Lab 5.7: Adding a Progress Control

Objectives

After completing this lab, you will be able to:

- Create and control a progress control.
- Create and control a label for a progress control.
- Display both the control and its label in the context of the status bar.

Prerequisites

To complete this lab, you should have mastered the material covered in Labs 5.1 through 5.3.

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: **45 minutes**.

Exercises

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

Exercise 1: Creating a Progress Control

In this exercise, you will integrate a progress control into the default status bar that AppWizard provides as part of the MainFrame. You will also provide the progress control with a label.

Exercise 2: Implementing the Progress Control

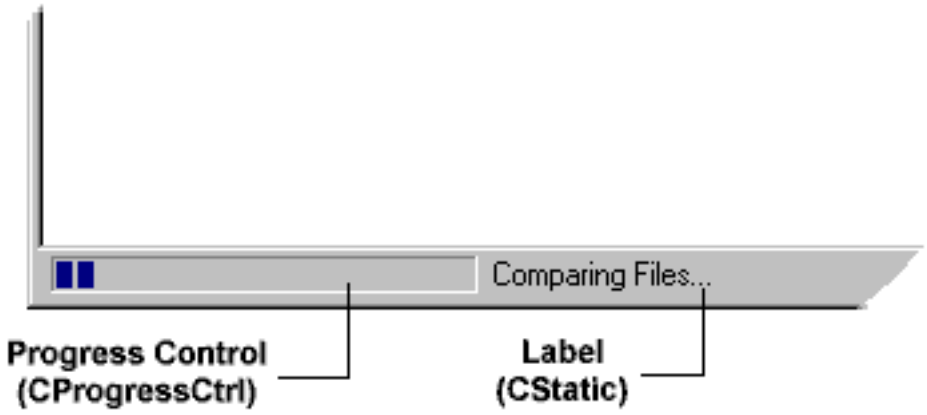
In this exercise, you will provide the supporting code and resources to implement the progress control from Exercise 1 in an application.

In this lab, you can use the project you created in Labs 5.1, 5.2, and 5.3. You can copy this project from \Labs\C05\Lab07\Baseline. The completed code for these exercises is in \Labs\C05\Lab07\Xxx, where the Xxx is the exercise number.

Exercise 1: Creating a Progress Control

This lab is provided for the advanced student and is optional. If you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C05\Lab07\Baseline.

In this exercise, you will integrate a progress control with the status bar that AppWizard provides as a default part of the MainFrame. You also will add a label to the progress control.



➢ **Create a new CProgressStatusBar class derived from the generic CWnd**

1. On the View menu, click ClassWizard; or, press CTRL+W.
2. Choose Add Class and add **CProgressStatusBar**. Base it on generic **CWnd**.
3. Click OK to create the new class.

➢ **Edit the generated files to derive CProgressStatusBar from CStatusBar**

1. Open ProgressStatusBar.h.
2. On the Edit menu, click Replace.
3. Replace all instances of **CWnd** with **CStatusBar**.
4. Save ProgressStatusBar.H.
5. Repeat this procedure with ProgressStatusBar.Cpp.

➢ **Add new members to CProgressStatusBar**

CProgressStatusBar needs to track the member controls in the derived class, and the state and position of these member controls. Right-click **CProgressStatusBar** in ClassView and add four protected variables:

1. Add a member for the progress control itself.

```
CProgressCtrl   m_ProgressCtrl
```

2. Add a member for the static control that displays the label.

```
CStatic     m_ProgressLabel
```

3. **CProgressStatusBar** suppresses the default painting of its **CStatusBar** parent when you display the progress indicator. To track this, add a member that is set when a progress control is in use.

```
BOOLm_bProgressMode
```

4. **CProgessStatusBar** needs to store the width of the progress control to position the label. Add:

```
int m_nProgressCtrlWidth
```

➢ **Create a method to set the width of the progress control**

1. Right-click **CProgressStatusBar** in ClassView and select Go to Definition. Immediately before the class definition, add three constants. First, declare a constant for the default width of the progress control.

```
const int PROGRESS_CTRL_CX = 160;
```

Establish constants for the position and size of the progress control and its label. In ProgressStatusBar.H, declare constants for the horizontal and vertical margins and the width of the progress control.

```
const int X_MARGIN = 5; // X value used for margins
                                // and control spacing
const int Y_MARGIN = 2; // Y value used for margins
                                // and control spacing
```

2. Right-click **CProgressStatusBar** in ClassView and add a public function to set the control width.

```
void SetProgressCtrlWidth(UINT nWidth =
                                PROGRESS_CTRL_CX)
```

3. Write the function body to set **m_nProgressCtrlCX** to the passed parameter.

```
void SetProgressCtrlWidth(UINT nWidth)
{
        m_nProgressCtrlWidth = nWidth;
}
```

4. Save ProgressStatusBar.Cpp

➢ **Create a method to lay out the progress control and its label**

1. Right-click **CProgressStatusBar** in ClassView and add a public function:

```
void RecalcProgressDisplay()
```

2. Go to the implementation of **RecalcProgressDisplay**. You may assume when this function is called that both the progress control and its labels have been created. Code the function to get the client rectangle of the status bar.

```
CRect ControlRect;
CRect ClientRect;
GetClientRect(&ClientRect);
ControlRect = ClientRect;
```

3. Adjust the rectangle for the progress control so that it reflects the X and Y margins, and the progress control width.

```
//  First the Progress bar

ControlRect.left += X_MARGIN;
ControlRect.right = ControlRect.left +
                            m_nProgressCtrlWidth;
ControlRect.top += Y_MARGIN;
ControlRect.bottom -= Y_MARGIN;
```

4. Position the progress control within this rectangle.

```
m_ProgressCtrl.MoveWindow(ControlRect, FALSE);
```

5. Adjust the rectangle so that it occupies the space remaining to the right of the progress control, except for the X margins.

```
ControlRect.left = ControlRect.right + X_MARGIN;
ControlRect.right = ClientRect.right - X_MARGIN;
```

6. Position the label within this rectangle.

```
m_ProgressLabel.MoveWindow(ControlRect, FALSE);
```

7. Save ProgressStatusBar.Cpp. The complete function follows.

```
void CProgressStatusBar::RecalcProgressDisplay()
```

```
{
    //  Adjust the postions of the Label and Progress Controls
    //  Place the Label Control to the right of the
    //  Progress Control
    //
    //      [Progress Control] Label Text...

    CRect ControlRect;
    CRect ClientRect;
    GetClientRect(&ClientRect);
    ControlRect = ClientRect;

    //  First the Progress bar

    ControlRect.left += X_MARGIN;
    ControlRect.right = ControlRect.left + m_nProgressCtrlWidth;
    ControlRect.top += Y_MARGIN;
    ControlRect.bottom -= Y_MARGIN;

    m_ProgressCtrl.MoveWindow(ControlRect, FALSE);

    //  Then the text label using the rest of the status
    //  bars client area

    ControlRect.left = ControlRect.right + X_MARGIN;
    ControlRect.right = ClientRect.right - X_MARGIN;

    m_ProgressLabel.MoveWindow(ControlRect, FALSE);
}
```

➢ **Create a method to set the label for the progress control**

1. Right-click **CProgressStatusBar** in ClassView and add a public function.

```
void SetProgressLabel(LPCSTR lpszProgressLabel)
```

2. Go to the implementation of **SetProgressLabel** and have it set the text of the static to the passed parameter.

```
m_ProgressLabel.SetWindowText(lpszProgressLabel);
```

3. If the program displays the progress bar at the time of the call, update the display.

```
if(m_bProgressMode)
{
    RecalcProgressDisplay();
    Invalidate();
    UpdateWindow();
}
```

4. Save ProgressStatusBar.Cpp. The complete function follows.

```
void CProgressStatusBar::SetProgressLabel(LPCSTR
                                                lpszProgressLabel)
{

    m_ProgressLabel.SetWindowText(lpszProgressLabel);

    //  If were currently displaying progress, update
    //  placement of label and progress control

    if(m_bProgressMode)
```

```
    {
        RecalcProgressDisplay();
        Invalidate();
        UpdateWindow();
    }
}
```

➢ **Create a method to show or hide the progress control**

1. Right-click **CProgressStatusBar** in ClassView and add **ShowProgressDisplay** as a public function.

```
void ShowProgressDisplay(BOOL bShow = TRUE)
```

2. Go to the implementation of **ShowProgressDisplay** and set the mode to the passed parameter.

```
m_bProgressMode = bShow;
```

3. If you want to show the progress control, recalculate the display.

```
if(m_bProgressMode)
{
    RecalcProgressDisplay();
}
```

4. Show or hide the static and the progress control, as appropriate.

```
m_ProgressLabel.ShowWindow(m_bProgressMode ?    SW_SHOW :
                                                        SW_HIDE);
m_ProgressCtrl.ShowWindow(m_bProgressMode ? SW_SHOW :
                                                        SW_HIDE);
```

5. Invalidate and update the status bar.

```
Invalidate();
UpdateWindow();
```

6. Save ProgressStatusBar.Cpp. The complete function follows.

```
void CProgressStatusBar::ShowProgressDisplay(BOOL
                                                    bShow)
{
    m_bProgressMode = bShow;
    if(m_bProgressMode)
    {
        RecalcProgressDisplay();
    }
    m_ProgressLabel.ShowWindow(m_bProgressMode ?    SW_SHOW :
                                                        SW_HIDE);
    m_ProgressCtrl.ShowWindow(m_bProgressMode ? SW_SHOW :
                                                        SW_HIDE);
    Invalidate();
    UpdateWindow();
}
```

➢ **Create a handler for WM_CREATE**

1. Start the ClassWizard and create a handler for the WM_CREATE message in **CProgressStatusBar.**
2. Edit the code for this handler. Create the progress control with no positions or percentages.

```
if(!m_ProgressCtrl.Create(
                0,                  // Style - Don't Show
                                    //  Position or Percent
                CRect(0,0,0,0), //Initial position
                this,               // Parent ID
                0))                 // Child ID
```

```
    {
        return -1;
    }
```

3. Create the static label control.

```
    if(!m_ProgressLabel.Create(
                    NULL,               // Text
                    WS_CHILD|SS_LEFT,   // Style
                    CRect(0,0,0,0), // Initial position
                    this))              // Parent
    {
        return -1;
    }
```

4. Set the font of the static to the same font as that of the status bar.

```
    m_ProgressLabel.SetFont(GetFont());
```

5. Tell the calling function that you succeeded.

```
      return 0;
```

6. Save ProgressStatusBar.Cpp. The complete text of the function follows.

```
int CProgressStatusBar::OnCreate(LPCREATESTRUCT
                                    lpCreateStruct)
{
    if (CStatusBar::OnCreate(lpCreateStruct) == -1)
        return -1;

    //  Create the Progress Control - we'll calculate its   //  size and
position later - in response to a
    //  ShowProgressDisplay() call.

    if(!m_ProgressCtrl.Create(
                    0,                  // Style - Don't Show
                                        //  Position or Percent
                    CRect(0,0,0,0), //Initial position
                    this,               // Parent ID
                    0))                 // Child ID
    {
        return -1;
    }

    //  Create the Progress Label - we'll calculate its
    //  size and position later - in response to a
    //ShowProgressDisplay() call.

    if(!m_ProgressLabel.Create(
                    NULL,               // Text
                    WS_CHILD|SS_LEFT,   // Style
                    CRect(0,0,0,0), // Initial position
                    this))              // Parent
    {
        return -1;
    }

    //  Use the same font as the Status Bar
```

```
    m_ProgressLabel.SetFont(GetFont());

    return 0;
}
```

➢ **Create a handler for WM_PAINT**

1. Start ClassWizard and create a handler for the WM_PAINT message in **CProgressStatusBar**.
2. Edit the code for this handler. If the progress control does not display, pass on the WM_PAINT message.

```
    if(!m_bProgressMode)
    {
        CStatusBar::OnPaint();
    }
```

3. Save ProgressStatusBar.Cpp. The complete function follows.

```
void CProgressStatusBar::OnPaint()
{
    //  If were displaying the progress control, then we    //  need to
handle painting of the Status Bar,
    //  otherwise defer to the base class

    if(!m_bProgressMode)
    {
        CStatusBar::OnPaint();
    }
}
```

➢ **Update the constructor to set defaults**

1. Set the mode to FALSE and the control width to the declared default.

```
    m_bProgressMode = FALSE;
    m_nProgressCtrlWidth = PROGRESS_CTRL_CX;
```

2. Save ProgressStatusBar.Cpp.

The completed code for this exercise is in \Labs\C05\Lab07\EX01.

Exercise 2: Implementing the Progress Control

Continue with the files you created in Exercise 1; if you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C05\Lab07\Ex01.

In this exercise, you will provide the supporting code and resources to use the **CProgressStatusBar** class in the ShowDIFF application.

➢ **Make the status bar available to other classes**

1. Open MainFrm.Cpp.
2. Include ProgressStatusBar.H before Splitter.H.
3. Save MainFrm.Cpp.
4. Open MainFrm.H.
5. Add a method to return a pointer to the status bar.

```
    CProgressStatusBar *GetStatusBar()
                        { return &m_wndStatusBar; }
```

6. Change the status bar type from **CStatusBar** to **CProgressStatusBar**.
7. Save MainFrm.H.
8. Open ProgressStatusBar.H.

9. Add a method to return a pointer to the progress control itself.

```
CProgressCtrl * GetProgressCtrl()
                        { return &m_ProgressCtrl; }
```

10. Save ProgressStatusBar.H.
11. Include ProgressStatusBar.H before Splitter.H in Diff.Cpp, DiffDoc.Cpp, and DiffView.Cpp.
12. Save Diff.Cpp, DiffDoc.Cpp, and DiffView.Cpp.

➢ **Start the String Table editor and add a label string**

1. Open the ResourceView to the String Tables.
2. Open the String Table resource.
3. The focus is on the last line in the String Table. Edit its properties by double-clicking the last line, or by pressing ENTER.
4. Set the ID to IDS_COMPARING, and the Caption to Comparing files.
5. Save the String Table.

➢ **Set the label of the status bar to the resource string**

1. Go to the implementation of **CDiffDoc::RunComparison**. All the code you add in the following steps should be placed before the code that is already there.
2. In **RunComparison**, before processing the files, get the status bar.

```
CProgressStatusBar *pStatus = CDiffApp::GetApp()->
            GetMainFrame()->GetStatusBar();
```

3. Add a statement to determine if pStatus if valid.

```
if (pStatus)
{
```

4. Declare a **CString**.

```
    CString Label;
```

5. Load the resource string.

```
    Label.LoadString(IDS_COMPARING);
```

6. Set the label of the progress bar.

```
    pStatus->SetProgressLabel(Label);
```

➢ **Show the status bar**

- After setting the label, set the status bar to "Progress mode."

```
pStatus->ShowProgressDisplay(TRUE);
```

➢ **Step the status bar through its range**

Normally, you update a status bar as the result of an action that your program performs. Because the key time-consuming action of ShowDIFF is the differencer itself, and the differencer is not yet implemented, you can simulate that action with a timing loop.

1. Get a pointer to the progress control.

```
    CProgressCtrl *pProgress = pStatus->GetProgressCtrl();
```

2. Set the range of the progress control to ten steps, starting at 0.

```
    pProgress->SetRange(0,10);
```

3. Display the first step of the range.

```
pProgress->SetStep(1);
```

4. In a timing loop, do ten steps of the progress control.

```
while (pProgress->StepIt() != 10)
{
    for(LONG i = 0; i <2000000; l++);
}
```

➢ **Restore the status bar to its normal behavior**

1. After you stop the progress control, send the status bar a message to turn off the progress bar.

```
pStatus->ShowProgressDisplay(FALSE);
```

2. Save DiffDoc.Cpp.
3. Build and run the application.
4. The code you inserted at the beginning of **RunComparison** follows.

```
void CDiffDoc::RunComparison(LPCSTR lpszFile1,
                             LPCSTR lpszFile2)
{
        //  TODO: Run a difference comparison and stick the
        //  results in the 2 splitter panes.

        //
        //  For effect, use the progress feedback available
        //  from our status bar class
        //
    CProgressStatusBar *pStatus = CDiffApp::GetApp()->
                        GetMainFrame()->GetStatusBar();
    if(pStatus)
    {
        CString Label;
        Label.LoadString(IDS_COMPARING);
        Label.Empty();
        pStatus->SetProgressLabel(Label);

            //
            //  Flip the StatusBar into Progress Mode
            //
        pStatus->ShowProgressDisplay(TRUE);

            //
            //  Simulate some bogus progress
            //
        CProgressCtrl *pProgress =
                        pStatus->GetProgressCtrl();
        if(pProgress)
        {
            pProgress->SetRange(0, 10);
            pProgress->SetStep(1);
            while(pProgress->StepIt() != 10)
            {
                for(LONG l = 0; l < 2000000; l++);
            }
        }
        pStatus->ShowProgressDisplay(FALSE);
```

```
}
```

The completed code for this exercise is in \Labs\C05\Lab07\Ex02.