Lab 5.6: Adding a Dialog Bar

Objectives

After completing this lab you will be able to:

- Create the resources to support a dialog bar.
- Use a menu command to toggle the dialog bar on and off.
- Use a combo box drop-down list in the dialog bar to display information.
- Use an event handler to handle dialog bar events.
- Use CWnd::OnCommand to handle dialog bar events.

Prerequisites

This lab assumes that you are familiar with the topics covered this chapter, and have implemented menus and command handlers. A basic understanding of combo boxes is also helpful, but not necessary.

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: **60 minutes**.

Exercises

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

Exercise 1: Creating Dialog Bar Resources

In this exercise, you will create the dialog bar resources for an application.

Exercise 2: Implementing a Dialog Bar

In this exercise, you will implement a nonfunctional dialog bar, which includes a menu to toggle the dialog bar on and off.

Exercise 3: Implementing a Combo Box for the Dialog Bar

In this exercise, you will implement a combo box on the left side of the dialog bar by manually adding a message-map entry and its associated command handler.

Exercise 4: Implementing a Combo Box with Command Handlers

In this exercise, you will implement a combo box on the right side of the dialog bar using an OnCommand handler.

Copy the contents of \Labs\C05\Lab06\Baseline to your working directory. The completed code for these exercises is in \Labs\C05\Lab06\Xxx, where the Xxx is the exercise number.

Exercise 1: Creating Dialog Bar Resources

If you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C05\Lab06\Baseline.

In this exercise, you will use the resource editors to add the menu option, and to create the dialog bar resources and symbols.

➢ **Add a Menu and Dialog Bar to the ShowDiff Application**

1. Use the ResourceView to open the menu resource IDR_MAINFRAME for editing.
2. In the menu editor, choose the top-level item, View, then double-click the empty box at the bottom (beneath Status Bar).
3. Use the Menu Item Properties dialog sheet to set the menu item ID to ID_VIEW_DIALOGBAR=0xE821. This ID will cause the associated resources to be used in calculating the layout of toolbars. If you do not want a menu item to toggle the dialog bar, you can just use View Resource Symbols to insert the name and ID of the menu item.

   For a more detailed explanation of toolbar layout, see Technical Note TN031: Control Bars in the Visual C++ online documentation.
4. Use the property sheet to set the Caption to &Dialog Bar.
5. Use the property sheet to set the Prompt to **Show or hide the dialog bar\nToggle Dialog Bar**.

➢ **Create the Dialog Bar Resources**

1. Use the ResourceView to insert a new dialog resource. Right-click the Dialog folder and choose Insert.
2. Expand the Dialog icon by clicking the plus (+) symbol, and choose IDD_DIALOGBAR.
3. Delete the **Static TODO** control from the dialog bar.
4. Drag a combo box from the control palette to the dialog. Place the combo box in the upper left corner. The position of the combo box should be 10, 1 and its size should be 142 by 12. Set its ID to **IDC_LEFT.** Set its styles to Type **Drop List**, and check **Auto HScroll.**
5. Select the combo box. Use the menu to copy and paste a second combo box. Name the second box **IDC_RIGHT**. Place it slightly to the right of the first combo box. Align the top edges of the combo boxes, then reduce the size of the dialog box to just contain the combo boxes.
6. Save your work.

➢ **Provide Prompts and Tootips for the Combo Boxes**

1. Use the String Table.
2. Add a new string with an ID of **IDC_LEFT** and a caption: **Select a file to display in the left window\nDisplay on Left**
3. Add a new string with an ID of **IDC_RIGHT** and a caption: **Select a file to display in the right window\nDisplay on Right**
4. Save your work.

The completed code for this exercise is in \Labs\C05\Lab06\Ex01.

Exercise 2: Implementing a Dialog Bar

Continue with the files created in Exercise 1, or if you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C05\Lab06\Ex01.

In this exercise you will create the dialog bar, and enable the menu to toggle the bar on and off.

➢ **Associate the Dialog Bar Resource with the CMainFrame Class**

1. Use the ResourceView to open the dialog resources. Highlight **IDD_DIALOGBAR** with a single click. Start the Class Wizard. Choose the radio button **Select an existing class,** choose **CMainFrame,** and then click Select.
2. Click Yes to create an association between **CMainFrame** and **CDialog**. Close ClassWizard, and then restart it.
3. In ClassWizard, make sure that the selected class name is **CMainFrame**. In the Object IDs, select the ID **ID_VIEW_DIALOGBAR** and double-click the word **COMMAND** in the Messages window. Accept the default name of **OnViewDialogbar**. (Do not select Edit Code yet.)
4. Double-click **UPDATE_COMMAND_UI**. Accept the default name of **OnUpdateViewDialogbar**.
5. Exit ClassWizard.

➢ **Add a Protected Variable to the CMainFrame class**

- Right-click **CMainFrame** in ClassView and add a protected variable.

  ```
  CDialogBar m_wndDialogBar
  ```

➢ **Implement the Dialog Bar in the CMainFrame::OnCreate Function**

1. In the function **CMainFrame::OnCreate**, call **CDialogBar::Create** using the data member **m_wndDialogBar**. Place the code at the bottom of the function, just before the **return 0** statement, as follows.

   ```
   m_wndDialogBar.Create( this, IDD_DIALOGBAR, CBRS_TOP, ID_VIEW_DIALOGBAR );
   ```

   The first argument, *this*, sets the mainframe as the parent of the dialog bar window. The second argument, *IDD_DIALOGBAR*, is the graphic resource used for the dialog bar. The third argument, *CBRS_TOP*, says the bar should initially align at the top of the main frame. The last argument, *ID_VIEW_DIALOGBAR*, is the control ID. You set this ID to 0xE821 when you created the menu resource to tell the operating system that the associated window should be treated as a control bar.

2. Because a dialog bar is similar to a toolbar, copy the code that AppWizard created for the toolbar to set docking and styles, and to set tool tips. Change the variable name of the toolbar to that of the dialog bar variable. Place this code after the call to **CDialogBar::Create**. Edit the docking style of the dialog bar to allow docking at only the top or bottom of the frame.

   ```
   m_wndDialogBar.EnableDocking(CBRS_ALIGN_TOP|CBRS_ALIGN_BOTTOM);
   EnableDocking(CBRS_ALIGN_ANY);
   DockControlBar(&m_wndDialogBar);
   m_wndDialogBar.SetBarStyle(m_wndDialogBar.GetBarStyle() |
       CBRS_TOOLTIPS | CBRS_FLYBY);
   ```

3. Open the header file for **CMainFrame**. Remove the following statement from the beginning of the file (if it is there).

   ```
       #define _BASELINE_CODE_
   ```

4. While you have the **CMainFrame** header file open, locate the definition of the enumerated values LEFT and RIGHT. Now that you have added controls with the identifiers IDC_LEFT and IDC_RIGHT, you can complete the declaration of this enumerated type as follows.

   ```
       enum { LEFT = IDC_LEFT, RIGHT = IDC_RIGHT };
   ```

5. Build and run the application. If it is working correctly, the dialog bar should start docked beneath the standard toolbar. It should tear away, float, and dock. If you float the dialog bar, and then close it, you will not be able to get it back.

   The next step implements toggling the dialog bar on and off from the menu.

➢ **Implement the Menu to Toggle the Dialog Bar On and Off**

1. Find the stub implementation of **CMainFrame::OnViewDialogbar**. Use the functions **CDialogBar::ShowControlBar** and **CWnd::IsWindowVisible** to toggle the dialog bar on and off.

   ```
       ShowControlBar( & m_wndDialogBar,
       ! m_wndDialogBar.IsWindowVisible( ), FALSE );
   ```

2. Find the stub implementation of **CMainFrame::OnUpdateViewDialogbar**. Use the functions **CCmdUI::SetCheck** and **CWnd::IsWindowVisible** to update the menu item.

   ```
       pCmdUI->SetCheck( m_wndDialogBar.IsWindowVisible( ) );
   ```

3. Build and run the application. Verify that the menu toggles the dialog bar on and off.

The completed code for this exercise is in \Labs\C05\Lab06\Ex02.

Exercise 3: Implementing a Combo Box for the Dialog Bar

Continue with the files created in Exercise 2, or if you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C05\Lab06\Ex02.

In this exercise, you will implement the functionality for the left-side combo box on the dialog bar by manually adding a command handler. (You also could code the right-side combo box functionality the same way, but you will learn a different method in Exercise 4.)

➢ Create message map entries for the left combo box

The combo boxes each contain a list of files opened by the user. When the user selects a filename from the list, the program will open and display that file.

1. Right-click the **CMainFrame** icon in the ClassView window. Click Add Member Function to add the following function header:

   ```
   afx_msg void OnSelendokLeft()
   ```

2. Insert a message map entry in the **CMainFrame** implementation file. Place the entry between the end of the code section created by ClassWizard and the end of the Message Map as follows.

   ```
       //}}AFX_MSG_MAP
       ON_CBN_SELENDOK ( IDC_LEFT, OnSelendokLeft )
   END_MESSAGE_MAP()
   ```

➢ Code the function CMainFrame::OnSelendokLeft

You added this message handler in the previous step. In this handler, you will extract the selected string from the combo box, then use that string as an argument to the function **CMainFrame::ResetFile. ResetFile** is not part of MFC; it is part of the baseline code written for this application.

1. Get a pointer to the combo box using **CWnd::GetDlgItem** and the enum **CMainFrame::LEFT**.
2. Create a **CString** variable.
3. Call the function **CWnd::GetWindowText** to modify the **CString** variable.
4. Invoke **CMainFrame:: ResetFile** using the enum and the string variable.
5. The completed code for the **CMainFrame::OnSelendokLeft** function follows.

   ```
   void CMainFrame::OnSelendokLeft()
   {
       // TODO: Add your control notification handler code here
       CComboBox * pCmb =
           ( CComboBox * ) m_wndDialogBar.GetDlgItem( LEFT );
       CString str;
       pCmb->GetWindowText( str );  //Get the selected text
       ResetFile( LEFT, str );
   }
   ```

6. Build and run the application. Use File Open at least twice. Drop the left list box to choose a file name. If the application is working correctly, when you click on the name, the indicated file will load into the view.

The completed code for this exercise is in \Labs\C05\Lab06\Ex03.

Exercise 4: Implementing a Combo Box with Command Handlers

Continue with the files created in Exercise 3, or if you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C05\Lab06\Ex03.

In this exercise, you will implement the functionality for the right-side combo box on the dialog bar by using the ClassWizard to add a command handler. You will use an **OnCommand** handler to implement combo box functionality.

➢ Add a Command handler to the CMainFrame Class

Use ClassWizard to add a handler to the **CMainFrame** class for the message event **OnCommand**. The parameter, **wParam**, contains the identifier of the message source in its low-order word. The high-order word of **wParam** contains the notification code. When the message is the selection notification from the right-hand combo box, get the selected file name from the combo box. Display the selected file in the right-hand view.

1. Initialize an **int** variable with the message value. Use the **HIWORD** macro to extract the message.
2. Create a second **int** variable. Use **LOWORD** to extract and store the control ID in the variable.
3. If the message is **CBN_SELENDOK** and the control is **IDC_RIGHT**, then get the selected text from the combo box.
4. Load the right-side view using the function **CMainFrame::ResetFile**. The completed code for adding the command handler and coding this function follows.

```
BOOL CMainFrame::OnCommand(WPARAM wParam, LPARAM lParam)
{
    // TODO: Add your specialized code here and/or call the base class

    int msg  = HIWORD( wParam );  //Get the message
    int ctrl = LOWORD( wParam );  //Get the message source

    //Check for notification message from the right-side combo
    if ( CBN_SELENDOK == msg && IDC_RIGHT == ctrl )
    {
        CString str;
        CComboBox * pCmb =
            ( CComboBox * ) m_wndDialogBar.GetDlgItem( ctrl );
        pCmb->GetWindowText( str );  //Get selected text
        ResetFile( ctrl, str );      //Change to the selected file
    }
    return CFrameWnd::OnCommand(wParam, lParam);
}
```

5. Build and run the application. Use File Open at least twice. Drop the right list box to choose a file name. If the application is working correctly, when you click on the name, the indicated file will be displayed. The left list box should work the same way.

The completed code for this exercise is in \Labs\C05\Lab06\Ex04.