Lab 5.3: Enabling Menu Items

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.
- Enable and disable menu items.
- Change text of menu items.
- Add buttons to the toolbar.
- Manage multiple toolbar resources.

Prerequisites

Before starting this lab, you must be able to implement menu functionality and dynamically change menu items.

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 State Indicators

In this exercise, you will use command ranges to consolidate the command handlers for a menu and then add radio buttons to the menu that indicate six different pen-width states.

Exercise 2: Toggling the Pen Width Option

In this exercise, you will enable the More Choices menu item to extend the user's options for selecting pen widths.

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.

Copy the contents of \Labs\C05\Lab03\Baseline to your working directory. The completed code for these exercises is in \Labs\C05\Lab03\Xxx, where the Xxx is the exercise number.

Exercise 1: Implementing State Indicators

If you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C05\Lab03\Baseline.

➢ Look at the Application

1. Build and run the program.

After you copy the source code from BASELINE, build and run the program. This program is similar to the application you created in Lab 5.2. Notice that none of the submenu items in the Pen menu is enabled because these items do not have command handlers. The first menu item, More Choices, will be used to limit the menu choices to two-pixel and five-pixel widths. The other six menu choices will set the pen width from to one to six pixels. Since the code required to do this is the same except for a single function argument, this is a good place to consolidate the command handlers using a command range.

2. Stop the program.

➢ **Edit the message map of the CScribbleDoc Class**

1. From the View menu, choose Resource Symbols. Notice that the identifiers ID_PEN_1PIXEL through ID_PEN_6PIXEL are consecutive integers. You willl make use of these identifiers using a command range. Using a command range requires you to manually edit the message map of a class. This is one of the few cases when editing the message map is the appropriate thing to do. Most of the time, when you want to add an entry to a class's message map, use Class Wizard to do so.
2. Open ScribDoc.Cpp and scroll down to the message map, which currently looks like this:

```
BEGIN_MESSAGE_MAP(CScribbleDoc, CDocument)
    //{{AFX_MSG_MAP(CScribbleDoc)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

3. Edit the message map so that it look like this:

```
BEGIN_MESSAGE_MAP(CScribbleDoc, CDocument)
    //{{AFX_MSG_MAP(CScribbleDoc)
    //}}AFX_MSG_MAP
    ON_COMMAND_RANGE(ID_PEN_1PIXEL, ID_PEN_6PIXEL, OnPenPixel)
END_MESSAGE_MAP()
```

Whenever you edit a message map, you must do so outside the AFX_MSG_MAP comment lines.

➢ **Add the Command Handler to the Class**

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

```
void OnPenPixel(UINT nID)
```

2. Associate pen width identifiers with command IDs. Insert the following code for the **OnPenPixel** function in ScribDoc.Cpp.

```
int penWidth = nID - ID_PEN_1PIXEL + 1;
ChangePen((CScribbleDoc::PENWIDTH)penWidth);
```

A simple calculation on the command identifier produces a number in the range of 1 to 6. Because the argument to **ChangePen** is an enumerated type, each number in the range must be cast.

3. Add enumerated values for PENWIDTH. Go to the implementation of PENWIDTH in ScribDoc.H, and add entries for 1, 3, 4, and 6 so that it looks like this:

```
enum PENWIDTH { VERY_THIN   = 1,
                THIN    = 2,
                MEDIUM      = 3,
                HEAVY       = 4,
                THICK       = 5,
                VERY_THICK  = 6 };
```

4. Build and run Scribble. Verify that you can set the pen width to each of the six possible values.
5. Use the command range to Implement the command UI handlers.

Command UI handlers also can use command ranges. Add this code to the message map of CScribbleDoc:

```
ON_UPDATE_COMMAND_UI_RANGE(ID_PEN_1PIXEL, ID_PEN_6PIXEL,
```

```
OnUpdatePenPixel)
```

As always, when editing a message map, place your new code outside the AFX_MSG_MAP comment lines.

6. Add the Command Handler to the Class

Right-click **CScribbleDoc** in ClassView and add **OnUpdatePenPixel** as a protected function.

```
void OnUpdatePenPixel(CCmdUI * pCmdUI)
```

➢ **Add Radio Buttons to Indicate the Pen Width States**

1. Insert the following code for the **OnUpdatePenPixel** function in ScribDoc.Cpp.

```
int nID = pCmdUI->m_nID;
pCmdUI->SetRadio(GetPenWidth() == (nID - ID_PEN_1PIXEL + 1) );
```

The CCmdUI member m_nID holds the command identifier. SetRadio is a member of the CCmdUI class and takes a BOOL argument.

2. Build and run Scribble. Verify that the selected pen width option has a radio button next to it on the menu.

The completed code for this exercise is in \Labs\C05\Lab03\Ex01.

Exercise 2: Toggling the Pen Width Option

Continue with the files you 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\Lab03\Ex01.

Your menu system is now complete, except that the pen widths are available at all times. In this exercise, you will enable the More Choices menu item. When More Choices is the choice, you will disable all but the 2-Pixel and 5-Pixel width options. When it is not chosen, you will set its text to Fewer Choices and enable all the pen-width options.

➢ **Create strings in the string table**

1. Open the Project Workspace to the Resource View.
2. Open the String Table resource in the String Table folder. Scroll to the end of the table and double-click in the blank line at the end to add a new string. The String Properties dialog box displays.
3. Set the ID of the new string to IDS_MORECHOICES and its caption to More &Choices. Press ENTER to insert this string into the table.
4. In the next string, set the ID to IDS_FEWERCHOICES and its caption to Fewer &Choices.
5. Save Scribble.Rc.

➢ **Add COMMAND and UPDATE_COMMAND_UI handlers**

1. Open the ClassWizard by selecting it from the View menu or by pressing CTRL+W.
2. Select the **CScribbleDoc** class and the ID_PEN_VARIABLEWIDTH object ID. Select the COMMAND message and press Add Function. Accept the default function name **OnPenVariablewidth**.
3. Repeat with the UPDATE_COMMAND_UI message.

➢ **Implement the COMMAND and UPDATE_COMMAND_UI handlers**

1. Select **OnPenVariablewidth** and press Edit Code.
2. In the COMMAND handler, you will flip the value of m_VariableWidth; if you enable the Thick and Thin options, make sure that the pen is set appropriately.

```
m_VariableWidth = !m_VariableWidth;
```

3. Get the current pen width.

```
PENWIDTH nPenWidth = GetPenWidth();
```

4. If you are not supporting all of the widths, include the following.

```
if (!m_VariableWidth)
```

5. Set the pen to Thin if it is less than four pixels wide. Otherwise, set it to thick.

```
ChangePen ((nPenWidth <= MEDIUM)?THIN:THICK);
```

6. The complete COMMAND handler follows.

```
void CScribbleDoc::OnPenVariablewidth()
{
    m_VariableWidth = !m_VariableWidth;

    PENWIDTH nPenWidth = GetPenWidth();
    if (!m_VariableWidth)
        ChangePen ((nPenWidth <= MEDIUM)?THIN:THICK);
}
```

7. Move to the UPDATE_COMMAND_UI handler. You will load the menu string based on m_VariableWidth. Declare a **CString** to hold the menu string.

```
CString  MenuString;
```

8. Load the **CString** from the string table resource based on m_VariableWidth.

```
MenuString.LoadString(m_VariableWidth?  IDS_FEWERCHOICES:
                                        IDS_MORECHOICES);
```

9. Set the text of the menu to the string.

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

10. Save ScribDoc.Cpp. The complete UPDATE_COMMAND_UI handler follows.

```
void CScribbleDoc::OnUpdatePenVariablewidth(CCmdUI* pCmdUI)
{
    CString MenuString;
    MenuString.LoadString(m_VariableWidth?  IDS_FEWERCHOICES:
                                            IDS_MORECHOICES);
    pCmdUI->SetText(MenuString);
}
```

➢ **Add m_VariableWidth member to CSribbleDoc**

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

```
BOOL m_VariableWidth
```

2. Open ScribDoc.Cpp.
3. In InitDocument, initialize m_VariableWidth to TRUE.

```
m_VariableWidth = TRUE;
```

4. Save ScribDoc.Cpp.

➢ **Modify the UPDATE_COMMAND_UI handler for pixel widths**

1. In OnUpdatePenPixels, check to see whether the item is neither the Thick or Thin menu item.

```
if (nID != ID_PEN_2PIXEL && nID != ID_PEN_5PIXEL)
```

2. If the item is not, you enable or disable it depending on m_VariableWidth.

```
pCmdUI->Enable(m_VariableWidth);
```

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

```
void CScribbleDoc::OnUpdatePenpixels(CCmdUI* pCmdUI)
{
    INT nID = pCmdUI->m_nID;

    if (nID != ID_PEN_2PIXEL && nID != ID_PEN_5PIXEL)
        pCmdUI->Enable(m_VariableWidth);

    pCmdUI->SetRadio(GetPenWidth() == (nID - ID_PEN_1PIXEL + 1));
}
```

4. Build and run Scribble.

The completed code for this exercise is in \Labs\C05\Lab03\Ex02.

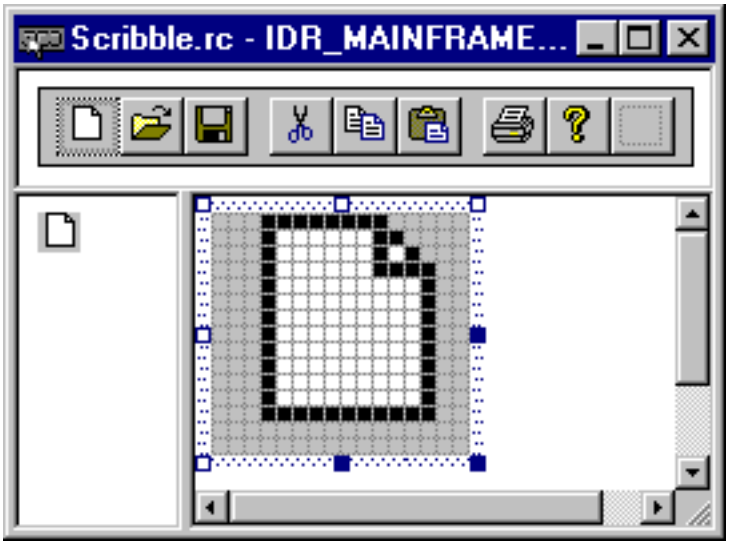
Exercise 3: Adding Toolbar Buttons

Continue with the files you 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\Lab03\Ex02.

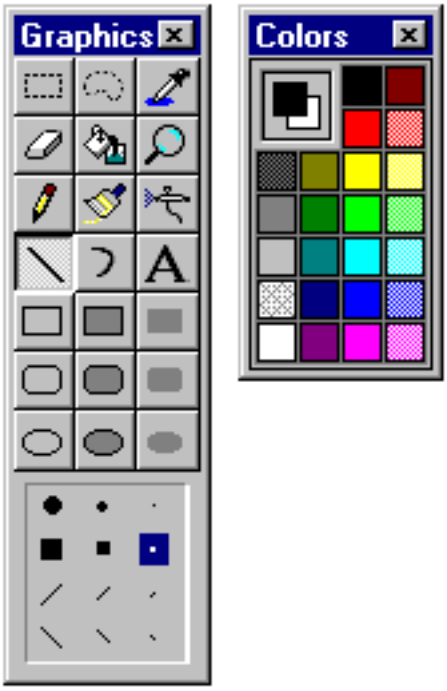
Your menu system is now complete; however, you have not yet implemented toolbar support. Toolbar support is straightforward. In this exercise, you will add two buttons to the toolbar and connect them to work with the menu.

➢ 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, or magnified view, of the editor window.

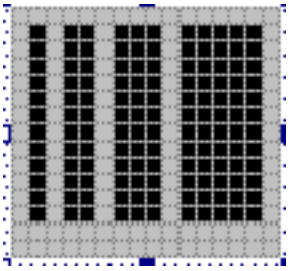


The graphics and color tools also open as part of the toolbar editor. If these graphics tools do not appear, choose Toolbars from the View menu and select Graphics and Colors in the dialog box. You can drag the graphics tools to either side of the screen and dock them to get a better view of the editor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

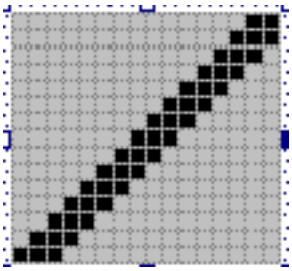
➢ **Delete and add toolbar buttons**

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 otherwise 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 the former location of the cut button. This new button gets 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. You will add three buttons to the IDR_MAINFRAME toolbar:

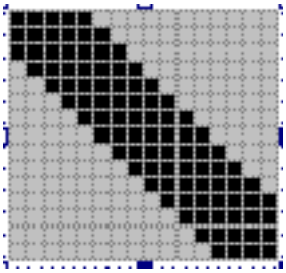
a. a toggle for More or Fewer choices:



b. a thin line button:



c. a thick line button:



Each of these buttons is drawn using the line tool.

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

5. Save Scribble.Rc.

➢ Associate toolbar buttons with command IDs

In the next step, you associate the new buttons 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 will bind the toggle button to ID_PEN_VARIABLEWIDTH, the thick line button to ID_PEN_5PIXEL, and the thin line button to ID_PEN_2PIXEL. You defined these IDs earlier for the Thick Line and Thin Line menu commands, so Visual C++ has already written a **#define** for the ID in Resource.H. Your only task at this point is to associate the ID with the button.

1. Double-click the toggle button to show the Toolbar Button Properties page. Visual C++ will assign 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 toggle button, choose ID_PEN_VARIABLEWIDTH.
2. Repeat with the thin line button. Set its ID to ID_PEN_2PIXEL.
3. Repeat with the thick line button. Set its ID to ID_PEN_5PIXEL.

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 appears next to the button.

➢ Add ToolTips

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

   In the Prompt: box, you will see the text "Toggle between a fewer and greater number of choices.” You entered this text for the toggle menu item on Scribble's Pen menu; it appears in the status line when the mouse passes over the menu command.
2. To add a ToolTip, at the end of the Prompt text, type a newline character (\n) plus the text that you want to display in the tip (there should be no space between the newline character and the tip text). If you want a ToolTip without a prompt string, simply start with the newline character.

Keep this text short. Type \nToggle Choices after the existing Prompt string.

3. Repeat this with the other two new buttons. Use \n2 pixels for the thin line button and \n5 pixels for the thick line.

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

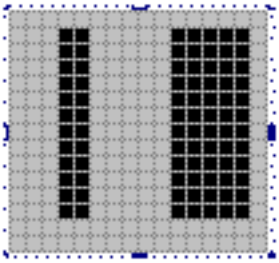
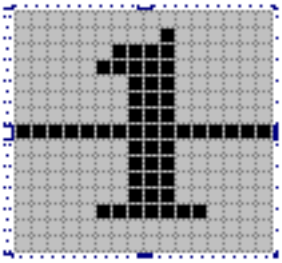
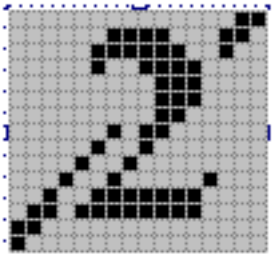
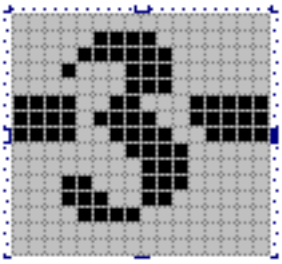
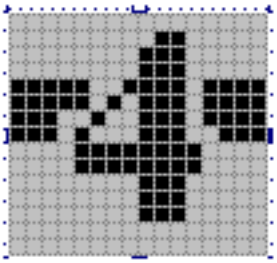
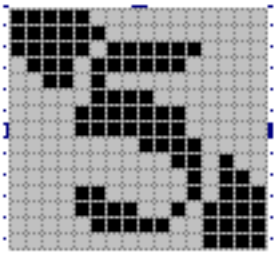
➢ Copy the toolbar resource

1. From the View menu, choose Resource Symbols. Select the IDR_MAINFRAME toolbar. Copy it to the clipboard by choosing the Copy command from the Edit menu.
2. Paste the toolbar by choosing Paste from the Edit menu. You will now have an IDR_MAINFRAME toolbar and an IDR_MAINFRAME1 toolbar. Show the Toolbar Properties dialog box by pressing selecting the resource, right-clicking and choosing Properties.
3. Change the ID for IDR_MAINFRAME to IDR_FEWERCHOICES. Change the ID for IDR_MAINFRAME1 to IDR_MORECHOICES.
4. Save Scribble.Rc.

➢ Add additional buttons, IDs and ToolTips

1. Open the IDR_MORECHOICES toolbar. You can create a toolbar with four more buttons. Upon completion, this toolbar will look like this:

2. Use the line tool and the character tool to add the buttons. Set the font to Times New Roman, 14-point bold. Set the IDs as shown in the illustration.

Bitmap	ID	ToolTip
	ID_PEN_VARIABLEWIDTH	Toggle Choices
	ID_PEN_1PIXEL	1 Pixel
	ID_PEN_2PIXEL	2 Pixels
	ID_PEN_3PIXEL	3 Pixels
	ID_PEN_4PIXEL	4 Pixels
	ID_PEN_5PIXEL	5 Pixels

3 Save Scribble.Rc.

➢ **Integrate the new toolbar into Scribble**

1. Open MainFrm.Cpp. In **CMainFrame::OnCreate**, set the startup toolbar to IDR_MORECHOICES.

```
if (!m_wndToolBar.Create(this) ||
    !m_wndToolBar.LoadToolBar(IDR_MORECHOICES))
```

2. Save MainFrm.Cpp.
3. You will need a reference to the main frame's toolbar in order to change the toolbar resource behind it. In the public attributes section of MainFrm.H, add a function to return a pointer to the toolbar.

```
CToolBar * GetToolBar() { return &m_wndToolBar; }
```

4. Save MainFrm.H.
5. Open ScribDoc.Cpp. You can reference the main frame window to change the toolbar; include MainFrm.H in ScribDoc.Cpp.

```
#include "mainfrm.h"
```

6. In **CScribbleDoc::OnPenVariableWidth**, get the main window frame.

```
CMainFrame * pMain;
pMain = (CMainFrame *)AfxGetMainWnd();
```

7. Get the toolbar and set the toolbar resource to the appropriate toolbar.

```
pMain->GetToolBar()->LoadToolBar(m_VariableWidth?
                                    IDR_MORECHOICES:
                                    IDR_FEWERCHOICES);
```

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

```
void CScribbleDoc::OnPenVariablewidth()
{
    m_VariableWidth = !m_VariableWidth;

    CMainFrame * pMain;
    pMain = (CMainFrame *)AfxGetMainWnd();
    pMain->GetToolBar()->LoadToolBar(m_VariableWidth?
                                        IDR_MORECHOICES:
                                        IDR_FEWERCHOICES);

    PENWIDTH nPenWidth = GetPenWidth();
    if (!m_VariableWidth)
        ChangePen ((nPenWidth <= MEDIUM)?THIN:THICK);
}
```

9. Build and run Scribble.

The completed code for this exercise is in \Labs\C05\Lab03\Ex03.