Lab 5.1: Adding Static Drop-down Menus

Objectives

After completing this lab, you will be able to:

- Edit a menu resource.
- Use the ClassWizard to add COMMAND handlers.
- Implement handlers.
- Use the ClassWizard to add UPDATE_COMMAND_UI handlers.
- Set radio button or checkbox indicators in menus.
- Add buttons to the toolbar.

Prerequisites

In Labs 5.1 through 5.3, you will use the Scribble application as the basis for the exercises. For more information, see the online Visual C++ Tutorials in \Samples\MFC Samples\Tutorials\Scribble.

In this lab, you will add more functionality to the Scribble application. Using the Menu Editor, you will provide a top-level menu and submenu that enables the user to set the width of the drawing pen. You can then add a radio button indicator to show the state of a menu item.

Before starting this lab, you must be able to build an application framework using AppWizard.

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: Implementing a Static Drop-down Menu

In this exercise, you will create and implement a menu that controls the pen width in the Scribble application.

Exercise 2: Implementing State Indicators

In this exercise, you will add a radio button to indicate the pen width that is currently in effect in the Scribble application.

Exercise 3: Adding Toolbar Buttons

In this exercise, you will implement toolbar support by adding two buttons to the toolbar and connecting them to work with the menu.

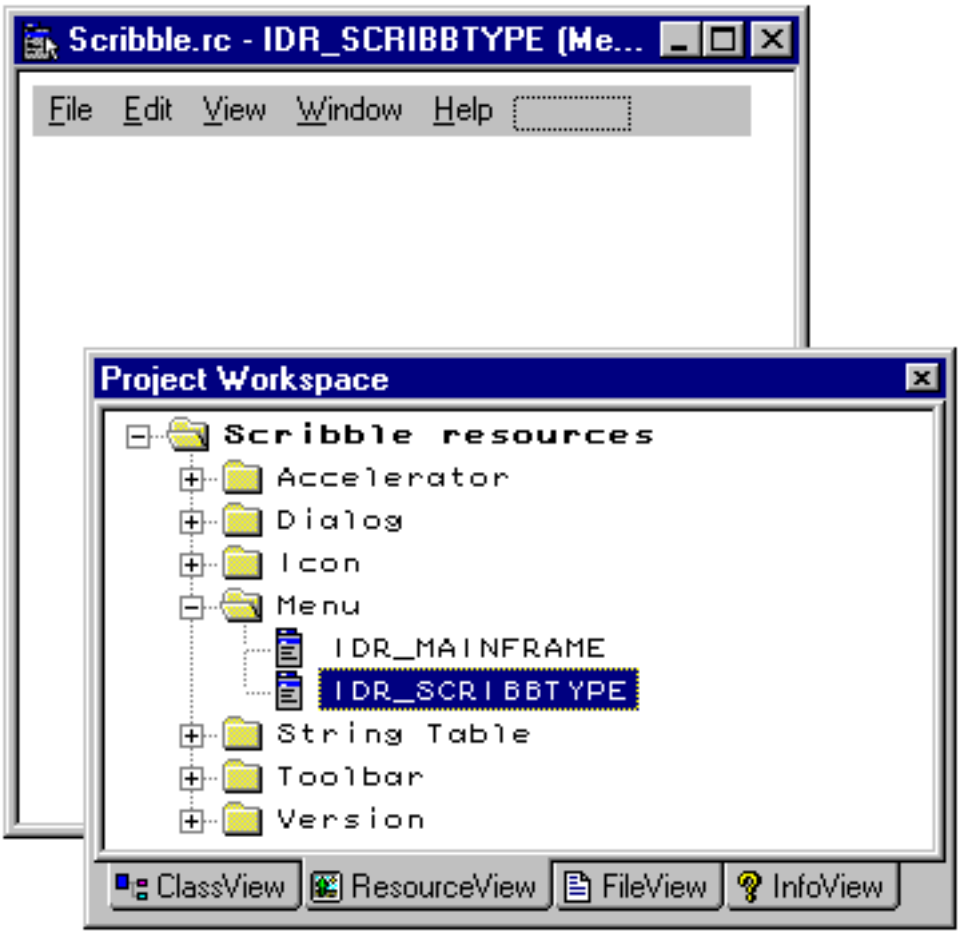
To do these labs, copy the contents of \Labs\C05\Lab01\Baseline to your working directory. The completed code for these exercises is in Labs\C05\Lab01\Xxx, where Xxx is the exercise number.

Exercise 1: Implementing a Static Drop-down Menu

If you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C05\Lab01\Baseline.

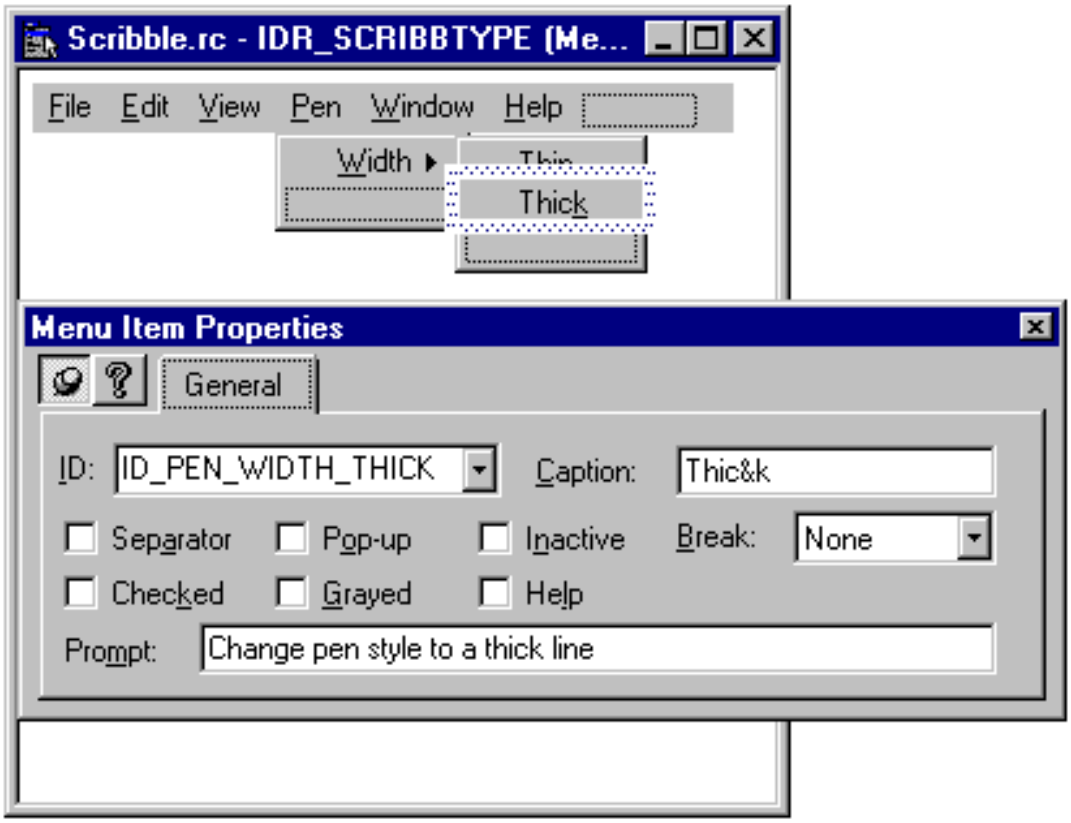
In this exercise, you will create and implement a menu that controls the pen width in the Scribble application.

➢ **Add a Pen menu with Width and Thick and Thin drop-down menus**



1. Open the Project Workspace to the ResourceView. Open the Menu folder, and then open the IDR_SCRIBBTYPE resource.
2. In the Menu editor, click the dotted box to the right of Help and drag it to the position between the View and Window items.
3. Double-click the dotted box to display the Menu Item Properties dialog box. Set the Caption to &Pen, which will display as Pen. To keep the Properties dialog box on top, press the pushpin in the upper-left corner of the dialog box.
4. Click the dotted box below the Pen menu. Set the Caption of this menu item to &Width. Because this item has a submenu, choose the Pop-up property.
5. Click the dotted box to the right of the Width menu item. Set this item's Caption to Thi&n. The menu editor assigns ID_PEN_WIDTH_THIN as this item's ID when you go to another menu item, so you can ignore the ID field. Set the Prompt of this item to "Change pen style to a thin line."

6. Click the dotted box below the Thin menu item. Set the caption of this item to Thic&k. The menu editor will assign ID_PEN_WIDTH_THICK as this item's ID when you select another menu item, so you can ignore the ID field. Set the Prompt of this item to "Change pen style to a thick line."



7. Close the Menu Item Properties dialog box.

➢ **Use ClassWizard to add handlers for the Thick and Thin menu items**

1. On the View menu, click ClassWizard; or, press CTRL+W. Click the Message Maps tab.
2. Select **CScribbleDoc** as the class name, ID_PEN_WIDTH_THIN as the object ID, and COMMAND as the message. Press Add Function to add the function and accept the default **OnPenWidthThin** as the function name.
3. Repeat this procedure to add the default handler for ID_PEN_WIDTH_THICK.

Note Consider the following guidelines when implementing handlers.

- In general, put handlers in the command-target class where they have the widest scope needed.
- When attributes are shared by multiple views or frame windows, put them in the common document.
- If attributes are not shared, put them in the view(s) or window(s) that use them.

---

4. Click Edit Code to go to ScribDoc.Cpp in the **OnPenWidthThick** handler. Replace the commented line:

```
// TODO: Add your command handler code here
```

with:

```
ChangePen(THICK);
```

You have not yet defined `ChangePen` or `THICK`; you can do this in the next section.

5. Replace the comment in **OnPenWidthThin** with:

```
ChangePen(THIN);
```

6. Save Scribdoc.Cpp. The complete functions follows.

```
void CScribbleDoc::OnPenWidthThin()
{
    ChangePen(THIN);
}

void CScribbleDoc::OnPenWidthThick()
{
    ChangePen(THICK);
}
```

➢ Provide a function to update the pen width

Scribble has a member that holds the document's current pen, **m_penCur**, and a member that holds the current pen's width.

1. Right-click **CScribbleDoc** in ClassView, and add **ChangePen** as a protected function.

```
void CScribbleDoc::ChangePen(PENWIDTH penWidth)
```

2. Set the current width member to the passed width. Write code for **ChangePen** as follows.

```
m_nPenWidth = penWidth;
```

3. Because Windows GDI objects are a limited resource, destroy them when they are no longer needed. Scribble does not need the old pen once a new one has been created, so delete the current GDI pen associated with the **m_penCur** object.

```
m_penCur.DeleteObject();
```

4. Create a new GDI pen to associate with the **m_penCur** object.

```
m_penCur.CreatePen( PS_SOLID,
                    m_nPenWidth,
                    RGB(0,0,0));
```

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

```
void CScribbleDoc::ChangePen(PENWIDTH penWidth)
{
    m_nPenWidth = penWidth;
    m_penCur.DeleteObject();
    m_penCur.CreatePen( PS_SOLID,
                        m_nPenWidth,
                        RGB(0,0,0));
}
```

➢ Integrate the changes into Scribble

While you can define a thick pen of five pixels and a thin pen of two pixels by hard-coding these values, this makes the code difficult to maintain. Instead, enumerate these values.

1. Open ScribDoc.H.
2. In the protected attributes section of the **CScribbleDoc** class declaration, define PENWIDTH.

```
enum PENWIDTH {THIN = 2, THICK = 5};
```

3. Change the type of **m_nPenWidth** from UINT to PENWIDTH.

```
PENWIDTH        m_nPenWidth;
```

4. Save ScribDoc.H.
5. Open ScribDoc.Cpp and find the **InitDocument** function.
6. Replace:

   ```
   m_nPenWidth = 2;
   ```

   with its enumerated equivalent:

   ```
   m_nPenWidth = THIN;
   ```

7. Click the header button in the WizardBar to go to ScribDoc.H.
8. Create a prototype for **ChangePen** in the protected Implementation section of the **CScribbleDoc** class declaration.

   ```
   void ChangePen(PENWIDTH penWidth);
   ```

9. Save ScribDoc.Cpp. and ScribDoc.H.
10. Build and run Scribble.

The completed code for this exercise is in \Labs\C05\Lab01\Ex01.

Exercise 2: Implementing State Indicators

The code that forms the basis for this exercise is in \Labs\C05\Lab01\Ex01.

The menu items you added are now functional. However, the user has no indication of the pen width without drawing a line. The easiest way to indicate the state of an option is to use radio button marks or check marks. Because the user can only choose between the two options of thick and thin pens, you will add a radio button as the indicator of the pen width.

➢ Use the ClassWizard to add UPDATE _COMMAND_UI handlers for Thick and Thin menu items

1. On the View menu, click ClassWizard; or, press CTRL+W.
2. Select the **CScribbleDoc** class and object ID ID_PEN_WIDTH_THICK.
3. Select the UPDATE _COMMAND_UI message.
4. Click Add Function and accept the default function name.
5. Repeat the previous steps for ID_PEN_WIDTH_THIN.

➢ Implement the UPDATE _COMMAND_UI handlers for Thick and Thin menu items

1. Choose the **OnUpdatePenWidthThick** member function and click Edit Code.
2. **SetRadio** and **SetCheck** are member functions of **CCmdUI** that take a **BOOL** value. To set the radio button indicator, check to see whether the pen width matches the menu item. Replace the `//TODO` comment with:

   ```
   pCmdUI->SetRadio(GetPenWidth() == THICK);
   ```

3. For the Thin menu item the same technique applies. Replace the //TODO comment with:

   ```
   pCmdUI->SetRadio(GetPenWidth() == THIN);
   ```

   If check marks were a more appropriate UI, you could use the same technique with **CCmdUI::SetCheck**.
4. Save ScribDoc.Cpp.

➢ Add a GetPenWidth member

1. Open ScribDoc.H.
2. Add to the public attributes section of **CScribbleDoc**.

   ```
   PENWIDTH    GetPenWidth() const { return m_nPenWidth; }
   ```

3. Save ScribDoc.H.
4. Build and run Scribble.

The completed code for this exercise is in \Labs\C05\Lab01\Ex02.

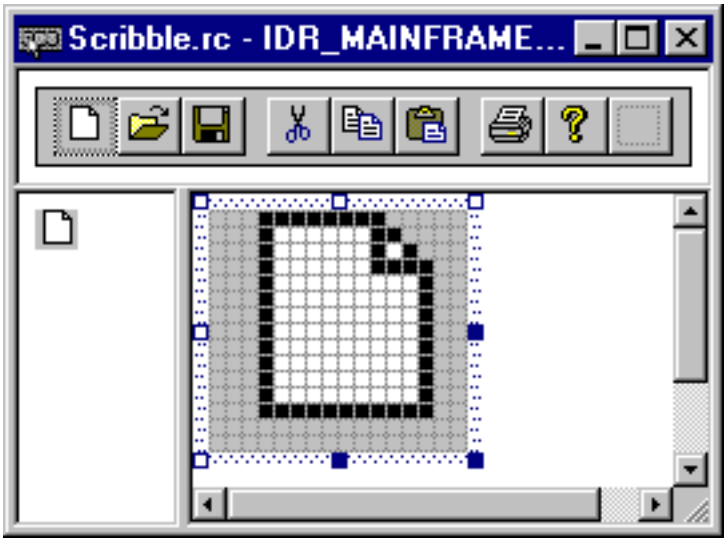
Exercise 3: Adding Toolbar Buttons

If you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C05\Lab01\Ex02.

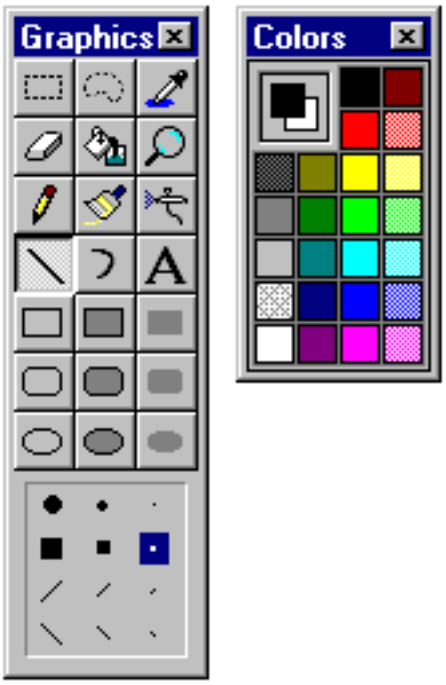
Your menu system is now complete; however, you have not yet implemented toolbar support. In this exercise, you will add two buttons to the toolbar and implement them to work with the menu.

➢ Open the toolbar resource

1. If you do not already have the resource file open, switch to ResourceView and expand the Scribble folder.
2. Expand the toolbar folder and double-click IDR_MAINFRAME.



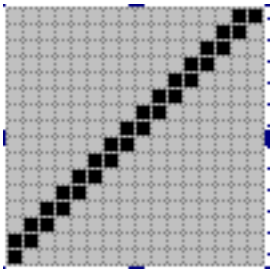
The toolbar editor opens and displays the default toolbar resource that AppWizard created for Scribble. The first button on the toolbar, selected by default, appears in the bottom pane (or magnified view) of the editor window.



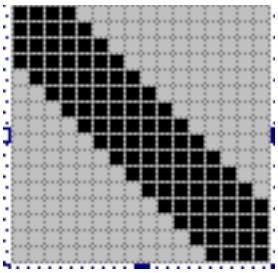
The graphics and color tools also open as part of the toolbar editor. If these graphics tools do not appear, click Toolbars on the View menu, and select Graphics and Colors in the dialog box. You can drag the graphics tools to either side of the screen and dock them to get a better view of the editor window.

➢ Delete and add toolbar buttons

1. Drag the button that you want to delete off the toolbar (in the top, or normal view pane). In this case, drag the Cut, Copy and Paste buttons off the Scribble toolbar. (This step is optional; if you do not remove these buttons, they will remain grayed in the running application but otherwise will not interfere with Scribble operations.)
2. To add a button, select the blank button at the right end of the toolbar resource. Drag this new button to the former location of the cut button. This new button receives the focus in the two split panes of the editing window. (If you want the button to appear larger in the editor, choose the Magnify tool, and select the magnification factor that you want.)
3. Choose the line tool from the graphics toolbar and choose the two-pixel wide pen.
4. Using the magnified view of the button, draw a line from the lower left corner to the upper right corner.



5. Repeat the process, creating a thick line from the upper-left to lower-right corner.



6. Your toolbar should now look like the toolbar below.

7. Save Scribble.Rc.

➢ Associate toolbar buttons with command IDs

In the next step, you associate the new Thick Line button with a command ID so that the button works when running the Scribble application. This step is identical to the one that you performed to associate a menu item with a command ID.

You bind the Thick Line button to ID_PEN_THICK and the Thin Line button toID_PEN_THIN. You defined these IDs earlier for the Thick Line and Thin Line menu commands, so Visual C++ already has written a **#define** for the ID in Resource.H. Your only task is to associate the ID with the button.

1. Double-click the Thin Line button to show the ToolBar Button Properties page. Visual C++ assigns an ID to the button, but you can choose an ID from the dropdown ID list that corresponds to the menu item that the toolbar imitates. For the Thin Line button, set its ID to ID_PEN_WIDTH_THIN.
2. Repeat with the Thick Line button. Set its ID to ID_PEN_WIDTH_THICK.

By associating the command ID with the toolbar button, the string resources become active for the button as well. When the mouse passes over the button, the prompt string displays in the status line, and the ToolTip displays next to the button.

➢ Add a Tool Tip

1. Select the Thin Line toolbar button in the editor window, and choose Properties from the View menu to display the Toolbar Button Properties page.

   In the Prompt: box, you will see the text "Change pen style to a thin line.” You entered this text as the Thin menu item in Scribble's Pen menu; it appears in the status line when the mouse passes over the menu command.
2. At the end of the Prompt text, type a newline character (\n) plus the text that you want to display in the ToolTip. (There should be no space between the newline character and the text of the ToolTip). If you want a ToolTip without a prompt string, simply start with the newline character.

   Keep this text short. For Scribble, type \nThin after the existing Prompt string.

Note You can have a ToolTip without a status bar string by starting the prompt string with \n.

---

3. Save Scribble.Rc and build Scribble.

Because of the built-in toolbar support in MFC, you were able to implement new toolbar buttons without coding. In the next lab, you can programmatically connect a toolbar button with a menu item.

The completed code for this exercise is in \Labs\C05\Lab01\Ex03.