Lab 6.1: Modifying Resources and Adding Dialog Boxes

Objectives

After completing this lab, you will be able to:

- Create a dialog box template.
- Build a new class based on that template.
- Remove unused menu and toolbar items.
- Add an error string to the string table.
- Write code to handle a modal dialog box within an application.
- Use the new dialog class in an application.

Prerequisites

Familiarity with the topics covered in this chapter including the CDialog class and the UpdateData function.

Lab Setup

To run the solution to this lab, click this icon.

To see a demonstration of the solution to this lab, click this icon.

Estimated time to complete this lab: **90 minutes**.

Exercises

The following exercises provide practice working with the concepts and techniques covered in this chapter.

Exercise 1: Creating the Dialog Box Template

In this exercise, you will create a dialog box in which you choose the files to compare in the two panes of a splitter window.

Exercise 2: Creating the Dialog Class and Providing for DDX and DDV

In this exercise, you will create a dialog class that is associated with your dialog box template. Then you will add member variables for all controls other than static text controls, and add simple DDX and DDV for the edit controls.

Exercise 3: Modifying the Menus and Toolbar

In this exercise, you will use the resource editor to modify the menus and toolbar for the Diff application.

Exercise 4: Implementing a Modal Dialog Box in an Application

In this exercise, you will write the code to enable the resources you have created in the previous exercises.

Exercise 5: Using the New Class in an Application

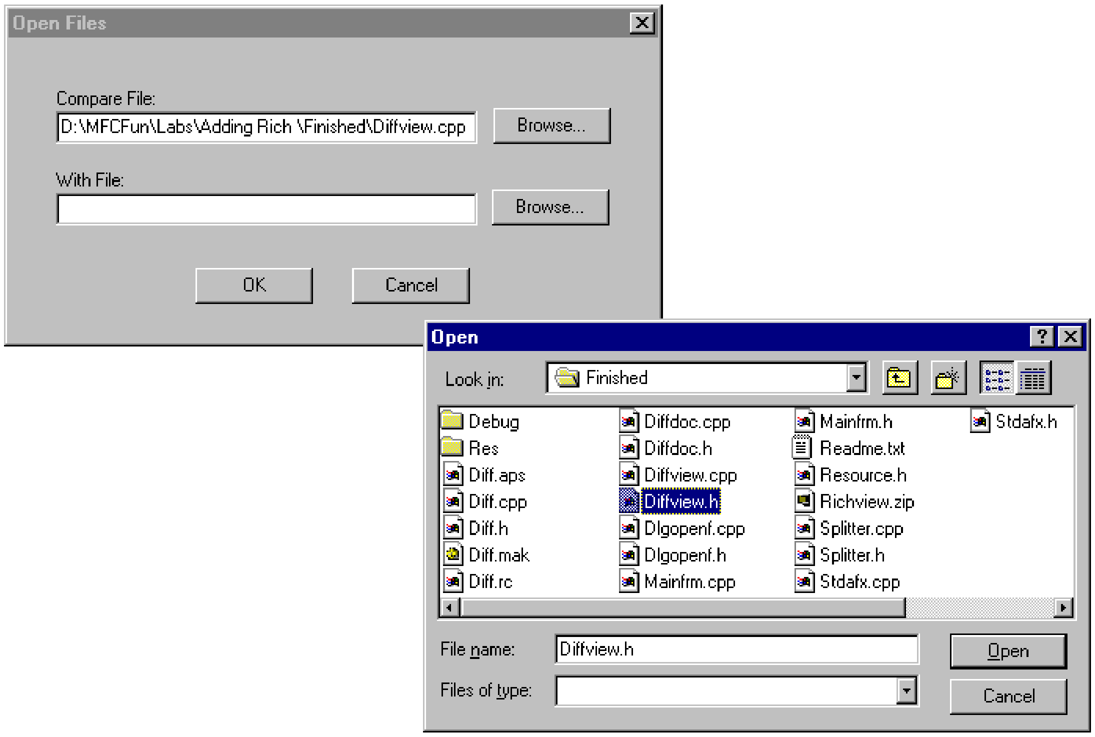
In this exercise, you will write the code to include the new class in an application.

For this lab, you will use the project from Chapter 7, Lab 2, Exercise 4. This project is in \Labs\C06\Lab01\Baseline. The completed code for these exercises is in \Labs\C06\Lab01\Xxx, where Xxx is the exercise number.

Exercise 1: Creating the Dialog Box Template

If you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C06\Lab01\Baseline.

In this exercise, you create a dialog box in which you choose the files to compare in the two panes of a splitter window. This dialog box cascades to the File Open dialog box from the Common Dialogs library, as shown in this illustration.



Compare File: D:\MFCFun\Labs\Adding Rich \Finished\Diffview.cpp

With File:

Look in: Finished

File name: Diffview.h

There are three parts to this exercise:

1. Creating the dialog box template
2. Adding the controls to the dialog box template
3. Testing the dialog box template and setting the tab order

Creating the Dialog Box Template

In Part 1 of this exercise, you will use the dialog editor to create the basic resource.

➢ Create a new (unadorned) dialog box resource

You can use any of the following techniques:

1. In the resource toolbar, click the dialog button to create a new dialog box template.
2. From the Insert menu, choose Resource. Select a dialog resource from the list box.
3. Right-click the dialog icon in the resource browser window. Select Insert Dialog from the shortcut menu.

➢ Change the title and ID of a dialog box using the resource editor

1. Display the Dialog Properties page by right-clicking anywhere in the window and choose Properties.
2. In the Properties window for the dialog box frame, in the Caption text box, type **Open Files**, the title for the dialog box.
3. In the ID text box, type **IDD_OPENFILES** as the ID value.

Adding Controls to the Dialog Box Template

By default, the dialog box template comes with OK and Cancel buttons in the upper-right-hand corner. You will add other common controls to produce a dialog box. The default buttons will be first and second in the tab order, followed by the other controls in the order they are added. For now, do not assign Group or Tabstop properties to any of the controls. Use the following table as a guide for adding the remaining controls. Resize the dialog box frame as needed to contain the required controls.

Note In the static text controls, the ampersand establishes ALT key access to the control that follows in the tab order. A static control cannot have focus, so focus automatically flows to the next control in the tab-order sequence. In this case, the shortcut keys in the static-text labels provide access to the associated edit-box controls.

Here are the controls to add, including their caption strings and IDs:

Control Type	Caption String	ID
Static text	**&Compare File:**	**IDC_STATIC** (default)
Edit box	none	**IDC_EDIT_FILE1**
Static text	**&With File:**	**IDC_STATIC** (default)
Edit box	none	**IDC_EDIT_FILE2**
Push button	**Browse...**	**IDC_BUTTON_FILE1_BROWSE**
Push button	**Browse...**	**IDC_BUTTON_FILE2_BROWSE**

➢ Open the IDD_OPENFILES dialog box resource

You will be adding a series of common controls to populate the dialog box. At any time in this process, resize the dialog-box frame to contain the controls.

➢ Add a common control to the dialog box template

These are general instructions, though the Checklist calls for a static text control. The type of tool you choose from the controls toolbar dictates the type of control that is drawn on the dialog box template.

1. From the controls toolbar, select the proper control. ToolTips provide hints about the functionality of each tool, as does Visual C++ Help.

   Note If the controls toolbar is not visible, go to the View menu and choose Toolbars. In the Toolbars dialog box, select the Controls option. Alternately, you can right-click a toolbar and choose Controls from the shortcut menu.

2. In the Dialog window, click the client area to add a static text control. You also can simply drag from the static text tool and drop at the destination in the client area.
3. Resize and reposition the control as needed.

➢ Set the caption text of a control

Control captions are set on the control's property page. Use one of the following two methods to gain access to a property page. (If a control has no caption field, this does not apply.)

1. After a control is placed, start typing. The Text property page appears, and the typed text is entered as the control caption.
2. Right-click the control and choose Properties. In the property page for that control, type the caption into the Caption edit box.

➢ Locate and identify tools on the controls toolbar

With the controls toolbar displayed, you can use one of the following techniques to find a tool (such as the one for group boxes) and display Help about the tool.

1. Rest the cursor on individual tool buttons. A Tool Tip that identifies each button should appear as you pause over each button. Find the group-box tool.

2. To view a prompt-string description of a control, rest the cursor above its tool and look in the status-bar pane. Holding down the left mouse-button on a control will accomplish the same thing.
3. To get Help, set focus to the controls toolbar window and press F1.

➢ **Add an edit control to the dialog box template**

Place it below the static text control.

➢ **Change the ID of the edit control to IDC_EDIT_FILE1**

➢ **Add a &With File: static text control and an edit control to its immediate right**

Use **IDC_EDIT_FILE2** as the edit-control ID.

➢ **Add two push buttons, placing one to the left of each edit control**

Assign to the buttons the captions &Browse... and Brow&se.... The three dots after these captions will display another dialog box. Assign the IDs **IDC_BUTTON_FILE1_BROWSE** and **IDC_BUTTON_FILE2_BROWSE**, respectively.

➢ **Align the controls**

Use commands in the Layout menu or tools in the dialog toolbar.

➢ **Save the current file**

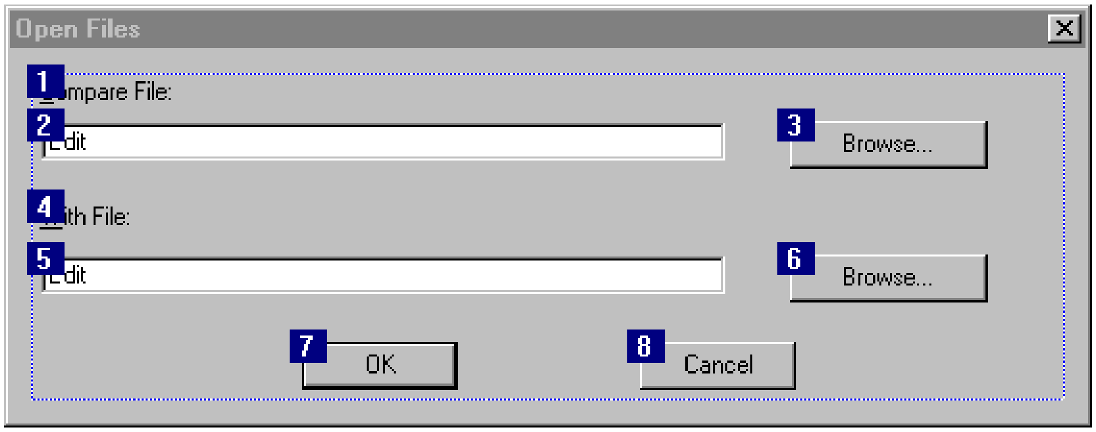
Testing the Dialog Box Template and Setting the Tab Order

In this final part of the exercise, you will use test mode to check various aspects of control functionality. After you exit test mode, you will change the tab order and group three radio buttons.

Set a new tab order for the controls in the following order:

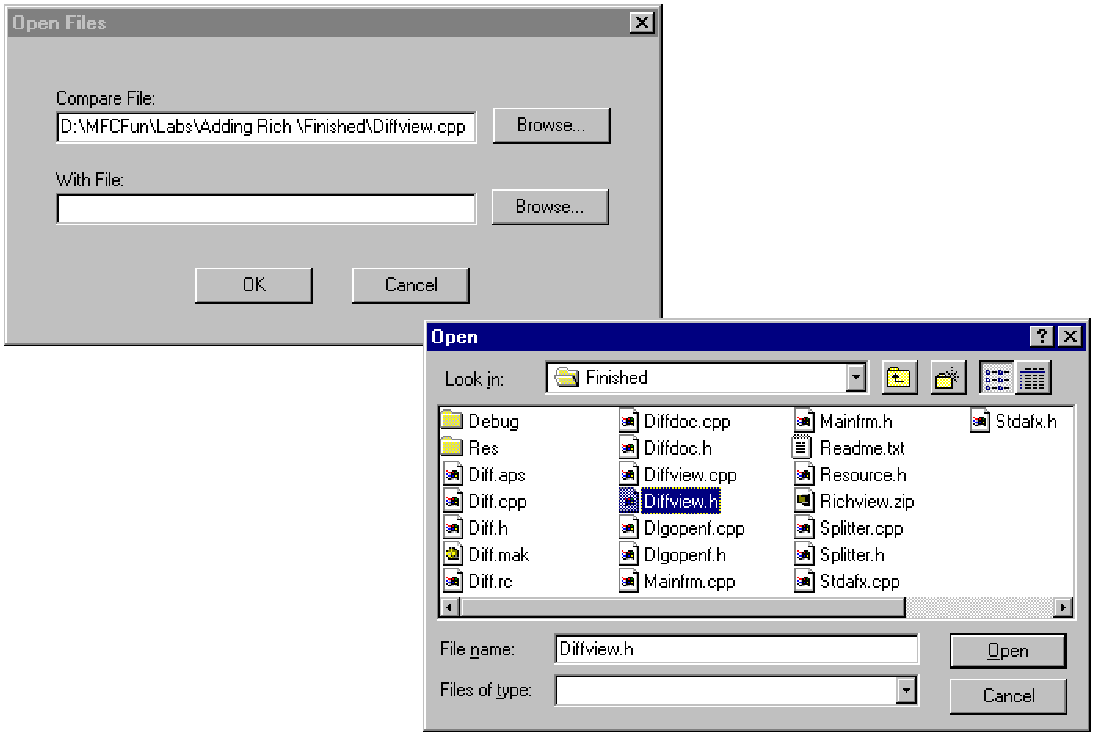
- Compare File
- Edit box (for Compare File)
- Browse button (for Compare File)
- With File
- Edit box (for With File)
- Browse button (for With File)
- OK button
- Cancel button

This illustration shows the correct tab order.



➢ **Test the dialog box resource**

1. Enter test mode, using one of these methods:



   a. Click the test tool on the dialog toolbar.

   b. From the Layout menu, choose Test.

   c. Press CTRL+T.

   A simulation of the dialog box created from this resource appears.

2. Note which control has initial keyboard focus. Also note, however, that the OK button is the default button, which is activated when the user presses the ENTER key.
3. Test the shortcut-key sequences.
4. Press the TAB key several times to cycle through the controls. Notice the effect and ordering. How do the radio buttons participate in the tab ordering?
5. Within the radio button group, use the arrow keys to navigate through the buttons.
6. Select an edit control. Type a text string, such as **Testing**.
7. Click either the OK or Cancel button to exit the test mode.

➢ Set the tab order for the controls on the dialog box template

1. From the Layout menu, choose Tab Order, or use the CTRL+D accelerator keys.

   The Properties page is hidden. The dialog editor now displays a number for each of the controls in your dialog box template. By default, the numbers indicate the order in which each control was added to the template.
2. Click the Compare File: static text control to set it as the first in the tab order. With this control now in the first position, the other numbers adjust accordingly.
3. Click the control that should be second in the tab order, in this case, the edit control paired with Compare File:.
4. Set the remaining controls in the same manner.
5. To end the tab-ordering operation, click inside the dialog editor window, but *outside* of the dialog box resource. (Pressing ESC also will end the session.)

➢ Test the dialog box template again

➢ **Save the current file**

The completed code for this exercise is in \Labs\C06\Lab01\Ex01.

Exercise 2: Creating the Dialog Class and Providing for DDX and DDV

If you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C06\Lab01\Ex01.

This exercise has two parts. In the first, you will use ClassWizard to create a dialog class that is associated with your dialog box template. In the second part, you will add member variables for all controls other than static text controls, and add simple dialog data exchange (DDX) and dialog data validation (DDV) for the edit controls.

Using ClassWizard to Create the Dialog Class

➢ **Run the dialog editor on the IDD_OPENFILES dialog resource**

➢ **Add a dialog class using ClassWizard**

1. In the dialog editor, be sure that your dialog box template window is the active child window.
2. Invoke ClassWizard. If the Adding A Class dialog box does not display automatically, click the Add Class button.
3. Choose the create a new class option, and click OK. The Create New Class dialog box appears. Note that the new dialog class is created in a new pair of .Cpp and .H files whose names are derived by default from the dialog class name.
4. In the Name box, type **CDlgOpenFiles** for the name of the associated C++ dialog class.
5. In the Base Class box, select **CDialog**. The Dialog ID already should be set to IDD_OPENFILES.
6. Set the filenames to DlgOpenF.Cpp and DlgOpenF.H by using the Change button and its dialog box.
7. Choose OK.

➢ **Examine the source files for the CDlgOpenFiles class**

Note that the class is derived from the base class **CDialog**.

To have ClassWizard automatically present the Adding A Class dialog box, open ClassWizard during the dialog editor session where you create the new dialog resource.

Adding Member Variables and DDX/DDV to the Dialog Class

In this part of the exercise, you will create C++ member variables for the new **CDlgOpenFiles** class and bind the variables to the controls on the dialog resource. Then, you will select dialog data validation (DDV) values or ranges where appropriate.

Add member variables as indicated in the following table. All the variables will be of category Value (*not* Control).

Member Variable Name	**Variable Type**	**DDV values**
m_File1	**CString**	255 characters
m_File2	**CString**	255 characters

Create handlers for the two Browse buttons as indicated in the following table.

Object ID	**Message**	**Function name**
IDC_BUTTON_FILE1_BROWSE	**BN_CLICKED**	OnButtonFile1Brows e
IDC_BUTTON_FILE2_BROWSE	**BN_CLICKED**	OnButtonFile2Brows

e

➢ To complete this exercise, perform the following steps.

1. Invoke ClassWizard for the Diff project.
2. Click the Member Variables tab in the ClassWizard dialog box.
3. Select the CDlgOpenFiles class.

You are now ready to create member variables for the dialog class and bind them to the actual dialog controls.

➢ **Create a member variable for the IDC_EDIT_FILE1 edit control**

1 Locate the control you want to bind in the Control IDs list box. Either double-click the control ID or select it and choose the Add Variable button. This invokes the Add Member Variable dialog box.

2. Type a name for the member variable in the first edit box. Usually, this is a permutation of the control ID. For example, **m_File1** is a likely name for the IDC_EDIT_FILE1 control.
3. In the Category box, select a category of the control. For the simple common controls that can be adequately represented by a single value, select Value. If you want to create a pointer to the actual control window, select Control.
4. In the Variable Type box, select the data type that best represents the information contained within the member variable being created. For example, an edit box generically returns a **CString**; but it can be further limited to return only numeric types.
5. Choose OK to save your input and return to ClassWizard.
6. At the bottom of the ClassWizard window, you will see edit boxes for entering dialog data validation (DDV) values. This will not be available for variables where DDV does not apply.

➢ **Go to the Maximum Characters box in ClassWizard and enter 255**

Note that 255 is the maximum file name length in Windows 95. This value is used for dynamic dialog validation (DDV).

➢ **Create a member variable for the IDC_EDIT_FILE2 edit control**

For the member variable name, use **m_File2**. The other values are the same as for **IDC_EDIT_FILE1**.

➢ **Move to the Message Maps tab in ClassWizard**

➢ **Create a handler for a dialog control**

1. Select the IDC_BUTTON_FILE1_BROWSE in the ObjectIDs list box.
2. Select the BN_CLICKED message in the Messages list box.
3. Choose the Add Function button.
4. The Add Member Function dialog box will be displayed. You can accept or change the Member Function name. Click OK to accept.
5. The function name will be displayed in the Member Functions list box.
6. Repeat the previous step for IDC_BUTTON_FILE2_BROWSE.
7. Repeat the previous step for IDOK.
8. Briefly examine the source files for the CDlgOpenFiles class.

The completed code for this exercise is in \Labs\C06\Lab01\Ex02.

Exercise 3: Modifying the Menus and Toolbar

If you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Lab\C06\Lab01\Ex02.

In this exercise, you will use the resource editor to modify the menus and toolbar for the Diff application. You also will add a string to the string table to display the following error message:

```
Either
```

```
<File 1>
OR
<File 2>
is an invalid file. Please check both file specifications and try again.
```

➢ **Delete File New from the menu**

1. Open the Menu folder in the Resource view.
2. Choose IDR_MAINFRAME to edit.
3. Open the File menu.
4. Choose New and delete it.
5. Save your changes to Diff.Rc.

➢ **Delete File New from the toolbar**

1. Open the toolbar folder.
2. Choose IDR_MAINFRAME to edit.
3. The File New icon will be selected; drag it off the sample toolbar to delete it.
4. Save your changes to Diff.Rc.

➢ **Delete File New from the Accelerator Table**

1. Open the Accelerator folder.
2. Choose IDR_MAINFRAME to edit.
3. Select the ID_FILE_NEW accelerator.
4. Press DELETE.
5. Save your changes to Diff.Rc.

➢ **Add an error string to the String Table**

1. Open the String Table folder.
2. Choose String Table to edit.
3. At the bottom of the String Table editor, double-click or choose Insert String. String properties db appears.
5. Set ID to IDS_ERRFMT_INVALIDFILE.
6. Set caption to:

```
Either \r\n\n\%1\r\n\t\OR\r\n\%2\r\n\nis an invalid file. Please check both
file specifications and try again.
```

Note afxFormatString2 uses %1 and %2 as its string substitution markers; if you were using wsprintf, you would use two %s markers.

7. Save your work and close the String Table editor.

The completed code for this exercise is in \Labs\C06\Lab01\Ex03.

Exercise 4: Implementing a Modal Dialog Box in an Application

In this exercise, you write the code that enables the resources that you have created.

If you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C06\Lab01\Ex03.

➢ **Get values from the edit controls**

In this step, you will create two functions, **GetFile1** and **GetFile2,** that are identical.

1. Declare two public methods in the attributes section of DlgOpenF.H.

```
public:
    void GetFile1 (CString& rFile);
    void GetFile2 (CString& rFile);
```

2. Save DlgOpenF.H.
3. In DlgOpenF.Cpp, implement the methods to set rFile to the corresponding edit control members.

```
void CDlgOpenFiles::GetFile1(CString& rFile)
{
        rFile = m_File1;
}

void CDlgOpenFiles::GetFile2(CString& rFile)
{
        rFile = m_File2;
}
```

4. Save DlgOpenF.Cpp.

➤ **Validate the files**

The simplest way to validate a file name for this application is to test for the file's existence.

1. Declare the method as protected in the implementation section in DlgOpenF.H.

```
protected:
    BOOL IsValidFileSpec (LPCSTR lpszFileSpec);
```

2. Save DlgOpenF.H.
3. In DlgOpenF.Cpp, check for the existence of the file.

```
BOOL CDlgOpenFiles::IsValidFileSpec (LPCSTR lpszFileSpec)
{
    OFSTRUCTof;
    if(OpenFile(lpszFileSpec, &of, OF_EXIST) == HFILE_ERROR)
```

4. If the file exists, the test will fail; if the file does not exist, the test will succeed. Return the inverse.

```
    {
        return FALSE;
    }
    else
    {
        return TRUE;
    }
}
```

5. Save DlgOpenF.Cpp.

➤ **Implement OnButton1FileBrowse and OnButton2FileBrowse handlers**

The browse buttons simply display the common dialog box, with the contents of the corresponding edit box. The function outlines have been provided by ClassWizard, and the details of this function are covered in [Check Ref] Chapter 3, Lab 2: Creating the Framework for DIFF. You can copy most of this function from **CDiffDoc::OnFileOpen** of DiffDoc.Cpp.

Update the class members of the dialog box so that you can provide the existing file name as a default in the File dialog box.

```
void CDlgOpenFiles::OnButtonFile1Browse()
{
    UpdateData();
```

1. Define the filter for the File dialog box.

```
static char BASED_CODE szFilter[] =
    "All Files (*.*)|*.*|C++ Files (*.cpp, *.h)"\
    "|*.cpp;*.h||";
```

2. Construct the File dialog box as an open dialog box with this filter.

```
CFileDialog dlg(TRUE,NULL,m_File1,NULL,szFilter);
```

3. Display the File dialog box as modal.

```
if (dlg.DoModal() == IDOK)
```

4. If the user clicked OK, then assign the path back to the members of your dialog box.

```
{
    m_File1 = dlg.GetPathName();
}
```

5. Update the data displayed in your dialog box.

```
UpdateData(FALSE);
```

6. Repeat this procedure for **OnButtonFile2Browse**. The complete function body follows.

```
void CDlgOpenFiles::OnButtonFile2Browse()
{
    UpdateData();
    static char BASED_CODE szFilter[] =
        "All Files (*.*)|*.*|C++ Files (*.cpp, *.h)"\
        "|*.cpp;*.h||";
    CFileDialog dlg(TRUE,NULL,m_File2,NULL,szFilter);

    if (dlg.DoModal() == IDOK)
    {
        m_File2 = dlg.GetPathName();
    }
    UpdateData(FALSE);
}
```

7. Save DlgOpenF.Cpp.

➢ **Respond to the user clicking OK**

1. Use ClassWizard to create a handler for IDOK. Accept the default **OnOK** function name.
2. Edit the code for this handler.
3. Update all the data in the members of the dialog box.
4. Check to see whether the files are valid using your **IsValidFileSpec** function. If the files are valid, simply pass on the message to **CDialog**.

```
if(IsValidFileSpec(m_File1) &&
    (IsValidFileSpec(m_File2)))
{
    CDialog::OnOK();
}
```

5. If either of the file names is not valid, display an error message.

```
else
{
    CString ErrMsg;
    AfxFormatString2(ErrMsg, IDS_ERRFMT_INVALIDFILE,
                        m_File1, m_File2);
    MessageBox (ErrMsg);
}
```

6. Save DlgOpenF.Cpp.

The completed code for this exercise is in \Labs\C06\Lab01\Ex04.

Exercise 5: Using the New Class in an Application

If you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Lab\C06\Lab01\Ex04.

In this exercise you will use the completed **CDlgOpenFiles** dialog class in Diff by modifying the simple **CDiffDoc::OnFileOpen** from Chapter 7, Lab 2, rather than modifying **CFileDialog**.

➢ **Include DlgOpenF.H in DiffDoc.Cpp**

➢ **Implement CDlgOpenFiles in OnFileOpen**

1. Construct **CDlgOpenFiles** rather than **CFileOpen**.

```
void CDiffDoc::OnFileOpen()
{
    CDlgOpenFiles   dlg;
```

2. Show the dialog box as modal.

```
if(dlg.DoModal() == IDOK)
```

3. If the user clicks OK, get the contents of the two file edit controls through their public interface.

```
dlg.GetFile1(m_File1);
dlg.GetFile2(m_File2);
```

4. Call RunComparison to load the files into the splitter windows.

```
RunComparison(m_File1, m_File2);
```

5. Save DiffDoc.Cpp. The complete function follows.

```
void CDiffDoc::OnFileOpen()
{
    CDlgOpenFiles   dlg;
    if(dlg.DoModal() == IDOK)
        dlg.GetFile1(m_File1);
        dlg.GetFile2(m_File2);
        RunComparison(m_File1, m_File2);
}
```

➢ **Build and run the Diff application**

The completed code for this exercise is in \Labs\C06\Lab01\Ex05.