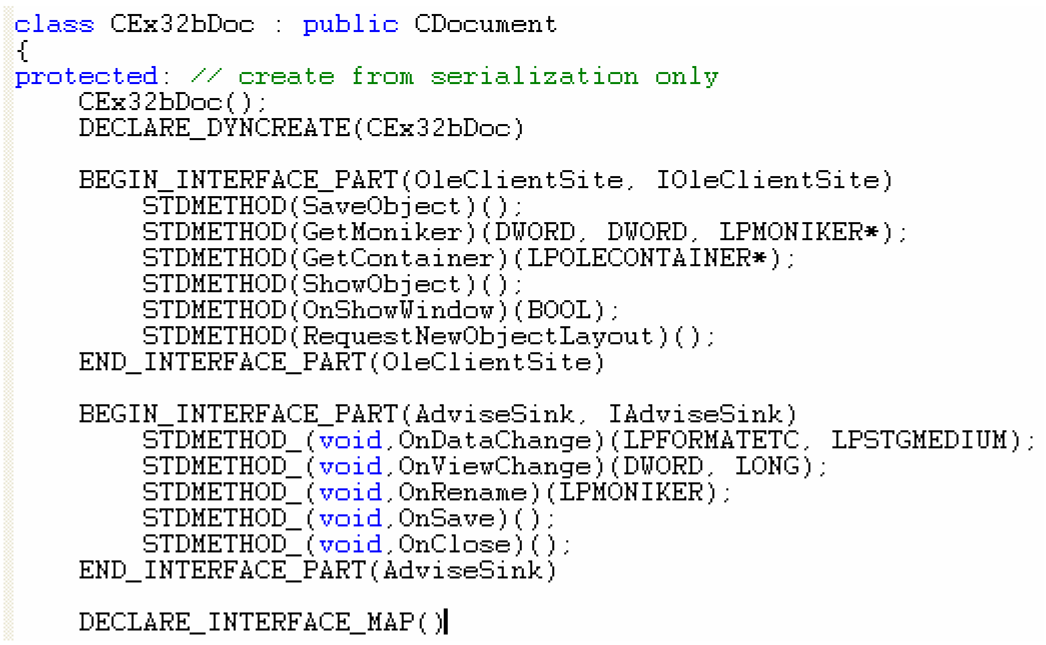


```
#endif // _MSC_VER > 1000

void ITrace(REFIID iid, const char* str);

class CEx32bDoc : public CDocument
{
```

Listing 26.

Manually add the following interface `OleClientSite` and `AdviseSink`.

```
	BEGIN_INTERFACE_PART(OleClientSite, IOleClientSite)
		STDMETHOD(SaveObject)();
		STDMETHOD(GetMoniker)(DWORD, DWORD, LPMONIKER*);
		STDMETHOD(GetContainer)(LPOLECONTAINER*);
		STDMETHOD(ShowObject)();
		STDMETHOD(OnShowWindow)(BOOL);
		STDMETHOD(RequestNewObjectLayout)();
	END_INTERFACE_PART(OleClientSite)

	BEGIN_INTERFACE_PART(AdviseSink, IAdviseSink)
		STDMETHOD_(void,OnDataChange)(LPFORMATETC, LPSTGMEDIUM);
		STDMETHOD_(void,OnViewChange)(DWORD, LONG);
		STDMETHOD_(void,OnRename)(LPMONIKER);
		STDMETHOD_(void,OnSave)();
		STDMETHOD_(void,OnClose)();
	END_INTERFACE_PART(AdviseSink)

	DECLARE_INTERFACE_MAP()
```



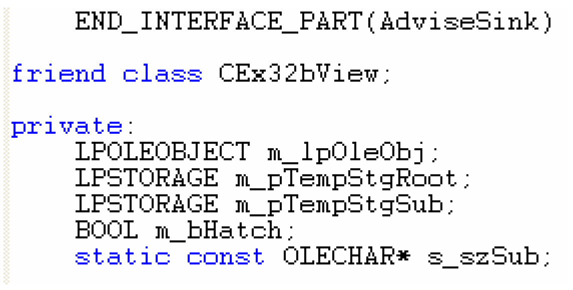
```
class CEx32bDoc : public CDocument
{
protected: // create from serialization only
    CEx32bDoc();
    DECLARE_DYNCREATE(CEx32bDoc)

    BEGIN_INTERFACE_PART(OleClientSite, IOleClientSite)
        STDMETHOD(SaveObject)();
        STDMETHOD(GetMoniker)(DWORD, DWORD, LPMONIKER*);
        STDMETHOD(GetContainer)(LPOLECONTAINER*);
        STDMETHOD(ShowObject)();
        STDMETHOD(OnShowWindow)(BOOL);
        STDMETHOD(RequestNewObjectLayout)();
    END_INTERFACE_PART(OleClientSite)

    BEGIN_INTERFACE_PART(AdviseSink, IAdviseSink)
        STDMETHOD_(void,OnDataChange)(LPFORMATETC, LPSTGMEDIUM);
        STDMETHOD_(void,OnViewChange)(DWORD, LONG);
        STDMETHOD_(void,OnRename)(LPMONIKER);
        STDMETHOD_(void,OnSave)();
        STDMETHOD_(void,OnClose)();
    END_INTERFACE_PART(AdviseSink)

    DECLARE_INTERFACE_MAP()
```

Listing 27.

Add the following friend class (in order to access the `private` variables and member functions).

```
friend class CEx32bView;

private:
	LPOLEOBJECT m_lpOleObj;
```

```
      LPSTORAGE m_pTempStgRoot;
      LPSTORAGE m_pTempStgSub;
      BOOL m_bHatch;
      static const OLECHAR* s_szSub;
```

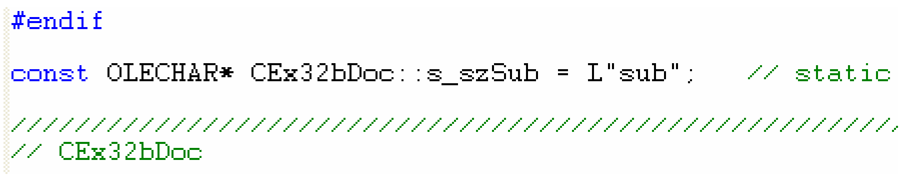


```
    END_INTERFACE_PART(AdviseSink)

friend class CEx32bView;

private:
    LPOLEOBJECT m_lpOleObj;
    LPSTORAGE m_pTempStgRoot;
    LPSTORAGE m_pTempStgSub;
    BOOL m_bHatch;
    static const OLECHAR* s_szSub;
```

Listing 28.

Add the following constant in **ex32bDoc.cpp**.

```
const OLECHAR* CEx32bDoc::s_szSub = L"sub"; // static
```

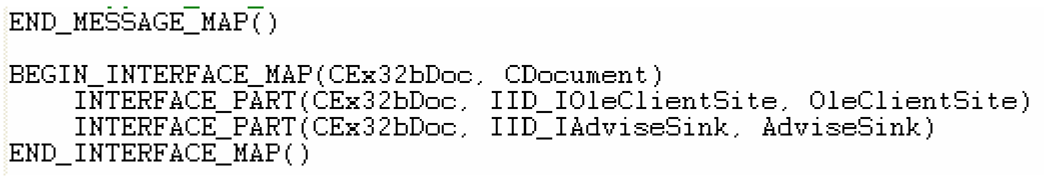


```
#endif

const OLECHAR* CEx32bDoc::s_szSub = L"sub";   // static

////////////////////////////////////////////////////////////////////
// CEx32bDoc
```

Listing 29.

Add the following interface mapping of the `OleClientSite` and `AdviseSink`.

```
BEGIN_INTERFACE_MAP(CEx32bDoc, CDocument)
      INTERFACE_PART(CEx32bDoc, IID_IOleClientSite, OleClientSite)
      INTERFACE_PART(CEx32bDoc, IID_IAdviseSink, AdviseSink)
END_INTERFACE_MAP()
```



```
END_MESSAGE_MAP()

BEGIN_INTERFACE_MAP(CEx32bDoc, CDocument)
    INTERFACE_PART(CEx32bDoc, IID_IOleClientSite, OleClientSite)
    INTERFACE_PART(CEx32bDoc, IID_IAdviseSink, AdviseSink)
END_INTERFACE_MAP()
```

Listing 30.

Then add the implementation of the `OleClientSite`.

```
////////////////////////////////////////////////////////////////////////////
// Implementation of IOleClientSite

STDMETHODIMP_(ULONG) CEx32bDoc::XOleClientSite::AddRef()
{
      TRACE("CEx32bDoc::XOleClientSite::AddRef\n");
      METHOD_PROLOGUE(CEx32bDoc, OleClientSite)
      return pThis->InternalAddRef();
}

STDMETHODIMP_(ULONG) CEx32bDoc::XOleClientSite::Release()
{
      TRACE("CEx32bDoc::XOleClientSite::Release\n");
```

```
	METHOD_PROLOGUE(CEx32bDoc, OleClientSite)
	return pThis->InternalRelease();
}

STDMETHODIMP CEx32bDoc::XOleClientSite::QueryInterface(
	REFIID iid, LPVOID* ppvObj)
{
	ITrace(iid, "CEx32bDoc::XOleClientSite::QueryInterface");
	METHOD_PROLOGUE(CEx32bDoc, OleClientSite)
	return pThis->InternalQueryInterface(&iid, ppvObj);
}

STDMETHODIMP CEx32bDoc::XOleClientSite::SaveObject()
{
	TRACE("CEx32bDoc::XOleClientSite::SaveObject\n");
	METHOD_PROLOGUE(CEx32bDoc, OleClientSite)
	ASSERT_VALID(pThis);

	LPPERSISTSTORAGE lpPersistStorage;
	pThis->m_lpOleObj->QueryInterface(IID_IPersistStorage,
		(void**) &lpPersistStorage);
	ASSERT(lpPersistStorage != NULL);
	HRESULT hr = NOERROR;
	if (lpPersistStorage->IsDirty() == NOERROR)
	{
		// NOERROR == S_OK != S_FALSE, therefore object is dirty!
		hr = ::OleSave(lpPersistStorage, pThis->m_pTempStgSub, TRUE);
		if (hr != NOERROR)
			hr = lpPersistStorage->SaveCompleted(NULL);

		// Mark the document as dirty, if save successful
		pThis->SetModifiedFlag();
	}
	lpPersistStorage->Release();
	pThis->UpdateAllViews(NULL);
	return hr;
}

STDMETHODIMP CEx32bDoc::XOleClientSite::GetMoniker(
	DWORD dwAssign, DWORD dwWhichMoniker, LPMONIKER* ppMoniker)
{
	TRACE("CEx32bDoc::XOleClientSite::GetMoniker\n");
	return E_NOTIMPL;
}

STDMETHODIMP CEx32bDoc::XOleClientSite::GetContainer(
	LPOLECONTAINER* ppContainer)
{
	TRACE("CEx32bDoc::XOleClientSite::GetContainer\n");
	return E_NOTIMPL;
}

STDMETHODIMP CEx32bDoc::XOleClientSite::ShowObject()
{
	TRACE("CEx32bDoc::XOleClientSite::ShowObject\n");
	METHOD_PROLOGUE(CEx32bDoc, OleClientSite)
	ASSERT_VALID(pThis);
	pThis->UpdateAllViews(NULL);
	return NOERROR;
}
```

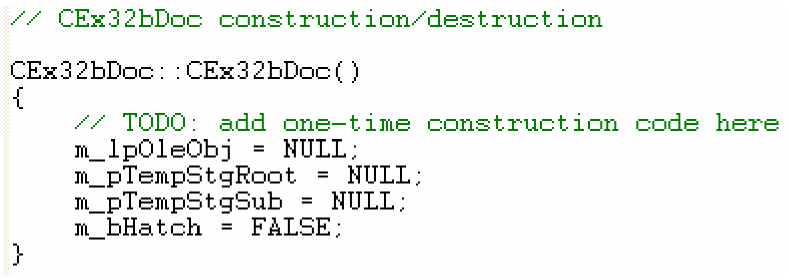
```
STDMETHODIMP CEx32bDoc::XOleClientSite::OnShowWindow(BOOL fShow)
{
      TRACE("CEx32bDoc::XOleClientSite::OnShowWindow\n");
      METHOD_PROLOGUE(CEx32bDoc, OleClientSite)
      ASSERT_VALID(pThis);
      pThis->m_bHatch = fShow;
      pThis->UpdateAllViews(NULL);
      return NOERROR;
}

STDMETHODIMP CEx32bDoc::XOleClientSite::RequestNewObjectLayout()
{
      TRACE("CEx32bDoc::XOleClientSite::RequestNewObjectLayout\n");
      return E_NOTIMPL;
}
```

And for AdviseSink.

```
//////////////////////////////////////////////////////////////////
// Implementation of IAdviseSink
STDMETHODIMP_(ULONG) CEx32bDoc::XAdviseSink::AddRef()
{
      TRACE("CEx32bDoc::XAdviseSink::AddRef\n");
      METHOD_PROLOGUE(CEx32bDoc, AdviseSink)
      return pThis->InternalAddRef();
}

STDMETHODIMP_(ULONG) CEx32bDoc::XAdviseSink::Release()
{
      TRACE("CEx32bDoc::XAdviseSink::Release\n");
      METHOD_PROLOGUE(CEx32bDoc, AdviseSink)
      return pThis->InternalRelease();
}

STDMETHODIMP CEx32bDoc::XAdviseSink::QueryInterface(
      REFIID iid, LPVOID* ppvObj)
{
      ITrace(iid, "CEx32bDoc::XAdviseSink::QueryInterface");
      METHOD_PROLOGUE(CEx32bDoc, AdviseSink)
      return pThis->InternalQueryInterface(&iid, ppvObj);
}

STDMETHODIMP_(void) CEx32bDoc::XAdviseSink::OnDataChange(
      LPFORMATETC lpFormatEtc, LPSTGMEDIUM lpStgMedium)
{
      TRACE("CEx32bDoc::XAdviseSink::OnDataChange\n");
      METHOD_PROLOGUE(CEx32bDoc, AdviseSink)
      ASSERT_VALID(pThis);

      // Interesting only for advanced containers.  Forward it such that
      //  containers do not have to implement the entire interface.
}

STDMETHODIMP_(void) CEx32bDoc::XAdviseSink::OnViewChange(
      DWORD aspects, LONG /*lindex*/)
{
      TRACE("CEx32bDoc::XAdviseSink::OnViewChange\n");
      METHOD_PROLOGUE(CEx32bDoc, AdviseSink)
      ASSERT_VALID(pThis);

      pThis->UpdateAllViews(NULL);  // the really important one
}

STDMETHODIMP_(void) CEx32bDoc::XAdviseSink::OnRename(
      LPMONIKER /*lpMoniker*/)
{
      TRACE("CEx32bDoc::XAdviseSink::OnRename\n");
      // Interesting only to the OLE link object. Containers ignore this.
}

STDMETHODIMP_(void) CEx32bDoc::XAdviseSink::OnSave()
{
      TRACE("CEx32bDoc::XAdviseSink::OnSave\n");
      METHOD_PROLOGUE(CEx32bDoc, AdviseSink)
      ASSERT_VALID(pThis);

      pThis->UpdateAllViews(NULL);
}

STDMETHODIMP_(void) CEx32bDoc::XAdviseSink::OnClose()
{
      TRACE("CEx32bDoc::XAdviseSink::OnClose\n");
      METHOD_PROLOGUE(CEx32bDoc, AdviseSink)
      ASSERT_VALID(pThis);

      pThis->UpdateAllViews(NULL);
}
```

Add/edit other codes.

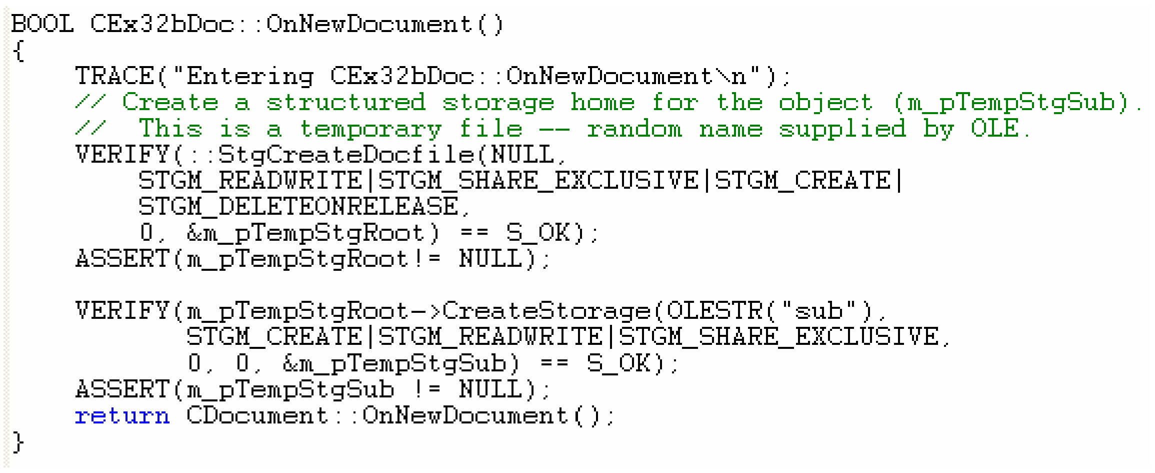
```
CEx32bDoc::CEx32bDoc()
{
      // TODO: add one-time construction code here
      m_lpOleObj = NULL;
      m_pTempStgRoot = NULL;
      m_pTempStgSub = NULL;
      m_bHatch = FALSE;
}
```



```
// CEx32bDoc construction/destruction

CEx32bDoc::CEx32bDoc()
{
    // TODO: add one-time construction code here
    m_lpOleObj = NULL;
    m_pTempStgRoot = NULL;
    m_pTempStgSub = NULL;
    m_bHatch = FALSE;
}
```

Listing 31.

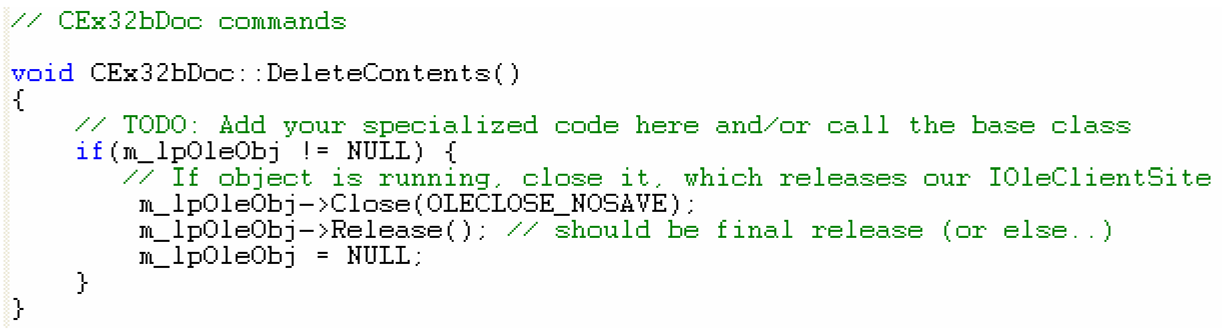
```
BOOL CEx32bDoc::OnNewDocument()
{
      TRACE("Entering CEx32bDoc::OnNewDocument\n");
      // Create a structured storage home for the object (m_pTempStgSub).
      //  This is a temporary file -- random name supplied by OLE.
      VERIFY(::StgCreateDocfile(NULL,
            STGM_READWRITE|STGM_SHARE_EXCLUSIVE|STGM_CREATE|
            STGM_DELETEONRELEASE,
            0, &m_pTempStgRoot) == S_OK);
      ASSERT(m_pTempStgRoot != NULL);
      VERIFY(m_pTempStgRoot->CreateStorage(OLESTR("sub"),
         STGM_CREATE|STGM_READWRITE|STGM_SHARE_EXCLUSIVE,
         0, 0, &m_pTempStgSub) == S_OK);
      ASSERT(m_pTempStgSub != NULL);
      return CDocument::OnNewDocument();
}
```



```
BOOL CEx32bDoc::OnNewDocument()
{
    TRACE("Entering CEx32bDoc::OnNewDocument\n");
    // Create a structured storage home for the object (m_pTempStgSub).
    //  This is a temporary file -- random name supplied by OLE.
    VERIFY(::StgCreateDocfile(NULL,
        STGM_READWRITE|STGM_SHARE_EXCLUSIVE|STGM_CREATE|
        STGM_DELETEONRELEASE,
        0, &m_pTempStgRoot) == S_OK);
    ASSERT(m_pTempStgRoot!= NULL);

    VERIFY(m_pTempStgRoot->CreateStorage(OLESTR("sub"),
           STGM_CREATE|STGM_READWRITE|STGM_SHARE_EXCLUSIVE,
           0, 0, &m_pTempStgSub) == S_OK);
    ASSERT(m_pTempStgSub != NULL);
    return CDocument::OnNewDocument();
}
```

Listing 32.

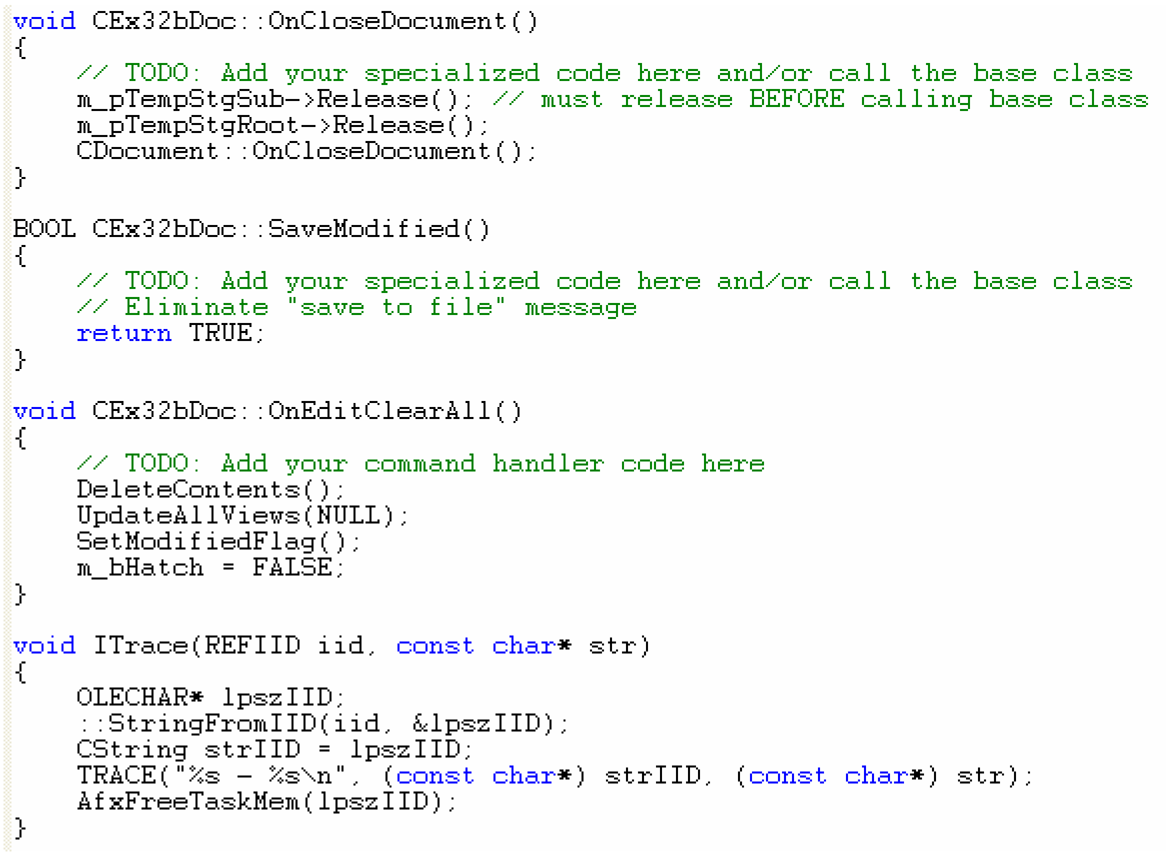
```
void CEx32bDoc::DeleteContents()
{
      // TODO: Add your specialized code here and/or call the base class
      if(m_lpOleObj != NULL) {
         // If object is running, close it, which releases our IOleClientSite
            m_lpOleObj->Close(OLECLOSE_NOSAVE);
            m_lpOleObj->Release(); // should be final release (or else..)
            m_lpOleObj = NULL;
      }
}
```



```
// CEx32bDoc commands

void CEx32bDoc::DeleteContents()
{
    // TODO: Add your specialized code here and/or call the base class
    if(m_lpOleObj != NULL) {
       // If object is running, close it, which releases our IOleClientSite
        m_lpOleObj->Close(OLECLOSE_NOSAVE);
        m_lpOleObj->Release(); // should be final release (or else..)
        m_lpOleObj = NULL;
    }
}
```

Listing 33.

```
void CEx32bDoc::OnCloseDocument()
{
      // TODO: Add your specialized code here and/or call the base class
      m_pTempStgSub->Release(); // must release BEFORE calling base class
      m_pTempStgRoot->Release();
      CDocument::OnCloseDocument();
}

BOOL CEx32bDoc::SaveModified()
{
      // TODO: Add your specialized code here and/or call the base class
      // Eliminate "save to file" message
      return TRUE;
}

void CEx32bDoc::OnEditClearAll()
{
    // TODO: Add your command handler code here
    DeleteContents();
    UpdateAllViews(NULL);
    SetModifiedFlag();
    m_bHatch = FALSE;
}

void ITrace(REFIID iid, const char* str)
{
      OLECHAR* lpszIID;
      ::StringFromIID(iid, &lpszIID);
      CString strIID = lpszIID;
      TRACE("%s - %s\n", (const char*) strIID, (const char*) str);
      AfxFreeTaskMem(lpszIID);
}
```

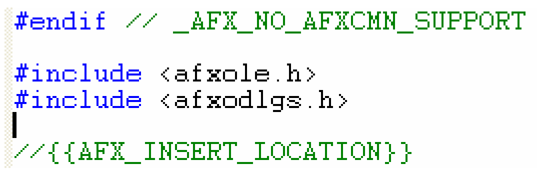


```
void CEx32bDoc::OnCloseDocument()
{
    // TODO: Add your specialized code here and/or call the base class
    m_pTempStgSub->Release(); // must release BEFORE calling base class
    m_pTempStgRoot->Release();
    CDocument::OnCloseDocument();
}

BOOL CEx32bDoc::SaveModified()
{
    // TODO: Add your specialized code here and/or call the base class
    // Eliminate "save to file" message
    return TRUE;
}

void CEx32bDoc::OnEditClearAll()
{
    // TODO: Add your command handler code here
    DeleteContents();
    UpdateAllViews(NULL);
    SetModifiedFlag();
    m_bHatch = FALSE;
}

void ITrace(REFIID iid, const char* str)
{
    OLECHAR* lpszIID;
    ::StringFromIID(iid, &lpszIID);
    CString strIID = lpszIID;
    TRACE("%s - %s\n", (const char*) strIID, (const char*) str);
    AfxFreeTaskMem(lpszIID);
}
```

Listing 34.

Add the following #include directives for automation support in **StdAfx.h**.

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

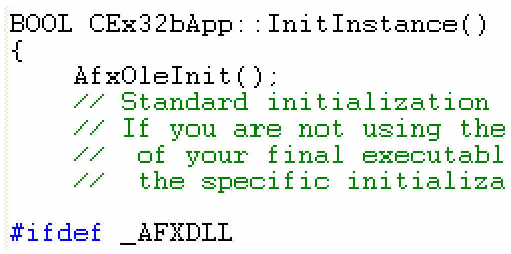


```
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxole.h>
#include <afxodlgs.h>

//{{AFX_INSERT_LOCATION}}
```

Listing 35.

Then add the following in **ex32b.cpp** at the beginning of the `InitInstance()`.

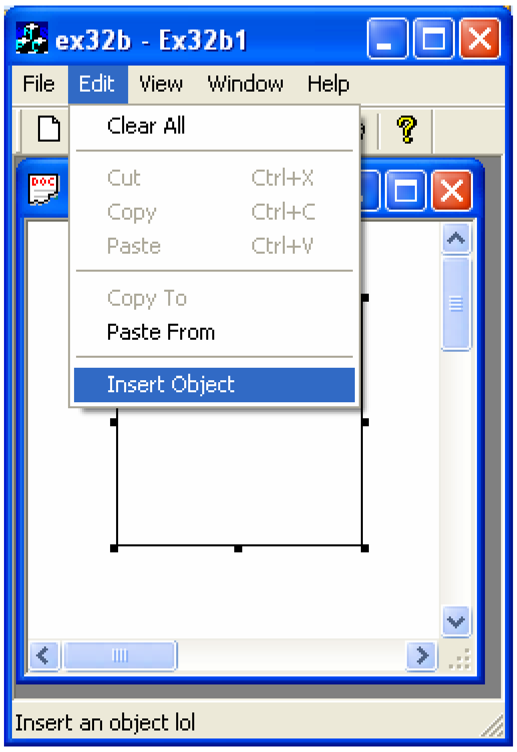
```
      AfxOleInit();
```



```
BOOL CEx32bApp::InitInstance()
{
    AfxOleInit();
    // Standard initialization
    // If you are not using the
    //  of your final executabl
    //  the specific initializa

#ifdef _AFXDLL
```

Listing 36.

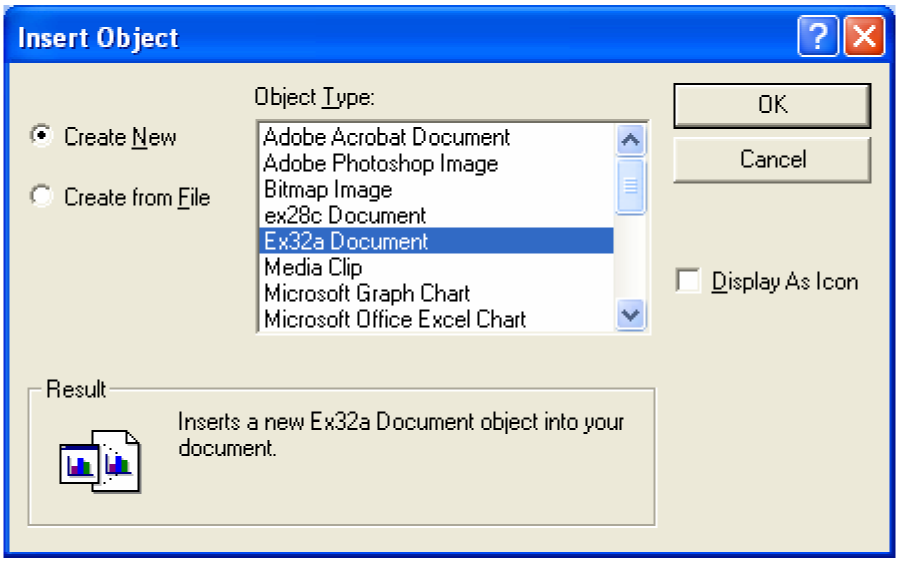
Now, it is time to test EX32B. Build and run, make sure there is no error lol! Select **Edit Insert Object** menu.



Insert an object lol

Figure 28: EX32B in action.

Select EX32A object, **Ex32a Document**. Click the **OK** button.


Insert Object

Create New [✓]

Create from File

Object Type:

- Adobe Acrobat Document
- Adobe Photoshop Image
- Bitmap Image
- ex28c Document
- Ex32a Document [✓]
- Media Clip
- Microsoft Graph Chart
- Microsoft Office Excel Chart

OK

Cancel

Display As Icon

Result

Inserts a new Ex32a Document object into your document.

Figure 29: Selecting EX32A object, **Ex32a Document**.

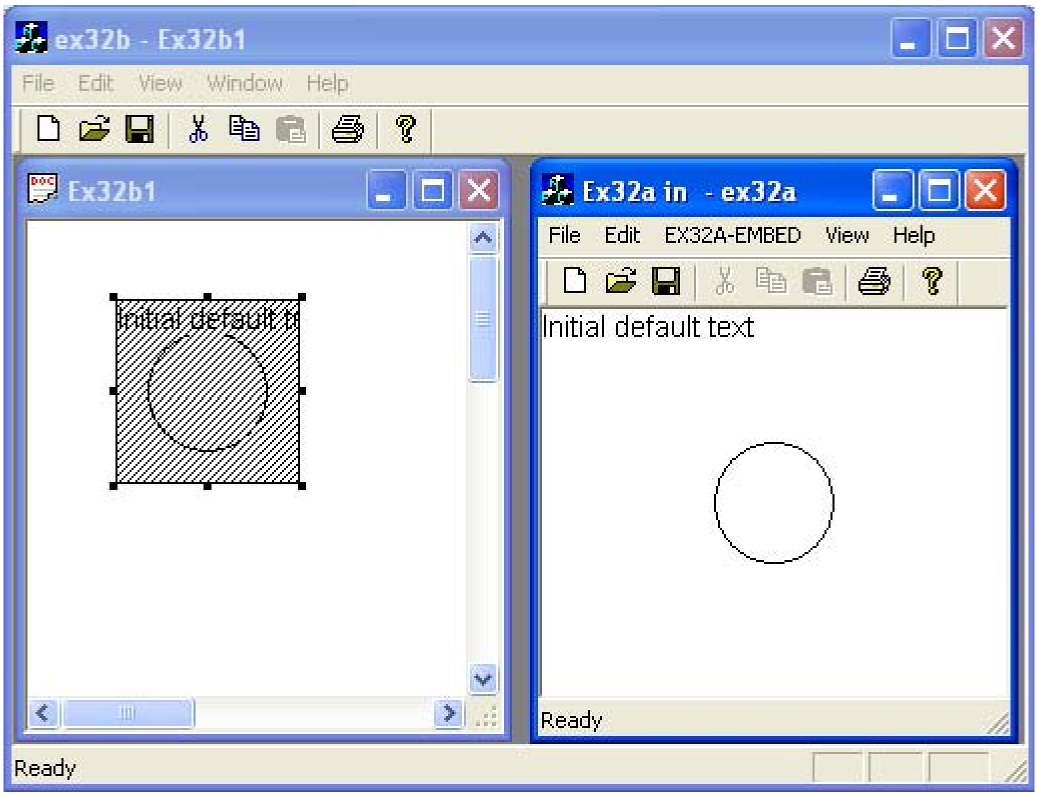
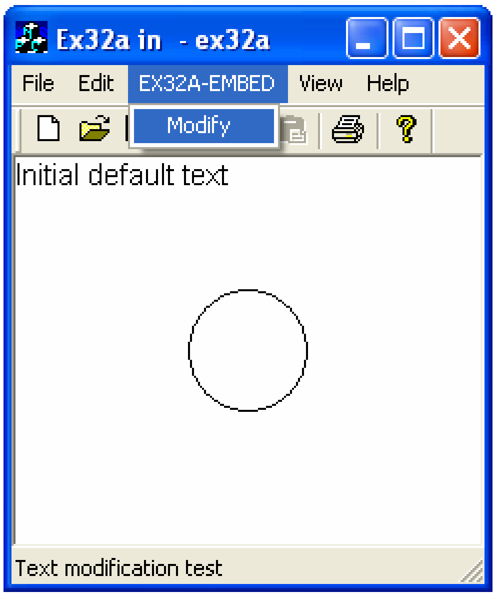


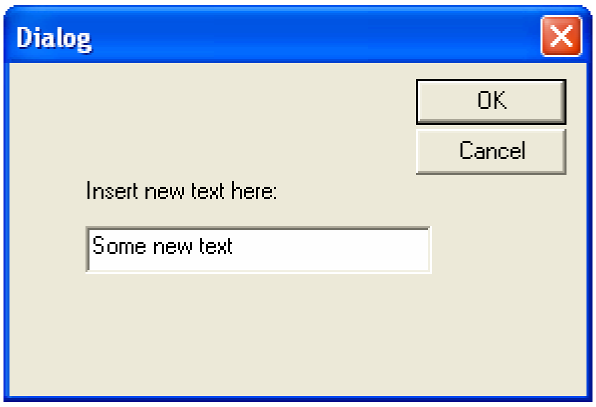
Figure 30: The embedded of the EX32A with in-place put side by side.



Initial default text

Text modification test

Figure 31: Modifying text in embedded mode.



Insert new text here:

Some new text

Figure 32: Trying some new strings.

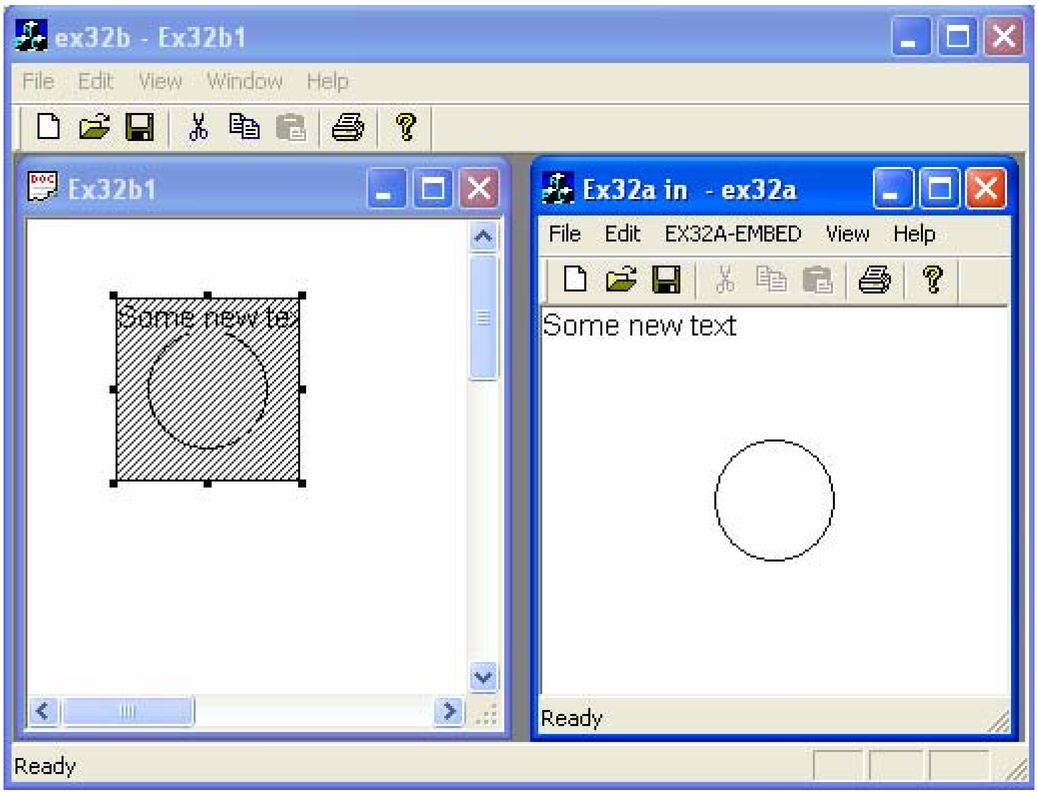


Figure 33: The new string in in-place and embedded modes.

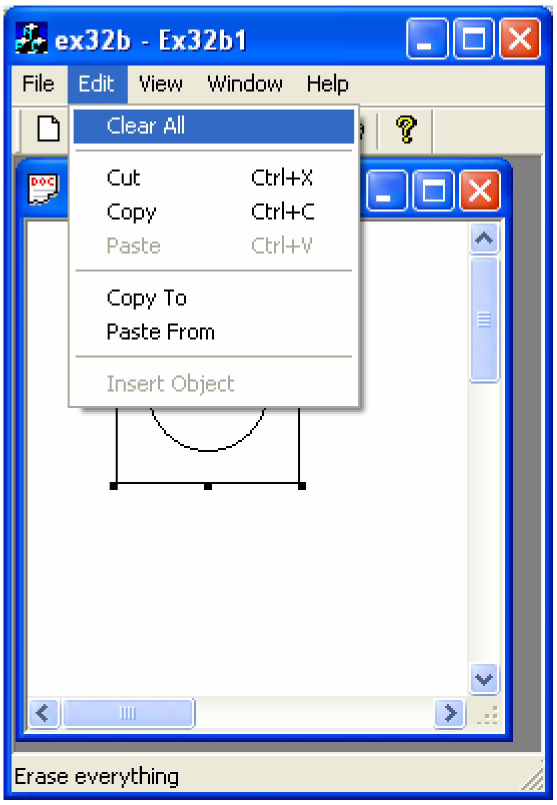
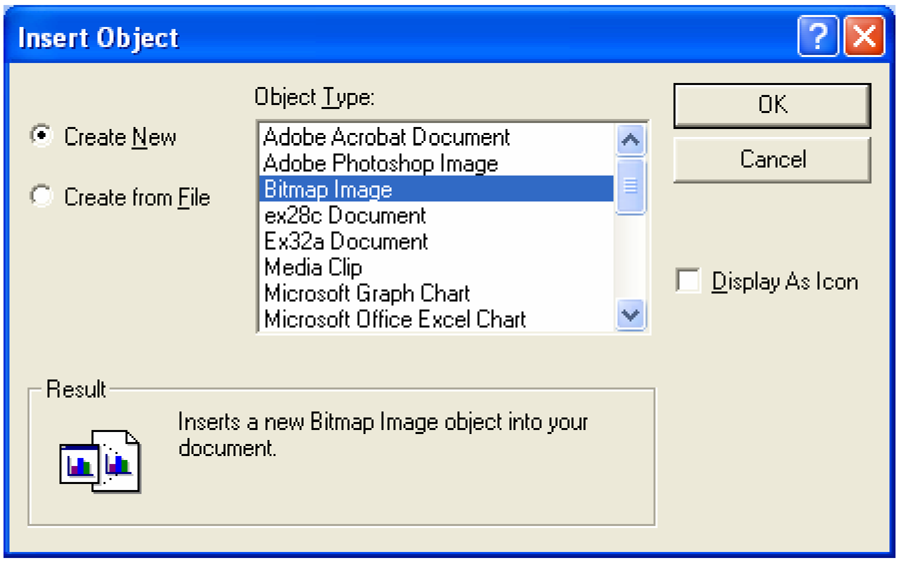


Figure 34: Testing the **Clear All** menu.

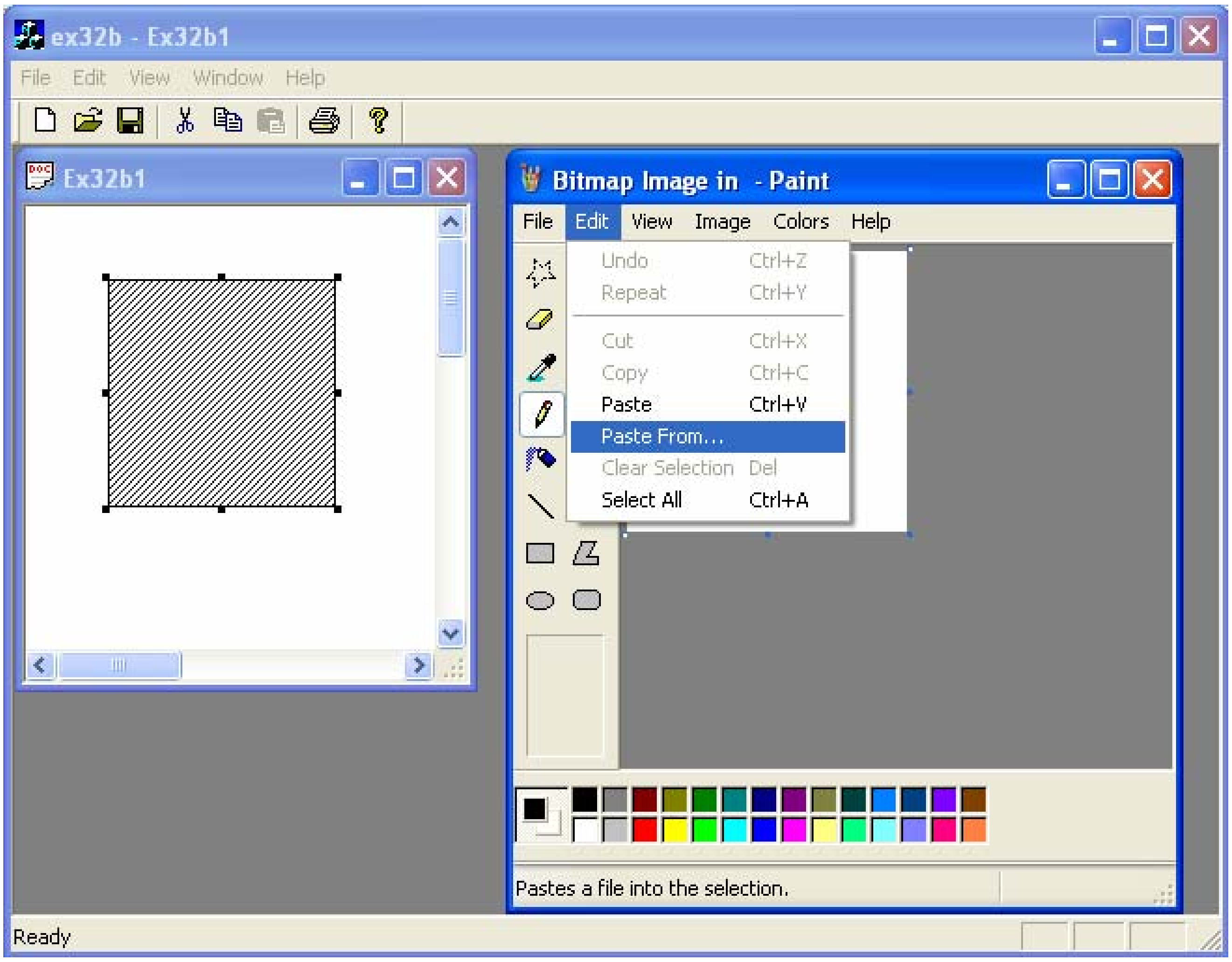
Next, let try other object. Click the **Edit Insert Object** menu. Select **Bitmap Image**.



Inserts a new Bitmap Image object into your document.

Figure 35: Selecting **Bitmap Image** object.

The default bitmap editor (Microsoft Paint) was launched. Then, select **Edit Paste From** menu of the **Paint**.



Pastes a file into the selection.

Ready

Figure 36: Default bitmap editor (Microsoft Paint) was launched.

Select any bitmap file sample. Do some editing for example, adding text as shown below.

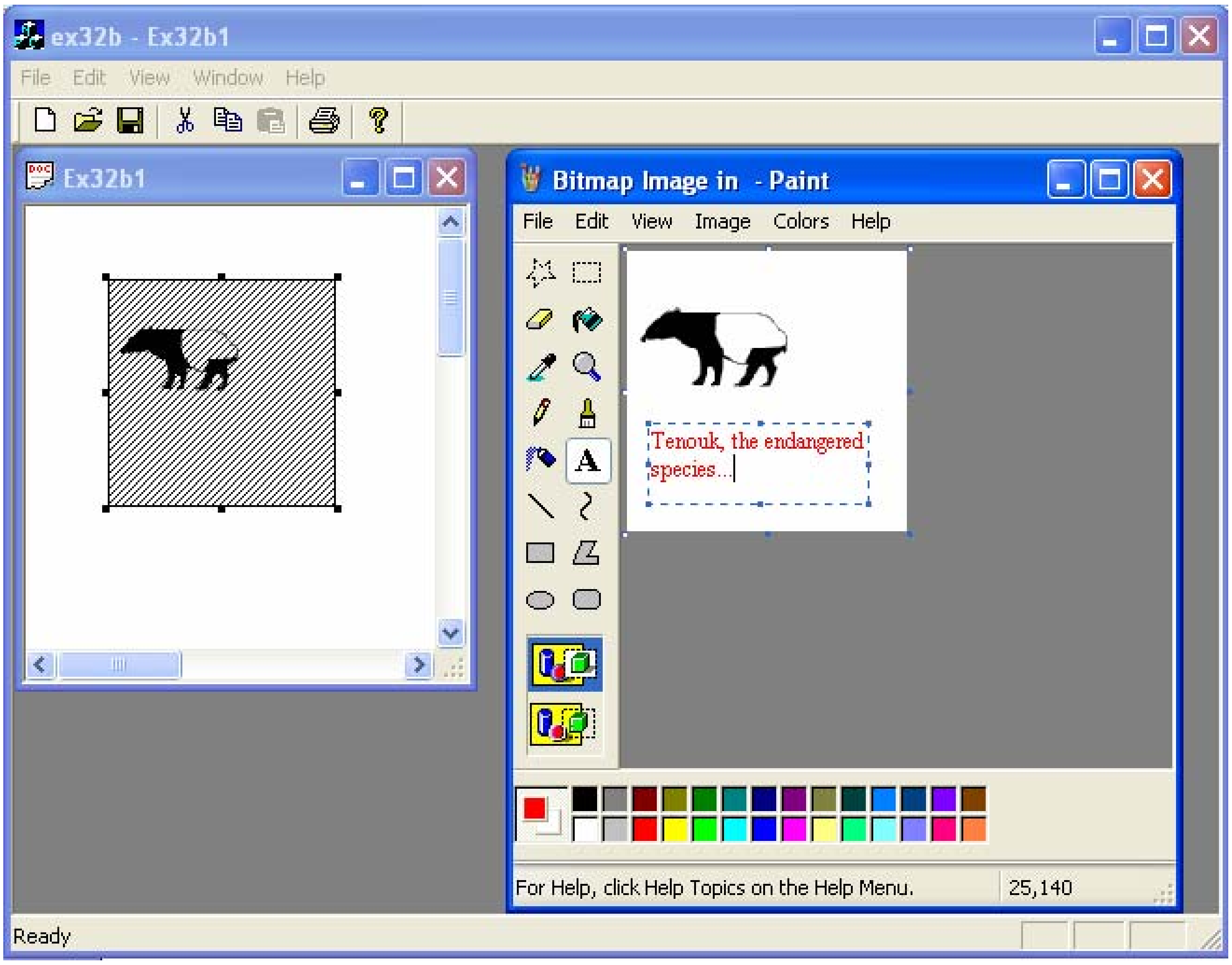

ex32b - Ex32b1
File Edit View Window Help
Ex32b1
Bitmap Image in - Paint
File Edit View Image Colors Help
Tenouk, the endangered species...
For Help, click Help Topics on the Help Menu.
25,140
Ready

Figure 37: Do some editing to the bitmap.

Then select **File Update** menu.

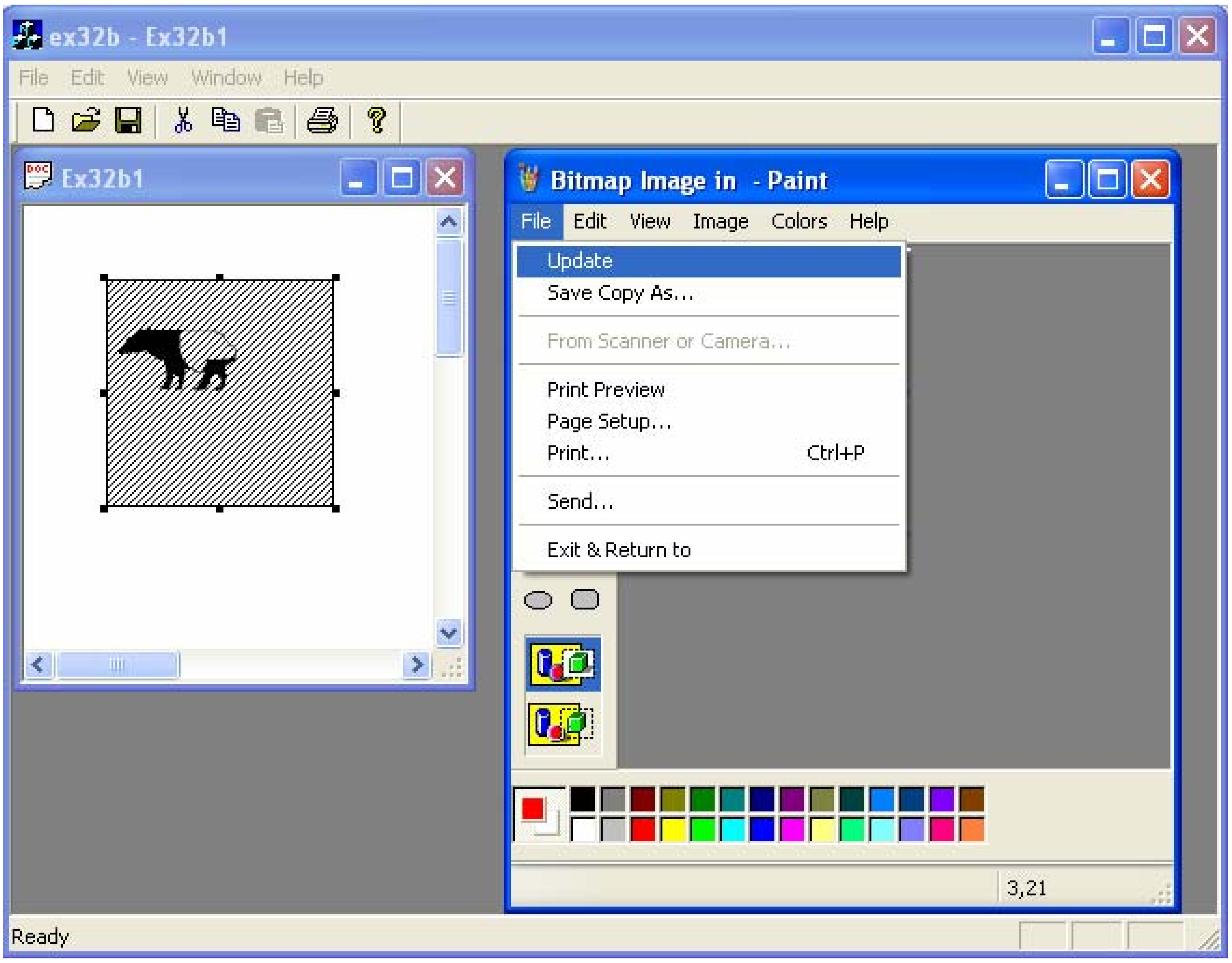


Figure 38: Updating the edited bitmap.

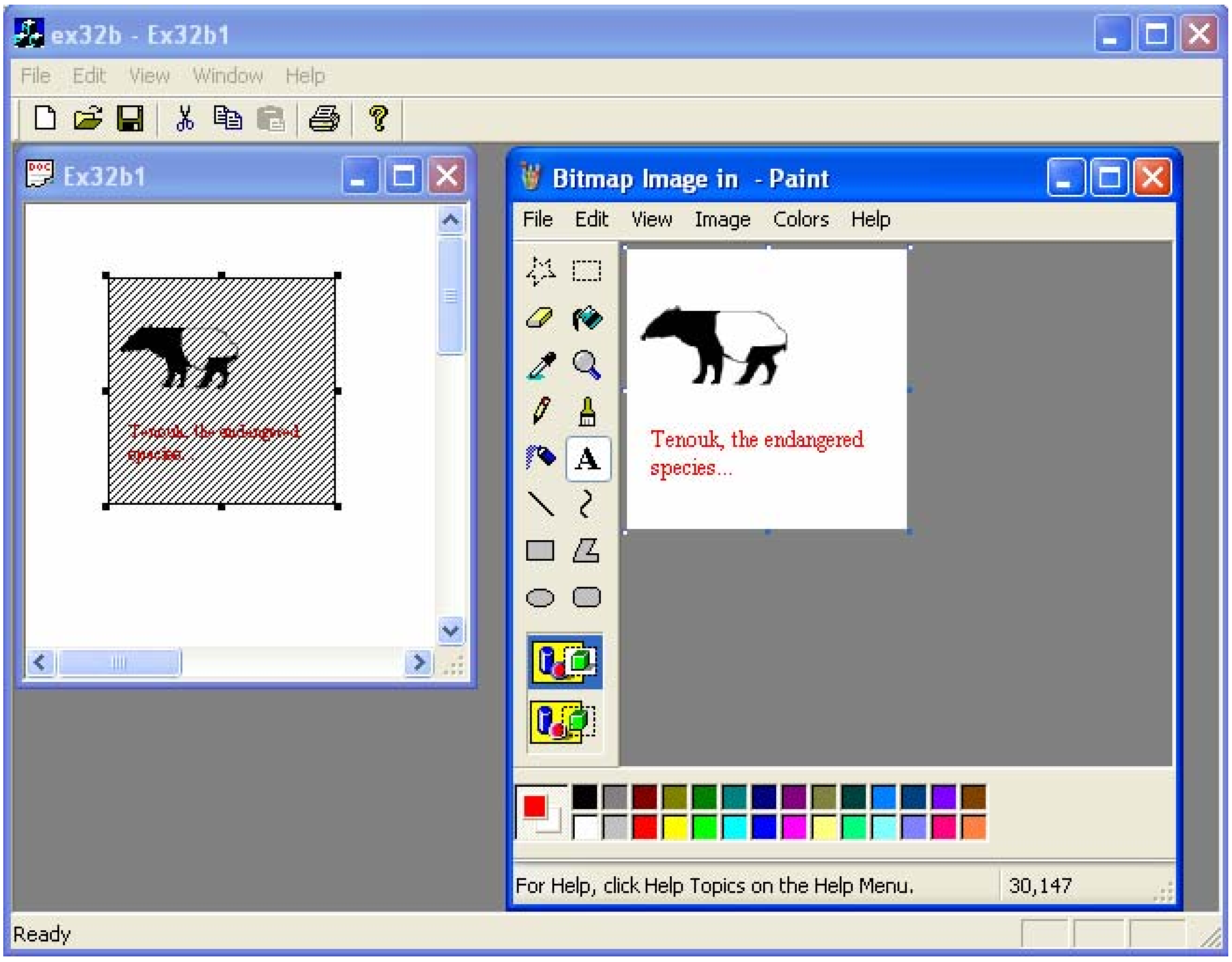

ex32b - Ex32b1
File Edit View Window Help
Ex32b1
Bitmap Image in - Paint
File Edit View Image Colors Help
Tenouk, the endangered species...
For Help, click Help Topics on the Help Menu.
30,147
Ready

Figure 39: The updated embedded and in-place modes.

For the **Paste From** menu, ASSERT failed for the `OnEditPasteFrom()`. You should debug this!

The Story

The CEx32bView Class

You can best understand the program by first concentrating on the view class. Look at the code in Listing 37, but ignore all `IOleClientSite` pointers. The container program will actually work if you pass `NULL` in every `IOleClientSite` pointer parameter. It just won't get notifications when the metafile or the native data changes. Also, components will appear displaying their stand-alone menus instead of the special embedded menus.

```
EX32BVIEW.H

// ex32bView.h : interface of the CEx32bView class
//
//////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_EX32BVIEW_H__FDA1A035_08CC_454C_88DF_8311CA5A0B9C__INCLUDED_)
#define AFX_EX32BVIEW_H__FDA1A035_08CC_454C_88DF_8311CA5A0B9C__INCLUDED_

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

#define CF_OBJECTDESCRIPTOR "Object Descriptor"
```

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

class CEx32bView : public CScrollView
{
public:
       CLIPFORMAT m_cfObjDesc;
       CLIPFORMAT m_cfEmbedded;
       CSize m_sizeTotal;  // document size
       CRectTracker m_tracker;
       CRect m_rectTracker; // logical coords

protected: // create from serialization only
       CEx32bView();
       DECLARE_DYNCREATE(CEx32bView)

// Attributes
public:
       CEx32bDoc* GetDocument();

// Operations
public:

// Overrides
       // ClassWizard generated virtual function overrides
       //{{AFX_VIRTUAL(CEx32bView)
       public:
       virtual void OnDraw(CDC* pDC);  // overridden to draw this view
       virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
       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 ~CEx32bView();
#ifdef _DEBUG
       virtual void AssertValid() const;
       virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
       //{{AFX_MSG(CEx32bView)
       afx_msg void OnEditCopy();
       afx_msg void OnUpdateEditCopy(CCmdUI* pCmdUI);
       afx_msg void OnEditCopyto();
       afx_msg void OnEditCut();
       afx_msg void OnEditInsertobject();
       afx_msg void OnUpdateEditInsertobject(CCmdUI* pCmdUI);
       afx_msg void OnEditPaste();
       afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);
       afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
       afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message);
       afx_msg void OnEditPastefrom();
       afx_msg void OnUpdateEditPaste(CCmdUI* pCmdUI);
       //}}AFX_MSG
       DECLARE_MESSAGE_MAP()
private:
	BOOL DoPasteObjectDescriptor(COleDataObject* pDataObject);
	BOOL DoPasteObject(COleDataObject* pDataObject);
	COleDataSource* SaveObject();
	BOOL MakeMetafilePict(COleDataSource* pSource);
	void SetViewAdvise();
	void SetNames();
	void GetSize();
};

#ifndef _DEBUG  // debug version in ex32bView.cpp
inline CEx32bDoc* CEx32bView::GetDocument()
   { return (CEx32bDoc*)m_pDocument; }
#endif

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

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

#endif //
!defined(AFX_EX32BVIEW_H__FDA1A035_08CC_454C_88DF_8311CA5A0B9C__INCLUDED_)
```

EX32BVIEW.CPP

```
// ex32bView.cpp : implementation of the CEx32bView class
//

#include "stdafx.h"
#include "ex32b.h"

#include <afxole.h>

#include "ex32bDoc.h"
#include "ex32bView.h"

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

/////////////////////////////////////////////////////////////////////////////
// CEx32bView

IMPLEMENT_DYNCREATE(CEx32bView, CScrollView)

BEGIN_MESSAGE_MAP(CEx32bView, CScrollView)
	//{{AFX_MSG_MAP(CEx32bView)
	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_INSERTOBJECT, OnEditInsertobject)
	ON_UPDATE_COMMAND_UI(ID_EDIT_INSERTOBJECT, OnUpdateEditInsertobject)
	ON_COMMAND(ID_EDIT_PASTE, OnEditPaste)
	ON_WM_LBUTTONDBLCLK()
	ON_WM_LBUTTONDOWN()
	ON_WM_SETCURSOR()
	ON_COMMAND(ID_EDIT_PASTEFROM, OnEditPastefrom)
	ON_UPDATE_COMMAND_UI(ID_EDIT_COPYTO, OnUpdateEditCopy)
	ON_UPDATE_COMMAND_UI(ID_EDIT_CUT, OnUpdateEditCopy)
	ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, OnUpdateEditPaste)
	//}}AFX_MSG_MAP
	// 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()

/////////////////////////////////////////////////////////////////////////////
// CEx32bView construction/destruction

CEx32bView::CEx32bView(): m_sizeTotal(20000, 25000),
    // 20 x 25 cm when printed
       m_rectTracker(0, 0, 0, 0)
{
       // TODO: add construction code here
       m_cfObjDesc = ::RegisterClipboardFormat(CF_OBJECTDESCRIPTOR);
       m_cfEmbedded = ::RegisterClipboardFormat(CF_EMBEDDEDOBJECT);
}

CEx32bView::~CEx32bView()
{
}

BOOL CEx32bView::PreCreateWindow(CREATESTRUCT& cs)
{
       // TODO: Modify the Window class or styles here by modifying
       //  the CREATESTRUCT cs

       return CScrollView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CEx32bView drawing

void CEx32bView::OnDraw(CDC* pDC)
{
       CEx32bDoc* pDoc = GetDocument();

       if(pDoc->m_lpOleObj != NULL)
    {
              VERIFY(::OleDraw(pDoc->m_lpOleObj, DVASPECT_CONTENT,
                             pDC->GetSafeHdc(), m_rectTracker) == S_OK);
       }

       m_tracker.m_rect = m_rectTracker;
      pDC->LPtoDP(m_tracker.m_rect);   // device
       if(pDoc->m_bHatch)
       {
              m_tracker.m_nStyle |= CRectTracker::hatchInside;
       }
       else
       {
              m_tracker.m_nStyle &= ~CRectTracker::hatchInside;
       }
       m_tracker.Draw(pDC);
}

void CEx32bView::OnInitialUpdate()
{
       TRACE("CEx32bView::OnInitialUpdate\n");
       m_rectTracker = CRect(1000, -1000, 5000, -5000);
       m_tracker.m_nStyle = CRectTracker::solidLine |
              CRectTracker::resizeOutside;
      SetScrollSizes(MM_HIMETRIC, m_sizeTotal);
       CScrollView::OnInitialUpdate();
}

/////////////////////////////////////////////////////////////////////////////
// CEx32bView printing

BOOL CEx32bView::OnPreparePrinting(CPrintInfo* pInfo)
{
        pInfo->SetMaxPage(1);
        return DoPreparePrinting(pInfo);
}

void CEx32bView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        // TODO: add extra initialization before printing
}

void CEx32bView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CEx32bView diagnostics

#ifdef _DEBUG
void CEx32bView::AssertValid() const
{
        CScrollView::AssertValid();
}

void CEx32bView::Dump(CDumpContext& dc) const
{
        CScrollView::Dump(dc);
}

CEx32bDoc* CEx32bView::GetDocument() // non-debug version is inline
{
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CEx32bDoc)));
        return (CEx32bDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CEx32bView message handlers

void CEx32bView::OnEditCopy()
{
        // TODO: Add your command handler code here
        COleDataSource* pSource = SaveObject();
        if(pSource)
    {
                pSource->SetClipboard(); // OLE deletes data source
    }
}

void CEx32bView::OnUpdateEditCopy(CCmdUI* pCmdUI)
{
        // TODO: Add your command update UI handler code here
        // serves Copy, Cut, and Copy To
    pCmdUI->Enable(GetDocument()->m_lpOleObj != NULL);
}

void CEx32bView::OnEditCopyto()
{
        // TODO: Add your command handler code here
        // Copy text to an .STG file (nothing special about STG ext)
    CFileDialog dlg(FALSE, "stg", "*.stg");
    if (dlg.DoModal() != IDOK) {
         return;
    }
    CEx32bDoc* pDoc = GetDocument();
    // Create a structured storage home for the object (m_pStgSub).
    // Create a root storage file, then a substorage named "sub."
        LPSTORAGE pStgRoot;
        VERIFY(::StgCreateDocfile(dlg.GetPathName().AllocSysString(),
                  STGM_READWRITE|STGM_SHARE_EXCLUSIVE|STGM_CREATE,
                  0, &pStgRoot) == S_OK);
        ASSERT(pStgRoot != NULL);

        LPSTORAGE pStgSub;
       VERIFY(pStgRoot->CreateStorage(CEx32bDoc::s_szSub,
           STGM_CREATE|STGM_READWRITE|STGM_SHARE_EXCLUSIVE,
           0, 0, &pStgSub) == S_OK);
        ASSERT(pStgSub != NULL);

        // Get the IPersistStorage* for the object
        LPPERSISTSTORAGE pPS = NULL;
        VERIFY(pDoc->m_lpOleObj->QueryInterface(IID_IPersistStorage,
                 (void**) &pPS) == S_OK);
       // Finally, save the object in its new home in the user's file

        VERIFY(::OleSave(pPS, pStgSub, FALSE) == S_OK);
        // FALSE means different stg
        pPS->SaveCompleted(NULL);  // What does this do?
        pPS->Release();

        pStgSub->Release();
        pStgRoot->Release();
}

void CEx32bView::OnEditCut()
{
        // TODO: Add your command handler code here
        OnEditCopy();
    GetDocument()->OnEditClearAll();
}

void CEx32bView::OnEditInsertobject()
{
        // TODO: Add your command handler code here
        CEx32bDoc* pDoc = GetDocument();
        COleInsertDialog dlg;
        if(dlg.DoModal() == IDCANCEL) return;
       // no addrefs done for GetInterface
        LPOLECLIENTSITE pClientSite = (LPOLECLIENTSITE)
               pDoc->GetInterface(&IID_IOleClientSite);
        ASSERT(pClientSite != NULL);
        pDoc->DeleteContents();
        VERIFY(::OleCreate(dlg.GetClassID(), IID_IOleObject,
                  OLERENDER_DRAW, NULL, pClientSite, pDoc->m_pTempStgSub,
                  (void**) &pDoc->m_lpOleObj) == S_OK);
        SetViewAdvise();

        pDoc->m_lpOleObj->DoVerb(OLEIVERB_SHOW, NULL, pClientSite, 0,
               NULL, NULL); // OleRun doesn't show it
        SetNames();
        GetDocument()->SetModifiedFlag();
        GetSize();
        pDoc->UpdateAllViews(NULL);
}

void CEx32bView::OnUpdateEditInsertobject(CCmdUI* pCmdUI)
{
        // TODO: Add your command update UI handler code here
        pCmdUI->Enable(GetDocument()->m_lpOleObj == NULL);
}

void CEx32bView::OnEditPaste()
{
        // TODO: Add your command handler code here
        CEx32bDoc* pDoc = GetDocument();
        COleDataObject dataObject;
        VERIFY(dataObject.AttachClipboard());
        pDoc->DeleteContents();
        DoPasteObjectDescriptor(&dataObject);
        DoPasteObject(&dataObject);
        SetViewAdvise();
        GetSize();
        pDoc->SetModifiedFlag();
        pDoc->UpdateAllViews(NULL);
}

void CEx32bView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
        // TODO: Add your command update UI handler code here
        // Make sure that object data is available
        COleDataObject dataObject;
        if (dataObject.AttachClipboard() &&
        dataObject.IsDataAvailable(m_cfEmbedded)) {
         pCmdUI->Enable(TRUE);
    } else {
        pCmdUI->Enable(FALSE);
    }
}

void CEx32bView::OnEditPastefrom()
{
        // TODO: Add your command handler code here
        CEx32bDoc* pDoc = GetDocument();
    // Paste from an .STG file
    CFileDialog dlg(TRUE, "stg", "*.stg");
    if (dlg.DoModal() != IDOK) {
         return;
    }
       // Open the storage and substorage
        LPSTORAGE pStgRoot;
        VERIFY(::StgOpenStorage(dlg.GetPathName().AllocSysString(), NULL,
                  STGM_READ|STGM_SHARE_EXCLUSIVE,
                  NULL, 0, &pStgRoot) == S_OK);
        ASSERT(pStgRoot != NULL);

        LPSTORAGE pStgSub;
        VERIFY(pStgRoot->OpenStorage(CEx32bDoc::s_szSub, NULL,
                  STGM_READ|STGM_SHARE_EXCLUSIVE,
                  NULL, 0, &pStgSub) == S_OK);
        ASSERT(pStgSub != NULL);

       // Copy the object data from the user storage to the temporary storage
        VERIFY(pStgSub->CopyTo(NULL, NULL, NULL,
                  pDoc->m_pTempStgSub) == S_OK);
       // Finally, load the object -- pClientSite not necessary
        LPOLECLIENTSITE pClientSite =
               (LPOLECLIENTSITE) pDoc->GetInterface(&IID_IOleClientSite);
        ASSERT(pClientSite != NULL);
        pDoc->DeleteContents();
        VERIFY(::OleLoad(pDoc->m_pTempStgSub, IID_IOleObject, pClientSite,
                 (void**) &pDoc->m_lpOleObj) == S_OK);
        SetViewAdvise();
        pStgSub->Release();
        pStgRoot->Release();
        GetSize();
    pDoc->SetModifiedFlag();
    pDoc->UpdateAllViews(NULL);
}


void CEx32bView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
        // TODO: Add your message handler code here and/or call default
        if(m_tracker.HitTest(point) == CRectTracker::hitNothing) return;
        // Activate the object
    CEx32bDoc* pDoc = GetDocument();
        if(pDoc->m_lpOleObj != NULL) {
               LPOLECLIENTSITE pClientSite =
                      (LPOLECLIENTSITE) pDoc->GetInterface(&IID_IOleClientSite);
               ASSERT(pClientSite != NULL);
               VERIFY(pDoc->m_lpOleObj->DoVerb(OLEIVERB_OPEN, NULL, pClientSite, 0,
                      GetSafeHwnd(), CRect(0, 0, 0, 0)) == S_OK);
               SetNames();
               GetDocument()->SetModifiedFlag();
        }
}

void CEx32bView::OnLButtonDown(UINT nFlags, CPoint point)
{
        // TODO: Add your message handler code here and/or call default
        TRACE("**Entering CEx32bView::OnLButtonDown -- point = (%d, %d)\n",
                point.x, point.y);
         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 coords
               GetDocument()->UpdateAllViews(NULL);
        }
        TRACE("**Leaving CEx32bView::OnLButtonDown\n");
}

BOOL CEx32bView::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 CEx32bView::GetSize()
{
        CEx32bDoc* pDoc = GetDocument();
        if(pDoc->m_lpOleObj != NULL) {
               SIZEL size;     // Ask the component for its size
               pDoc->m_lpOleObj->GetExtent(DVASPECT_CONTENT, &size);
               m_rectTracker.right = m_rectTracker.left + size.cx;
               m_rectTracker.bottom = m_rectTracker.top - size.cy;
        }
}

void CEx32bView::SetNames()
{
        CEx32bDoc* pDoc = GetDocument();
        CString strApp = AfxGetApp()->m_pszAppName;
        if(pDoc->m_lpOleObj != NULL) {
               pDoc->m_lpOleObj->SetHostNames(strApp.AllocSysString(), NULL);
        }
}

void CEx32bView::SetViewAdvise()
{
        CEx32bDoc* pDoc = GetDocument();
        if(pDoc->m_lpOleObj != NULL) {
               LPVIEWOBJECT2 pViewObj;
               pDoc->m_lpOleObj->QueryInterface(IID_IViewObject2,
                      (void**) &pViewObj);
               LPADVISESINK pAdviseSink =
                      (LPADVISESINK) pDoc->GetInterface(&IID_IAdviseSink);
               VERIFY(pViewObj->SetAdvise(DVASPECT_CONTENT, 0, pAdviseSink)
                      == S_OK);
               pViewObj->Release();
        }
}

BOOL CEx32bView::MakeMetafilePict(COleDataSource *pSource)
{
        CEx32bDoc* pDoc = GetDocument();
        COleDataObject dataObject;
        LPDATAOBJECT pDataObj; // OLE object's IDataObject interface
        VERIFY(pDoc->m_lpOleObj->QueryInterface(IID_IDataObject,
                 (void**) &pDataObj) == S_OK);
        dataObject.Attach(pDataObj);
     FORMATETC fmtem;
     SETFORMATETC(fmtem, CF_METAFILEPICT, DVASPECT_CONTENT, NULL,
         TYMED_MFPICT, -1);
     if (!dataObject.IsDataAvailable(CF_METAFILEPICT, &fmtem)) {
               TRACE("CF_METAFILEPICT format is unavailable\n");
        return FALSE;
     }
        // Just copy the metafile handle from the OLE object
        //  to the clipboard data object
     STGMEDIUM stgmm;
        VERIFY(dataObject.GetData(CF_METAFILEPICT, &stgmm, &fmtem));
        pSource->CacheData(CF_METAFILEPICT, &stgmm, &fmtem);
        return TRUE;
}

COleDataSource* CEx32bView::SaveObject()
{
        TRACE("Entering CEx32bView::SaveObject\n");
        CEx32bDoc* pDoc = GetDocument();
     if (pDoc->m_lpOleObj != NULL) {
               COleDataSource* pSource = new COleDataSource();
         // CODE FOR OBJECT DATA
         FORMATETC fmte;
         SETFORMATETC(fmte, m_cfEmbedded, DVASPECT_CONTENT, NULL,
             TYMED_ISTORAGE, -1);
         STGMEDIUM stgm;
         stgm.tymed = TYMED_ISTORAGE;
         stgm.pstg = pDoc->m_pTempStgSub;
         stgm.pUnkForRelease = NULL;
               pDoc->m_pTempStgSub->AddRef();   // must do both!
               pDoc->m_pTempStgRoot->AddRef();
               pSource->CacheData(m_cfEmbedded, &stgm, &fmte);
               // metafile needed too
               MakeMetafilePict(pSource);
               // CODE FOR OBJECT DESCRIPTION DATA
               HGLOBAL hObjDesc = ::GlobalAlloc(GMEM_SHARE,
sizeof(OBJECTDESCRIPTOR));
               LPOBJECTDESCRIPTOR pObjDesc =
                   (LPOBJECTDESCRIPTOR) ::GlobalLock(hObjDesc);
               pObjDesc->cbSize = sizeof(OBJECTDESCRIPTOR);
               pObjDesc->clsid      = CLSID_NULL;
               pObjDesc->dwDrawAspect = 0;
               pObjDesc->dwStatus = 0;
               pObjDesc->dwFullUserTypeName = 0;
               pObjDesc->dwSrcOfCopy = 0;
               pObjDesc->sizel.cx = 0;
        pObjDesc->sizel.cy = 0;
        pObjDesc->pointl.x = 0;
        pObjDesc->pointl.y = 0;
               ::GlobalUnlock(hObjDesc);
               pSource->CacheGlobalData(m_cfObjDesc, hObjDesc);
               return pSource;
        }
        return NULL;
}
```

```
BOOL CEx32bView::DoPasteObject(COleDataObject *pDataObject)
{
       TRACE("Entering CEx32bView::DoPasteObject\n");
    // Update command UI should keep us out of here if not
       //  CF_EMBEDDEDOBJECT
    if (!pDataObject->IsDataAvailable(m_cfEmbedded)) {
              TRACE("CF_EMBEDDEDOBJECT format is unavailable\n");
       return FALSE;
    }
       CEx32bDoc* pDoc = GetDocument();
    // Now create the object from the IDataObject*.
       //  OleCreateFromData will use CF_EMBEDDEDOBJECT format if available.
       LPOLECLIENTSITE pClientSite =
                     (LPOLECLIENTSITE) pDoc->GetInterface(&IID_IOleClientSite);
       ASSERT(pClientSite != NULL);
       VERIFY(::OleCreateFromData(pDataObject->m_lpDataObject,
                 IID_IOleObject,OLERENDER_DRAW, NULL, pClientSite,
                 pDoc->m_pTempStgSub, (void**) &pDoc->m_lpOleObj) == S_OK);
       return TRUE;
}

BOOL CEx32bView::DoPasteObjectDescriptor(COleDataObject *pDataObject)
{
       TRACE("Entering CEx32bView::DoPasteObjectDescriptor\n");
       STGMEDIUM stg;
       FORMATETC fmt;
       CEx32bDoc* pDoc = GetDocument();
    if (!pDataObject->IsDataAvailable(m_cfObjDesc)) {
              TRACE("OBJECTDESCRIPTOR format is unavailable\n");
       return FALSE;
    }
       SETFORMATETC(fmt, m_cfObjDesc, DVASPECT_CONTENT, NULL,
              TYMED_HGLOBAL, -1);
       VERIFY(pDataObject->GetData(m_cfObjDesc, &stg, &fmt));

       return TRUE;
}
```

Listing 37: The container's `CEx32bView` class listing.

Study the message map and the associated command handlers. They're all relatively short, and they mostly call the OLE functions described earlier. A few private helper functions need some explanation, however.
You'll see many calls to a `GetInterface()` function. This is a member of class `CCmdTarget` and returns the specified OLE interface pointer for a class in your project. It's used mostly to get the `IOleClientSite` interface pointer for your document. It's more efficient than calling `ExternalQueryInterface()`, but it doesn't increment the object's reference count.

`GetSize()`

This function calls `IOleObject::GetSize` to get the embedded object's extents, which it converts to a rectangle for storage in the tracker.

`SetNames()`

The SetNames function calls `IOleObject::SetHostNames` to send the container application's name to the component.

`SetViewAdvise()`

This function calls the embedded object's `IViewObject2::SetAdvise` function to set up the advisory connection from the component object to the container document.

MakeMetafilePict()

The MakeMetafilePict() function calls the embedded object's IDataObject::GetData function to get a metafile picture to copy to the clipboard data object. A metafile picture, by the way, is a Windows METAFILEPICT structure instance, which contains a pointer to the metafile plus extent information.

SaveObject()

This function acts like the SaveDib() function in the EX25A example. It creates a COleDataSource object with three formats: embedded object, metafile, and object descriptor.

DoPasteObjectDescriptor()

The DoPasteObjectDescriptor() function pastes an object descriptor from the clipboard but doesn't do anything with it. This function must be called prior to calling DoPasteObject().

DoPasteObject()

This function calls OleCreateFromData() to create an embedded object from an embedded object format on the clipboard.

The CEx32bDoc Class

This class implements the IOleClientSite and IAdviseSink interfaces. Because of our one-embedded-item-per-document simplification, we don't need to track separate site objects. The document is the site. We're using the standard MFC interface macros, and, as always, we must provide at least a skeleton function for all interface members.
Look carefully at the functions XOleClientSite::SaveObject, XOleClientSite::OnShowWindow, and XAdviseSink::OnViewChange in Listing 38. They're the important ones. The other ones are less important, but they contain TRACE statements as well, so you can watch the functions as they're called by the handler. Look also at the OnNewDocument(), OnCloseDocument(), and DeleteContents() functions of the CEx32bView class. Notice how the document is managing a temporary storage. The document's m_pTempStgSub data member holds the storage pointer for the embedded object, and the m_lpOleObj data member holds the embedded object's IOleObject pointer.

EX32BDOC.H

```
// ex32bDoc.h : interface of the CEx32bDoc class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_EX32BDOC_H__5C6DE978_9660_4229_887E_9A63ECF07A05__INCLUDED_)
#define AFX_EX32BDOC_H__5C6DE978_9660_4229_887E_9A63ECF07A05__INCLUDED_

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

void ITrace(REFIID iid, const char* str);

class CEx32bDoc : public CDocument
{
protected: // create from serialization only
	CEx32bDoc();
	DECLARE_DYNCREATE(CEx32bDoc)

	BEGIN_INTERFACE_PART(OleClientSite, IOleClientSite)
		STDMETHOD(SaveObject)();
		STDMETHOD(GetMoniker)(DWORD, DWORD, LPMONIKER*);
		STDMETHOD(GetContainer)(LPOLECONTAINER*);
		STDMETHOD(ShowObject)();
		STDMETHOD(OnShowWindow)(BOOL);
```

```
              STDMETHOD(RequestNewObjectLayout)();
       END_INTERFACE_PART(OleClientSite)

       BEGIN_INTERFACE_PART(AdviseSink, IAdviseSink)
              STDMETHOD_(void,OnDataChange)(LPFORMATETC, LPSTGMEDIUM);
              STDMETHOD_(void,OnViewChange)(DWORD, LONG);
              STDMETHOD_(void,OnRename)(LPMONIKER);
              STDMETHOD_(void,OnSave)();
              STDMETHOD_(void,OnClose)();
       END_INTERFACE_PART(AdviseSink)

       DECLARE_INTERFACE_MAP()

friend class CEx32bView;

private:
       LPOLEOBJECT m_lpOleObj;
       LPSTORAGE m_pTempStgRoot;
       LPSTORAGE m_pTempStgSub;
       BOOL m_bHatch;
       static const OLECHAR* s_szSub;


// Attributes
public:

// Operations
public:

// Overrides
       // ClassWizard generated virtual function overrides
       //{{AFX_VIRTUAL(CEx32bDoc)
       public:
       virtual BOOL OnNewDocument();
       virtual void Serialize(CArchive& ar);
       virtual void DeleteContents();
       virtual void OnCloseDocument();
       protected:
       virtual BOOL SaveModified();
       //}}AFX_VIRTUAL

// Implementation
public:
       virtual ~CEx32bDoc();
#ifdef _DEBUG
       virtual void AssertValid() const;
       virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
       //{{AFX_MSG(CEx32bDoc)
       afx_msg void OnEditClearAll();
       //}}AFX_MSG
       DECLARE_MESSAGE_MAP()
};

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

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

#endif //
!defined(AFX_EX32BDOC_H__5C6DE978_9660_4229_887E_9A63ECF07A05__INCLUDED_)
```

EX32BDOC.CPP

```
// ex32bDoc.cpp : implementation of the CEx32bDoc class
//

#include "stdafx.h"
#include "ex32b.h"

#include "ex32bDoc.h"

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

const OLECHAR* CEx32bDoc::s_szSub = L"sub";   // static

/////////////////////////////////////////////////////////////////////////////
// CEx32bDoc

IMPLEMENT_DYNCREATE(CEx32bDoc, CDocument)

BEGIN_MESSAGE_MAP(CEx32bDoc, CDocument)
       //{{AFX_MSG_MAP(CEx32bDoc)
       ON_COMMAND(ID_EDIT_CLEAR_ALL, OnEditClearAll)
       //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BEGIN_INTERFACE_MAP(CEx32bDoc, CDocument)
       INTERFACE_PART(CEx32bDoc, IID_IOleClientSite, OleClientSite)
       INTERFACE_PART(CEx32bDoc, IID_IAdviseSink, AdviseSink)
END_INTERFACE_MAP()

/////////////////////////////////////////////////////////////////////////////
// Implementation of IOleClientSite

STDMETHODIMP_(ULONG) CEx32bDoc::XOleClientSite::AddRef()
{
       TRACE("CEx32bDoc::XOleClientSite::AddRef\n");
       METHOD_PROLOGUE(CEx32bDoc, OleClientSite)
       return pThis->InternalAddRef();
}

STDMETHODIMP_(ULONG) CEx32bDoc::XOleClientSite::Release()
{
       TRACE("CEx32bDoc::XOleClientSite::Release\n");
       METHOD_PROLOGUE(CEx32bDoc, OleClientSite)
       return pThis->InternalRelease();
}

STDMETHODIMP CEx32bDoc::XOleClientSite::QueryInterface(
       REFIID iid, LPVOID* ppvObj)
{
       ITrace(iid, "CEx32bDoc::XOleClientSite::QueryInterface");
       METHOD_PROLOGUE(CEx32bDoc, OleClientSite)
       return pThis->InternalQueryInterface(&iid, ppvObj);
}

STDMETHODIMP CEx32bDoc::XOleClientSite::SaveObject()
{
       TRACE("CEx32bDoc::XOleClientSite::SaveObject\n");
       METHOD_PROLOGUE(CEx32bDoc, OleClientSite)
       ASSERT_VALID(pThis);

       LPPERSISTSTORAGE lpPersistStorage;
       pThis->m_lpOleObj->QueryInterface(IID_IPersistStorage,
                (void**) &lpPersistStorage);
        ASSERT(lpPersistStorage != NULL);
        HRESULT hr = NOERROR;
        if (lpPersistStorage->IsDirty() == NOERROR)
        {
                // NOERROR == S_OK != S_FALSE, therefore object is dirty!
                hr = ::OleSave(lpPersistStorage, pThis->m_pTempStgSub, TRUE);
                if (hr != NOERROR)
                        hr = lpPersistStorage->SaveCompleted(NULL);

                // Mark the document as dirty, if save successful
                pThis->SetModifiedFlag();
        }
        lpPersistStorage->Release();
        pThis->UpdateAllViews(NULL);
        return hr;
}

STDMETHODIMP CEx32bDoc::XOleClientSite::GetMoniker(
        DWORD dwAssign, DWORD dwWhichMoniker, LPMONIKER* ppMoniker)
{
        TRACE("CEx32bDoc::XOleClientSite::GetMoniker\n");
        return E_NOTIMPL;
}

STDMETHODIMP CEx32bDoc::XOleClientSite::GetContainer(
        LPOLECONTAINER* ppContainer)
{
        TRACE("CEx32bDoc::XOleClientSite::GetContainer\n");
        return E_NOTIMPL;
}

STDMETHODIMP CEx32bDoc::XOleClientSite::ShowObject()
{
        TRACE("CEx32bDoc::XOleClientSite::ShowObject\n");
        METHOD_PROLOGUE(CEx32bDoc, OleClientSite)
        ASSERT_VALID(pThis);
        pThis->UpdateAllViews(NULL);
        return NOERROR;
}

STDMETHODIMP CEx32bDoc::XOleClientSite::OnShowWindow(BOOL fShow)
{
        TRACE("CEx32bDoc::XOleClientSite::OnShowWindow\n");
        METHOD_PROLOGUE(CEx32bDoc, OleClientSite)
        ASSERT_VALID(pThis);
        pThis->m_bHatch = fShow;
        pThis->UpdateAllViews(NULL);
        return NOERROR;
}

STDMETHODIMP CEx32bDoc::XOleClientSite::RequestNewObjectLayout()
{
        TRACE("CEx32bDoc::XOleClientSite::RequestNewObjectLayout\n");
        return E_NOTIMPL;
}

/////////////////////////////////////////////////////////////////////////////
// Implementation of IAdviseSink

STDMETHODIMP_(ULONG) CEx32bDoc::XAdviseSink::AddRef()
{
        TRACE("CEx32bDoc::XAdviseSink::AddRef\n");
        METHOD_PROLOGUE(CEx32bDoc, AdviseSink)
        return pThis->InternalAddRef();
}

STDMETHODIMP_(ULONG) CEx32bDoc::XAdviseSink::Release()
```

```
{
	TRACE("CEx32bDoc::XAdviseSink::Release\n");
	METHOD_PROLOGUE(CEx32bDoc, AdviseSink)
	return pThis->InternalRelease();
}

STDMETHODIMP CEx32bDoc::XAdviseSink::QueryInterface(
	REFIID iid, LPVOID* ppvObj)
{
	ITrace(iid, "CEx32bDoc::XAdviseSink::QueryInterface");
	METHOD_PROLOGUE(CEx32bDoc, AdviseSink)
	return pThis->InternalQueryInterface(&iid, ppvObj);
}

STDMETHODIMP_(void) CEx32bDoc::XAdviseSink::OnDataChange(
	LPFORMATETC lpFormatEtc, LPSTGMEDIUM lpStgMedium)
{
	TRACE("CEx32bDoc::XAdviseSink::OnDataChange\n");
	METHOD_PROLOGUE(CEx32bDoc, AdviseSink)
	ASSERT_VALID(pThis);
	// Interesting only for advanced containers.  Forward it such that
	//  containers do not have to implement the entire interface.
}

STDMETHODIMP_(void) CEx32bDoc::XAdviseSink::OnViewChange(
	DWORD aspects, LONG /*lindex*/)
{
	TRACE("CEx32bDoc::XAdviseSink::OnViewChange\n");
	METHOD_PROLOGUE(CEx32bDoc, AdviseSink)
	ASSERT_VALID(pThis);

	pThis->UpdateAllViews(NULL); // the really important one
}

STDMETHODIMP_(void) CEx32bDoc::XAdviseSink::OnRename(
	LPMONIKER /*lpMoniker*/)
{
	TRACE("CEx32bDoc::XAdviseSink::OnRename\n");
	// Interesting only to the OLE link object. Containers ignore this.
}

STDMETHODIMP_(void) CEx32bDoc::XAdviseSink::OnSave()
{
	TRACE("CEx32bDoc::XAdviseSink::OnSave\n");
	METHOD_PROLOGUE(CEx32bDoc, AdviseSink)
	ASSERT_VALID(pThis);

	pThis->UpdateAllViews(NULL);
}

STDMETHODIMP_(void) CEx32bDoc::XAdviseSink::OnClose()
{
	TRACE("CEx32bDoc::XAdviseSink::OnClose\n");
	METHOD_PROLOGUE(CEx32bDoc, AdviseSink)
	ASSERT_VALID(pThis);

	pThis->UpdateAllViews(NULL);
}

/////////////////////////////////////////////////////////////////////////////
// CEx32bDoc construction/destruction

CEx32bDoc::CEx32bDoc()
{
	// TODO: add one-time construction code here
	m_lpOleObj = NULL;
	m_pTempStgRoot = NULL;
	m_pTempStgSub = NULL;
```

```
       m_bHatch = FALSE;
}

CEx32bDoc::~CEx32bDoc()
{
}

BOOL CEx32bDoc::OnNewDocument()
{
       TRACE("Entering CEx32bDoc::OnNewDocument\n");
       // Create a structured storage home for the object (m_pTempStgSub).
       //  This is a temporary file -- random name supplied by OLE.
       VERIFY(::StgCreateDocfile(NULL,
              STGM_READWRITE|STGM_SHARE_EXCLUSIVE|STGM_CREATE|
              STGM_DELETEONRELEASE,
              0, &m_pTempStgRoot) == S_OK);
       ASSERT(m_pTempStgRoot!= NULL);

    VERIFY(m_pTempStgRoot->CreateStorage(OLESTR("sub"),
          STGM_CREATE|STGM_READWRITE|STGM_SHARE_EXCLUSIVE,
          0, 0, &m_pTempStgSub) == S_OK);
       ASSERT(m_pTempStgSub != NULL);
       return CDocument::OnNewDocument();
}

/////////////////////////////////////////////////////////////////////////////
// CEx32bDoc serialization

void CEx32bDoc::Serialize(CArchive& ar)
{
       if (ar.IsStoring())
       {
              // TODO: add storing code here
       }
       else
       {
              // TODO: add loading code here
       }
}

/////////////////////////////////////////////////////////////////////////////
// CEx32bDoc diagnostics

#ifdef _DEBUG
void CEx32bDoc::AssertValid() const
{
       CDocument::AssertValid();
}

void CEx32bDoc::Dump(CDumpContext& dc) const
{
       CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CEx32bDoc commands

void CEx32bDoc::DeleteContents()
{
       // TODO: Add your specialized code here and/or call the base class
       if(m_lpOleObj != NULL) {
          // If object is running, close it, which releases our IOleClientSite
              m_lpOleObj->Close(OLECLOSE_NOSAVE);
              m_lpOleObj->Release(); // should be final release (or else..)
              m_lpOleObj = NULL;
       }
}
```

```
void CEx32bDoc::OnCloseDocument()
{
       // TODO: Add your specialized code here and/or call the base class
       m_pTempStgSub->Release(); // must release BEFORE calling base class
       m_pTempStgRoot->Release();
       CDocument::OnCloseDocument();
}

BOOL CEx32bDoc::SaveModified()
{
       // TODO: Add your specialized code here and/or call the base class
       // Eliminate "save to file" message
       return TRUE;
}

void CEx32bDoc::OnEditClearAll()
{
       // TODO: Add your command handler code here
       DeleteContents();
    UpdateAllViews(NULL);
    SetModifiedFlag();
       m_bHatch = FALSE;
}

void ITrace(REFIID iid, const char* str)
{
       OLECHAR* lpszIID;
       ::StringFromIID(iid, &lpszIID);
       CString strIID = lpszIID;
       TRACE("%s - %s\n", (const char*) strIID, (const char*) str);
       AfxFreeTaskMem(lpszIID);
}
```

Listing 38: The container's `CEx32bDoc` class listing.

Continue on next module...part 3

--------------------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.