Lab 6.2: Adding a Modeless Dialog Box

Objectives

After completing this lab, you will be able to:

- Add a dialog box to an existing application
- Implement the dialog box as a Modeless dialog box
- Respond to user actions from the dialog box in the application

Prerequisites

You may want to preview Chapter 7: View Classes doing the third exercise in this lab.

Lab Setup

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

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

Estimated time to complete this lab: **30 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 that can step through the differences between two files.

Exercise 2: Implementing the Dialog Class

In this exercise, you will implement the dialog class, **CFindDifferenceDialog**, which you created in Exercise 1. Because this dialog is modeless, all its actions will take place while its window still exists. Therefore, you will not need to use DDX synchronization, because you can query controls directly.

Exercise 3: Integrating the Dialog Box into an Application

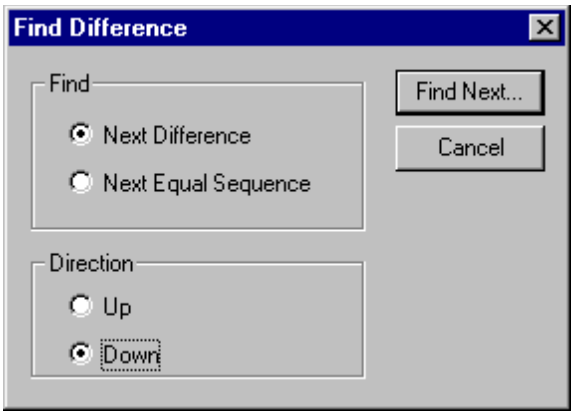
In this exercise, you will provide the code that responds to user actions in the dialog box.

For this lab, you will need to use the project from Chapter 6, Lab 3: Using Common Dialogs. If you have not done the lab on Using Common Dialogs, you can copy this code from \Labs\C06\Lab02\Baseline. The completed code for these exercises is in \Labs\C06\Lab02\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\Lab02\Baseline.

In this exercise, you will create a dialog box that can step through the differences between two files. As with of the ShowDiff programs, the differencing is simulated.



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 and setting the tab order.

Creating the Dialog Box Template

In this section, you will use the dialog editor to create the basic resource.

➢ **Create a new (unadorned) dialog box resource**

Use any of the following techniques:

1. In the resource toolbar, click the dialog button to create a 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**

You can display the Dialog Properties page by right-clicking anywhere in the window and choosing Properties.

1. In the Properties window for the dialog box frame, locate the Caption text box and type **Find Difference**, the title for the dialog box.
2. In the ID text box, type **IDD_NEXTDIFF** as the ID value.

Adding Controls to the Dialog Box Template

The dialog box template comes with default OK and Cancel buttons in the upper-right corner. You will add other common controls to produce a dialog box like the one shown above. 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.

Note For the static text controls, the ampersand on the static text 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, including their caption strings, and IDs:

Control Type	ID	Caption
Dialog	IDD_NEXTDIFF	Next Difference
Default command button (push button)	IDOK	Find Next...:
Command button	IDCANCEL	Cancel
Group box	IDC_STATIC	Find
Radio button	IDC_RADIO_NEXTDIFF	&Next Difference
Radio button	IDC_RADIO_NEXTEQUAL	Next &Equal Sequence
Group box	IDC_STATIC	Direction
Radio button	IDC_RADIO_UP	&Up
Radio button	IDC_RADIO_DOWN	&Down

➢ **Add controls to the dialog box template**

1. Open the IDD_NEXTDIFF dialog box resource.

   You will be adding a series of common controls to populate the dialog box. Use the preceding diagram and tables as a guide for adding common controls to the dialog box resource. At any time in this process, resize the dialog box frame to contain the controls.

2. Add a group box control to the dialog box template.

   These are general instructions for adding a control, though this exercise 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.

3. From the controls toolbar, select the proper control. ToolTips provide information about the functionality of each tool.

   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.

   ---

4. 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.
5. Resize and position the control in the upper-left corner of the dialog box image.

➢ **Change the caption text of the group box to Find**

Control captions are set on the property page for the control. 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.

➢ **Add and modify radio buttons for the dialog box template**

1. Add a radio button to the dialog box template and place it inside the group box.
2. Change the ID of the radio button to IDC_RADIO_NEXTDIFF.
3. Change the caption of the radio button to &Next Difference.
4. Repeat with another radio button.

   Set its ID to IDC_RADIO_NEXTEQUAL and its caption to Next &Equal Sequence.

5. Repeat with the Direction group box and its two radio buttons, &Up and &Down.
6. Change the caption of the OK button to Find Next.
7. Align the controls using the commands in the Layout menu or tools in the dialog toolbar

➢ **Save the current file**

➢ **Locate and identify tools on the controls toolbar**

With the controls toolbar displayed, 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 ToolTip window, containing an identification string, 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 the 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.

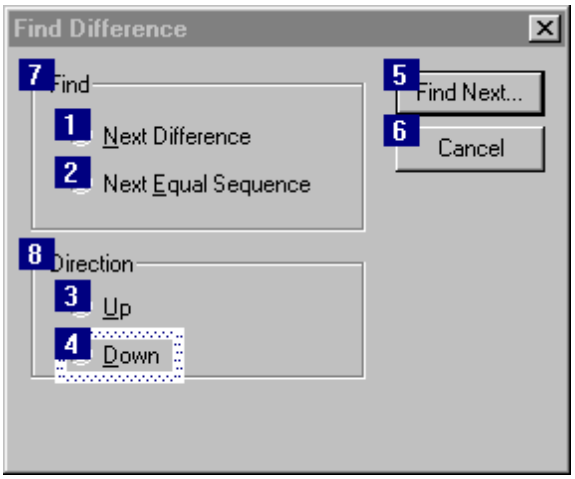
Testing the Dialog Box Template and Setting the Tab Order

In this 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:

1. The Next Difference radio button
2. The Next Equal Sequence radio button
3. The Up radio button
4. The Down radio button
5. The Find Next button
6. The Cancel button
7. The “Find” group label
8. The “Direction” group label

This illustration shows the correct tab order for the controls.



➢ **Test the dialog box resource**

Use any of these methods to enter test mode:

1. Click the test tool on the dialog toolbar.
2. Go to the Layout menu and choose Test.
3. Press CTRL+T.

   A simulation of the dialog box created from this resource appears.
4. Note which control has initial keyboard focus. Also note, however, that the OK button is the default button. It is activated when the user presses the ENTER key.
5. Test the shortcut-key sequences.

6. Press the TAB key several times to cycle through the controls. Notice the effect and ordering.
7. Within the radio button group, use the arrow keys to navigate among the buttons.
8. Select an edit control. Type a text string, such as **Testing**.
9. Click the OK or Cancel button to exit test mode.

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

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

   The property 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. While setting the tab order, note that the OK button automatically has the Default Button option selected.
2. Click a control to set it as the first in the tab order. (For purposes of this exercise, click the Next Difference radio button.) 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 radio button Next Equal Sequence.
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.)

➢ **Group controls on the dialog box template**

1. Right-click the Next Differences radio button and choose Properties.
2. Check the Group option.
3. Double-click the Up radio button and set its group option.
4. Double-click the Find Next button and set its group option.
5. Save the dialog box template.

➢ **Check and Save Your Work**

1. Test the dialog box template again.
2. Save the current file when you satisfied with your work.

Using ClassWizard to Create the Dialog Class

In this part of Exercise 1, you will use ClassWizard to create a dialog class in the Diff application.

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.

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

➢ **Create 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. It should automatically display the Adding A Class dialog box. (If this dialog box does not appear, click the Add Class button.)
3. Choose the Create A New Class option, and click OK. The Create New Class dialog box appears.
4. In the Name box, type **CFindDifferenceDialog** for the name of the associated C++ dialog class.
5. In the Base Class box, select **CDialog**. The Dialog ID should already be set to IDD_ NEXTDIFF.
6. Shorten CfindDifferenceDialog.Cpp and .H to CfindDiff.Cpp and .H by using the Change button and its dialog box.
7. Choose OK.

➢ **Briefly examine the source files for the CFindDifferenceDialog class**

The completed code for this exercise is in \Labs\C06\Lab02\Ex01.

Exercise 2: Implementing the Dialog Class

If you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C06\Lab02\Ex01.

In this exercise, you will implement the dialog class, **CFindDifferenceDialog**, which you created in the previous exercise. Because this dialog is modeless, all its actions will take place while its window still exists. Thus, you will not need to use DDX synchronization; you can simply query controls directly.

A modeless dialog is its own standalone window with its own window procedure. A modeless dialog box can communicate with its parent window in many ways; the simplest (and safest) is to use registered window messages known by the dialog class and the main application.

Another major difference between modal and modeless dialog boxes is how they close. A modal dialog box closes itself by calling **CDialog::EndDialog**, and the dialog object is normally destroyed when it passes out of scope. Because it cannot be assumed that modal dialog boxes are limited to the scope of their creating function, they must not only close their window (by calling **CWnd::DestroyWindow**), they also must delete themselves after their window is gone.

➢ **Set up members for control states in the dialog box**

1. Open FindDiff.H.
2. Create member variables to determine whether terminating or finding next.

```
//attributes
protected:
    BOOLm_bTerminating;
    BOOLm_bFindNext;
```

3. Declare member functions to return the internal state to the application.

```
public:
    BOOL    IsTerminating() const;  //TRUE if terminating
    BOOL    SearchDown() const;     //TRUE if searching down,
                                    //FALSE if searching up
    BOOL    FindDifference() const; //TRUE if finding next difference
                                    //FALSE if finding next equal
    BOOL    FindNext() const;       //TRUE if find next button pressed
```

4. Save FindDiff.H.
5. Open FindDiff.Cpp.
6. Implement **CFindDifferenceDialog::IsTerminating** to return m_bTerminating.

```
BOOL CFindDifferenceDialog::IsTerminating() const
{
    return m_bTerminating;
}
```

7. Implement **CFindDifferenceDialog::SearchDown** to return the state of the Search Down radio button.

```
BOOL CFindDifferenceDialog::SearchDown() const
{
    return(IsDlgButtonChecked(IDC_RADIO_DOWN));
}
```

8. Implement **CFindDifferenceDialog::FindDifference** to return the state of the Next Difference radio button.

```
BOOL CFindDifferenceDialog::FindDifference() const
{
```

```
    return(IsDlgButtonChecked(IDC_RADIO_NEXTDIFF));
}
```

9. Implement **CFindDifferenceDialog::FindNext** to return m_bFindNext.

```
BOOL CFindDifferenceDialog::FindNext() const
{
    return m_bFindNext;
}
```

10. Save FindDiff.Cpp.

➢ **Implement the Find Next button handler**

When the main application window is notified that the user has clicked a button, it will query the dialog box about the action. You could use the WPARAM parameter of the message to pass this information, but this would entail a more complicated maintenance scheme.

1. Open FindDiff.H.
2. Define a constant string for the name of the message.

```
const char* const FINDDIFF_MSGSTRING = "diffapp_FindDifference";
```

3. Save FindDiff.Cpp.
4. Start ClassWizard from the View menu or press CTRL+W.
5. Add a function for the IDOK BN_CLICKED message. Give it the name **OnFindNext**.
6. Click the Edit Code button to move to the function. Set m_bFindNext to TRUE.

```
m_bFindNext = TRUE;
```

7. Send the private message to the parent of the dialog box with a pointer to the dialog object in LPARAM.

```
GetParent()->SendMessage(
    ::RegisterWindowMessage(FINDDIFF_MSGSTRING),
    0, (LPARAM)this);
```

8. Reset m_bFindNext.

```
m_bFindNext = FALSE;
```

9. Save FindDiff.Cpp.

➢ **Implement dialog initialization**

1. In the constructor, initialize m_bTerminating and m_bFindNext to false.

```
CFindDifferenceDialog::CFindDifferenceDialog(CWnd* pParent /*=NULL*/)
    : CDialog(CFindDifferenceDialog::IDD, pParent)
{
    //{{AFX_DATA_INIT(CFindDifferenceDialog)
        // NOTE: ClassWizard will add member initialization here
    //}}AFX_DATA_INIT

    m_bTerminating = FALSE;
    m_bFindNext = FALSE;
}
```

2. Run ClassWizard from the View menu or press CTRL+W. Create handlers for WM_INITDIALOG and Create.
3. Edit the code for OnInitDialog. Check the Next Difference and Down radio buttons.

```
BOOL CFindDifferenceDialog::OnInitDialog()
```

```
{
    CDialog::OnInitDialog();

    //  Our initialization
    CheckDlgButton(IDC_RADIO_NEXTDIFF, TRUE);
    CheckDlgButton(IDC_RADIO_DOWN, TRUE);

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
```

4. Delete the parameters from the **Create** handler. Because all the members are set up in **CDialog::CDialog**, the call to Create uses these variables.

```
BOOL CFindDifferenceDialog::Create()
{
    //  m_lpszTemplateName and m_pParent are set
    //  up by CDialog during construction
    return CDialog::Create (m_lpszTemplateName, m_pParentWnd);
}
```

5. Save FindDiff.Cpp.
6. In the file FindDiff.H, remove the parameters from **Create**.
7. Save FindDiff.H.

➢ **Implement dialog shutdown**

1. Use ClassWizard to create handlers for BN_CLICKED on the IDCANCEL button (OnCancel) and the PostNcDestroy messages.
2. Edit the code of **CFindDifferenceDialog::OnCancel**. Use **CWnd::DestroyWindow** to close the window when the Cancel button is pressed.

```
void CFindDifferenceDialog::OnCancel()
{
    DestroyWindow();
}
```

3. Edit the code of **CFindDifferenceDialog::PostNcDestroy**. PostNCDestroy is sent by **CWnd** after the window has been destroyed. Because your window no longer exists, you cannot follow your window tree with **CWnd::GetParent**. You will use the stored m_mParentWnd handle to communicate with the parent window. Set the m_bTerminating member to TRUE, and then send the message to your parent window.

```
m_bTerminating = TRUE;
m_pParentWnd->SendMessage(
    ::RegisterWindowMessage(FINDDIFF_MSGSTRING),
    0, (LPARAM)this);
```

4. As the last step, delete your object.

```
delete this;
```

5. Save FindDiff.Cpp.

The complete **PostNcDestroy** handler is shown in this sample code.

```
void CFindDifferenceDialog::PostNcDestroy()
{
    m_bTerminating = TRUE;
    m_pParentWnd->SendMessage(
        ::RegisterWindowMessage(FINDDIFF_MSGSTRING),
```

```
        0, (LPARAM)this);

    delete this;
}
```

The completed code for this exercise is in \Labs\C06\Lab02\Ex02.

Exercise 3: Integrating the Dialog Box into the Application

If you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C06\Lab02\Ex02.

A modeless dialog box shares code with its parent window. In this exercise, you will provide the code that responds to actions in the dialog box. This code is placed in **CMainFrame** for simplicity; however, any **CWnd**-derived window (such as **CDiffView**) could be used as the target.

➢ **Resolve dependencies on CdifferenceDialog in MAINFRM.H**

1. You will be adding references to **CFindDifferenceDialog** to MainFrm.H. You will need to include FindDiff.H in any file that includes MainFrm.H prior to that include. Add the include statements to these files.
   - MainFrm.Cpp
   - DiffView.Cpp
   - DiffDoc.Cpp
   - Diff.Cpp

➢ **Add menu items to the IDR_MAINFRAME menu**

1. Open the IDR_MAINFRAME menu.
2. Add a separator to the end of the Edit menu.
3. Add these two menu items after the separator:

ID	Caption
ID_EDIT_FIND	&Find...
ID_EDIT_FIND_DIFFERENCE	Find &Difference...

4. Save Diff.Rc.

➢ **Integrate CFindDifferenceDialog into CMainFrame**

1. Open MainFrm.H.
2. Declare a pointer to a **CFindDifferenceDialog** object in the protected implementation section.

```
CFindDifferenceDialog*  m_pFindDiffDlg;
```

3. Declare a handler for the FINDDIFF_MSGSTRING registered message before DECLARE_MESSAGE_MAP.

```
afx_msg LRESULT OnFindDifferenceCmd (WPARAM, LPARAM lParam);
```

4. Declare a member function to find the next difference.

```
void OnFindNextDifference(BOOL bSearchDown,
                          BOOL bNextDifference);
```

5. Save MainFrm.H.
6. Open MainFrm.Cpp.
7. Initialize a variable to hold the ID of the registered message.

```
static const UINT nMsgFindDifference =
```

```
::RegisterWindowMessage(FINDDIFF_MSGSTRING);
```

8. Map the message to **CMainFrame**.

```
ON_REGISTERED_MESSAGE (nMsgFindDifference, OnFindDifferenceCmd)
```

9. Show the CommandWizard from the View menu. Add a command handler for ID_EDIT_FINDDIFF (OnEditFindDiff).
10. In the constructor, set m_pFindDiffDlg to NULL.

```
m_pFindDiffDlg = NULL;
```

➢ **Implement the menu handler for ID_EDIT_FINDDIFF**

1. Edit the code for **CMainFrame::OnEditFindDiff**. Check to see whether the dialog box is already displayed.

```
if(m_pFindDiffDlg == NULL)
```

2. If the dialog box is not already displayed, dynamically construct a **CFindDifferenceDialog** and assign the pointer to m_pFindDiffDlg.

```
m_pFindDiffDlg = new CFindDifferenceDialog(this);
```

3. Once you have constructed the dialog box, call **CFindDifferenceDialog::Create** to initialize it.

```
if(m_pFindDiffDlg)
{
    m_pFindDiffDlg->Create();
}
```

4. Show the dialog window.

```
if(m_pFindDiffDlg)
{
    m_pFindDiffDlg->SetActiveWindow();
    m_pFindDiffDlg->ShowWindow(SW_SHOW);
}
```

5. Save MainFrm.Cpp.

The complete function is shown in this sample code.

```
void CMainFrame::OnEditFindDiff()
{
    //  Create the dialog if needed

    if(m_pFindDiffDlg == NULL)
    {

        m_pFindDiffDlg = new CFindDifferenceDialog(this);
        if(m_pFindDiffDlg)
        {
            m_pFindDiffDlg->Create();
        }
    }

    //  Show it

    if(m_pFindDiffDlg)
    {
        m_pFindDiffDlg->SetActiveWindow();
        m_pFindDiffDlg->ShowWindow(SW_SHOW);
```

```
    }
}
```

➢ **Implement a handler for the registered message**

1. Define **CMainFrame::OnFindDifferenceCmd**.

```
LRESULT CMainFrame::OnFindDifferenceCmd(WPARAM, LPARAM lParam)
```

2. Cast the LPARAM to a pointer to **CFindDifferenceDialog**.

```
CFindDifferenceDialog* pDialog = (CFindDifferenceDialog *)lParam;
```

3. The dialog box sends its message when the Find Next button is clicked, or when the dialog box is closing. In the latter case, clear m_pFindDiffDlg.

```
if (pDialog->IsTerminating())
{
    m_pFindDiffDlg = NULL;
}
```

4. When the Find Next button is clicked, check the state of the radio buttons and dispatch to **CMainFrame::OnFindNextDifference**.

```
else if (pDialog->FindNext())
{
    OnFindNextDifference(pDialog->SearchDown(),
                         pDialog->FindDifference());
}
```

5. Return 0 for the message.
6. Save MainFrm.Cpp.

The complete function is shown in this sample code.

```
LRESULT CMainFrame::OnFindDifferenceCmd(WPARAM, LPARAM lParam)
{

    CFindDifferenceDialog* pDialog = (CFindDifferenceDialog *)lParam;

    if (pDialog->IsTerminating())
    {
        m_pFindDiffDlg = NULL;
    }
    else if (pDialog->FindNext())
    {
        OnFindNextDifference(pDialog->SearchDown(),
                             pDialog->FindDifference());
    }

    return 0;
}
```

7. Add the prototype for **OnFindDifferenceCmd** to MainFrm.H. It should be added after the //}} AFX_MSG_MAP comment and before the DECLARE_MESSAGE_MAP macro.

➢ **Implement visual feedback**

1. Since differencing is simulated, finding the next difference also will need to be simulated. You will simply move randomly forward or backward through the current view in response to the message from the dialog box. Define **CMainFrame::OnFindNextDifference**.

```
void CMainFrame::OnFindNextDifference(BOOL bSearchDown,
                                      BOOL bNextDifference)
```

2. Get the active view.

```
CDiffView * pView = (CDiffView *)GetActiveView();
if(pView)
{
```

3. Find out which line (if any) is currently highlighted.

```
int nLineCnt = pView->GetRichEditCtrl().GetLineCount();
LONG lStart = 0;
LONG lEnd   = 0;
pView->GetRichEditCtrl().GetSel(lStart, lEnd);
LONG lCurLine = pView->GetRichEditCtrl().LineFromChar(lStart);
```

4. If you are searching forward, find a line randomly between the current line and the end of the text; if you are searching back, find a line randomly between the current line and the beginning of the text.

```
int nNewLine;
if(bSearchDown)
{
    nNewLine = lCurLine + (rand() % (nLineCnt-lCurLine)+1);
}
else
{
    nNewLine = rand() % (lCurLine+1) + 1;
}
```

5. Find the starting and ending characters of that line and select them.

```
lStart = pView->GetRichEditCtrl().LineIndex(nNewLine);
lEnd = lStart + pView->GetRichEditCtrl().LineLength(nNewLine);

pView->GetRichEditCtrl().SetSel(lStart, lEnd);
```

6. There is no difference between the next difference and the next equal text in this simulation. Use the status bar text to show which is the current option.

```
if (bNextDifference)
{
    m_wndStatusBar.SetWindowText(_T("Found next difference"));
}
else
{
    m_wndStatusBar.SetWindowText(_T("Found next equal run"));
}
```

7. Save MainFrm.Cpp.

The complete function is shown in this sample code.

```
void CMainFrame::OnFindNextDifference(BOOL bSearchDown,
                                      BOOL bNextDifference)
{
    CDiffView * pView = (CDiffView *)GetActiveView();
    if(pView)
    {
        int nLineCnt = pView->GetRichEditCtrl().GetLineCount();
        LONG lStart = 0;
        LONG lEnd   = 0;
```

```
        pView->GetRichEditCtrl().GetSel(lStart, lEnd);
        LONG lCurLine = pView->GetRichEditCtrl().LineFromChar(lStart);

        int nNewLine;
        if(bSearchDown)
        {
            nNewLine = lCurLine + (rand() % (nLineCnt-lCurLine)+1);
        }
        else
        {
            nNewLine = rand() % (lCurLine+1) + 1;
        }

        lStart = pView->GetRichEditCtrl().LineIndex(nNewLine);
        lEnd = lStart + pView->GetRichEditCtrl().LineLength(nNewLine);

        pView->GetRichEditCtrl().SetSel(lStart, lEnd);

        if (bNextDifference)
        {
            m_wndStatusBar.SetWindowText(_T("Found next difference"));
        }
        else
        {
            m_wndStatusBar.SetWindowText(_T("Found next equal run"));
        }
    }
}
```

8. Build and run Diff.Exe.

The completed code for this exercise is in \Labs\C06\Lab02\Ex03.