Lab 5.2: Changing Text in 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.
- Add buttons to the toolbar.
- Add and implement COMMAND and UPDATE_COMMAND_UI handlers.

Prerequisites

Before starting this lab, you must know how to add a menu, a submenu, and menu functionality to an MFC application.

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: **30 minutes**.

Exercises

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

Exercise 1: Implementing Toggling Text

In this exercise, you will toggle the text of a menu based on the state of the pen width. When the pen width is thin, the text will read "Change to Thick." When the pen width is thick, the text will read "Change to Thin."

Exercise 2: Adding Toolbar Buttons

In this exercise, you will make the toolbar and the menu work together by using a single button's pressed and unpressed states to reflect the current state of the pen width: up for thin, down for thick.

Copy the contents of \Labs\C05\Lab02\Baseline to your working directory. The completed code for these exercises is in \Labs\C05\Lab02\Xxx, where Xxx is the exercise number.

Exercise 1: Implementing Toggling Text

If you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C05\Lab02\Baseline.

In the baseline code, the menu items have been implemented and are active. However, the user has no indication of the pen width without drawing a line. One way to provide this indication is to change the text of the menu item. When the pen is thin, the menu can say "Change to Thick" and when thick, "Change to Thin."

UPDATE_COMMAND_UI is sent for each menu item before the menu is displayed. This enables you to set the text of the menu based on the state of m_nPenWidth.

▶ Add the menu strings to the string table resource

1. Choose the ResourceView in the Project Workspace.
2. Open the String Table folder and double-click the String Table item.
3. Display the properties for the blank line at the end of the String Table resource. Set the ID to IDS_CHANGETHICK and the Caption to &Change to Thick.
4. Press ENTER and the properties for the new blank line display. Set the ID to IDS_CHANGETHIN, and the Caption to &Change to Thin.
5. Save Scribble.Rc.

▶ Use the ClassWizard to add UPDATE _COMMAND_UI handlers for the Thick or Thin menu items

1. From the View menu, select ClassWizard or press CTRL+W.
2. Select the **CScribbleDoc** class and object ID ID_PEN_CHANGETOTHICKORTHIN.
3. Select the UPDATE _COMMAND_UI message.
4. Click Add Function and accept the default function name.

▶ Implement the UPDATE _COMMAND_UI handlers for Thick or Thin menu items

1. Choose the **OnUpdatePenChangetoThickorThin** member function in the ClassWizard. Click Edit Code, or open ScribDoc.Cpp and scroll to **OnUpdatePenChangetoThickorThin**.
2. Declare a string object to hold the menu string.

```
CString MenuCaption;
```

3. As with the menu COMMAND handler, you will check the current pen width.

```
if (GetPenWidth() == THIN)
```

4. If the width is THIN, then you will load the menu string for the THICK choice.

```
    MenuCaption.LoadString(IDS_CHANGETHICK);
```

5. Otherwise, you should load the menu string for the THIN choice.

```
    MenuCaption.LoadString(IDS_CHANGETHIN);
```

6. Set the menu text to the menu string.

```
pCmdUI->SetText(MenuCaption);
```

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

```
void CScribbleDoc::OnUpdatePenChangetothickorthin(CCmdUI* pCmdUI)
{
    CString MenuCaption;
    if (GetPenWidth() == THIN)
    {
        MenuCaption.LoadString(IDS_CHANGETHICK);
    }
    else
    {
        MenuCaption.LoadString(IDS_CHANGETHIN);
    }
    pCmdUI->SetText(MenuCaption);
}
```

8. Build and run Scribble.

The completed code for this exercise is in \Labs\C05\Lab02\Ex01.

Exercise 2: Adding Toolbar Buttons

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\Lab02\Ex01.

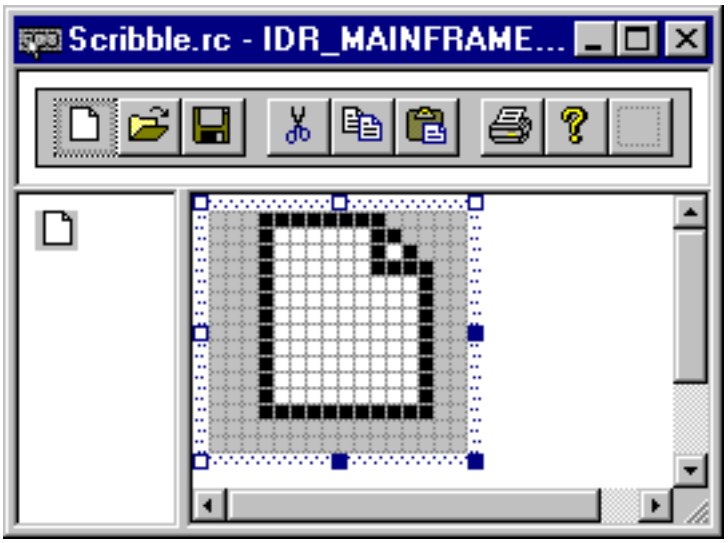
Your menu system is now complete; however, you have not yet implemented toolbar support. In this lab, there is no simple one-to-one relationship between a toolbar button and the menu function. There are two methods for making the toolbar and the menu work together:

1. Use a single button's pressed and unpressed state to reflect the current pen state: up for thin, down for thick.
2. Use two buttons, one for thick and one for thin, with the pressed state reflecting the current pen.

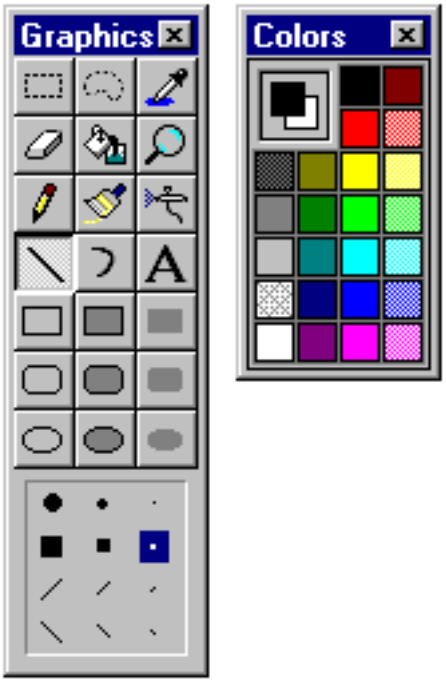
In this exercise, you will implement the first method. The coding techniques are easily applicable to the second method.

▶ Open the toolbar resource

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



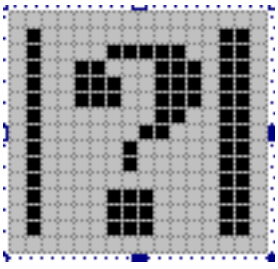
The toolbar editor opens and displays the default toolbar resource that AppWizard created for Scribble. The first button on the toolbar, selected by default, appears in the bottom pane (magnified view) of the editor window.



The graphics and color tools also open as part of the toolbar editor. If these graphics tools do not appear, choose Toolbars from 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 appear dimmed in the running application but will not interfere with Scribble operations.)
2. To add a button, select the blank button to the right of the toolbar resource. Drag this new button to where the cut button was. This new button receives 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 one-pixel-wide pen.
4. Using the magnified view of the button, draw a single-pixel line on the left of the button.
5. Choose the line tool from the graphics toolbar and choose the two-pixel-wide pen.
6. Using the magnified view of the button, draw a two-pixel line on the right of the button.
7. Use the Text tool to type a question mark between the two lines. Set the font to Times New Roman, 14 point bold.



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

9. Save Scribble.Rc.

▶ Assign an ID and a ToolTip

1. Double-click the toggle line button to show the Toolbar Button Properties page. Visual C++ will assign an ID to the button, but you will want to create a meaningful ID. Set the ID to ID_PEN_TOGGLE.
2. A toolbar button prompt has two parts: a status bar string and a ToolTip string. These two strings are separated by a newline (\n); you can omit the newline if you do not have a ToolTip. Set the prompt to:

```
Toggle line between thick and thin\nToggle pen
```

▶ Implement COMMAND and UPDATE_COMMAND_UI handlers

1. Start ClassWizard by choosing ClassWizard from the View menu or by pressing CTRL+W.
2. Choose the **CScribbleDoc** class, Object ID ID_PEN_TOGGLE, and the COMMAND message. Press Add Function and change the function name to that of the menu COMMAND handler, **OnPenChangetothickorthin**. This will cause MFC to use the same COMMAND handler for the toolbar and the menu.
3. Choose the UPDATE_COMMAND_UI message. Press Add Function and accept the default handler.
4. With OnUpdatePenToggle selected, press Edit Code.
5. SetRadio(TRUE) causes a toolbar button to appear as depressed. Check the width of the pen, and set the radio button state based on the width.

```
pCmdUI->SetRadio((GetPenWidth() == THIN)? FALSE: TRUE);
```

6. Save ScribDoc.Cpp and build Scribble.

The completed code for this exercise is in \Labs\C05\Lab02\Ex02.