Chapter 7 — Icons, Cursors, and Dialogs

Introduction

The Windows environment offers a wealth of graphical resources from simple items such as icons, cursors, and bitmaps, to more complex items such as list boxes, edit boxes, and buttons of all kinds (radio buttons, push buttons, and so on), several new controls such as the progress control, to complex items such as dialog boxes and property sheets or tabbed dialogs with many control elements. While this wide variety makes the Windows visually appealing for a user, this same wide variety offers a challenge to the Windows programmer.

This portion of Windows is so involved that an entire text could be written on using the controls, dialogs and property sheets. As you look over the books and their presentations of Windows resources, one can easily be overwhelmed by its complexity. There are many different approaches to resource construction to say nothing about the mechanical and artistic styles involved. Construction and use of Windows resources in an application range from the tedious bit by bit, manual construction from the “ground up” approach to the almost “one coding line” approach using a previously created resource built in the Workshop.

Resources can be used all by themselves within part of the main window area, such as a push button off to one side in the window. Such are known as **child window controls**. Often, the resources are organized into a package presented to the user upon demand, in other words, a **dialog box**. An application can even have these resources, especially a dialog box, as its main window! Windows provides several **Common Dialog Boxes** to simplify several universal situations: selecting a file to open, choosing a color, choosing a font, and printing a file, for example.

If we were to discuss all aspects of resource creation and usage from the C API style and the MFC styles and the various methods of construction via resource editors, we would consume all the remaining pages of this text. Thus, it is at this point that some restrictions on our coverage must be made.

1. Whenever possible, resources are created using the Developer Studio.
2. Because of Restriction 1, only the basic syntax involved in manual construction of resources is discussed and then only as it applies to using resources already constructed.
3. The major coding discussions illustrate the MFC approach to using resources. C API style is not covered.
4. Only the basics of how to use these resources is covered. This is likely one area that you will continually be learning new and fancier ways of doing things.

Realize that in the creation of a real application, a great deal of time is spent designing the optimum visual resources and their intended methods of use. Do not be surprised to discover that you are spending nearly 1/3 of your program development time on this task.

Using Developer Studio to Browse Existing Resources

An excellent place to begin your study of resources is to examine the resources of known applications. Their resources are stored in their EXE files. Bring up the Developer Studio and Choose File|Open; browse for your favorite Windows application and choose Open. Before you click Ok, in the Open As combo box, select Resource. The workshop then decompiles the resources and displays the resources in the small window. Now select the different dialogs and icons and cursors to view. Notice that it is possible to extract a resource from an application and copy it into your own resource file. Browse until you get a feel of what different resources look like and the various styles and the look and feel of Windows dialogs. Then, make a play project and Insert a New Resource and experiment creating some resources.

Creation of Icons, Bitmaps, and Cursors

Let’s begin with the simpler resources: icons, cursors, and bitmaps. What does the application require? Normally, one icon represents the application in all of the various locations, not only when it is minimized. If the standard mouse cursors are not sufficient, then perhaps you will need a cursor or two. Bitmaps are often small pictures that represent something. For now, we have almost no use for them. However, when we add tool bars (control bars) and tool boxes to our frame windows in a later chapter, bitmaps play a vital role. When we discuss the graphics interface in a later chapter, bitmaps become crucial.

See if you can create an icon and a cursor. Figure 7.1 and 7.2 show what the pair that I created for this sample program look like.

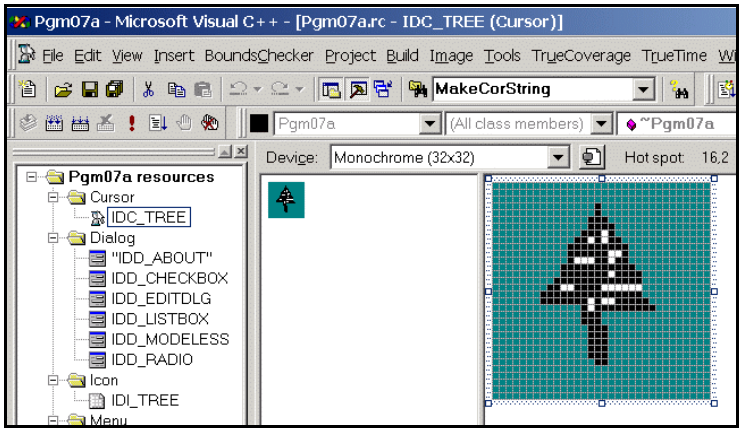

Figure 7.1 The Tree Cursor

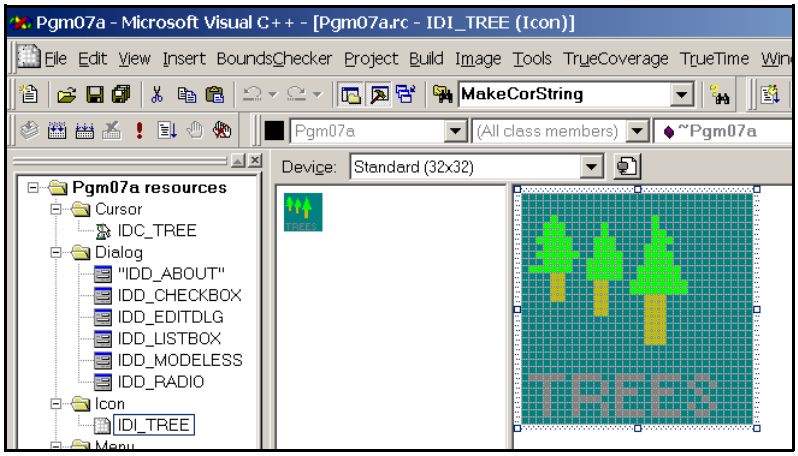

Figure 7.2 The Tree Icon

In the RC file, they are defined as separate files to be included. The ID numbers are given in the **resource.h** file.

```
IDI_TREE  ICON   "trees.ico"
IDC_TREE  CURSOR "tree.cur"
```

Observe the convention: the prefix **IDI_** identifies icons; the prefix **IDC_** identifies cursors. (It is only a convention.) As with the name of a menu, the identifiers, **IDI_TREE** and **IDC_TREE**, can be used as shown — in other words, as string identifiers — or one could define these names in the **resource.h** file, in which case they become resource ID numbers. I chose the latter, actual numbers.

The numbering scheme: Within a dialog, menu, string table, and so on, the ID numbers must be different. However, the same numbers can be reused on different type objects. That is, the one menu, icon, bitmap, and cursor can all have the same number, say 100, and still be independent items.

When the resource compiler is run as part of the Build program option, it includes the above two resource files. You can also code a partial or full path to these files. For example, if the files were in a resource subdirectory, one could code the following

```
IDI_TREE  ICON  "resource\trees.ico"
```

or

```
IDI_TREE  ICON  "d:\learnwin\vc\pgm07a\resource\trees.ico"
```

However, supplying the full path makes the program development non-portable; it now is tied to a specific drive and set of subdirectories. I recommend using only the first method.

Using Icons and Cursors

We already know how to install the application icon by using the **LoadIcon** function on the assignment to the **wndclass.hIcon** member using the C style approach. Under the MFC, the icon

would be loaded in the **FrameWin** constructor's invocation of the **Create** function as the fourth parameter to **AfxRegisterWndClass** as has been done in all examples thus far.

```
AfxGetApp()->LoadIcon (IDI_TREE)),// set min icon
```

After the invocation of **Create**, the cursor can be loaded by using the **LoadCursor** function and installed by using the **SetClassLong** function.

```
hcursor = LoadCursor (AfxGetApp()->m_hInstance,
                      MAKEINTRESOURCE (IDC_TREE));
SetClassLong (m_hWnd, GCL_HCURSOR, (long) hcursor);
```

However, let's examine some possible dynamic uses of a cursor. Assume that the arrow cursor is the default. Whenever an application enters a relatively long processing process, it is common to change the cursor to the hourglass. When the process is finished, the arrow cursor is returned. Since this is so frequently desired, the MFC has provided an extremely easy method to perform this switching. The **CWnd** class is derived from the **CCmdTarget** class. It is this base class that provides the wait cursor capabilities. To switch cursors, use the following.

```
BeginWaitCursor ();    // installs the hourglass cursor
EndWaitCursor ();      // restores the previous cursor
RestoreWaitCursor ();  // restores wait cursor after an
                       // interruption such as a MessageBox
                       // or dialog box operations are done
                       // during a wait sequence
```

These functions are best utilized within the processing sequences of a single message. When Windows is allowed to process other messages during the long wait period, the cursor can be altered by other framework actions we are about to see.

Another common use of changing cursors is to provide a special cursor while within a particular window. For example, in my WWII game, the arrow cursor is the normal cursor. However, when the cursor is over a map window, a special location cross hair cursor appears. When the mouse moves out of the map window, the arrow cursor reappears. This behavior comes about because of the default behavior of the **CWnd** message handler **OnSetCursor**.

If the mouse is not captured and mouse movement is within a window, the framework calls the window's **OnSetCursor** handler. The default implementation calls the parent window's handler before it calls the child window's handler. It uses the arrow cursor if the mouse in not in the client area of the window. If the mouse is in the client area, then the registered class cursor is used. Thus, if the application is using a **CFrameWnd** only and has installed a special cursor, then when the mouse is over the client area, the special cursor is automatically used; when the mouse is over the frame window elements, such as the menu bar, the arrow cursor appears.

This behavior is implemented in the sample programs in this chapter. The sample programs install a special cursor for use while within the client area; the arrow cursor appears automatically when the mouse moves outside the client window area. Simply by changing the cursor registered to the **CWnd** class, our applications can make use of this behavior. In case you

wish to refine the process further, the signature of the **OnSetCursor** function is this.

```
afxmsg  BOOL  OnSetCursor (CWnd* ptrwin, UINT hittest, UNIT msg);
```

The **msg** parameter is one of the various mouse messages, while the **hittest** parameter is one of the many mouse hit identifiers, such as **HTCLIENT,** meaning it is within the client area. See the documentation given in **CWnd::OnNcHitTest** for the rather long list of hit codes.

The new cursor must be loaded. If the cursor is not going to change, one can install it directly into the **WNDCLASS** structure. If the cursor is being loaded once only, it makes sense to install it directly into the **WNDCLASS** structure as the window is being registered at creation time.

Method 1 — Using the C API Functions

The **LoadCursor** function requires the **m_hInstance** value and the cursor identifier which can be given in one of two ways depending upon whether the cursor identifier is a string or an ID number. If the identifier is a number, use the macro **MAKEINTRESOURCE** to convert it into a string.

```
     HCURSOR  hcursor;
     hcursor = LoadCursor (AfxGetApp()->m_hInstance,
                    MAKEINTRESOURCE (IDC_TREE)); // id is numeric
```

or

```
     hcursor = LoadCursor (AfxGetApp()->m_hInstance, IDC_TREE);
```

If you have access to **WNDCLASS**, code it this way.

```
     wndclass.hCursor = hcursor;
     wndclass.hCursor = LoadCursor (AfxGetApp()->m_hInstance,
                                    MAKEINTRESOURCE (IDC_TREE));
```

or

```
     ::SetCursor (hcursor);
```

Method 2 — Use the CWinApp functions to load the cursor

```
     HCURSOR  LoadCursor (stringid);
     HCURSOR  LoadCursor (UINT id);
```

where **stringid** is a **char*** or **LPCTSTR**. In the above example, one could code it this way.

```
     wndclass.hCursor = AfxGetApp()->LoadCursor (IDC_TREE);
```

If the cursor is one of the Windows supplied cursors (there are a lot of them — use Explorer and look in the \WINDOWS\CURSORS folder), then use the following **CWinApp** function.

```
     HCURSOR LoadStandardCursor (LPCTSTR lpszCursorName) const;
```

The identifiers are defined in **windows.h** and include the following.

```
IDC_ARROW    Standard arrow cursor
```

```
IDC_IBEAM    Standard text-insertion cursor
IDC_WAIT     Hourglass cursor used when Windows performs
             time-consuming task
IDC_CROSS    Cross-hair cursor for selection
IDC_UPARROW  Arrow that points straight up
IDC_SIZE     Cursor to use to resize a window
IDC_ICON     Cursor to use to drag a file
IDC_SIZENWSE Two-headed arrow with ends at upperleft & lowerright
IDC_SIZENESW Two-headed arrow with ends at upperright & lowerleft
IDC_SIZEWE   Horizontal two-headed arrow
IDC_SIZENS   Vertical two-headed arrow
```

Method 3 — the SetClassLong

This method is best used when you wish to dynamically alter the cursor while the application is running.

```
SetClassLong (type id, (long) cursor object);
```

The type id can be any one of the following identifiers.

```
GCL_HCURSOR     - changes cursor
GCL_HICON       - changes the minimize icon
GCL_HBACKGROUND - changes the background brush
GCL_STYLE       - changes window styles
```

To permanently alter the cursor, code this call.

```
SetClassLong (GCL_HCURSOR, (long) HCURSOR);
```

These approaches are used in **Pgm07a**; experiment with the mouse cursor. Notice that when it is over the client window area, the “tree” cursor is visible. When the cursor passes over the menu bar or out of the window all together, the standard arrow cursor reappears.

Controls and Dialog Boxes

The basic controls that communicate user-entered information include: push buttons, list boxes, edit boxes, check boxes, radio button groups, combo boxes, and scroll bars. Windows 95 added some new controls: progress bars, spin bars (up/down control), track bars, and tree views. Also, property sheets or tabbed dialog boxes were new with Windows 95. Note that any needed scroll bars within list and combo boxes are automatically handled by Windows; we examine general scroll bars later. For now, concentrate on the others.

The basic procedure to create a new dialog begins with Insert — New Resource and select Dialog. A new, mostly empty, dialog is created which has two buttons: Ok and Cancel. In the left Resource View tab, select the new dialog name and right click on it and choose Properties. In the Properties dialog, enter the name you wish to call the dialog. In Figure 7.3, I called it IDD_PLAY. By convention, all dialog Ids begin with **IDD_**.

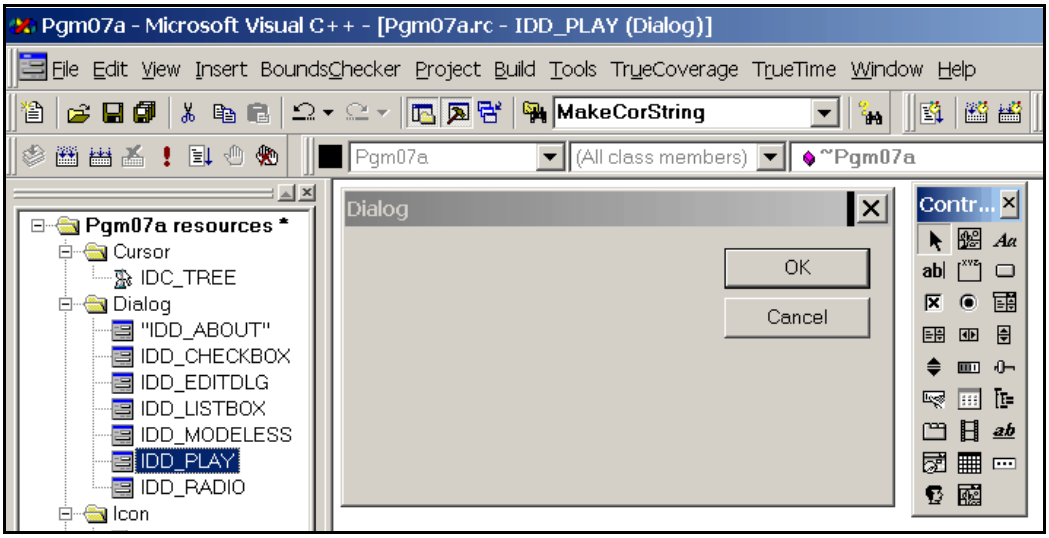

Figure 7.3 Creating a new IDD_PLAY Dialog

Next, in some empty space of the dialog area itself, right click and choose Properties. In this Properties dialog, you specify all of the overall dialog features. In the General tab, you should enter a Caption for the dialog. Then, select the More Styles tab and select 3D-Look and Center. These two options give the dialog a three-dimensional appearance and cause the dialog to be centered when it is launched. From the Layout menu, select the last item, Test. This action runs the dialog in a preview mode. You can play with it to see how it is working and appearing. When you click on Ok or Cancel, the test is terminated. This is shown in Figure 7.4.

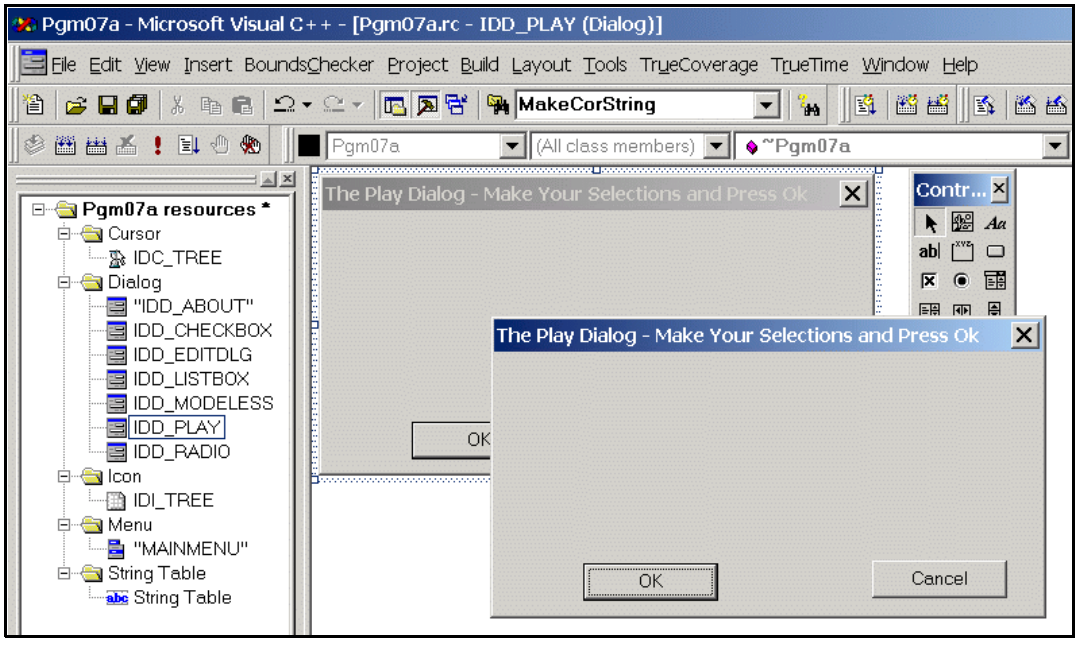

Figure 7.4 The Play Dialog with Caption and Layout-Test in Operation

Next, experiment with some of the other many style options. After selecting an option, use Layout-Test to see how that choice will appear when the dialog actually runs.

Now notice the buttons on the Controls tool box on the right side. This box contains icons representing many of the controls that can be placed within the dialog. To insert one, click on the desired control in the Control box, drag it to the dialog, and release it. Then, resize and reposition the new control as desired. Try adding a static control and an edit control similar to Figure 7.5. When the controls are where you want them, select each control in turn and right click and choose properties. With the static text control, you should enter the text that the control is to display. You can also adjust the appearance of the control with the various style options. On the static control, I used the Sunken option. Experiment to see what looks best for your dialog. When you choose the properties on the edit control, you should enter the control id number that you wish to use. By convention, the ids for controls of a dialog all begin **IDC_**. You can also set various styles to make the edit control appear and act as you desire. In this case I entered IDC_EDITNAME.

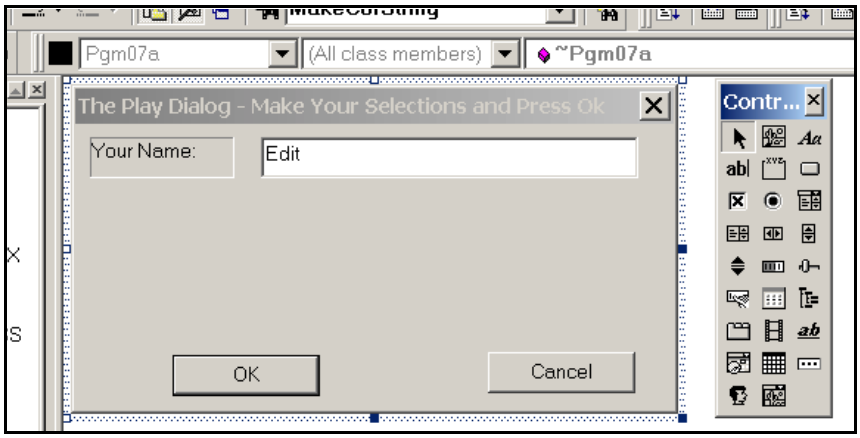

Figure 7.5 The Play Dialog with Static Control and Edit Control

On the Layout menu are numerous control alignment choices. Typically, one selects a group of controls by left-click and drag, placing a focus rectangle around the desired controls. Then, choose Layout and the alignment action to perform on the group of controls. In the above figure, I selected both controls and made them have the same height.

Next, let's add three radio buttons. Begin by inserting a new group box that will house all of the three radio buttons. Then, insert three radio button controls positioned inside the group box. Choose properties on each of these four controls. The group box should provide the overall title for the button set. Each button caption should indicate its meaning. Here, the group box says "Select Color" while each of the buttons displays a different color choice. With the radio button properties, the first button in the group must have Group and Tab Stop checked. All other radio buttons in this group must have the Tab Stop checked. This is **crucial**. Windows uses the **Group flag** to recognize that **this specific radio button** is the **first** of a group of radio buttons and that the radio buttons coming after this are part of this group of radio buttons. The ids that I used for

these buttons were **IDC_RADIO_RED**, **IDC_RADIO_GREEN**, and **IDC_RADIO_BLUE**. This is shown in Figure 7.6.

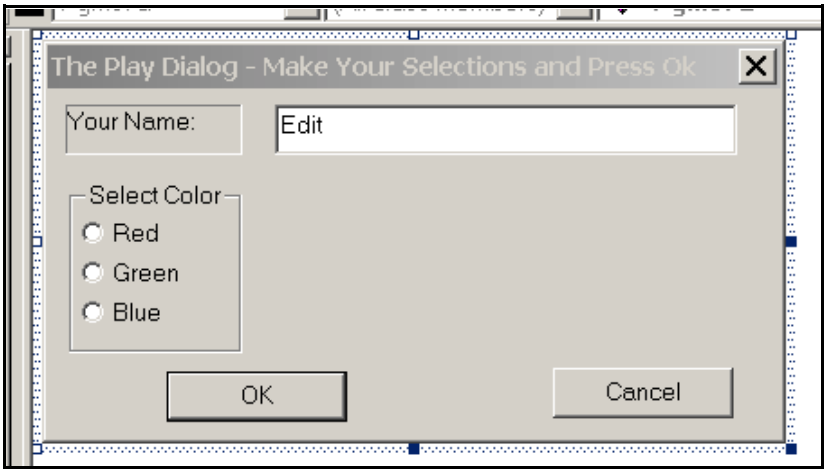

Figure 7.6 The Play Dialog with Three Color Radio Buttons

Next, add in a list box below the edit control. Here I called it **IDC_LIST_CITY**. I added the Client Edge and Static Edge styles. The one crucial item to examine is the Sorted property. If you leave this item checked, then when you add strings into the list box, they are automatically inserted in alphabetical order. While this is convenient, it poses a major problem when the user has made a selection. And that problem is how do you correlate that selection to what it is that you are to do with it. If the list box is selecting a record from an array to subsequently edit, how do you determine what the subscript of the selected item is if the list box items are in a different order than the array of records is in? Sometimes, unsorted is more convenient. Figure 7.7 shows the list box.

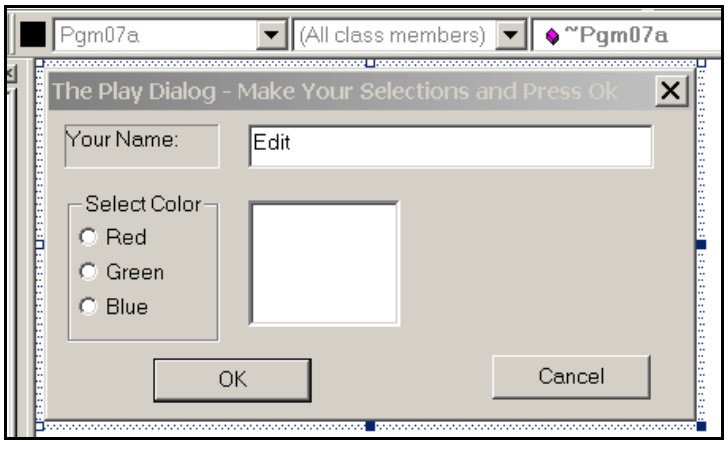

Figure 7.7 The Play Dialog with List Box

Finally, add in a check box to the right of the list box. I called this one **IDC_CHECK_EMAIL**. The final dialog view is shown in Figure 7.8.

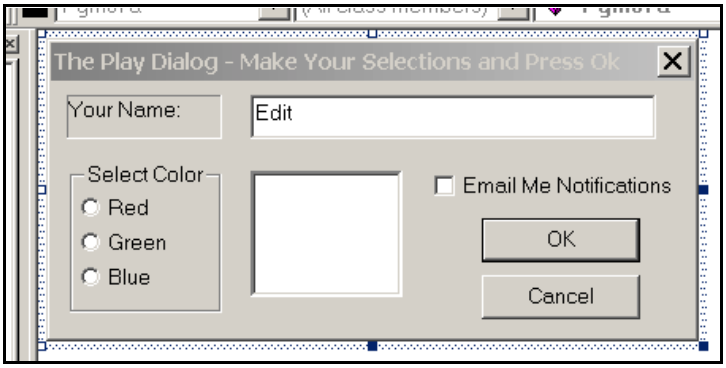

Figure 7.8 The Play Dialog with Check Box

The final step is to get the tab order set properly. When the dialog is in operation, users often tab from control to control as they make their selections. The tab order is adjustable by choosing Layout — Tab Order. When you select this option, large numbers appear beside each control. You change the order by sequential clicking on each control in the order you wish them to be. Figure 7.9 shows the final settings of my tab order. To end the tab order selection, choose Layout — Tab Order once again. Note that this menu item is actually a checked menu item.

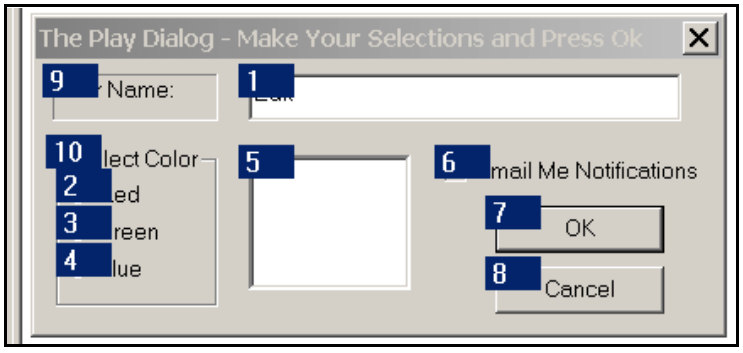

Figure 7.9 Setting the Tab Order of the Play Dialog

After you save everything, let's see what is actually stored in the text RC file. Open the RC file as text. Scroll down to the new dialog. Here is what my RC file now contains.

```
IDD_PLAY DIALOGEX 0, 0, 203, 81
STYLE DS_MODALFRAME | DS_3DLOOK | DS_CENTER | WS_POPUP | WS_CAPTION |
    WS_SYSMENU
CAPTION "The Play Dialog - Make Your Selections and Press Ok"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT        IDC_EDITNAME,63,4,128,14,ES_AUTOHSCROLL
    CONTROL         "Red",IDC_RADIO_RED,"Button",BS_AUTORADIOBUTTON |
                    WS_GROUP | WS_TABSTOP,9,35,29,10
    CONTROL         "Green",IDC_RADIO_GREEN,"Button",BS_AUTORADIOBUTTON |
                    WS_TABSTOP,9,46,35,10
    CONTROL         "Blue",IDC_RADIO_BLUE,"Button",BS_AUTORADIOBUTTON |
                    WS_TABSTOP,9,57,30,10
    LISTBOX         IDC_LIST_CITY,63,28,48,40,LBS_NOINTEGRALHEIGHT |
                    WS_VSCROLL | WS_TABSTOP,WS_EX_CLIENTEDGE |
```

```
                    WS_EX_STATICEDGE
    CONTROL         "Email Me Notifications",IDC_CHECK_EMAIL,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,121,28,75,10
    DEFPUSHBUTTON   "OK",IDOK,136,43,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,136,61,50,14
    LTEXT           "Your Name:",IDC_STATIC,5,4,49,14,SS_SUNKEN
    GROUPBOX        "Select Color",IDC_STATIC,6,25,48,48
END
```

From this listing, notice that the tab order is given by the sequential placement of the control definition lines. The **EDITTEXT** control is the first entry in the dialog and first in the tab stop order. The Red radio button comes next. Thus, you can manually cut and paste these control definitions into a different order to change the tab stop order. Also notice the vitally important styles of the radio buttons, highlighted in bold face above. The first radio button in a group of radio buttons must have the **WS_GROUP** as well as **WS_TABSTOP**. All other radio buttons in the group just have **WS_TABSTOP**.

The Text Syntax of Dialogs and Controls

The dialog box definition syntax is as follows.

```
Id of dialog DIALOGEX x, y, width, height
STYLE    ORed string of WS_ DS_ styles
CAPTION  "caption for the title bar"
FONT     points, "name"
BEGIN or {
```

One can easily reset the coordinates that the dialog box pops-up at, the (x, y) values. But with the centered option, these are not actually used. However, be careful about resetting the width and height; making these too small can result in some controls being truncated or not appearing at all! Further, all of these values are in “dialog units” and not in pixels.

As you examine the syntax of the controls, you should also see the “x, y, width, height” series of numbers representing the location of the upper left corner of the control and its size. By making all of the x values of a series of controls have the same value, you are aligning them together. So this is an alternative to the Layout menu choices.

General Design Guidelines for Dialogs

There are two major dialog types: **modal** and **modeless**. A **modal** dialog is the most common. When activated, the user is only allowed to make entries in the dialog box; the rest of the application is locked out from receiving any messages until the dialog box has completed and disappears. The **MessageBox** is modal, for example. **Modeless** dialog boxes appear and stay visible on the screen until the user terminates them. Unlike a modal dialog, a modeless dialog allows the application to continue receiving and processing events. The title bar highlighting

notifies the user which window has the input focus or is currently active. The Find-Replace dialogs are a familiar example. One very common usage of modeless boxes is to display some additional information often as informational boxes.

There are some dialog design guidelines that should be followed. Expect to spend some time getting your dialogs just the way that you want them to look.

Design Rule 32: Dialog Box Design Guidelines

1. Group all radio button items of which the user is to select one into a visible group box.

2. Buttons are usually on the right or bottom.

3. User entry Edit boxes and other control items should have some form of static text prompt or descriptive label.

4. If there are many entries or groups for the user to enter, the TAB key should proceed in an orderly manner from one to the next — top down, left to right.

5. Follow the industry guidelines for good screen data entry design — if in doubt, consult relevant texts on good screen design layouts.

If you discover that the Tab feature is not moving through the controls in a reasonable manner, top-down and left-right, you can use a text editor on the dialog in the RC file and cut and paste the controls into the correct order. Even better, use Layout|Tab Order. The Tab feature begins on the first control in the dialog given in the resource file and moves on down the sequence.

Programming Tip: When developing a new application with dialogs and controls, first create a basic shell and construct the shell **resource.h** and RC file, perhaps with a menu. Get this shell running. Next, construct the dialogs with the Developer Studio and adjust them with a text editor, if needed. Finally, compile the basic shell with the RC file and remove any accidental resource file errors. Only then go on to code the actual usage of the dialogs. Use a building block approach; it is easier to debug.

Implementing Dialogs — a Myriad of Choices

With the dialog resource built, the next step is to implement it in the application. And here in lies all of the complexity for there are many different MFC ways to actually handle the dialog operations. In order to grasp what and how you are going to actually implement a dialog, let’s

examine the fundamentals of dialog operations. To begin our discussion, assume that the dialog contains four edit controls that make up a cost record: the **itemNumber**, **quantity**, **cost**, and **description**. Further, assume that the **FrameWin** has an array of **COSTREC** structures called **costRecDataBase**. The dialog class is called **AddCostRec** and is derived from **CDialog**.

In response to a menu item, Add a Cost Record, the function **CmAddCostRecord** is called via the Message Map. To launch and execute an instance of a dialog, we code the following.

```
void FrameWin::CmAddCostRecord () {
 AddCostRec dlg (this);
 if (dlg.DoModal () == IDOK) {

 }
}
```

The **CDialog** function **DoModal** launches and executes a dialog instance returning **IDOK** if the Ok button was pressed or **IDCANCEL** if Cancel was chosen. Normally, one wishes to do further actions only if the user clicked Ok in the dialog. The above is, therefore, a very standard handling.

The **CDialog** class, while it is derived from **CWnd**, operates very differently. Instead of **OnCreate**, it uses another function to actually build the real **hwnd** of the dialog, **OnInitDialog**. In other words, the real dialog window does not exist until the call to **CDialog::OnInitDialog** returns to our derived class's **OnInitDialog**. And herein lies the major task of dialog operations — how is the necessary information or data passed back and forth between the real dialog's controls and the C++ dialog wrapper and the invoker of the dialog, **FrameWin**? Further, what if this dialog were going to update an existing record in the array. How does the caller **FrameWin** get the current record's information into the C++ wrapper and then on down into the actual real dialog controls? This mechanism is known as **data transfer** and is accomplished via one or more **transfer buffers**. Illustration 7.1 shows the complexity that must be examined.

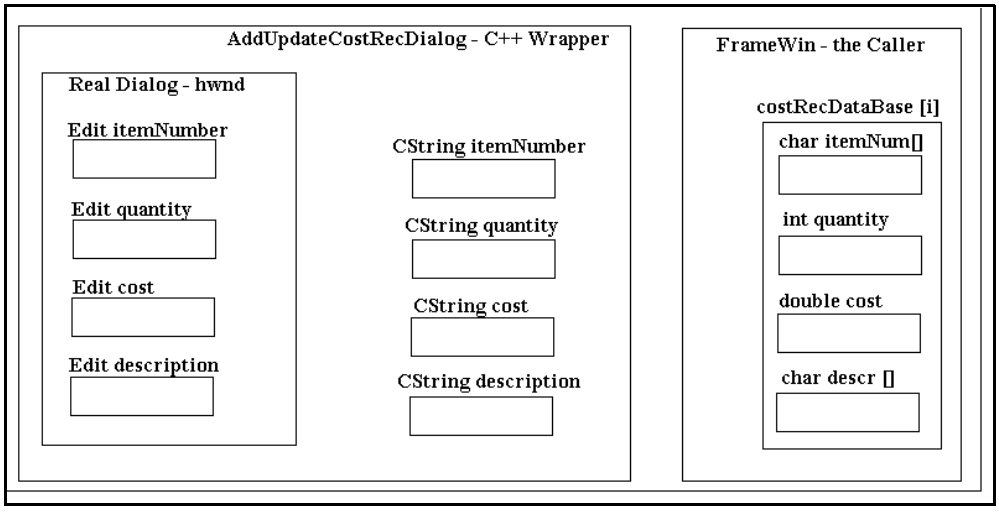

Illustration 1 The Dialog General Situation

The **FrameWin** has an area in which the new data is to be placed (Add and Update) or from which the initial values to be shown in the controls are to come (Update). Perhaps, it is the i^{th} element of an array of **COSTREC** structures. Our dialog class has member fields to store that data, very often they are **CStrings**. The real edit controls show only character strings in them. So numerical values must somehow be converted into strings before they can be inserted into the controls. When the user clicks Ok, the strings in the real edit controls must be retrieved and moved into the dialog's fields before the base class **CDialog::OnOk** is called because when that function returns, the real dialog and controls have been destroyed.

When an Update type of dialog is to be launched, the data in the i^{th} element of **FrameWin**'s array of **COSTREC** structures must be passed into our C++ dialog class. However, the real edit controls do not exist until later on, when **CDialog::OnInitDialog** is called. Thus, these initial values are often stored in data members of our C++ wrapper class. Then, when the real edit controls do exist, that data must be moved into the edit controls. Illustration 2 shows this process.

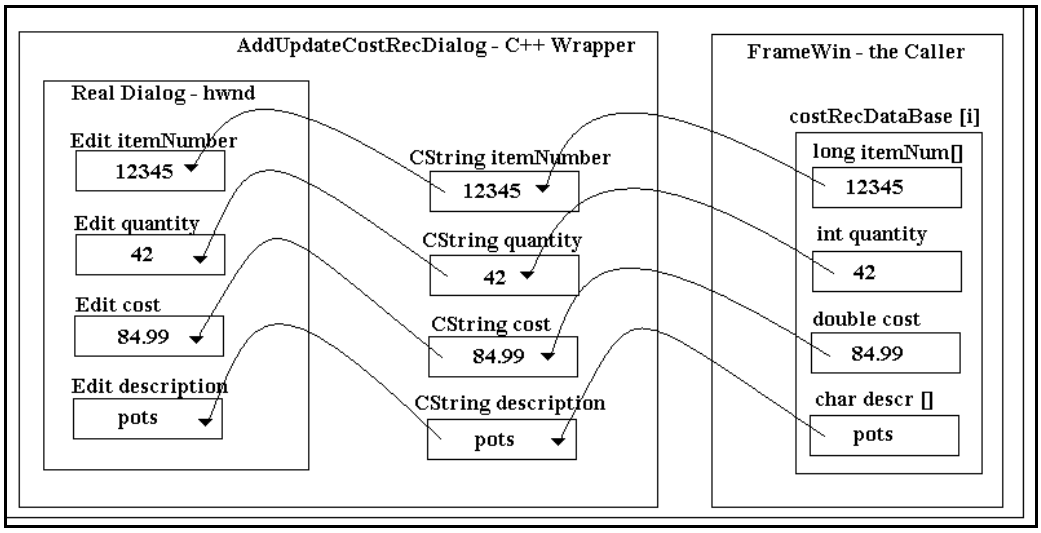

Illustration 7.2 Transferring the FrameWin's Initial Data into the Edit Controls

This transfer of data can be done in many different ways! Let's examine three of these many different approaches. One can do it manually at each step of the process, which yields maximum flexibility and control. One can do it using the MFC's built-in dynamic data transfer as implemented by the Class Wizard. One can do it the way that the Windows Common Dialogs do it. Windows provides a few common dialogs for application handling of very common actions. These common dialogs are the File-Open-Save dialog, Choose Font dialog, Find-Replace dialog, Choose Color dialog and those for File-Print and Print Setup.

The First Approach is called the manual approach. It invokes the appropriate Get/Set functions for each type of control. For an Edit control, the functions are **GetDlgItemText** and **SetDlgItemText**.

```
SetDlgItemText (control's id, char* newstring);
GetDlgItemText (control's id, char* string to be filled,
                max length of string);
```

With these two functions, one can place new data into a control or retrieve a control's contents at any time that control is in existence. This method is very handy when the dialog has a couple additional buttons: Clear and Reset. Typically, the Clear button causes the entire contents of the edit controls to be cleared out. The Reset button typically resets the contents of the edit controls back to the original contents that they initially contained when the dialog was launched. When performing data entry, these are handy. If one is updating the contents of the edit controls and one has made too many goofs, pressing reset puts all the data values back to the beginning values so one can start over. Other times, it is easier to just wipe out all of the contents and type in the new values. So these are two handy buttons users like to see in dialogs.

Illustrations 7.1 and 7.2 above are not set up to handle the Reset operation. Another set of four dialog members would likely have to be defined. These would hold the original contents with which the four fields began when the dialog was launched. Either that or provide some mechanism for the dialog member functions to get access back to the **FrameWin**'s original data to be updated.

The Second Approach is to use the Class Wizard method which uses the MFC's built-in automatic data transfer method. The **CDialog** class has member functions that can transfer data from dialog member fields to and from the actual edit controls on the screen. This method, however, does not handle in any way the transfer of the data into or out of the dialog C++ class itself. We must provide means to do that step so that when the user clicks Ok, the data ends up automatically back in the **FrameWin**'s appropriate data areas.

The Third Approach that is followed by the Windows Common Dialog classes is to totally ignore the transfer of data to and from the dialog and the caller, **FrameWin**. Instead, all the data are stored in the dialog C++ wrapper in public access data areas. When **FrameWin** wishes to launch a dialog of this type, it first allocates an instance of the dialog. The **FrameWin** must copy the necessary data into the dialog's public data members and then invoke **DoModal**. When the dialog is finished, while the real edit controls are long gone, the C++ instance of the dialog class is not. So the **FrameWin** now copies the data from the public dialog members back into **FrameWin** data areas.

The Third Approach often uses the Second Approach to actually transfer the data from the dialog data members into the real edit controls and from the controls back to the data members. Realize that there are still other methods.

The beginning point centers around the MFC's method of automatic data transfer of data to the actual edit controls from the C++ data members and vice versa. The MFC **CWnd** class, from which **CDialog** is descended has a function called **DoDataExchange** that is called to handle the data transfer. It is coded like this.

```
virtual void DoDataExchange (CDataExchange *ptrdata);
void MyDlg::DoDataExchange (CDataExchange *ptrdata) {
 CDialog::DoDataExchange (ptrdata);
 ... place the macros to transfer the data here
}
```

After the call to the base class, the instance of **CDataExchange** is initialized and ready for operations. A **BOOL** data member of this class keeps track of the intended direction of data flow — from the controls to the data member (**TRUE**)or from the data members to the controls (**FALSE**). We merely pass this instance of **CDataExchange** onto the actual macros that perform the data transfer. These macros are called **DDX**es short for dynamic data exchange. Each type of control has its own particular **DDX** macro.

Text operations

```
DDX_Text (CDataExchange* pDX, int nIDC, BYTE& value);
DDX_Text (CDataExchange* pDX, int nIDC, int& value);
DDX_Text (CDataExchange* pDX, int nIDC, UINT& value);
DDX_Text (CDataExchange* pDX, int nIDC, long& value);
DDX_Text (CDataExchange* pDX, int nIDC, DWORD& value);
DDX_Text (CDataExchange* pDX, int nIDC, CString& value);
DDX_Text (CDataExchange* pDX, int nIDC, float& value);
DDX_Text (CDataExchange* pDX, int nIDC, double& value);
```

Special control types — check box, radio button, list and combo box

```
DDX_Check (CDataExchange* pDX, int nIDC, int& value);
DDX_Radio (CDataExchange* pDX, int nIDC, int& value);
DDX_LBString (CDataExchange* pDX, int nIDC, CString& value);
DDX_CBString (CDataExchange* pDX, int nIDC, CString& value);
DDX_LBIndex (CDataExchange* pDX, int nIDC, int& index);
DDX_CBIndex (CDataExchange* pDX, int nIDC, int& index);
DDX_LBStringExact (CDataExchange* pDX, int nIDC,
                   CString& value);
DDX_CBStringExact (CDataExchange* pDX, int nIDC,
                   CString& value);
DDX_Scroll (CDataExchange* pDX, int nIDC, int& value);
```

Getting a pointer to the actual control's window

```
DDX_Control (CDataExchange* pDX, int nIDC, CWnd& rControl);
```

For edit controls, we may use any of the Text **DDX** versions. The most common one is the **CString** one. Remember that all text in the real edit control is character string data. So it makes sense for the transfer buffer in the dialog class to be a **CString**.

```
DDX_Text (CDataExchange* pDX, int nIDC, CString& value);
```

Here, the **nIDC** is the id for the specific edit control in question and the **value** is our dialog's data member. Based upon whether it is saving the control's values into the data members (**TRUE**) or whether it is updating the control's contents from the data members (**FALSE**), the **DDX** does the data transfer.

Another **CWnd** function calls **DoDataExchange** passing it the current request. That function is called **UpdateData**. It takes a **BOOL** indicating the direction of the data transfer which defaults to **TRUE**, meaning to transfer the data from the controls into the class member fields.

```
BOOL UpdateData (BOOL saveAndValidate = TRUE);
```

We may call this function anytime that we wish to transfer the data.

However, **CDialog** calls **UpdateData** from two locations. When the user clicks on the Ok button, **OnOk** is called. One of the steps within the default implementation of **OnOk** is to call **UpdateData (TRUE)**. Further, in **CDialog::OnInitDialog**, once the real dialog controls have been created, **UpdateData (FALSE)** is called.

So what must we code in our **CDialog** derived class in order for this mechanism to function properly? We must define member fields of the correct data type to match what the control's **DDX** macro expects, such as a **CString** for an edit control. We must override the **DoDataExchange** function in our class and insert the correct **DDX** macros that tie each edit control to its corresponding data member in our class. Additionally, in the constructor, these data members must be initialized to their starting values. The starting values can be null strings for an Add Record dialog or the existing data that we wish to present in an Update Record dialog.

These steps are indeed fairly simple to implement manually. However, the Class Wizard does a terrific job of setting all this up for us. So it is now time to learn how to use the Class Wizard with our dialogs.

Using Class Wizard to Create and Maintain Our CDialog Classes

Go ahead and make yourself a play dialog similar to the one I did above in Figure 7.8. Then you can follow along with this discussion step by step.

With the new dialog completed in the resource editor and with your screen appearing as shown in Figure 7.8, right click in some empty space within the dialog proper. If you right click on a control, cancel that request and try again. Choose Class Wizard from the context pop-up menu.

If this is the first time that this has been done on a new project, you will get a message box saying that the xxx.clw file does not exist and do you want to build one now. Click Yes. The Class Wizard file for a project has the extension clw. When the Select Source Files dialog box appears, just click Ok. Then, the Class Wizard immediately discovers that this dialog we just right clicked upon does not yet have a C++ class wrapped around it and asks you in a message box Adding a Class. This box is shown below.

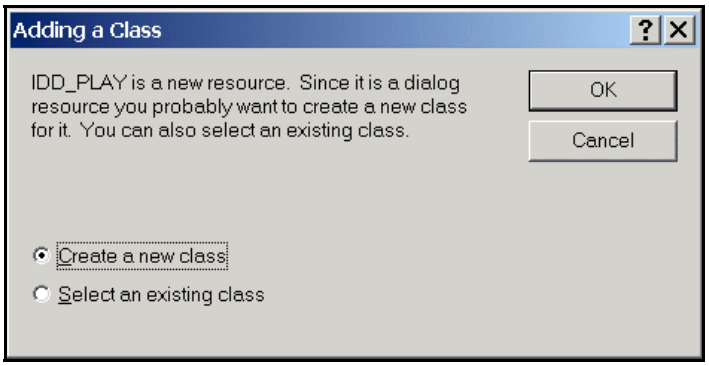

Figure 7.10 Class Wizard Add a Class Message Box

Click Ok because we wish to create a new class for this dialog. Next, in Figure 7.11, we are asked to provide a name for this class. Notice that the Class Wizard has already preselected **CDialog** as the base class and has also tied this new class to the dialog by using its id **IDD_PLAY**. I entered **PlayDialog** for the new class name. Click Ok.

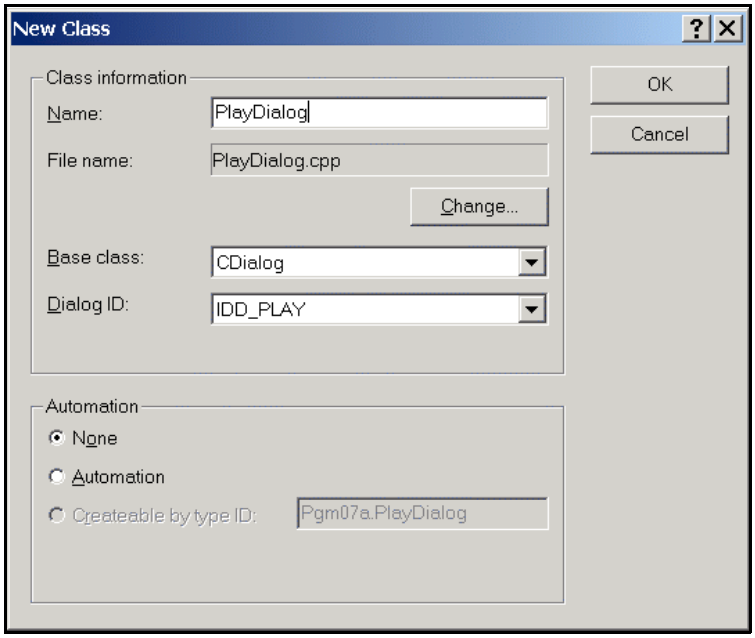

Figure 7.11 Naming the New Dialog Class

The Class Wizard then builds the new derived **PlayDialog** and opens the wizard dialog or property sheet. Your screen should appear similar to Figure 7.12 below. It is from these property pages that we control all of the options to be generated for us automatically. Specifically, we can create the data members to be used in the transfer of data. We can add overridden functions and more. From these pages, we could also select other classes within out project to work on, if there were any others. At the moment, there are none. Had we used the App Wizard to build the skeletal project, then all of those generated classes would be available here for modifications.

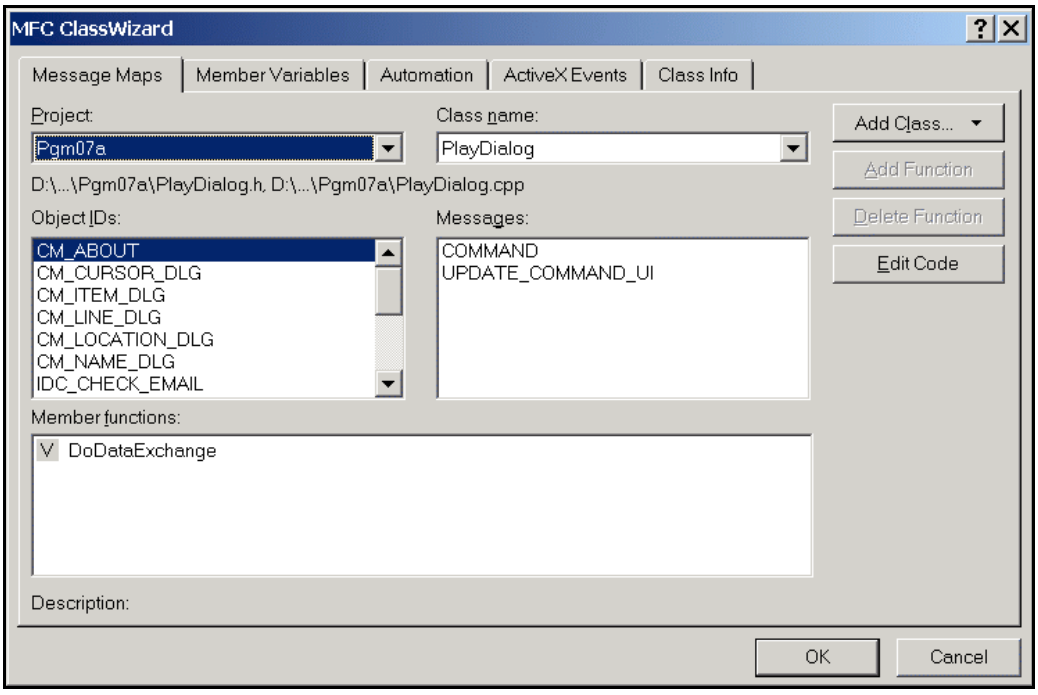

Figure 7.12 The Main Class Wizard Property Sheet

Our first action is to create the data members and tie them to the actual controls. So click on the Member Variables tab. On the left appear all of the control ids that you created in this dialog. Select the one for the edit control that is to contain the Name; mine was called **IDC_EDIT_NAME**. Then, click Add Variable. Your screen should now appear as shown in Figure 7.13 below.

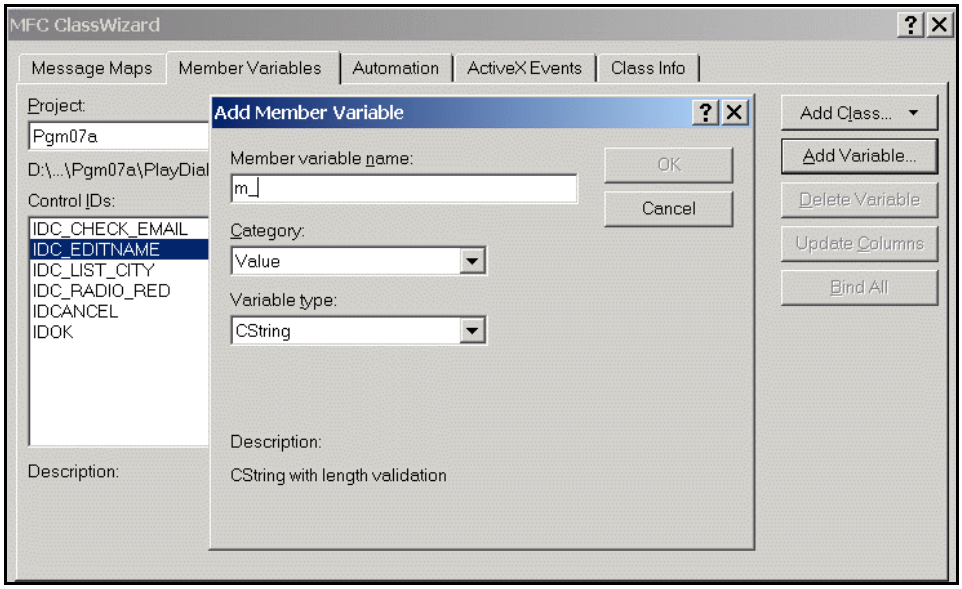

Figure 7.13 Adding a Member Variable for the Name Control

The Class Wizard prefers to prefix all member names with **m_** short for member. Since the Class Wizard is always going to begin this way, we might as well play along with it. Enter the name you wish for this variable. I called mine **m_name**. The category defaults to Value which is what we want. Click on the Variable Type combo box. Examine all of the possibilities. Notice that each of these are the exact same data types that were in the text **DDX** macros shown a few pages ago. Since this edit control is to store a person's name, we really do want **CString**.

When you click Ok, notice that now the selected line shows not only the **IDC_EDITNAME**, but also **CString** and **m_name**. Further, since this is a string, at the bottom of the window a Maximum Chars edit control has appeared. Here you can limit the total number of characters that this field could contain. It does not count the null terminator.

Next, select the id you entered for the Send Email check box; I used **IDC_CHECK_EMAIL** and click on Add Variable. Notice this time the variable is of type **BOOL**. Enter a name for this variable. I used **m_email**. Click Ok. The transfer buffer for a check box is a **BOOL**. When it is **TRUE**, the check appears. When it is **FALSE**, it is unchecked.

Now select the id for the radio button. Notice that only the id of the first radio button of the group is shown. Then, choose Add Variable. This time, notice that the variable type is **int**. I called this variable **m_color**. The transfer buffer is an **int** which contains the zero-based offset of the radio button that is selected. If the **int** contains 0, then the first radio button of the group is selected. If the **int** contains 2, then it is the third radio button that is on in the group.

Finally, select the id for the list box and choose add variable. The data type is **int** which represents the zero-based index of which string in the list box the user has selected. However, with list boxes, we do not want this as the variable type. Why? There are no strings in the list box as yet. We are going to have to add in those strings. Here comes the catch-22. We must add in the strings into the list box. So the real list box must exist. But the real list box does not exist when the constructor for the dialog class executes. The real list box only comes into existence when the **CDialog::OnInitDialog** function call has finished. So we must add our desired strings into the list box in our derived class's **OnInitDialog** after the base class finishes creation of the real dialog and its controls. This means, that we must dynamically get a pointer to this list box. However, there is an easier way. Click on the Category combo box. Notice that there is another choice — Control. Select Control and enter a name; I used **m_list**. Notice that the variable type is now **CListBox**.

Your view should now be similar to Figure 7.14 below. If yours does not, go ahead and make any needed changes. With all of the variables defined, we are ready to add in the necessary functions using the Class Wizard.

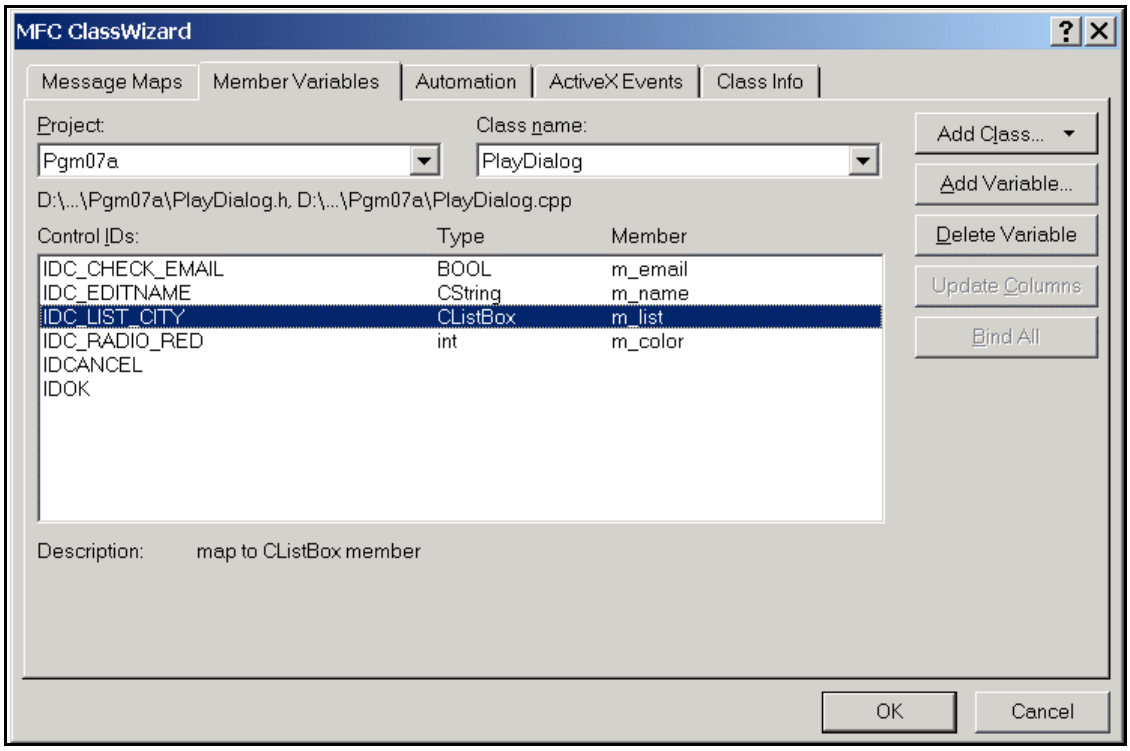

Figure 7.14 The Play Dialog With All Data Members Defined

Click on the Message Map tab and scroll down the left Object Ids column until you find the name of the dialog. Select it; here it is **PlayDialog**. In the right side list box are all of the possible member functions that the Class Wizard knows about. Notice that **DoDataExchange** is highlighted in bold face. This means that that function is already defined in the class. The Class Wizard has already added that one for us. Scroll down the list of possibilities. Notice that the Class Wizard only knows about a very tiny fraction of the total number of member functions that are available in a **CWnd** derived class. The Class Wizard can only be used to add or remove functions from within this subset. If we need a function not in this subset, we must add it manually ourselves. This is the limitation of the Class Wizard.

Scroll down to **WM_INITDIALOG**, select it, and choose Add Function. The Class Wizard immediately adds this function as shown in the bottom left list box labeled Member Functions. With this function, there is no choice in function name because we are overriding an existing member function of **CDialog**.

Finally, find the **IDOK** item in the left side Object Ids list box. Select it. Notice that since it is a button, there are two possibilities that we can respond to: a button click and a button double click. Select the button click and choose Add Function. We need to override this function in order to develop a way to get the data from our dialog class member variable back into the caller's data area(s), **FrameWin** in this case. Notice that this time, you get to provide the name of the function that is called when the Ok button is pressed. The Class Wizard assumes that you will

follow the normal naming convention and makes its suggestion for a reasonable name based upon the id of the control. **OnOk** is fine with me, so I clicked Ok. At this point your screen should appear similar to Figure 7.15 below.

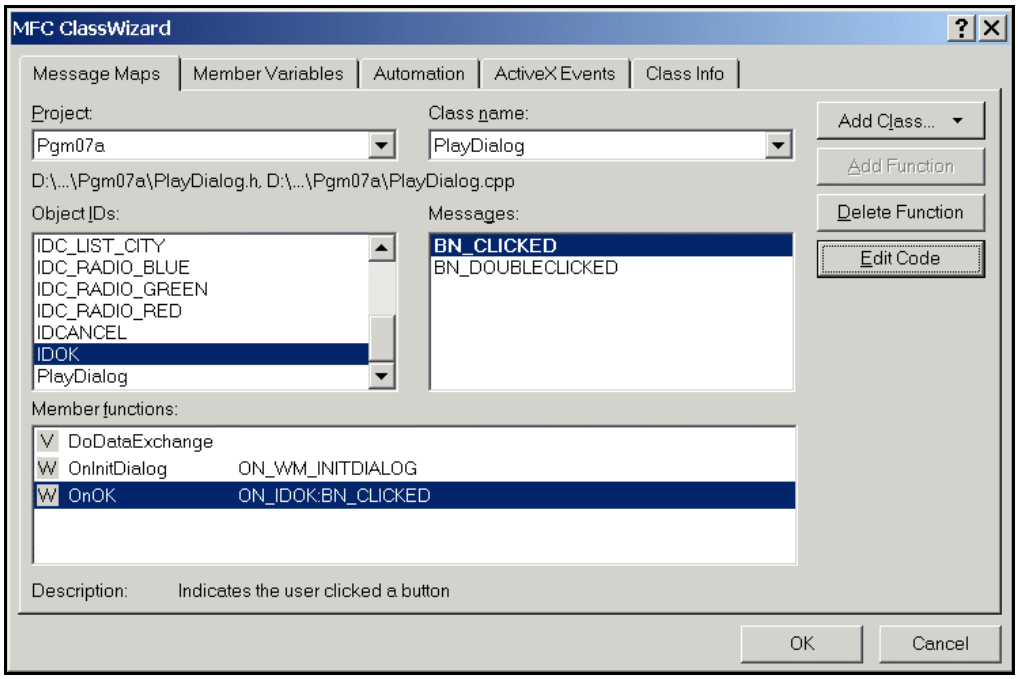

Figure 7.15 The PlayDialog Final Actions

If all is okay, now press the Edit Code button. This closes the Class Wizard and Visual Studio now opens the new cpp file we just created with the Class Wizard. Open the **PlayDialog.h** header file. Here is what mine contains. We know we need to include #**ifndef** logic. But look at the "unique name" the Class Wizard has come up with!

```
#if !defined(AFX_PLAYDIALOG_H__8CF3D2F4_FA2A_4513_BA1E_6BD99F95F1
FD__INCLUDED_)
#define AFX_PLAYDIALOG_H__8CF3D2F4_FA2A_4513_BA1E_6BD99F95F1FD__I
NCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PlayDialog.h : header file
//

/////////////////////////////////////////////////////////////////////
////////////
// PlayDialog dialog

class PlayDialog : public CDialog
{
```

```
// Construction
public:
     PlayDialog(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
     //{{AFX_DATA(PlayDialog)
     enum { IDD = IDD_PLAY };
     CListBox  m_list;
     CString   m_name;
     BOOL m_email;
     int       m_color;
     //}}AFX_DATA


// Overrides
     // ClassWizard generated virtual function overrides
     //{{AFX_VIRTUAL(PlayDialog)
     protected:
     virtual void DoDataExchange(CDataExchange* pDX);    //
DDX/DDV support
     //}}AFX_VIRTUAL

// Implementation
protected:
     // Generated message map functions
     //{{AFX_MSG(PlayDialog)
     virtual BOOL OnInitDialog();
     virtual void OnOK();
     //}}AFX_MSG
     DECLARE_MESSAGE_MAP()
};

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

#endif //
!defined(AFX_PLAYDIALOG_H__8CF3D2F4_FA2A_4513_BA1E_6BD99F95F1FD__
INCLUDED_)
```

Notice that the sections I have bold faced above are, in fact, gray in color in your editor. The cardinal rule when working with Class Wizard generated classes is do not change things contained within these funny looking comments nor do not delete the funny looking comments. If you alter or remove these funny looking comments, then the Class Wizard is totally lost and cannot ever again be used to modify this class! You see the Class Wizard is not really a wizard after all. It just finds things by looking for its unique comments!

Now look at the actual **PlayDialog.cpp** file and see what the Class Wizard has created for us. We actually have a working dialog class that will transfer data into and out of the controls into our data members — well, all except that list box. Here is the major portion of my version of **PlayDialog.cpp** file. Again notice the funny comments. Do not alter them! Notice that the constructor is initializing the data members to a default initial value. The **DoDataExchange** function provides all of the **DDX** macros needed. **OnInitDialog** has a comment showing us where we must add in our coding to add in the strings into the list box. And finally, in **OnOk**, the comments tell us to insert our extra coding before calling the base class **OnOk** function. Here is the location in which we must somehow transfer the data in our data members back to the caller's data areas.

```
PlayDialog::PlayDialog(CWnd* pParent /*=NULL*/)
     : CDialog(PlayDialog::IDD, pParent)
{
     //{{AFX_DATA_INIT(PlayDialog)
     m_name = _T("");
     m_email = FALSE;
     m_color = -1;
     //}}AFX_DATA_INIT
}

void PlayDialog::DoDataExchange(CDataExchange* pDX)
{
     CDialog::DoDataExchange(pDX);
     //{{AFX_DATA_MAP(PlayDialog)
     DDX_Control(pDX, IDC_LIST_CITY, m_list);
     DDX_Text(pDX, IDC_EDITNAME, m_name);
     DDX_Check(pDX, IDC_CHECK_EMAIL, m_email);
     DDX_Radio(pDX, IDC_RADIO_RED, m_color);
     //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(PlayDialog, CDialog)
     //{{AFX_MSG_MAP(PlayDialog)
     //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////
////////////
// PlayDialog message handlers

BOOL PlayDialog::OnInitDialog()
{
     CDialog::OnInitDialog();
     // TODO: Add extra initialization here
     return TRUE;  // return TRUE unless you set the focus to a
control
              // EXCEPTION: OCX Property Pages should return FALSE
```

```
}

void PlayDialog::OnOK()
{
     // TODO: Add extra validation here
     CDialog::OnOK();
}
```

Transferring Data from Dialog Data Members to the Caller's Data Members

Okay. We have the first problem solved easily by using the automatic **DDX** transfer mechanism. Data now can flow smoothly between the C++ data members and the real controls of the dialog on the screen. But we still have not addressed the second issue of how does the caller of the dialog get initial values into those C++ members and how do the results get back into the caller's answer areas. Remember that we are going to examine three approaches to dialogs.

The Third Approach in which all of the dialog's data members are public access is the simplest from the view point of the dialog class. Notice in the above **PlayDialog.h** class definition that the Class Wizard built for us, all of the transfer data members have been made public access. Thus, if we were going to implement the **PlayDialog** using the Third Approach, we are done with the dialog coding! All further actions occur in the caller function. When we allocate an instance of the dialog class, the C++ data members are ready for action. The actual real dialog controls, however, are not yet in existence because **DoModal** has not been called which means that **OnInitDialog** has not yet been called.

Further, once **DoModal** has returned with either an **IDOK** or **IDCANCEL**, though the real dialog controls no longer exist, the C++ class still does exist. The following represents how all of the data can be handled — that is, all except the list box strings. Let's look at the **FrameWin** coding for an Update Record type function in which the dialog data members must be given their initial values which are not nulls. In this case, assume **PlayDialog** needs to be passed the initial name and other values. Further, assume that there are many sets of Play data stored in an array of PLAY structures. The PLAY structure is defined this way.

```
     enum Color {Red, Green, Blue};
     struct PLAY {
      char name[100];
      bool email;
      Color color;
      ...
     };
     // these two are protected data members of FrameWin
     PLAY play[1000]; // the array of Play data
     int thisOne;     // index to the current one to be updated

     void FrameWin::CmPlayDialog () {
```

```
 PlayDialog dlg (this);
 // copy all need data into dialog data members
 dlg.m_name = play[thisOne].name;
 dlg.m_email = play[thisOne].email ? TRUE : FALSE;
 dlg.m_color = (int) play[thisOne].color;
 // ignore list box for now
 if (dlg.DoModal () == IDOK) {
  // copy all data from dialog members into answer area
  play[thisOne].email = dlg.m_email ? true : false;
  play[thisOne].color = (Color) dlg.m_color;
  if (dlg.m_name.GetLength() > 49) {
   // oops
   strncpy (play[thisOne].name, dlg.m_name, 49);
   play[thisOne].name[49] = 0;
  }
  else strcpy (play[thisOne].name, dlg.m_name);
  Invalidate (); // update the screen perhaps
 }
}
```

This method is straightforward, but causes a lot of coding to occur in this function. Imagine what would be in this function's body if there were a dozen data items that needed transferring? This is the drawback of the Third Approach, the caller has to do a lot of coding and the caller must know the exact dialog member variables and their data types and their possible values and ranges and so on. In other words, the dialog is not such a black box any more from an OOP point of view.

The Second Approach migrates this coding into the dialog itself so that the internal workings of the dialog are not being exposed. However, the dialog must have access to a transfer buffer that initially contains the data to be loaded into the controls and which contains the results after Ok is pressed. Note that if Cancel is pressed, this transfer buffer is not updated. In the **OnOk** function, the dialog must have access to the caller's transfer buffer. This generally means that the dialog constructor must be passed a pointer to the caller's transfer buffer or a reference to it. Further, the constructor must then save that pointer in a protected data member so that it is available in **OnInitDialog** (to be able to initialize list boxes) and **OnOk**.

Good design dictates that the **PLAY** structure definition with its **enum** is contained in a separate header file and not in the **FrameWin.h**. Thus, the **PlayDialog** must now include **Play.h** to get access to the structure definition. Here is how the **PlayDialog** class is invoked and how it can be modified for the Second Approach.

```
void FrameWin::CmPlayDialog () {
 PlayDialog dlg (this);
 if (dlg.DoModal () == IDOK) {
  // here play[thisOne] contains the data
  Invalidate ();
 }
```

```
}

PlayDialog::PlayDialog (PLAY* ptrp, CWnd* pParent /*=NULL*/)
   : CDialog(PlayDialog::IDD, pParent) {
 //{{AFX_DATA_INIT(PlayDialog)
 m_name = _T("");
 m_email = FALSE;
 m_color = -1;
 //}}AFX_DATA_INIT
 ptrplay = ptrp;
 m_name = ptrp->name;
 m_email = ptrp->email ? TRUE : FALSE;
 m_color = (int) ptrp->color;
}

void PlayDialog::OnOK() {
 UpdateData (TRUE); // transfer controls to members
 ptrplay->email = m_email ? true : false;
 ptrplay->color = (Color) m_color;
 if (m_name.GetLength() > 49) {
   strncpy (ptrplay->name, m_name, 49);
   ptrplay->name[49] = 0;
  }
  else strcpy (ptrplay->name, m_name);
 }
 CDialog::OnOK();
}
```

What about situations in which there are only a few simple incoming data fields from **FrameWin**? One could pass the address of each of these to the constructor and have it store a pointer to each item for later use. What about situations in which there are a lot of data members to be transferred to the dialog, perhaps several different structures and even some items that are not part of any structure? In this case, I usually make up a **XFER** structure that contains an instance of everything that needs to be passed, fill it up in the **FrameWin** and pass its address to the dialog. That way, the dialog does not have to store lots of pointers for use in **OnOk**.

In summary, you are going to see both the Second Approach and the Third Approach in widespread use. They are very common. However, you are also likely to see the manual approach also used. So now that we understand the basics of the automatic approaches, let’s see how the manual method operates.

The First Approach also must have ways and means of gaining access to the caller’s initial data items and a way to store the results back into the caller’s answer or transfer buffer. Thus, the data variable must still be defined as they are already in **PlayDialog**. All that changes is there is no **DoDataExchange** function.

```
PlayDialog::PlayDialog (PLAY* ptrp, CWnd* pParent /*=NULL*/)
```

```
        : CDialog(PlayDialog::IDD, pParent) {
       ptrplay = ptrp;
       m_name = ptrp->name;
       m_email = ptrp->email ? TRUE : FALSE;
       m_color = (int) ptrp->color;
      }

      BOOL PlayDialog::OnInitDialog() {
       CDialog::OnInitDialog();
       SetDlgItemText (IDC_EDITNAME, m_name);
or
       SetDlgItemText (IDC_EDITNAME, ptrplay->name);
       ...
       return TRUE;
      }

      void PlayDialog::OnOK() {
       GetDlgItemText (IDC_EDITNAME, ptrplay->name);
       ...
       CDialog::OnOK();
      }
```

When using the manual approach, once the pointer to the caller's transfer buffer is saved, there really is no need, necessarily, to have member data fields because you can transfer directly to and from the caller's buffer. However, you do need to figure out how to manually set each different type of control. Each has a different set of functions to get/set the data in the control.

Okay. So much for "playing" around. We should get some real coding done. So let's examine the dialogs and actions of **Pgm07a** which illustrates these many points and also how to deal with list boxes.

Pgm07a Dialogs to Control Window Actions

In this program, I have implemented an icon to represent the application, one that resembles a tree. I have created a "tree" cursor that looks horrible. Thus, one action the user likely desires at once is the ability to switch back to using the good old arrow cursor. So one dialog box controls switching between the arrow and the tree cursor.

Next, there is a dialog to permit the user to enter their first and last name. The screen display can show which cursor is in use along with the name of the user. However, there is a dialog box that the user can invoke that handles toggling the display of which kind of cursor is in use off and on. The user's name is always visible. But the additional line identifying the cursor is under the user's control.

Next, on which line in the window does the user's name appear? Well actually, I display three lines: a heading line, a blank line, and the user's first and last names. The size of the window at any point in processing dictates the available lines. If the window's height is small, there are only a few lines possible. If the height is large, the name could be on a large number of lines. Thus, a dialog with a list box that shows all available lines permits the user to choose on which line the name is to be displayed.

There is also a simple Help-About dialog which identifies the author of the program. And finally, the user can launch a modeless dialog that tracks mouse cursor locations.

I created these dialogs using many different dialog styles so you can see a variety of appearances. The next series of six figures shows what all of these dialogs look like.

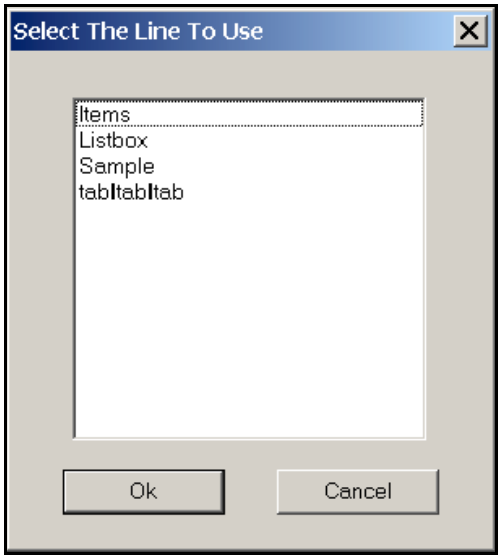

Figure 7.16 The List Box Dialog

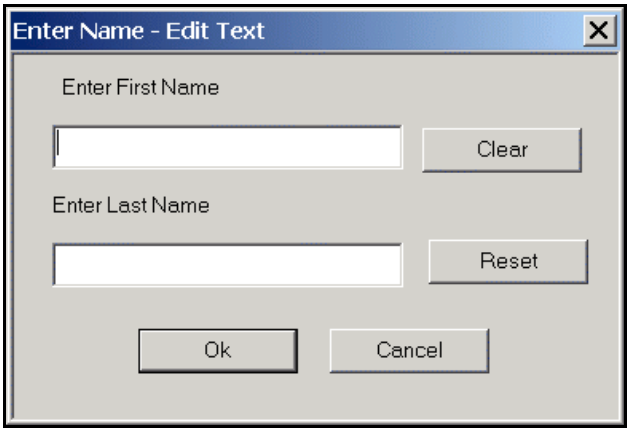

Figure 7.17 The Edit Names Dialog

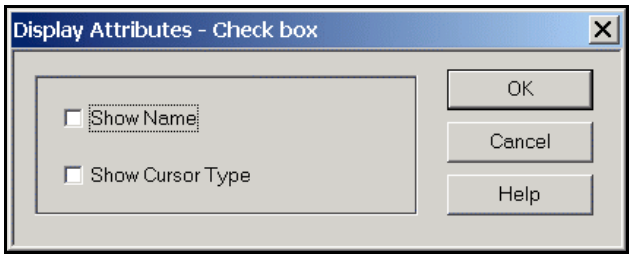

Figure 7.18 The Check Box Show Dialog

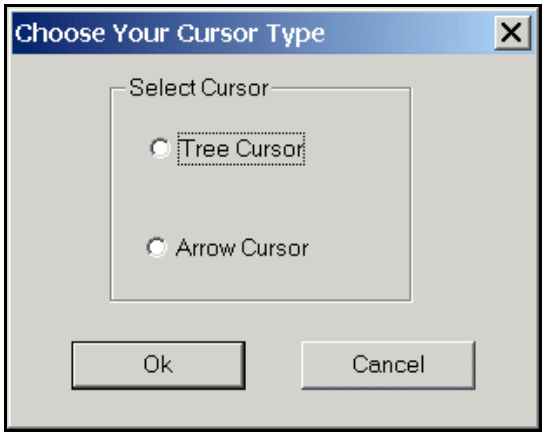

Figure 7.19 The Choose Cursor Dialog

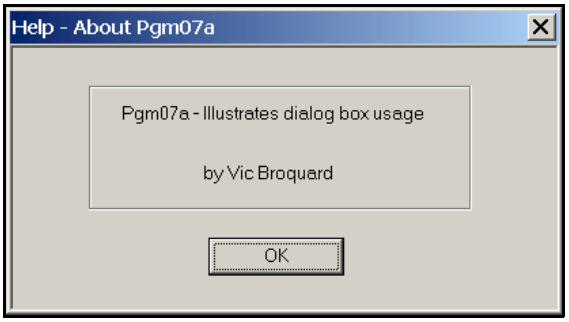

Figure 7.20 The Help About Dialog

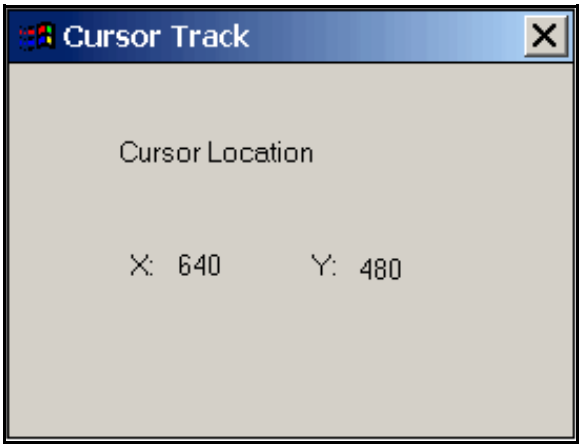

Figure 7.21 The Cursor Location Dialog

Here are the relevant portions of the **resource.h** and RC files.

```
#define CM_CURSOR_DLG                   101
#define CM_NAME_DLG                     102
#define CM_ITEM_DLG                     103
#define CM_LINE_DLG                     104
#define CM_LOCATION_DLG                 105
#define CM_ABOUT                        106
#define IDC_LISTBOX                     201
#define IDC_CHECK_NAME                  202
#define IDC_CHECK_CURSOR                203
#define IDC_FIRSTNAME                   204
#define IDC_LASTNAME                    205
#define IDC_CLEARBUTTON                 206
#define IDC_RESETBUTTON                 207
#define IDC_RADIOBUTTON_TREE            208
#define IDC_RADIOBUTTON_ARROW           209
#define IDC_CURSOR_X                    210
#define IDC_CURSOR_Y                    211
#define IDC_GRPBTN                      212
#define IDI_TREE                        300
#define IDC_TREE                        301
#define IDD_PLAY                        401
#define IDD_RADIO                       501
#define IDD_CHECKBOX                    502
#define IDD_MODELESS                    503
#define IDD_EDITDLG                     504
#define IDD_LISTBOX                     505
#define IDC_EDITNAME                    1000
#define IDC_RADIO_RED                   1001
#define IDC_RADIO_GREEN                 1002
#define IDC_RADIO_BLUE                  1003
#define IDC_LIST_CITY                   1005
#define IDC_CHECK_EMAIL                 1006
#define IDS_MAINTITLE                   2000
#define IDS_WINNAME                     2001
```

```
#define IDS_MSG_QUIT                    2002
#define IDS_MSG_QUERY                   2003
#define IDS_ISTREE                      2004
#define IDS_ISARROW                     2005
#define IDS_NAMEID                      2006
#define IDS_LINEMSG                     2007


//////////////////////////////////////////////////////////////////////
///////
//
// Menu
//

MAINMENU MENU DISCARDABLE
BEGIN
    POPUP "&Dialogs"
    BEGIN
        MENUITEM "Choose &Cursor",              CM_CURSOR_DLG
        MENUITEM "Enter &Name",                 CM_NAME_DLG
        MENUITEM "Set &Display Items",          CM_ITEM_DLG
        MENUITEM "Choose Display &Line",        CM_LINE_DLG
        MENUITEM "Show &Location of Cursor",    CM_LOCATION_DLG
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "Help &About",                 CM_ABOUT
    END
END


//////////////////////////////////////////////////////////////////////
///////
//
// Dialog
//

IDD_EDITDLG DIALOGEX 23, 81, 189, 114
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
EXSTYLE WS_EX_CLIENTEDGE | WS_EX_STATICEDGE
CAPTION "Enter Name - Edit Text"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT        IDC_FIRSTNAME,12,22,110,14
    EDITTEXT        IDC_LASTNAME,12,59,110,14
    PUSHBUTTON      "Ok",IDOK,39,86,50,14
    PUSHBUTTON      "Clear",IDC_CLEARBUTTON,128,23,50,14
    PUSHBUTTON      "Reset",IDC_RESETBUTTON,130,58,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,99,86,50,14
    CTEXT           "Enter First Name",-1,12,6,56,10
    LTEXT           "Enter Last Name",-1,12,43,56,10
```

```
END

IDD_ABOUT DIALOGEX 48, 113, 200, 96
STYLE DS_MODALFRAME | DS_3DLOOK | DS_CENTER | WS_POPUP | WS_VISIBLE |
    WS_CAPTION | WS_SYSMENU
EXSTYLE WS_EX_CLIENTEDGE | WS_EX_STATICEDGE
CAPTION "Help - About Pgm07a"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,72,70,50,14
    LTEXT           "by Vic Broquard",IDC_STATIC,70,42,57,8
    LTEXT          "Pgm07a - Illustrates dialog box usage",IDC_STATIC,
                    40,20,128,11
    GROUPBOX        "",IDC_STATIC,28,10,141,50
END

IDD_RADIO DIALOGEX 34, 29, 150, 106
STYLE DS_MODALFRAME | WS_POPUP | WS_CLIPSIBLINGS | WS_CAPTION |
WS_SYSMENU
EXSTYLE WS_EX_STATICEDGE
CAPTION "Choose Your Cursor Type"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "Tree Cursor",IDC_RADIOBUTTON_TREE,"Button",
                BS_AUTORADIOBUTTON | WS_GROUP | WS_TABSTOP,39,22,62,10
    CONTROL         "Arrow Cursor",IDC_RADIOBUTTON_ARROW,"Button",
                    BS_AUTORADIOBUTTON | WS_TABSTOP,38,48,66,15
    PUSHBUTTON      "Ok",IDOK,17,82,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,83,82,50,14
    GROUPBOX        "Select Cursor",IDC_GRPBTN,28,5,87,66,WS_GROUP
END

IDD_CHECKBOX DIALOGEX 32, 94, 207, 66
STYLE DS_MODALFRAME | DS_3DLOOK | DS_CENTER | WS_POPUP | WS_VISIBLE |
    WS_CAPTION | WS_SYSMENU
EXSTYLE WS_EX_CLIENTEDGE
CAPTION "Display Attributes - Check box"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "Show Name",IDC_CHECK_NAME,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,17,16,61,14
    CONTROL         "Show Cursor Type",IDC_CHECK_CURSOR,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,17,35,79,15
    DEFPUSHBUTTON   "OK",IDOK,148,6,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,148,24,50,14
    PUSHBUTTON      "Help",ID_HELP,148,42,50,14
    CONTROL        "",-1,"Static",SS_BLACKFRAME | SS_SUNKEN,7,8,131,48
END

IDD_MODELESS DIALOGEX 34, 51, 134, 89
STYLE DS_3DLOOK | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
EXSTYLE WS_EX_STATICEDGE
```

```
CAPTION "Cursor Track"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "Cursor Location",-1,"Static",SS_SIMPLE,26,17,56,9
    LTEXT           "X:",-1,29,44,9,9,NOT WS_GROUP
    LTEXT           "640",IDC_CURSOR_X,40,44,25,9,NOT WS_GROUP
    LTEXT           "Y:",-1,72,44,8,8,NOT WS_GROUP
    LTEXT           "480",IDC_CURSOR_Y,83,45,16,8,NOT WS_GROUP
END

IDD_LISTBOX DIALOG DISCARDABLE  20, 38, 148, 153
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Select The Line To Use"
FONT 8, "MS Sans Serif"
BEGIN
    LISTBOX         IDC_LISTBOX,19,14,109,106,LBS_SORT |
                    LBS_NOINTEGRALHEIGHT | WS_VSCROLL | WS_TABSTOP
    PUSHBUTTON      "Ok",IDOK,16,129,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,82,129,50,14
END

/////////////////////////////////////////////////////////////////////
///////
//
// Icon
//

// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDI_TREE                ICON    DISCARDABLE     "TREES.ICO"

/////////////////////////////////////////////////////////////////////
///////
//
// Cursor
//

IDC_TREE                CURSOR  DISCARDABLE     "TREE.CUR"

/////////////////////////////////////////////////////////////////////
///////
//
// String Table
//

STRINGTABLE DISCARDABLE
BEGIN
    IDS_MAINTITLE           "Resources and Dialogs - MFC style"
    IDS_WINNAME             "ResourcesPgm"
    IDS_MSG_QUIT            "Do you want to quit the application?"
    IDS_MSG_QUERY           "Query?"
    IDS_ISTREE              "The cursor is the TREE cursor."
```

```
    IDS_ISARROW             "The cursor is the ARROW cursor."
    IDS_NAMEID              "First Name  and  Last Name"
    IDS_LINEMSG             "Line %02d"
END
```

Next, since I used various dialog styles and options (3D and Centered), let's see how these dialogs actually appear when the program is run. Help About shown in Figure 7.22 is 3D and centered using both the Client and Static edges. Notice that it is centered on the screen, not the application.

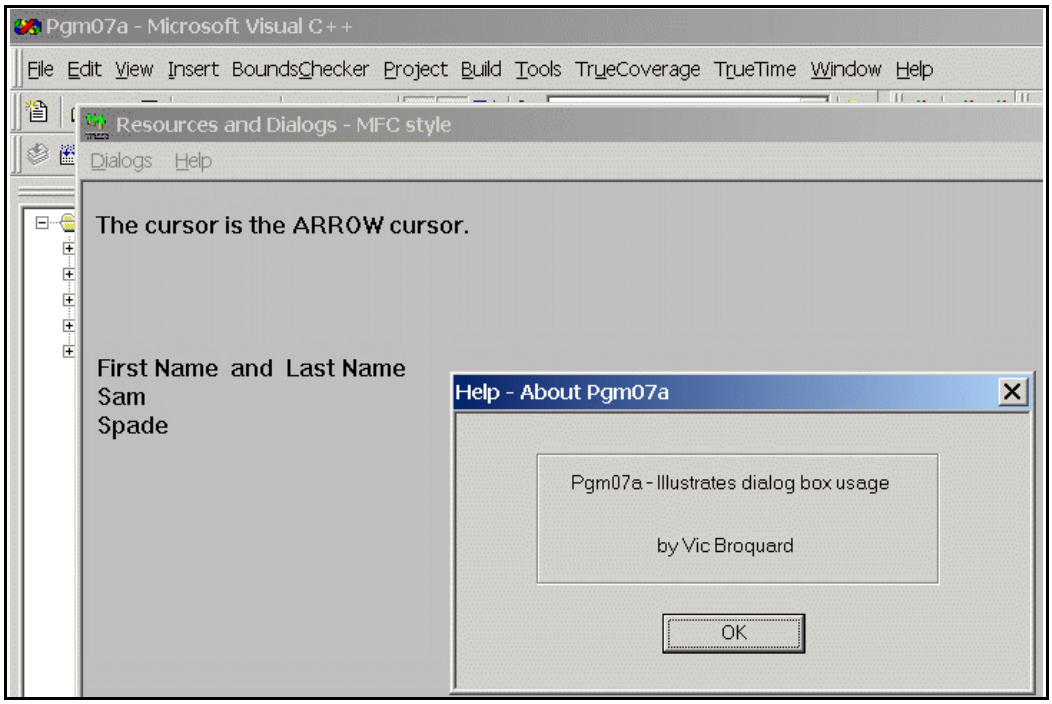

Figure 7.22 Help About Dialog

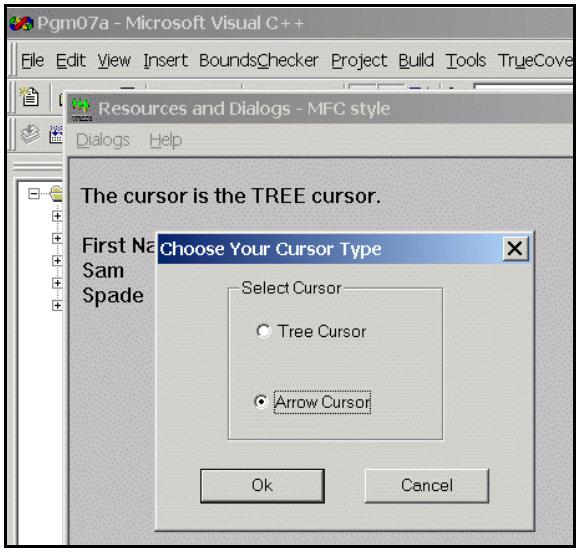

Figure 7.23 Choose Cursor Dialog

The Choose Cursor dialog shown in Figure 7.23 uses a Static Edge and is not centered. The Choose User Name dialog shown in Figure 7.24 uses both a Static and Client Edge.

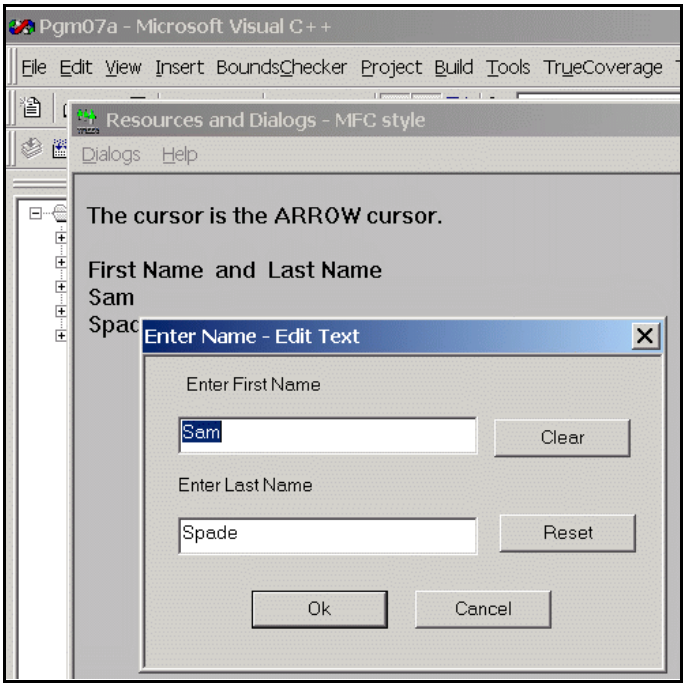

Figure 7.24 Choose User Name Dialog

The Choose Display Attributes Dialog shown in Figure 7.25 has a Client edge, is centered, and is 3D.

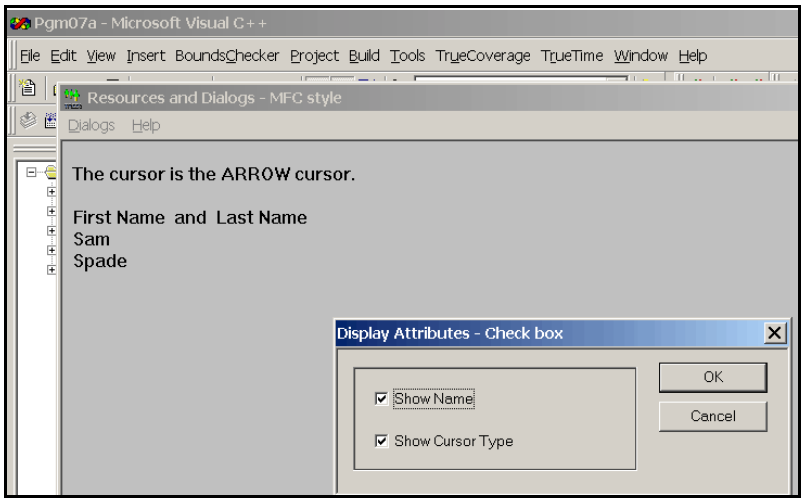

Figure 7.25 Choose Display Attributes Dialog

The Choose Line Dialog shown in Figure 7.26 uses all default style settings.

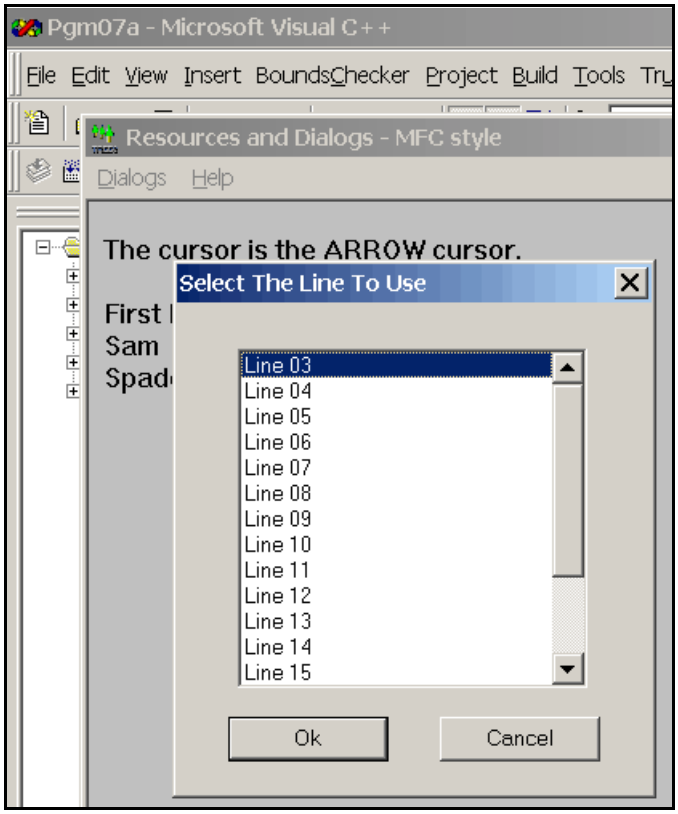

Figure 7.26 Choose Line Dialog

Finally, the Mouse Tracking Dialog in Figure 7.27 uses a 3D look with Static edge.

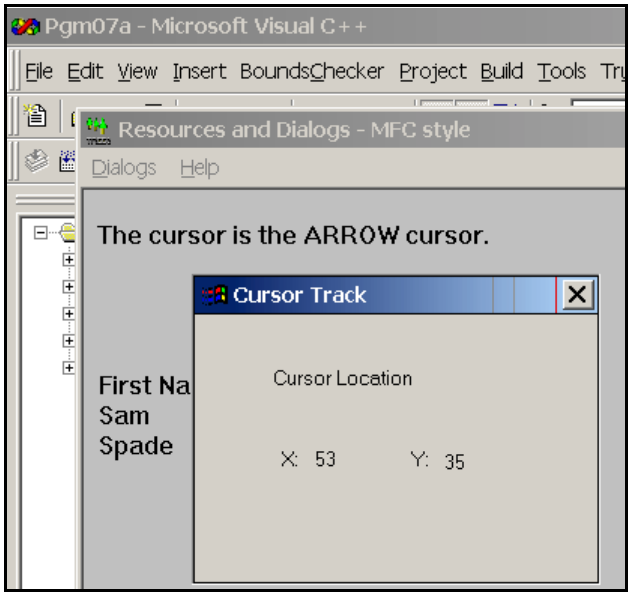

Figure 7.27 Mouse Tracking Modeless Dialog

Sometimes the data to be transferred into and out of a dialog is best handled by passing a pointer to a special transfer buffer structure. If there are numerous values to be passed, of if the values to be passed are within several different structures, of if the data needs to be altered in some manner, such as formatting, before it is given to the dialog, use a transfer structure to simplify it. I did just this with the Choose User Name dialog. Here is the special transfer buffer I made for it.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* NamesXfer.h                                                               *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #ifndef NAMESXFERH                                                     *
*  2 #define NAMESXFERH                                                     *
*  3                                                                        *
*  4 #define MAX_NAME_LEN 40   // max length of first and last names       *
*  5                                                                        *
*  6 // edit dialog transfer buffer - a convenient method to transfer      *
*  7 // data to/from a dialog when there is a lot of data or the           *
*  8 // data is inconvenient to pass or needs to be altered in someway     *
*  9 struct  TRANSFER_NAMES {                                              *
* 10  char first_name[MAX_NAME_LEN];                                       *
* 11  char last_name[MAX_NAME_LEN];                                        *
* 12 };                                                                    *
* 13                                                                       *
* 14 #endi                                                                 *
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

The application derived class is identical to the previous chapters and is not shown here. The **FrameWin** class definition defines the basic data. Pay particular attention to those values that are in some way altered by the various dialogs. These are shown in bold face below. Notice that a pointer to the **MouseTrackingDialog** must be saved because it is a modeless dialog and can exist beyond the function in which it is allocated and launched.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* FrameWin.h                                                               *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #ifndef FRAMEWIN                                                       *
*  2 #define FRAMEWIN                                                       *
*  3                                                                        *
*  4 #include "NamesXfer.h"                                                *
*  5                                                                        *
*  6 class MouseTrackingDialog;                                            *
*  7                                                                        *
*  8 /*************************************************************/*
*  9 /*                                                           */*
* 10 /* FrameWin Class Definition                                 */*
* 11 /*                                                           */*
* 12 /*************************************************************/*
* 13                                                                       *
* 14 class FrameWin : public CFrameWnd {                                   *
* 15                                                                       *
* 16  /*************************************************************/*
* 17  /*                                                           */*
```

```
* 18  /* Class Data Members                                          */*
* 19  /*                                                            */*
* 20  /************************************************************/*
* 21                                                                 *
* 22 protected:                                                      *
* 23 int  height;         // current client window height in pixels  *
* 24 int  width;          // current client window width in pixels   *
* 25                                                                 *
* 26 int  avg_caps_width; // average capital letter width           *
* 27 int  avg_char_width; // average character width                 *
* 28 int  avg_char_height;// average character height                *
* 29                                                                 *
* 30 int  which_cursor;   // radio button index of which btn is on   *
* 31 int  show_name;      // show names when TRUE - for checkboxes   *
* 32 int  show_cursor;    // show cursor when TRUE                   *
* 33 char firstName[MAX_NAME_LEN]; // user's first name              *
* 34 char lastName[MAX_NAME_LEN];  // user's last name               *
* 35                                                                 *
* 36 public: // transfer areas to/from dialogs                      *
* 37 // these need to be public so they can be accessed from the dlgs *
* 38 bool track_on;        // when true, displays mouse coordinates   *
* 39 char line[50][10];    // lines for listbox strings               *
* 40 int  linetot;        // total lines in array that are used      *
* 41 int  linenumnew;     // set by CmOk of ChooseLineDialog         *
* 42 int  linenum;        // current line number to show names on    *
* 43                                                                 *
* 44 protected:                                                      *
* 45 HCURSOR hcursor;     // current cursor in use                   *
* 46 HBRUSH  hbkgrndbrush;// current background brush                *
* 47 MouseTrackingDialog  *ptrMouseTracking; // ptr to modeless dialog*
* 48                                                                 *
* 49  /************************************************************/*
* 50  /*                                                          */*
* 51  /* Class Functions:                                         */*
* 52  /*                                                          */*
* 53  /************************************************************/*
* 54                                                                 *
* 55 public:                                                         *
* 56               FrameWin ();               // constructor         *
* 57               ~FrameWin () {}            // destructor          *
* 58                                                                 *
* 59 protected:                                                      *
* 60 afx_msg void  OnPaint ();                // paint the window    *
* 61 afx_msg int   OnCreate (LPCREATESTRUCT); // initial class members*
* 62 afx_msg void  OnDestroy ();              // delete MouseTracking *
* 63 afx_msg void  OnClose ();                // can app quit?       *
* 64 afx_msg void  OnSize (UINT, int, int);   // calc number lines   *
* 65 afx_msg void  OnMouseMove (UINT, CPoint);// track current pos   *
* 66                                                                 *
* 67 afx_msg void  CmChooseCursor ();         // choose which cursor  *
* 68 afx_msg void  CmChooseName ();           // enter user name     *
```

```
* 69 afx_msg void  CmChooseDisplay ();        // choose what to dsply *
* 70 afx_msg void  CmChooseLine ();           // pick which line      *
* 71 afx_msg void  CmMouseTracking ();        // start cursor tracking*
* 72 afx_msg void  CmAbout ();                // help about dialog    *
* 73                                                                  *
* 74 DECLARE_MESSAGE_MAP();                                          *
* 75 };                                                              *
* 76 #endif                                                          *
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

In the **FrameWin** constructor notice that all of the data members are initialized **before** the call to the **Create** function. Remember that **Create** eventually calls **OnCreate** which sends the first paint message. So some of these must be already set to their initial values. **OnCreate** also illustrates how you can set a different cursor than the one installed during the creation process.

OnDestroy checks to see if there is an instance of the **MouseTrackingDialog** still running when the application is shutting down. If so, it is deleted.

OnSize determines how many lines of text can be shown on the screen with the current window size. It then builds the array **lines** which contain the strings to be shown in the Choose Lines dialog.

OnMouseMove passes the coordinates of the mouse to the Display Tracking dialog, but only if that dialog exists.

Next, come the command handlers that launch the various dialogs. Let’s examine these along with the dialogs themselves, from the simplest to the more complex.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* FrameWin.cpp                                                       *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #include "stdafx.h"                                             *
*  2 #include "framewin.h"                                           *
*  3 #include "resource.h"                                           *
*  4 #include "MouseTrackingDialog.h"                               *
*  5 #include "ChooseNameDialog.h"                                  *
*  6 #include "DisplayAttributesDialog.h"                           *
*  7 #include "SelectCursorDialog.h"                                *
*  8 #include "ChooseLineDialog.h"                                  *
*  9                                                                 *
* 10 /*****************************************************************/*
* 11 /*                                                             */*
* 12 /* FrameWin Events Response Table                              */*
* 13 /*                                                             */*
* 14 /*****************************************************************/*
* 15                                                                 *
* 16 BEGIN_MESSAGE_MAP(FrameWin, CFrameWnd)                          *
* 17  ON_WM_SIZE ()                                                  *
```

```
* 18  ON_WM_PAINT ()                                                    *
* 19  ON_WM_CREATE ()                                                   *
* 20  ON_WM_CLOSE ()                                                    *
* 21  ON_WM_DESTROY ()                                                  *
* 22  ON_WM_MOUSEMOVE ()                                                *
* 23  ON_COMMAND(CM_CURSOR_DLG,   CmChooseCursor)                       *
* 24  ON_COMMAND(CM_NAME_DLG,     CmChooseName)                         *
* 25  ON_COMMAND(CM_ITEM_DLG,     CmChooseDisplay)                      *
* 26  ON_COMMAND(CM_LINE_DLG,     CmChooseLine)                         *
* 27  ON_COMMAND(CM_LOCATION_DLG, CmMouseTracking)                      *
* 28  ON_COMMAND(CM_ABOUT,        CmAbout)                              *
* 29 END_MESSAGE_MAP();                                                 *
* 30                                                                    *
* 31 /*****************************************************************/*
* 32 /*                                                               */*
* 33 /* Framewin: Construct the window object                         */*
* 34 /*                                                               */*
* 35 /*****************************************************************/*
* 36                                                                    *
* 37 FrameWin::FrameWin () : CFrameWnd () {                             *
* 38  ptrMouseTracking = 0; // set no MouseTrackingDialog allocated     *
* 39  track_on = false;  // indicate mouse position tracking is not on*
* 40  firstName[0] = 0;  // set for no inital user name                 *
* 41  lastName[0] = 0;                                                  *
* 42  show_name    = FALSE;  // init check box xfer buf                 *
* 43  show_cursor  = TRUE;   // show cursor type but not names          *
* 44  linenum      = 3;      // set default line for showing names      *
* 45                                                                    *
* 46  // set the default cursor to the Tree cursor                     *
* 47  which_cursor = SelectCursorDialog::Tree;                          *
* 48                                                                    *
* 49                                                                    *
* 50  DWORD style = WS_OVERLAPPEDWINDOW; // set basic window style      *
* 51  CString title;                                                    *
* 52  title.LoadString (IDS_MAINTITLE);  // load caption of window     *
* 53                                                                    *
* 54  Create ( AfxRegisterWndClass (                                    *
* 55                CS_VREDRAW | CS_HREDRAW,                            *
* 56                AfxGetApp()->LoadStandardCursor (IDC_ARROW),        *
* 57                ::CreateSolidBrush (GetSysColor(COLOR_WINDOW)),     *
* 58                AfxGetApp()->LoadIcon (IDI_TREE)),                  *
* 59           title,          // window caption                        *
* 60           style,          // wndclass DWORD style                  *
* 61           rectDefault,    // set initial window position           *
* 62           0,              // the parent window, here none          *
* 63           "MAINMENU");    // assign the main menu                  *
* 64                                                                    *
* 65  // load the Tree cursor and install it replacing Arrow cursor     *
* 66  hcursor = ::LoadCursor (AfxGetApp()->m_hInstance,                  *
* 67                          MAKEINTRESOURCE (IDC_TREE));              *
* 68  ::SetClassLong (m_hWnd, GCL_HCURSOR, (long) hcursor);             *
* 69 }                                                                  *
```

```
* 70                                                                  *
* 71 /*****************************************************************/*
* 72 /*                                                               */*
* 73 /* OnDestroy: remove dynamically allocated MouseTrackingDialog */*
* 74 /*                                                               */*
* 75 /*****************************************************************/*
* 76                                                                  *
* 77 void FrameWin::OnDestroy () {                                    *
* 78  if (ptrMouseTracking) { // only delete it if it still exists    *
* 79   delete ptrMouseTracking;                                       *
* 80  }                                                               *
* 81  CFrameWnd::OnDestroy ();                                        *
* 82 }                                                               *
* 83                                                                  *
* 84 /*****************************************************************/*
* 85 /*                                                              */*
* 86 /* OnCreate: get average character dimensions                   */*
* 87 /*                                                              */*
* 88 /*****************************************************************/*
* 89                                                                  *
* 90 int FrameWin::OnCreate (LPCREATESTRUCT lpCS) {                   *
* 91  if (CFrameWnd::OnCreate (lpCS) == -1) return -1;                *
* 92                                                                  *
* 93  TEXTMETRIC  tm;                                                 *
* 94                                                                  *
* 95  // get the system font's characteristics in tm                  *
* 96  CClientDC dc (this);         // acquire a DC                    *
* 97  dc.GetTextMetrics (&tm);     // get the information              *
* 98                                                                  *
* 99  // calculate average character parameters                       *
*100  avg_char_width  = tm.tmAveCharWidth;                            *
*101  avg_char_height = tm.tmHeight + tm.tmExternalLeading;           *
*102  avg_caps_width  = (tm.tmPitchAndFamily & 1 ? 3 : 2) *           *
*103                    avg_char_width / 2;                           *
*104                                                                  *
*105  // the following illustrates how to change the background color *
*106  // AFTER it has been set initially in the wndclass structure    *
*107  // you need to get the handle of the original brush so that it  *
*108  // can be deleted, if not, you will get a memory leak           *
*109  hbkgrndbrush = ::CreateSolidBrush (RGB (192, 192, 192));        *
*110                                                                  *
*111  // get old brush so we can delete it after installing new brush *
*112  HBRUSH oldbrush = (HBRUSH) ::GetClassLong (m_hWnd,              *
*113                                             GCL_HBRBACKGROUND);  *
*114  // install new brush                                            *
*115  ::SetClassLong (m_hWnd, GCL_HBRBACKGROUND, (long) hbkgrndbrush);*
*116  ::DeleteObject (oldbrush);                                      *
*117                                                                  *
*118  return 0;                                                       *
*119 }                                                               *
*120                                                                  *
*121 /*****************************************************************/*
```

```
*122 /*                                                                 */*
*123 /* OnSize: acquire the current dimensions of the client window */*
*124 /*    calc number of available lines for name showing          */*
*125 /*                                                                 */*
*126 /*******************************************************************/*
*127                                                                     *
*128 void FrameWin::OnSize (UINT, int cx, int cy) {                      *
*129  width = cx;  // save current height                                *
*130  height = cy; // save current width                                 *
*131                                                                     *
*132  char msg[10];                                                      *
*133  int  i, j;                                                         *
*134                                                                     *
*135  // now dynamically adjust the number of possible lines upon       *
*136  // which the names can be displayed:                              *
*137                                                                     *
*138  // retrieve printf control string "Line %02d"                     *
*139  LoadString (AfxGetApp()->m_hInstance, IDS_LINEMSG, msg,           *
*140              sizeof(msg));                                          *
*141                                                                     *
*142  // insert line choices into the list box lines strings that       *
*143  // will be loaded into the list box                               *
*144  // -3 for the cursor msgs; -2 since 3 lines of names              *
*145  j = height / avg_char_height - 5;                                  *
*146                                                                     *
*147  if (j<0) j=1; // force at least line 3                            *
*148  linetot = j;                                                       *
*149  for (i=0; i<j; i++) {                                              *
*150   wsprintf (line[i], msg, i+3);                                     *
*151  }                                                                  *
*152 }                                                                   *
*153                                                                     *
*154 /*******************************************************************/*
*155 /*                                                               */*
*156 /* OnMouseMove: display current mouse position                   */*
*157 /*                                                               */*
*158 /*******************************************************************/*
*159                                                                     *
*160 void      FrameWin::OnMouseMove (UINT, CPoint pt) {                 *
*161  // if tracking is on, force the modeless dialog to display pos    *
*162  if (track_on) ptrMouseTracking->ShowPos (pt);                      *
*163 }                                                                   *
*164                                                                     *
*165 /*******************************************************************/*
*166 /*                                                               */*
*167 /* OnPaint: display what the user has selected                   */*
*168 /*                                                               */*
*169 /*******************************************************************/*
*170                                                                     *
*171 void FrameWin::OnPaint () {                                        *
*172  CPaintDC dc (this);                                                *
*173  CString msg;                                                       *
```

```
*174  dc.SetBkMode (TRANSPARENT); // let background color show through*
*175                                                                  *
*176  if (show_cursor) {                                             *
*177   if (which_cursor == SelectCursorDialog::Tree)                  *
*178    msg.LoadString (IDS_ISTREE);                                 *
*179   else msg.LoadString (IDS_ISARROW);                            *
*180   dc.TextOut (avg_char_width, avg_char_height, msg);            *
*181  }                                                              *
*182                                                                  *
*183  if (show_name) {                                               *
*184   msg.LoadString (IDS_NAMEID);                                  *
*185   dc.TextOut (avg_char_width, avg_char_height*(linenum), msg);   *
*186   dc.TextOut (avg_char_width, avg_char_height*(linenum+1),       *
*187               firstName);                                       *
*188   dc.TextOut (avg_char_width, avg_char_height*(linenum+2),       *
*189               lastName);                                        *
*190  }                                                              *
*191 }                                                              *
*192                                                                  *
*193                                                                  *
*194 /*****************************************************************/*
*195 /*                                                             */*
*196 /* OnClose: determine if the app can be shut down              */*
*197 /*                                                             */*
*198 /*****************************************************************/*
*199                                                                  *
*200 void FrameWin::OnClose () {                                     *
*201  CString msgtitle;                                              *
*202  CString msgtext;                                               *
*203  msgtext.LoadString (IDS_MSG_QUIT);                             *
*204  msgtitle.LoadString (IDS_MSG_QUERY);                           *
*205  if (MessageBox (msgtext, msgtitle, MB_YESNO | MB_ICONQUESTION)  *
*206      == IDYES) {                                                *
*207   CFrameWnd::OnClose ();                                        *
*208  }                                                              *
*209 }                                                              *
*210                                                                  *
*211 /*****************************************************************/*
*212 /*                                                             */*
*213 /* CmAbout: Help About dialog                                  */*
*214 /*                                                             */*
*215 /*****************************************************************/*
*216                                                                  *
*217 void      FrameWin::CmAbout () {                                 *
*218  CDialog aboutdlg ("IDD_ABOUT", this);                          *
*219  aboutdlg.DoModal ();                                           *
*220 }                                                              *
*221                                                                  *
*222 /*****************************************************************/*
*223 /*                                                             */*
*224 /* CmMouseTrack: cursor location modeless dialog activation    */*
*225 /*  No data xfer - ShowPos updates dlg controls from mouse move*/*
```

```
*226 /*                                                                  */*
*227 /********************************************************************/*
*228                                                                     *
*229 void      FrameWin::CmMouseTracking () {                            *
*230  if (!track_on) {          // avoid multiple instances of the dlg*
*231   track_on = true;         // indicate tracking is active        *
*232   ptrMouseTracking = new MouseTrackingDialog (this);              *
*233   POINT p;                                                        *
*234   ::GetCursorPos (&p);     // retrieve current mouse position    *
*235   CPoint pt (p);           // convert to CPoint                  *
*236   ptrMouseTracking->ShowPos (pt);         // display new position*
*237   ptrMouseTracking->ShowWindow (SW_SHOW); // make dialog visible *
*238  }                                                                *
*239 }                                                                 *
*240                                                                     *
*241 /********************************************************************/*
*242 /*                                                                  */*
*243 /* CmChooseName: enter names dialog                                */*
*244 /*   We fill a xfer buffer with the needed data and give to dlg*/*
*245 /*   The dlg replaces the xfer buffer with results              */*
*246 /*                                                                  */*
*247 /********************************************************************/*
*248                                                                     *
*249 void      FrameWin::CmChooseName () {                               *
*250  TRANSFER_NAMES  xfer_edit_names;  // edit names transfer buffer *
*251  // copy into transfer buf the current name in use                 *
*252  strcpy (xfer_edit_names.first_name, firstName);                    *
*253  strcpy (xfer_edit_names.last_name, lastName);                      *
*254                                                                     *
*255  ChooseNameDialog dlg (&xfer_edit_names, this);                     *
*256  if (dlg.DoModal () == IDOK) {                                      *
*257   // copy the new data in the xfer buffer to the real db location*
*258   strcpy (firstName, xfer_edit_names.first_name);                   *
*259   strcpy (lastName, xfer_edit_names.last_name);                     *
*260   Invalidate(); // force the new name strings to be shown         *
*261  }                                                                 *
*262 }                                                                  *
*263                                                                     *
*264 /********************************************************************/*
*265 /*                                                                  */*
*266 /* CmChooseCursor: choose cursor radio button dialog              */*
*267 /*    After creating an instance of the dlg, we must fill the  */*
*268 /*    public dlg data member with the initial data; and if ok, */*
*269 /*    we must copy the result out of the dlg data member       */*
*270 /*                                                                  */*
*271 /*  Uses a public enum of SelectCursorDialog to id the cursor  */*
*272 /*                                                                  */*
*273 /********************************************************************/*
*274                                                                     *
*275 void      FrameWin::CmChooseCursor () {                             *
*276  SelectCursorDialog dlg (this);                                     *
*277  // store our current settings into dlg's public data member      *
```

```
 dlg.m_which_btn_is_on = which_cursor;
 if (dlg.DoModal () == IDOK) { // create & execute dlg
  // update our members from dlg's public data member
  which_cursor = dlg.m_which_btn_is_on;
  // install the correct cursor
  if (which_cursor == SelectCursorDialog::Tree) {
   hcursor = ::LoadCursor (AfxGetApp()->m_hInstance,
                           MAKEINTRESOURCE (IDC_TREE));
   ::SetClassLong (GetSafeHwnd(), GCL_HCURSOR, (long) hcursor);
  }
  else {
   hcursor = AfxGetApp()->LoadStandardCursor (IDC_ARROW);
   ::SetClassLong (m_hWnd, GCL_HCURSOR, (long) hcursor);
  }
  Invalidate (); // force paint to display msg of new cursor
 }
}

/*************************************************************/
/*                                                           */
/* CmChooseDisplay: choose which items to display in paint   */
/*    After creating an instance of the dlg, we must fill the */
/*    public dlg data members with the initial data and if ok, */
/*    we must copy the results out of the dlg data members   */
/*                                                           */
/*************************************************************/

void      FrameWin::CmChooseDisplay () {
 DisplayAttributesDialog dlg (this);
 // transfer our current settings into dlg's public members
 dlg.m_showName   = show_name;
 dlg.m_showCursor = show_cursor;
 if (dlg.DoModal () == IDOK) {    //  create & execute dlg
  // update our members from the dlg's public data members
  show_name   = dlg.m_showName ;
  show_cursor = dlg.m_showCursor;
  Invalidate (); // repaint window with these options
 }
}

/*************************************************************/
/*                                                           */
/* CmChooseLine: pick which line on which to display name    */
/*    dlg uses GetParent and typecast to FrameWin to access  */
/*    the public data members that contain available lines and */
/*    the line on which to display the name                  */
/*                                                           */
/*************************************************************/

void      FrameWin::CmChooseLine () {
 ChooseLineDialog dlg (this);
```

```
*330  if (dlg.DoModal () == IDOK) {                                    *
*331   if (linenumnew >= 0) {      // if OK, set our new line number  *
*332    linenum = linenumnew + 3;  // save new line number            *
*333    Invalidate();              // force names to show on this line*
*334   }                                                               *
*335  }                                                                *
*336 }                                                                 *
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

The simplest dialog is the Help About dialog. Notice that there is no class for it and that there is no #define for its name, **IDD_ABOUT**, either. This is an example of the simplest possible dialog — one that has no data to be transferred in or out of it. One can just wrap the basic **CDialog** class around it and run it.

```
CDialog aboutdlg ("IDD_ABOUT", this);
aboutdlg.DoModal ();
```

Next, let's look at the Choose Names dialog. This one is an example in which a transfer buffer is used to simplify data transfer. This is a very common approach — copy all the needed fields into a structure instance and pass its address to the constructor who saves it in a protected data member for all the other dialog functions to use as needed. When the dialog completes, the caller must move the results from this structure instance back into their proper variables. **FrameWin** calls it this way.

```
TRANSFER_NAMES  xfer_edit_names;  // edit names transfer buffer
strcpy (xfer_edit_names.first_name, firstName);
strcpy (xfer_edit_names.last_name, lastName);
ChooseNameDialog dlg (&xfer_edit_names, this);
if (dlg.DoModal () == IDOK) {
 strcpy (firstName, xfer_edit_names.first_name);
 strcpy (lastName, xfer_edit_names.last_name);
 Invalidate(); // force the new name strings to be shown
}
```

Notice that the final **Invalidate** function call is what actually gets the new data painted on the screen.

Here is the Choose Names dialog definition. In bold are the saved transfer buffer pointer and the corresponding C++ dialog data members that are used to transfer the data to the edit controls.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* ChooseNameDialog.h                                                 *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #if !defined(AFX_ChooseNameDialog_H__CD484406_4367_4679_9BCC_0796*
*  2 #define AFX_ChooseNameDialog_H__CD484406_4367_4679_9BCC_079618154*
*  3                                                                  *
*  4 #if _MSC_VER > 1000                                             *
*  5 #pragma once                                                    *
*  6 #endif // _MSC_VER > 1000                                       *
*  7                                                                 *
```

```
*  8 #include "resource.h"                                                *
*  9 #include "NamesXfer.h"                                              *
* 10                                                                      *
* 11 /*************************************************************/*
* 12 /*                                                           */*
* 13 /* ChooseNameDialog - enter user first and last name         */*
* 14 /*  Address of a caller transfer buffer is passed and stored */*
* 15 /*  the edit controls are filled from this and results replace */*
* 16 /*  the original data in the xfer buffer                     */*
* 17 /*                                                           */*
* 18 /*  Uses Client Edge and Static Edge styles - no 3d no centered*/*
* 19 /*                                                           */*
* 20 /*************************************************************/*
* 21                                                                      *
* 22 class ChooseNameDialog : public CDialog {                           *
* 23 // Construction                                                      *
* 24 public:                                                             *
* 25  ChooseNameDialog(TRANSFER_NAMES* ptrb, CWnd* pParent = NULL);   *
* 26  TRANSFER_NAMES* ptrnames;                                          *
* 27                                                                     *
* 28 // Dialog Data                                                      *
* 29  //{{AFX_DATA(ChooseNameDialog)                                    *
* 30  enum { IDD = IDD_EDITDLG };                                        *
* 31  CString m_firstName;                                               *
* 32  CString m_lastName;                                                *
* 33  //}}AFX_DATA                                                      *
* 34                                                                     *
* 35 // Overrides                                                       *
* 36  // ClassWizard generated virtual function overrides              *
* 37  //{{AFX_VIRTUAL(ChooseNameDialog)                                 *
* 38 protected:                                                          *
* 39  virtual void DoDataExchange(CDataExchange* pDX);                   *
* 40  //}}AFX_VIRTUAL                                                   *
* 41                                                                     *
* 42 // Implementation                                                  *
* 43 protected:                                                         *
* 44                                                                     *
* 45  // Generated message map functions                                *
* 46  //{{AFX_MSG(ChooseNameDialog)                                     *
* 47  virtual void OnOK();                                               *
* 48  afx_msg void OnReset();                                            *
* 49  afx_msg void OnClear();                                            *
* 50  //}}AFX_MSG                                                       *
* 51  DECLARE_MESSAGE_MAP()                                             *
* 52 };                                                                  *
* 53                                                                     *
* 54 //{{AFX_INSERT_LOCATION}}                                         *
* 55 #endif // !defined(AFX_ChooseNameDialog_H__CD484406_4367_4679_9BC*
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

The constructor copies the initial values for the first and last names into the member variables and saves the pointer to the transfer buffer. **DoDataExchange** is called from **OnInitDialog** to load the strings into the edit controls initially. I will discuss the **DDV**, dynamic data validators, later on. Since we need to copy the results back into the passed structure instance, we respond to the Ok button in **OnOk**. The Reset button causes **OnReset** to copy the original data in the transfer buffer back into the edit controls. The Clear button causes the edit controls to be emptied. Notice that I just copy the needed data into the C++ data members and then call **UpdateData** to get the data transferred. This is shown in bold.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* ChooseNameDialog.cpp                                                      *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #include "stdafx.h"                                                    *
*  2 #include "pgm07a.h"                                                   *
*  3 #include "ChooseNameDialog.h"                                         *
*  4                                                                        *
*  5 #ifdef _DEBUG                                                         *
*  6 #define new DEBUG_NEW                                                 *
*  7 #undef THIS_FILE                                                      *
*  8 static char THIS_FILE[] = __FILE__;                                   *
*  9 #endif                                                                *
* 10                                                                       *
* 11 /*************************************************************/*
* 12 /*                                                           */*
* 13 /* ChooseNameDialog: enters user's first and last name       */*
* 14 /*                                                           */*
* 15 /*************************************************************/*
* 16                                                                       *
* 17 ChooseNameDialog::ChooseNameDialog (TRANSFER_NAMES* ptrb,             *
* 18                                     CWnd* pParent)                    *
* 19                 : CDialog(ChooseNameDialog::IDD, pParent) {           *
* 20  //{{AFX_DATA_INIT(ChooseNameDialog)                                  *
* 21  m_firstName = ptrb->first_name;                                      *
* 22  m_lastName = ptrb->last_name;                                        *
* 23  //}}AFX_DATA_INIT                                                   *
* 24  ptrnames = ptrb;                                                     *
* 25 }                                                                     *
* 26                                                                       *
* 27 void ChooseNameDialog::DoDataExchange(CDataExchange* pDX) {           *
* 28  CDialog::DoDataExchange(pDX);                                        *
* 29  //{{AFX_DATA_MAP(ChooseNameDialog)                                   *
* 30  DDX_Text(pDX, IDC_FIRSTNAME, m_firstName);                           *
* 31  DDV_MaxChars(pDX, m_firstName, MAX_NAME_LEN-1);                      *
* 32  DDX_Text(pDX, IDC_LASTNAME, m_lastName);                             *
* 33  DDV_MaxChars(pDX, m_lastName, MAX_NAME_LEN-1);                       *
* 34  //}}AFX_DATA_MAP                                                    *
* 35 }                                                                     *
* 36                                                                       *
* 37 BEGIN_MESSAGE_MAP(ChooseNameDialog, CDialog)                          *
* 38  //{{AFX_MSG_MAP(ChooseNameDialog)                                    *
* 39  ON_BN_CLICKED(IDC_RESETBUTTON, OnReset)                              *
```

```
* 40  ON_BN_CLICKED(IDC_CLEARBUTTON, OnClear)                              *
* 41  //}}AFX_MSG_MAP                                                      *
* 42 END_MESSAGE_MAP()                                                     *
* 43                                                                       *
* 44 void ChooseNameDialog::OnOK() {                                       *
* 45  UpdateData (TRUE);                                                   *
* 46  strcpy (ptrnames->first_name, m_firstName);                          *
* 47  strcpy (ptrnames->last_name, m_lastName);                            *
* 48  CDialog::OnOK();                                                     *
* 49 }                                                                     *
* 50                                                                       *
* 51 void ChooseNameDialog::OnReset() {                                    *
* 52  m_firstName = ptrnames->first_name;                                  *
* 53  m_lastName = ptrnames->last_name;                                    *
* 54  UpdateData (FALSE);                                                  *
* 55 }                                                                     *
* 56                                                                       *
* 57 void ChooseNameDialog::OnClear() {                                    *
* 58  m_firstName = "";                                                    *
* 59  m_lastName = "";                                                     *
* 60  UpdateData (FALSE);                                                  *
* 61 }                                                                     *
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

The next two dialogs, Choose Display Attributes and Choose Cursor, are handled in much the same way. Both follow the Third Approach in which the caller stored the needed initial values into the public data members before executing the dialog and then the caller retrieves the results from these same data members when the dialog finishes.

For check boxes, the transfer buffer is an **int** containing usually 0 or 1. However, there are really three values possible: **MF_CHECKED**, **MF_UNCHECKED**, **MF_GRAYED**. The first yields 0 while the second yields 1.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* DisplayAttributesDialog.h                                               *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #if !defined(AFX_DisplayAttributesDialog_H__553109ED_C157_434B_B9*
*  2 #define AFX_DisplayAttributesDialog_H__553109ED_C157_434B_B936_51*
*  3                                                                     *
*  4 #if _MSC_VER > 1000                                                 *
*  5 #pragma once                                                        *
*  6 #endif // _MSC_VER > 1000                                           *
*  7                                                                     *
*  8 #include "resource.h"                                              *
*  9                                                                     *
* 10 /*************************************************************/*
* 11 /*                                                           */*
* 12 /* DisplayAttributesDialog: choose which items to display   */   *
* 13 /*                                                           */*
* 14 /*  Caller stores initial values and Caller retrieves new ones */*
* 15 /*                                                           */*
```

```
* 16 /*  Uses Client Edge style and 3d and is centered              */*
* 17 /*                                                             */*
* 18 /*************************************************************/*
* 19                                                                 *
* 20 class DisplayAttributesDialog : public CDialog {                *
* 21 // Construction                                                 *
* 22 public:                                                         *
* 23  DisplayAttributesDialog(CWnd* pParent = NULL);                 *
* 24                                                                 *
* 25 // Dialog Data                                                  *
* 26  //{{AFX_DATA(DisplayAttributesDialog)                         *
* 27  enum { IDD = IDD_CHECKBOX };                                   *
* 28  BOOL m_showCursor;                                            *
* 29  BOOL m_showName;                                              *
* 30  //}}AFX_DATA                                                  *
* 31                                                                 *
* 32 // Overrides                                                   *
* 33  // ClassWizard generated virtual function overrides           *
* 34  //{{AFX_VIRTUAL(DisplayAttributesDialog)                       *
* 35 protected:                                                      *
* 36  virtual void DoDataExchange(CDataExchange* pDX);               *
* 37  //}}AFX_VIRTUAL                                               *
* 38                                                                 *
* 39 // Implementation                                              *
* 40 protected:                                                     *
* 41                                                                 *
* 42  // Generated message map functions                            *
* 43  //{{AFX_MSG(DisplayAttributesDialog)                          *
* 44  //}}AFX_MSG                                                   *
* 45  DECLARE_MESSAGE_MAP()                                         *
* 46 };                                                             *
* 47                                                                 *
* 48 //{{AFX_INSERT_LOCATION}}                                     *
* 49                                                                 *
* 50 #endif // !defined(AFX_DisplayAttributesDialog_H__553109ED_C157_4*
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

Notice that the dialog is very simple and that we do not even need to respond to the Ok button.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* DisplayAttributesDialog.cpp                                       *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #include "stdafx.h"                                            *
*  2 #include "pgm07a.h"                                           *
*  3 #include "DisplayAttributesDialog.h"                          *
*  4                                                                *
*  5 #ifdef _DEBUG                                                 *
*  6 #define new DEBUG_NEW                                        *
*  7 #undef THIS_FILE                                              *
*  8 static char THIS_FILE[] = __FILE__;                           *
*  9 #endif                                                        *
```

```
* 10                                                                    *
* 11 /*********************************************************************/*
* 12 /*                                                                  */*
* 13 /* DisplayAttributesDialog: choose what is shown on screen          */*
* 14 /*                                                                  */*
* 15 /*********************************************************************/*
* 16                                                                    *
* 17 DisplayAttributesDialog::DisplayAttributesDialog (CWnd* pParent)   *
* 18              : CDialog (DisplayAttributesDialog::IDD, pParent) {*
* 19  //{{AFX_DATA_INIT(DisplayAttributesDialog)                        *
* 20  m_showCursor = FALSE;                                             *
* 21  m_showName = FALSE;                                               *
* 22  //}}AFX_DATA_INIT                                                *
* 23 }                                                                  *
* 24                                                                    *
* 25 void DisplayAttributesDialog::DoDataExchange(CDataExchange* pDX){*
* 26  CDialog::DoDataExchange(pDX);                                     *
* 27  //{{AFX_DATA_MAP(DisplayAttributesDialog)                         *
* 28  DDX_Check(pDX, IDC_CHECK_CURSOR, m_showCursor);                   *
* 29  DDX_Check(pDX, IDC_CHECK_NAME, m_showName);                       *
* 30  //}}AFX_DATA_MAP                                                 *
* 31 }                                                                  *
* 32                                                                    *
* 33 BEGIN_MESSAGE_MAP(DisplayAttributesDialog, CDialog)                *
* 34  //{{AFX_MSG_MAP(DisplayAttributesDialog)                          *
* 35  //}}AFX_MSG_MAP                                                  *
* 36 END_MESSAGE_MAP()                                                 *
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

The Choose Cursor dialog has one extra twist. Since there are two possible cursors, rather than trying to remember which numerical value means which cursor, I chose to use an enumerated data type. This is a very common action, public class **enum**s. It is shown in bold.

The transfer buffer for a group of radio buttons is an **int** containing the zero-based offset of which button is on. A value of -1 means no selection has yet been made.

> **Design Rule 33: It is vitally important that only the first radio button of the group of buttons has the WS_GROUP style attribute in the resource file. It and the remainder of the radio buttons in the group do have the WS_TABSTOP style**

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* SelectCursorDialog.h                                                 *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #if !defined(AFX_SELECTCURSORDIALOG_H__A68E1D7A_EE22_4B0B_B16A_D3*
*  2 #define AFX_SELECTCURSORDIALOG_H__A68E1D7A_EE22_4B0B_B16A_D3CE3E8*
*  3                                                                    *
*  4 #if _MSC_VER > 1000                                               *
*  5 #pragma once                                                      *
```

```
*  6 #endif // _MSC_VER > 1000                                             *
*  7                                                                       *
*  8 #include "resource.h"                                                 *
*  9                                                                       *
* 10 /////////////////////////////////////////////////////////////////////*
* 11 //                                                                   */*
* 12 // SelectCursorDialog: Select which cursor to use                    */*
* 13 //                                                                   */*
* 14 // Caller stores initial cursor value before running dialog         */*
* 15 // Caller retrieves the new selection                               */*
* 16 //                                                                   */*
* 17 // Uses Static Edge and no 3d no centered                           */*
* 18 //                                                                   */*
* 19 /////////////////////////////////////////////////////////////////////*
* 20                                                                       *
* 21 class SelectCursorDialog : public CDialog {                          *
* 22 // Construction                                                       *
* 23 public:                                                              *
* 24  SelectCursorDialog(CWnd* pParent = NULL);                           *
* 25                                                                       *
* 26  // convenience public enum to determine which button is on          *
* 27  enum Order {Tree, Arrow};                                           *
* 28                                                                       *
* 29 // Dialog Data                                                       *
* 30  //{{AFX_DATA(SelectCursorDialog)                                    *
* 31  enum { IDD = IDD_RADIO };                                           *
* 32  int m_which_btn_is_on;                                              *
* 33  //}}AFX_DATA                                                       *
* 34                                                                       *
* 35 // Overrides                                                         *
* 36  // ClassWizard generated virtual function overrides                 *
* 37  //{{AFX_VIRTUAL(SelectCursorDialog)                                 *
* 38 protected:                                                           *
* 39  virtual void DoDataExchange(CDataExchange* pDX);                    *
* 40  //}}AFX_VIRTUAL                                                    *
* 41                                                                       *
* 42 // Implementation                                                    *
* 43 protected:                                                           *
* 44                                                                       *
* 45  // Generated message map functions                                  *
* 46  //{{AFX_MSG(SelectCursorDialog)                                     *
* 47  //}}AFX_MSG                                                        *
* 48  DECLARE_MESSAGE_MAP()                                               *
* 49 };                                                                   *
* 50 //{{AFX_INSERT_LOCATION}}                                           *
* 51 #endif // !defined(AFX_SELECTCURSORDIALOG_H__A68E1D7A_EE22_4B0B_B*
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

Again the implementation is very simple because the caller is responsible for setting the initial values and getting the results.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
```

```
* SelectCursorDialog.cpp                                                    *
/*************************************************************************1
*  1 #include "stdafx.h"                                                    *
*  2 #include "pgm07a.h"                                                   *
*  3 #include "SelectCursorDialog.h"                                       *
*  4                                                                       *
*  5 #ifdef _DEBUG                                                         *
*  6 #define new DEBUG_NEW                                                *
*  7 #undef THIS_FILE                                                      *
*  8 static char THIS_FILE[] = __FILE__;                                   *
*  9 #endif                                                                *
* 10                                                                       *
* 11 /*************************************************************/*
* 12 /*                                                           */*
* 13 /* SelectCursorDialog: select which mouse cursor to use      */*
* 14 /*                                                           */*
* 15 /*************************************************************/*
* 16                                                                       *
* 17 SelectCursorDialog::SelectCursorDialog (CWnd* pParent)                *
* 18                   : CDialog(SelectCursorDialog::IDD, pParent) {       *
* 19  //{{AFX_DATA_INIT(SelectCursorDialog)                               *
* 20  m_which_btn_is_on = -1;                                              *
* 21  //}}AFX_DATA_INIT                                                   *
* 22 }                                                                     *
* 23                                                                       *
* 24 void SelectCursorDialog::DoDataExchange(CDataExchange* pDX) {         *
* 25  CDialog::DoDataExchange(pDX);                                        *
* 26  //{{AFX_DATA_MAP(SelectCursorDialog)                                *
* 27  DDX_Radio(pDX, IDC_RADIOBUTTON_TREE, m_which_btn_is_on);             *
* 28  //}}AFX_DATA_MAP                                                    *
* 29 }                                                                     *
* 30                                                                       *
* 31 BEGIN_MESSAGE_MAP(SelectCursorDialog, CDialog)                        *
* 32  //{{AFX_MSG_MAP(SelectCursorDialog)                                 *
* 33  //}}AFX_MSG_MAP                                                     *
* 34 END_MESSAGE_MAP()                                                    *
.*************************************************************************-
```

Let's review again how **FrameWin** calls this dialog and how the public **enum** operates. Notice how it sets the dialog's public data before it executes the dialog. Notice also how it obtains the results from the dialog. This is precisely how we must work with the Windows Common dialogs in the next chapter.

```
SelectCursorDialog dlg (this);
dlg.m_which_btn_is_on = which_cursor;
if (dlg.DoModal () == IDOK) { // create & execute dlg
 which_cursor = dlg.m_which_btn_is_on;
 if (which_cursor == SelectCursorDialog::Tree) {
  ...
```

The Choose Line dialog uses a list box which shows text messages indicating on which line the names are to be shown. Sorting is not going to be a problem in this case because of the nature of the strings themselves. They all say Line 03, Line 04, and so on. Since the array of strings was built sequentially, they are in alphabetical order anyway.

A list box must be populated with the strings that it is to show. This must be done when the actual list box exists which is after the **CDialog::OnInitDialog** call is complete. By having the **DDX** mechanism automatically tie the class **CListBox** data member, **m_list**, to the actual list box itself, we can then call member functions to work with the list box. The **AddString** function adds a string (either a **char*** or a **CString**) to the list box. The function **SetCurSel** sets a current selection in the list box. Here I chose to set as the current selection the line that the **FrameWin** is currently using. The function **GetCurSel** returns the current selection. Both of these are zero-based integers. That is, if the user selects the first string in the list box, the index returned is 0. Note that one can also have multi-selection list boxes, but that is beyond this text.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* ChooseLineDialog.h                                                         *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #if !defined(AFX_ChooseLineDialog_H__D9A22B25_BC15_4B5B_8AC5_9870*
*  2 #define AFX_ChooseLineDialog_H__D9A22B25_BC15_4B5B_8AC5_987013862*
*  3                                                                          *
*  4 #include "resource.h"                                                   *
*  5                                                                          *
*  6 #if _MSC_VER > 1000                                                     *
*  7 #pragma once                                                            *
*  8 #endif // _MSC_VER > 1000                                               *
*  9                                                                          *
* 10                                                                         *
* 11 class ChooseLineDialog : public CDialog {                               *
* 12 // Construction                                                         *
* 13 public:                                                                 *
* 14  ChooseLineDialog(CWnd* pParent = NULL);                                *
* 15                                                                         *
* 16 // Dialog Data                                                         *
* 17  //{{AFX_DATA(ChooseLineDialog)                                        *
* 18  enum { IDD = IDD_LISTBOX };                                            *
* 19  ClistBox m_list;                                                       *
* 20  //}}AFX_DATA                                                          *
* 21                                                                         *
* 22                                                                         *
* 23 // Overrides                                                           *
* 24  // ClassWizard generated virtual function overrides                    *
* 25  //{{AFX_VIRTUAL(ChooseLineDialog)                                      *
* 26 protected:                                                              *
* 27  virtual void DoDataExchange(CDataExchange* pDX);                       *
* 28  //}}AFX_VIRTUAL                                                       *
* 29                                                                         *
* 30 // Implementation                                                       *
* 31 protected:                                                              *
```

```
* 32                                                                    *
* 33  // Generated message map functions                                *
* 34  //{{AFX_MSG(ChooseLineDialog)                                     *
* 35  virtual void OnOK();                                              *
* 36  virtual BOOL OnInitDialog();                                      *
* 37  //}}AFX_MSG                                                       *
* 38  DECLARE_MESSAGE_MAP()                                             *
* 39 };                                                                 *
* 40                                                                    *
* 41 //{{AFX_INSERT_LOCATION}}                                         *
* 42                                                                    *
* 43 #endif // !defined(AFX_ChooseLineDialog_H__D9A22B25_BC15_4B5B_8AC*
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

This implementation shows how a dialog can get a pointer to its parent or caller and then access public data members of the caller. This is a more limited technique because the dialog must include the caller class header and is thus tied directly to a single parent and is not easily reused in other situations. However, this situation does occur, so make sure you follow how this dialog gets access to the **FrameWin** variables.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* ChooseLineDialog.cpp                                                 *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #include "stdafx.h"                                               *
*  2 #include "pgm07a.h"                                              *
*  3 #include "ChooseLineDialog.h"                                    *
*  4                                                                   *
*  5 #include "FrameWin.h" // requires intimate knowledge of its data *
*  6                                                                   *
*  7 #ifdef _DEBUG                                                     *
*  8 #define new DEBUG_NEW                                            *
*  9 #undef THIS_FILE                                                  *
* 10 static char THIS_FILE[] = __FILE__;                               *
* 11 #endif                                                           *
* 12                                                                  *
* 13 /*****************************************************************/*
* 14 /*                                                               */*
* 15 /* ChooseLineDialog - pick which line on which to display names*/*
* 16 /*                                                               */*
* 17 /* Uses GetParent to access FrameWin's public data members     */*
* 18 /* Uses all default styles and no 3d or centered               */*
* 19 /*                                                               */*
* 20 /*****************************************************************/*
* 21                                                                  *
* 22 ChooseLineDialog::ChooseLineDialog(CWnd* pParent)                *
* 23                 : CDialog(ChooseLineDialog::IDD, pParent) {      *
* 24  //{{AFX_DATA_INIT(ChooseLineDialog)                             *
* 25  //}}AFX_DATA_INIT                                              *
* 26 }                                                                *
* 27                                                                  *
* 28 void ChooseLineDialog::DoDataExchange(CDataExchange* pDX) {      *
* 29  CDialog::DoDataExchange(pDX);                                   *
```

```
* 30  //{{AFX_DATA_MAP(ChooseLineDialog)                                  *
* 31  DDX_Control(pDX, IDC_LISTBOX, m_list);                             *
* 32  //}}AFX_DATA_MAP                                                   *
* 33 }                                                                   *
* 34                                                                     *
* 35 BEGIN_MESSAGE_MAP(ChooseLineDialog, CDialog)                        *
* 36  //{{AFX_MSG_MAP(ChooseLineDialog)                                  *
* 37  //}}AFX_MSG_MAP                                                   *
* 38 END_MESSAGE_MAP()                                                  *
* 39                                                                     *
* 40 void ChooseLineDialog::OnOK() {                                     *
* 41  // if no selection is made, abort Ok                               *
* 42  if (m_list.GetCurSel() == LB_ERR) return;                          *
* 43                                                                     *
* 44  // get a ptr to the parent                                         *
* 45  FrameWin *ptrparent = (FrameWin*) (GetParent ());                   *
* 46  if (!ptrparent) return;                                            *
* 47                                                                     *
* 48  // save the current user selection index                           *
* 49  ptrparent->linenumnew = m_list.GetCurSel ();                       *
* 50                                                                     *
* 51  CDialog::OnOK();                                                   *
* 52 }                                                                   *
* 53                                                                     *
* 54 BOOL ChooseLineDialog::OnInitDialog() {                             *
* 55  CDialog::OnInitDialog();                                           *
* 56                                                                     *
* 57  // get a ptr to the parent                                         *
* 58  FrameWin *ptrparent = (FrameWin*) (GetParent ());                   *
* 59  if (!ptrparent) return TRUE;                                       *
* 60                                                                     *
* 61  // fill up list box from parent's line array                       *
* 62  for (int i=0; i<ptrparent->linetot; i++) {                         *
* 63   m_list.AddString (ptrparent->line[i]);                            *
* 64   // set the current line in use as the selected one                *
* 65   if (i+3 == ptrparent->linenum) m_list.SetCurSel (i);              *
* 66  }                                                                  *
* 67  return TRUE;                                                      *
* 68 }                                                                   *
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

All of the above were modal dialogs. When they are running, the application is forced to wait until the dialog finishes. Sometimes, the application needs to continue to execute while the dialog is present, as in the Find/Replace dialog. Thus, the Mouse Tracking Dialog is modeless so that it can remain on the screen and actively tracking the mouse while the user does other actions. This means that we must dynamically construct an instance of the dialog and save a class member pointer to it so that access to the dialog is available from other member functions.

Since its purpose is to track the location of the mouse in the **FrameWin**, **OnMouseMove** must somehow get the dialog updated with each new mouse location. This is most easily done

my calling a helper function of the dialog, **ShowPos**. Also, note that the user could shut down the application while the dialog is still in operation. Thus, in **OnDestroy**, we must check for this situation and manually delete the dialog.

Another complexity is that when the user closes the dialog, the dialog must somehow communicate that fact to the **FrameWin** so that it ceases to invoke **ShowPos** as the mouse moves. This is accomplished by overriding **OnDestroy** and before actually destroying the dialog, reset the appropriate data items in the **FrameWin** by using **GetParent** to gain access to it.

Similarly, **OnCancel** is overridden to make sure **OnDestroy** is called. Why? Since this is a modeless dialog, the **CDialog::OnCancel** hides the dialog box and does not destroy it. Here, we need it destroyed, not hidden.

The following illustrates the real distinction between modal and modeless dialogs. Modal dialogs have a sequential operation.

constructor + DoModal() + return value => point A

with no other possible functions invoked. At point A, the dialog box is now totally gone. Modeless dialogs are NEVER executed via **DoModal**! They are run as follows.

constructor + Create + ShowWindow => point B

at a later point in time, user closes dialog

At point B the modeless dialog is visible and active and remains fully operational. The box remains visible and potentially active. Only at some later point in time or even application termination, is the box destroyed.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* MouseTrackingDialog.h                                                      *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #if !defined(AFX_MOUSETRACKINGDIALOG_H__BCB18397_8423_48C2_B52D_4*
*  2 #define AFX_MOUSETRACKINGDIALOG_H__BCB18397_8423_48C2_B52D_40FAF4*
*  3                                                                        *
*  4 #if _MSC_VER > 1000                                                   *
*  5 #pragma once                                                          *
*  6 #endif // _MSC_VER > 1000                                             *
*  7                                                                       *
*  8 #include "resource.h"                                                 *
*  9                                                                       *
* 10 /*************************************************************/*
* 11 /*                                                           */*
* 12 /* MouseTrackingDialog: display mouse position in the FrameWin */*
* 13 /*                                                           */*
* 14 /* Modeless dialog - must be created - no data transfer     */*
* 15 /* separate function to display mouse coordinates           */*
* 16 /*                                                           */*
* 17 /* Uses Static Edge and 3d look - no centered               */*
* 18 /*                                                           */*
* 19 /*************************************************************/*
* 20                                                                       *
```

```
* 21 class MouseTrackingDialog : public CDialog {                        *
* 22 // Construction                                                      *
* 23 public:                                                             *
* 24  MouseTrackingDialog(CWnd* pParent = NULL);                         *
* 25                                                                     *
* 26 // Dialog Data                                                      *
* 27  //{{AFX_DATA(MouseTrackingDialog)                                  *
* 28  enum { IDD = IDD_MODELESS };                                       *
* 29  //}}AFX_DATA                                                       *
* 30                                                                     *
* 31 void      ShowPos (CPoint&);// update coordinates shown on screen*
* 32 BOOL      DestroyWindow (); // notify parent no longer tracking  *
* 33                                                                     *
* 34 protected:                                                          *
* 35                                                                     *
* 36 void      OnCancel ();      // must trap and destroy modeless dlg*
* 37                                                                     *
* 38 // Overrides                                                       *
* 39  // ClassWizard generated virtual function overrides              *
* 40  //{{AFX_VIRTUAL(MouseTrackingDialog)                              *
* 41  //}}AFX_VIRTUAL                                                   *
* 42                                                                     *
* 43 // Implementation                                                  *
* 44 protected:                                                          *
* 45                                                                     *
* 46  // Generated message map functions                                *
* 47  //{{AFX_MSG(MouseTrackingDialog)                                  *
* 48  //}}AFX_MSG                                                       *
* 49  DECLARE_MESSAGE_MAP()                                            *
* 50 };                                                                  *
* 51 //{{AFX_INSERT_LOCATION}}                                          *
* 52 #endif // !defined(AFX_MOUSETRACKINGDIALOG_H__BCB18397_8423_48C2_*
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

In the constructor, we explicitly call the **Create** function. This differs from the modal dialogs in which **DoModal** calls the **Create** function as part of the launching process. Modeless dialogs are created and shown. Modal dialogs are launched with **DoModal**.

Notice in **DestroyWindow** that a pointer to the parent window is obtained again using **GetParent** and cast into a **FrameWin** pointer. The resultant pointer is used to reset **track_on** back to **false** to cause the **FrameWin** to stop calling the **ShowPos** function as the mouse moves over the client area.

Finally, notice how **ShowPos** is written. This is an example of manual transferring of data into controls. In this case the controls are Static Control. In order for the text in a Static Control to be changed, that control must have an Id number. **SetDlgItemText** then is used to alter the text which is the mouse coordinates.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* MouseTrackingDialog.cpp                                              *
```

```
/))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*    #include "stdafx.h"                                                  *
*    #include "pgm07a.h"                                                   *
*    #include "MouseTrackingDialog.h"                                      *
*    #include "FrameWin.h"                                                 *
*                                                                          *
*    #ifdef _DEBUG                                                         *
*    #define new DEBUG_NEW                                                 *
*    #undef THIS_FILE                                                      *
*    static char THIS_FILE[] = __FILE__;                                   *
*    #endif                                                                *
*                                                                          *
*    /*****************************************************************/*
*    /*                                                               */*
*    /* MouseTrackingDialog: display mouse position in the FrameWin */*
*    /*                                                               */*
*    /* Modeless dialog - must be created - no data transfer         */*
*    /* separate function to display mouse coordinates               */*
*    /*                                                               */*
*    /* Uses Static Edge and 3d look - no centered                   */*
*    /*                                                               */*
*    /*****************************************************************/*
*                                                                          *
*    MouseTrackingDialog::MouseTrackingDialog (CWnd* pParent)              *
*                       : CDialog(MouseTrackingDialog::IDD, pParent) {*
*     //{{AFX_DATA_INIT(MouseTrackingDialog)                               *
*      // NOTE: the ClassWizard will add member initialization here       *
*     //}}AFX_DATA_INIT                                                    *
*     // modeless dialogs must call Create                                 *
*     Create (IDD_MODELESS, pParent);                                      *
*    }                                                                     *
*                                                                          *
*    BEGIN_MESSAGE_MAP(MouseTrackingDialog, CDialog)                       *
*     //{{AFX_MSG_MAP(MouseTrackingDialog)                                 *
*     //}}AFX_MSG_MAP                                                      *
*    END_MESSAGE_MAP()                                                     *
*                                                                          *
*    /*****************************************************************/*
*    /*                                                               */*
*    /* DestroyWindow: activated when user closes modeless dialog   */*
*    /*                                                               */*
*    /* reset parent's tracking on indicator to false or off         */*
*    /*                                                               */*
*    /*****************************************************************/*
*                                                                          *
*    BOOL     MouseTrackingDialog::DestroyWindow () {                      *
*     // set parent's tracking indicator off to cease sending pos msgs*
*     // gain access to a pointer to the track_on public member           *
*     FrameWin *ptrparent = (FrameWin*) GetParent ();                      *
*     if (ptrparent)                                                       *
*     ptrparent->track_on = FALSE;                                         *
*     return CDialog::DestroyWindow ();                                    *
```

```
* 52 }                                                                  *
* 53                                                                    *
* 54 /*******************************************************************/*
* 55 /*                                                                 */*
* 56 /* OnCancel: handle ESC and closing by calling DestroyWindows     */*
* 57 /*                                                                 */*
* 58 /* because default handler does not delete the modeless dlg box*/*
* 59 /* rather it just hides it                                        */*
* 60 /*                                                                 */*
* 61 /*******************************************************************/*
* 62                                                                    *
* 63 void      MouseTrackingDialog::OnCancel () {                       *
* 64  DestroyWindow();  // modeless dlgs must be destroyed              *
* 65 }                                                                  *
* 66                                                                    *
* 67 /*******************************************************************/*
* 68 /*                                                                 */*
* 69 /* ShowPos: updates the static x,y mouse position fields          */*
* 70 /*                                                                 */*
* 71 /*******************************************************************/*
* 72                                                                    *
* 73 void      MouseTrackingDialog::ShowPos (CPoint &pt) {              *
* 74  char x[5];                                                        *
* 75  char y[5];                                                        *
* 76  itoa (pt.x, x, 10);                                              *
* 77  itoa (pt.y, y, 10);                                              *
* 78  SetDlgItemText (IDC_CURSOR_X, x);                                *
* 79  SetDlgItemText (IDC_CURSOR_Y, y);                                *
* 80 }                                                                  *
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

Data Validation

There are some data validator macros, **DDV**, to help ensure that the data being entered into an Edit control is valid. Most simply validate numerical ranges, verifying the number is between **minVal** and **maxVal** inclusive of the endpoints. (If the data type is **UINT**, **DWORD**, or one of the floating point types, typecasts are required on **minVal** and **maxVal**.)

> **Design Rule 34: The DDV function must immediately follow the DDX function for the same control.**

The validator functions are shown below.

```
DDV_MinMaxByte   (CDataExchange* pDX, BYTE value, BYTE minVal,
                  BYTE maxVal);
DDV_MinMaxInt    (CDataExchange* pDX, int value, int minVal,
                  int maxVal);
DDV_MinMaxLong   (CDataExchange* pDX, long value, long minVal,
```

```
                  long maxVal);
DDV_MinMaxUInt   (CDataExchange* pDX, UINT value, UINT minVal,
                  UINT maxVal);
DDV_MinMaxDWord  (CDataExchange* pDX, DWORD value, DWORD minVal,
                  DWORD maxVal);
DDV_MinMaxFloat  (CDataExchange* pDX, float const& value,
                  float minVal, float maxVal);
DDV_MinMaxDouble (CDataExchange* pDX, double const& value,
                  double minVal, double maxVal);
DDV_MaxChars     (CDataExchange* pDX, CString const& value,
                  int nChars);
```

A dialog can also force the **DDX/DDV** operation at other times that on Ok button presses. The **CWnd** member function **UpdateData** causes the data to be transferred as follows.

```
UpdateData (TRUE);
UpdateData (FALSE);
```

where **TRUE** causes the transfer of data from the controls to the dialog transfer buffer and **FALSE** causes the transfer of data to the controls from the dialog transfer buffer.

CDialog::OnOK calls **UpdateData (TRUE);** while **CDialog::OnInitDialog** calls **UpdateData (FALSE);**.

Writing Your Own Data Validator Functions

Frequently, these simple data validators are insufficient for the task at hand. Suppose that an item number was composed of 4 digits, a dash and a letter. When an edit control is used to allow the user to enter the item number, a user-written data validator can ensure that the data is in the proper format.

For example, assume that in the **FrameWin**, we had the following **CString** data items initialized as shown.

```
FrameWin::FrameWin () : CFrameWnd () {
 ...
 strcpy (description, "Pots");
 strcpy (itemNum, "1234-A");
...
```

Next, **OnLaunch** responds to the user clicking on the menu item to update the current data. An instance of the ControlsDialog is launched and is passed the address of the two needed data members.

```
void      FrameWin::OnLaunch() {
 ControlsDialog dlg (&description, &itemNum, this);
 if (dlg.DoModal () == IDOK) {
  Invalidate (); // repaint using new description and itemNum
```

```
 }
}
```

The class definition of the **ControlsDialog** is shown below. Notice the new **DDV** function.

```
#ifndef CONTROLDLGH
#define CONTROLDLGH

#include "resource.h"

class ControlsDialog : public CDialog {
public:
 ControlsDialog(CString* ptrd, Cstring* ptri,
                CWnd* pParent = NULL);
 enum { IDD = IDD_DIALOG };
 CString  m_description;
 CString  m_itemNum;
 CString* ptrDescription;
 CString* ptrItemNum;

protected:
 virtual void DoDataExchange(CDataExchange* pDX);
 virtual void OnOK();
 virtual BOOL OnInitDialog();
 void DDV_Fancy(CDataExchange*, CString, int);
 DECLARE_MESSAGE_MAP()
};
#endif
```

In the implementation of **ControlsDialog** the pointers are saved. If the corresponding pointer is not null, then its contents are copied into the dialog's members. The in **DoDataExchange**, immediately following the **DDX** for the item number control is the corresponding **DDV** macro for the item number validation.

In the **DDV** function, if the data is invalid, the function sets a flag in the **CDataExchange** instance by calling its **Fail** function and then throws a C++ exception. The MFC C++ exception handling is discussed more fully in a later chapter.

```
#include "stdafx.h"
#include "ControlsDialog.h"
#include "resource.h"

ControlsDialog::ControlsDialog(CString* ptrd, Cstring* ptri,
                               CWnd* pParent)
               : CDialog(ControlsDialog::IDD, pParent) {
 m_description = _T("");
```

```
 m_itemNum = _T("");
 ptrDescription = ptrd;
 ptrItemNum = ptri;
 if (ptrDescription)
  m_description = *ptrDescription;
 if (ptrItemNum)
  m_itemNum = *ptrItemNum;
}

void ControlsDialog::DoDataExchange(CDataExchange* pDX) {
 CDialog::DoDataExchange(pDX);
 DDX_Text(pDX, IDC_EDIT_FANCY, m_itemNum);
 DDV_Fancy(pDX, m_name2, 6);
 DDX_Text(pDX, IDC_EDIT_NAME, m_description);
}

void ControlsDialog::DDV_Fancy(CDataExchange *pDX, CString fancy,
                               int len) {
 if (pDX->m_bSaveAndValidate) {
  int ok = 0;
  if (fancy.GetLength() == len) {
   char *str = fancy.GetBuffer (7);
   if (str[0] >= '0' && str[0] <= '9' &&
       str[1] >= '0' && str[1] <= '9' &&
       str[2] >= '0' && str[2] <= '9' &&
       str[3] >= '0' && str[3] <= '9' &&
       str[4] == '-' &&
       ((str[5] >= 'a' && str[5] <= 'z') ||
        (str[5] >= 'A' && str[5] <= 'Z'))) ;
   else ok = 2;
   fancy.ReleaseBuffer ();
  }
  else ok = 1;
  if (ok!=0) {
   CString msg;
   if (ok == 1) msg = "Fancy must be 6 characters long";
   else msg = "Fancy must be of the form nnnn-a, where n is"
              " numeric and a is alpha";
   AfxMessageBox (msg);
   pDX->Fail ();
   AfxThrowUserException ();
  }
 }
}

BEGIN_MESSAGE_MAP(ControlsDialog, CDialog)
```

```
END_MESSAGE_MAP()

void ControlsDialog::OnOK() {
 UpdateData (TRUE);
 if (ptrDescription)
  *ptrDescription = m_description;
 if (ptrItemNum)
  *ptrItemNum = m_itemNum;
 CDialog::OnOK();
}
```

Sub-classing Controls — Deriving your Own Control Classes

Sometimes even a **DDV** is insufficient to handle the data editing needs of the dialog. Notice that a **DDV** is only called when the user clicks on the Ok button. If the data needs to be edited as the individual keystrokes are being input, then one must derive your own class from **CEdit** and provide that functionality by overriding **OnChar** and other necessary functions. This is known as sub-classing a control.

Sub classing is an advanced topic. But for the curious, one would define an instance of your derived edit class in the dialog definition. Then, use the **DDX** control version, like the list box example above, to have your derived class instance tied into the edit control when it is created.

Programming Problems

Problem Pgm07-1 — Acme Inventory Data Entry

We are going to construct an Inventory Data Base system for the ACME Construction Company during the series of programming problems through successive chapters. This initial portion implements the data entry user interface.

The **inventory record** consists of four fields in binary data format; in successive problems, the data is stored in a binary file.

item number: **char [7**]
    format: AA-###, where A is any uppercase letter and # is any digit — all three digits must be present as well as both letters and the dash — convert lowercase entries to uppercase
item description: **char [21]**
    format: any 20 characters what so ever
quantity on hand: **short int**
item cost: **double**

Make a structure containing these four fields. In the **FrameWin** class, create an instance of an array of 15 of these structures. When the application begins, the array should be empty.

The application should run **maximized** or **minimized only**. Use a light gray background. When the application wants to terminate, if there any data has been entered into the array, if that data has been altered and not yet saved, query the user and save or terminate based upon the user's response. For now, just pretend that the data has been saved and then terminate. We will add file support in the next version of the problem.

The main menu consists of File, Edit, Font, and Help. The File pop-up menu offers New, Open, Close, Save, SaveAs, Print, Print Setup, and Exit. The Edit pop-up menu offers: Add, Update, and Delete. The Help pop-up menu offers only About. Note that Font is just a menu item, not a pop-up. Use accelerators only if wanted. This problem implements Add, Update, Delete, Exit, and Help menu items only. The others are implemented in the next problem.

Create and install a dialog box for Help About as you see fit.

When the user selects Edit|Add, execute an Add dialog box with Ok, Cancel, and Clear buttons. The Add dialog box should provide edit controls for entering the four inventory fields in the above order. The Ok button ends the dialog as expected, passing the entered data back. The Cancel button terminates the dialog and sends no data back. The Clear button blanks (zeros) out the edit controls.

Additionally, three data fields must be validated for correct information. Force the item number to conform to the above specifications. The numerical fields, quantity and cost, should be forced to be numerical. Force quantity to be less than or equal to 32,767. You can check for validity of all fields after Ok is pressed. However, data validation is much easier if you use **DDV** data validation macros and corresponding functions. The easiest method of handling the item number is to make your own **DDV** function to handle its validation.

Assume that new records are appended to the end of the array. It the user attempts to add more than the current maximum of 15 records, display an appropriate message box. Make sure that you set the number of records in the array to 0 **before** calling the **Create** function because that function calls **OnCreate** which in turn calls **OnPaint**.

The main window displays of all the records and should contain an appropriate company title, such as, ACME Construction Company, on one line and appropriate column headings above the columns of data.

For extra credit, derive your own class from **CEdit** to perform the data validation as the keystrokes are entered. The easiest way to do this is in your dialog class have an instance of your dreived class, **MyEdit** edit. Then, in **OnInitDialog** after the base class call, tie your instance to the actual physical control by edit. **SubclassDlgItem (IDC_MYEDIT, this);** or use the **DDX** control macro in **DoDataExchange**.

Next, implement the Update and Delete menu choices. Of course, use an update handler to gray out Update and Delete menu items if there are no elements in the array as yet. The first step is to launch a Choose Record to delete or update dialog with a list box showing the item number and description as one long string. If the user makes a selection, then handle it accordingly.

If the user is deleting a record, display a confirmation message box and if Yes, delete that record from the array and repaint the screen. If the user is updating a record, then show the Update dialog. Notice that there is only a slight difference between the Update dialog and the Add dialog. In the update dialog, the dialog caption is different and there is one additional button, the Reset button which resets the four dialog controls back to the original values. Thus, the easiest way to handle the Update and Add dialogs is to make a single AddUpdate dialog and pass it a **bool isUpdate**. If **isUpdate** is on, then set the window text (caption) to reflect that this is an update. Otherwise, leave the original Add title appear as usual. This can be done in **OnInitDialog**. Also in **OnInitDialog**, if this is not an update, then hide the Reset button by getting a pointer to this control by calling **GetDlgItem** passing it the Reset button id. Then, use the returned **CWnd** pointer to call **ShowWindow** passing it **SW_HIDE**.