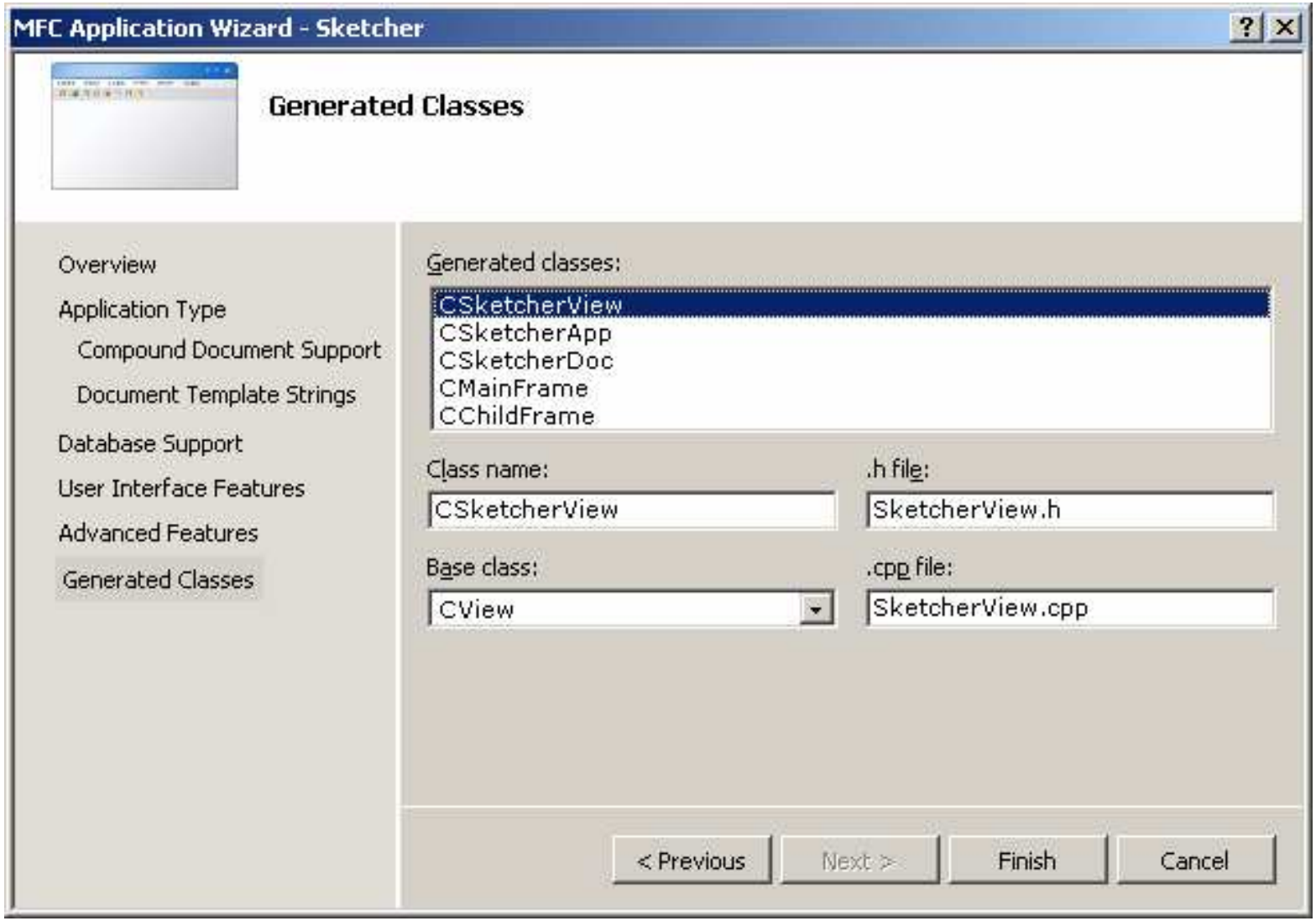
Chapter 14

Working with Menus and Toolbars

Creating an MDI Application

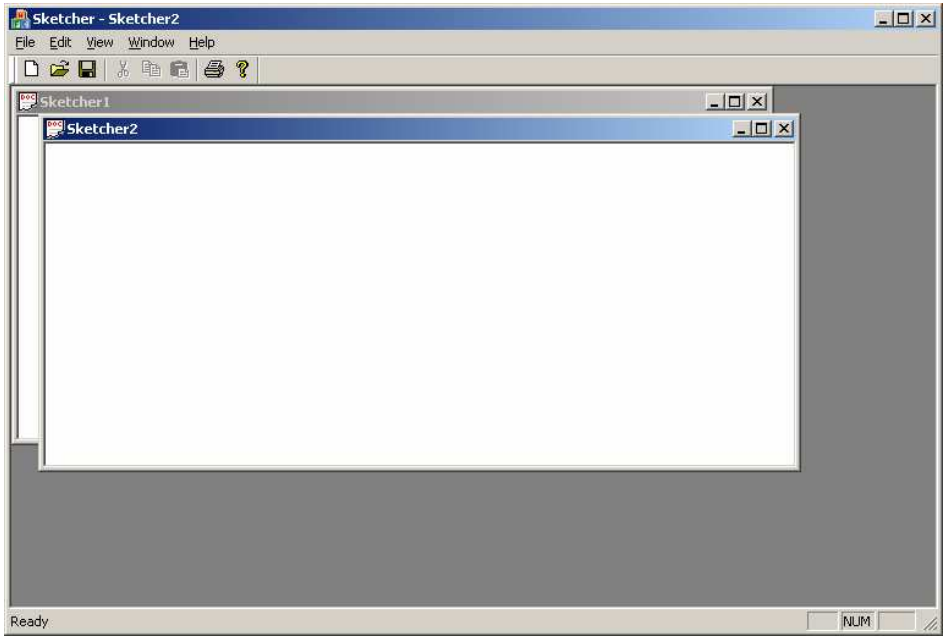
- Let us create a project **Sketcher**.
- The procedure is similar to that for creating an SDI application, with only three different things:
  - In `Application Type`, leave the default option MDI (rather than changing to the SDI option).
  - Under the `Document Template Strings`, specify the file extension as `ske`.
  - Under the `Generated Classes`, choose `CView` as the base class for the `CSketcherView` class.

Figure 13-12 (P.801)



Running the Program

- You can create new documents by selecting `File > New`.
  - The document can be saved with the extension `.ske`.
- However, you can't create any data in the application because we haven't added any code to do that.



Communicating with Windows

- Windows communicates with your program by sending messages to it.
  - `WM_PAINT`
  - `WM_LBUTTONDOWN`
- You have to provide functions to handle these messages that you're interested in - message handlers.
  - You ignore the other messages which you don't want to handle.

Message Map

- A message map for a class
  - A table of member functions that handle Windows messages.
- The start of a message map is indicated by a BEGIN_MESSAGE_MAP() macro, and the end is marked by an END_MESSAGE_MAP() macro.
- A message map is established for each of the main classes
  - CSketcherApp
  - CSketcherDoc
  - CSketcherView
  - CMainFrame
  - CChildFrame

CSketcherApp Class

- Look at the definition

```
class CSketcherApp : public CWinApp
{
public:
  CSketcherApp();


// Overrides
public:
  virtual BOOL InitInstance();

// Implementation
  afx_msg void OnAppAbout();
  DECLARE_MESSAGE_MAP()
};
```

- The word **afx_msg** is just to distinguish a message handler from other member functions.
- It will be converted to white space by the preprocessor.

Message Handler Definitions

- If a class definition includes the macro DECLARE_MESSAGE_MAP(), the class implementation must include the macros BEGIN_MESSAGE_MAP() and END_MESSAGE_MAP().
- Look at Sketcher.cpp:

```
BEGIN_MESSAGE_MAP(CSketcherApp, CWinApp)
   ON_COMMAND(ID_APP_ABOUT, &CSketcherApp::OnAppAbout)
   // Standard file based document commands
   ON_COMMAND(ID_FILE_NEW, &CWinApp::OnFileNew)
   ON_COMMAND(ID_FILE_OPEN, &CWinApp::OnFileOpen)
   // Standard print setup command
   ON_COMMAND(ID_FILE_PRINT_SETUP,
   &CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
```

current class name

base class

which menu or key is pressed

Message Categories

- Windows messages (Chapter 15)
  - Standard Windows messages such as
    - WM_PAINT – you need to redraw the client area
    - WM_LBUTTONUP – the left mouse button is released
- Control notification messages (Chapter 17)
  - Messages sent from controls (e.g. a list box) to the window which created it, or from a child window to a parent window.
- Command messages (this chapter)
  - Messages originating from the user interface elements, such as menu items and toolbar buttons.

Handling Messages in Your Program

- You can't put a handler for a message anywhere you like.
- Standard Windows messages and control notification messages are always handled by objects of classes derived from `CWnd`.
  - Frame window classes and view class.
  - On the contrary, application classes, document classes, and document template classes are not derived from `CWnd`, so they can't handle these messages.
  - See P.784
- Handling command messages is much more flexible. You can put handlers in the five classes mentioned above.

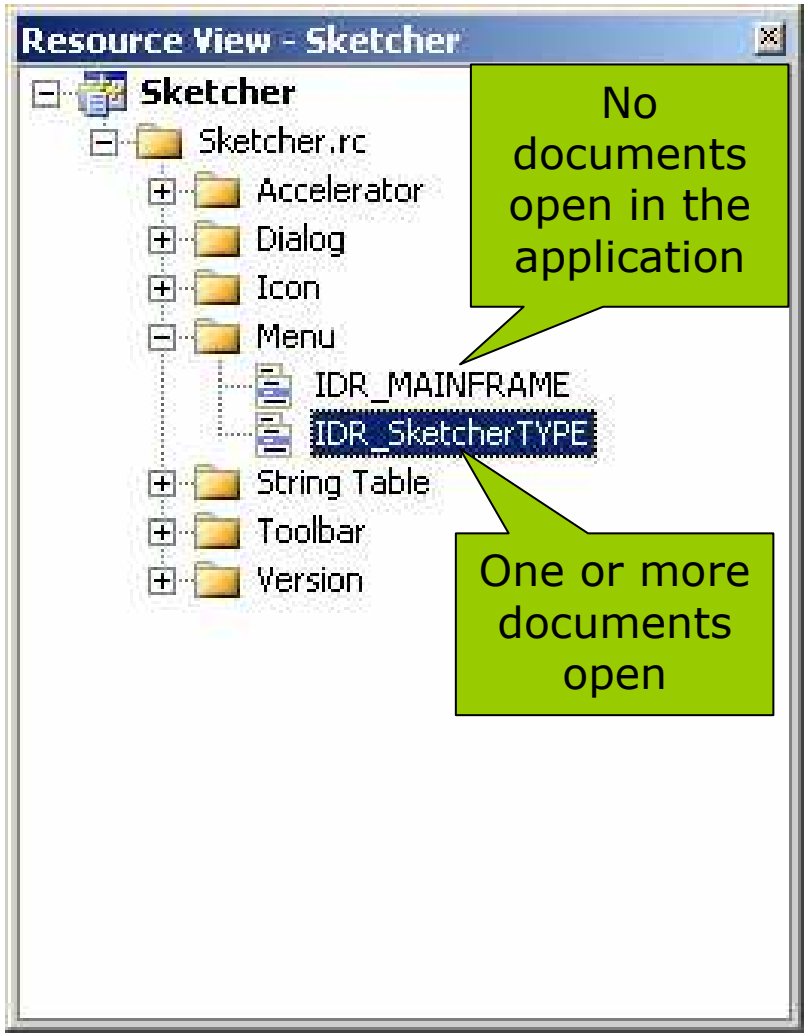
Notes for Message Map

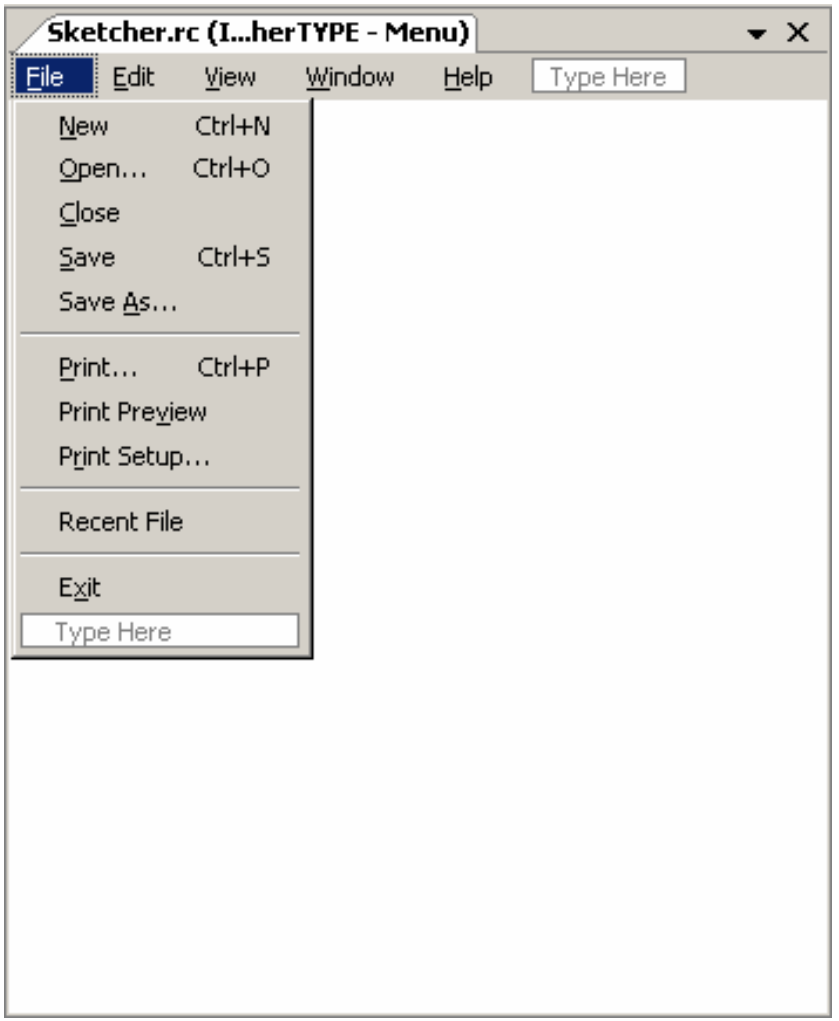
- You should not map a message to more than one message handler in a class.
  - The second message handler is never called.
- You may add a message handler by selecting the Messages button at the top of the Properties window.
  - In this way, you will not make the mistake of mapping a message to more than one message handler.
  - Another advantage is that you don't need to remember where to place handlers, as it only offers you the IDs allowed for the class.

Adding Menu Items to the Sketcher Program

- Menus are defined external to the program code in a resource file `Sketcher.rc`.
  - You could change to your menu items from English to French or Chinese without having to modify or recompile the program code.

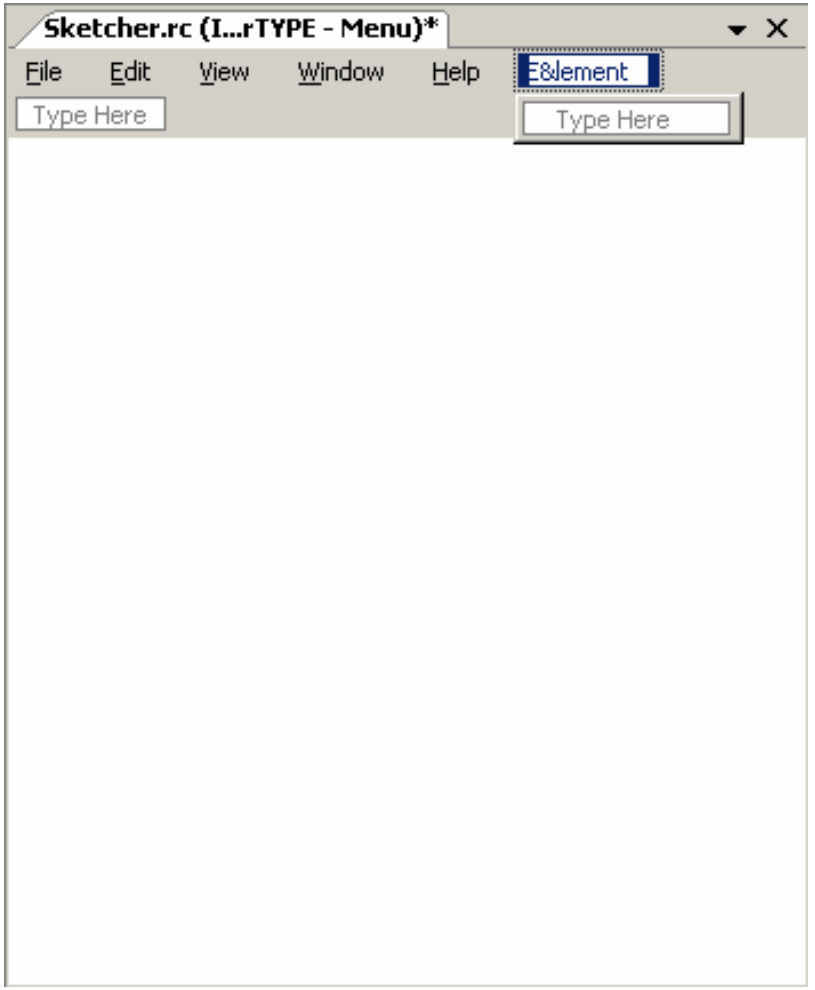
Resource View





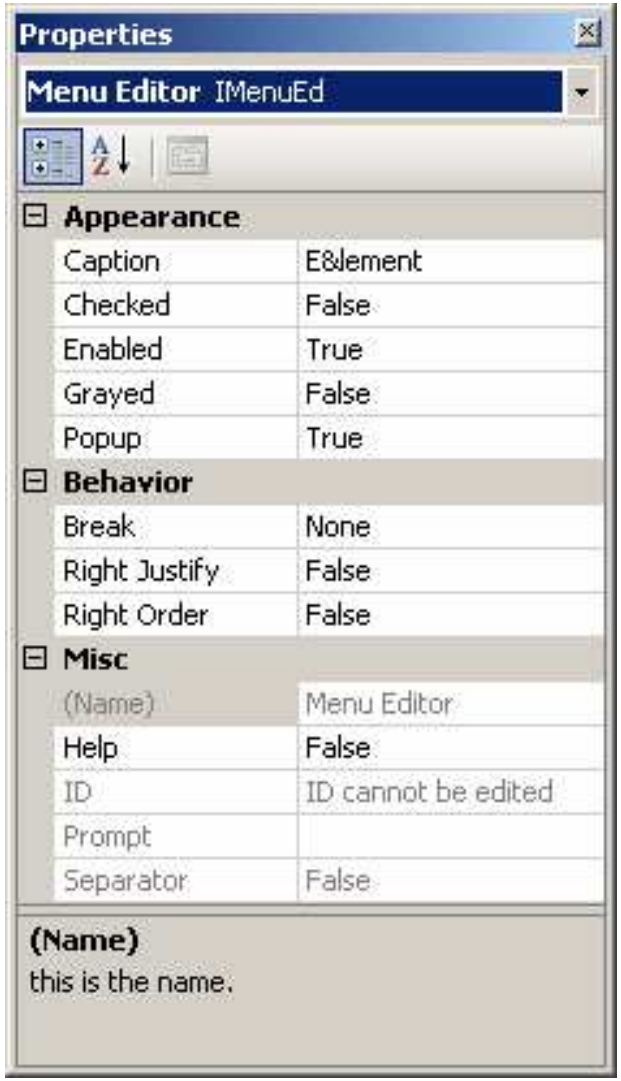
Adding a Menu Item to the Menu Bar

- Click the box with the text “Type Here” and type in your menu name.
  - Insert the ampersand (&) in front of a letter to make it a shortcut key.
  - Now , you may invoke the menu item by typing `Alt+L`.
- Drag it to a position between the `View` and `Window` menu items.



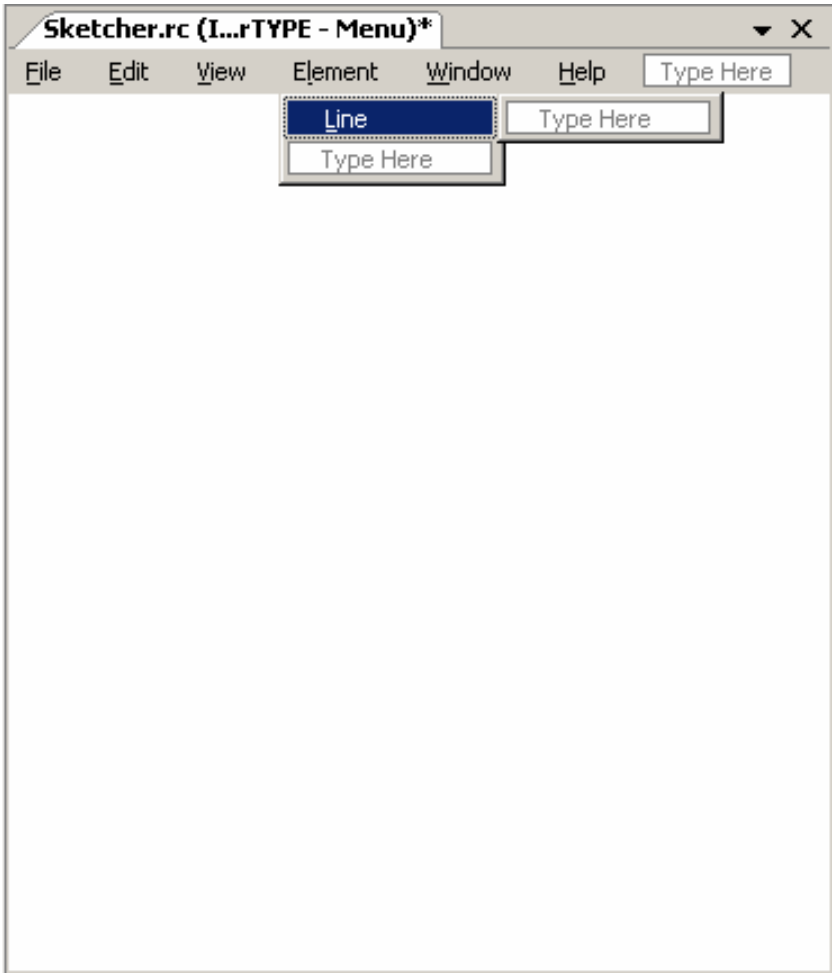
Properties of a Menu Item

- Double-click the menu item to display its properties.
- Properties are parameters that determine how the menu item will appear and behave.
  - They can be grouped by category or alphabetical order.
- No ID is necessary for a pop-up menu item
  - Selecting it just displays the menu beneath. No event for your code to handle.



Adding Items to the Element Menu

- Select the first item (currently labeled "Type Here") in the `Element` pop-up menu, then type `&Line` and press Enter.



Adding Items to the Element Menu

- This item is part of a pop-up menu, the Popup property is `False` by default.
- You could make it another pop-up menu with a further list of items
  - By setting the `Popup` property as `True`.
- You can enter a text string for the value of the `Prompt` property that appears in the status bar.
- The ID `ID_ELEMENT_LINE` is automatically specified.
  - You may change it if you like.

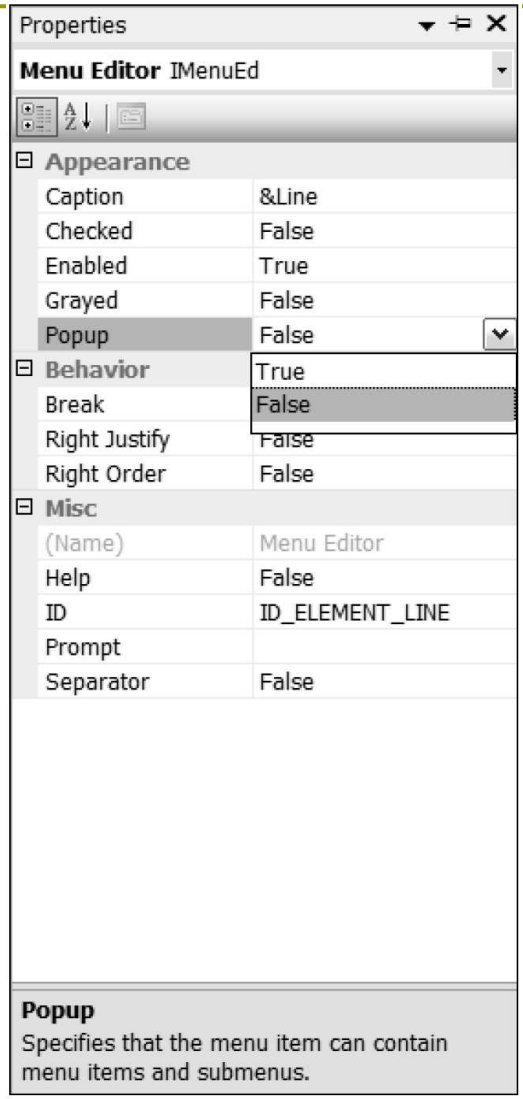


Figure 13-5

Completing the Menu

- Create the remaining items in `Element` menu:
  - `&Rectangle`
  - `&Circle`
  - `Cur&ve`
- The `Checked` property of `&Line` should be set to `True`, while the others have their `Checked` properties left as `False`.
- Create a `Color` menu on the menu bar, with items
  - Black
    - Set this one as `Checked` to indicate that it is the default color.
  - Red
  - Green
  - Blue
- Drag `Color` to the right of `Element`.

Figure 14-6 (P.815)

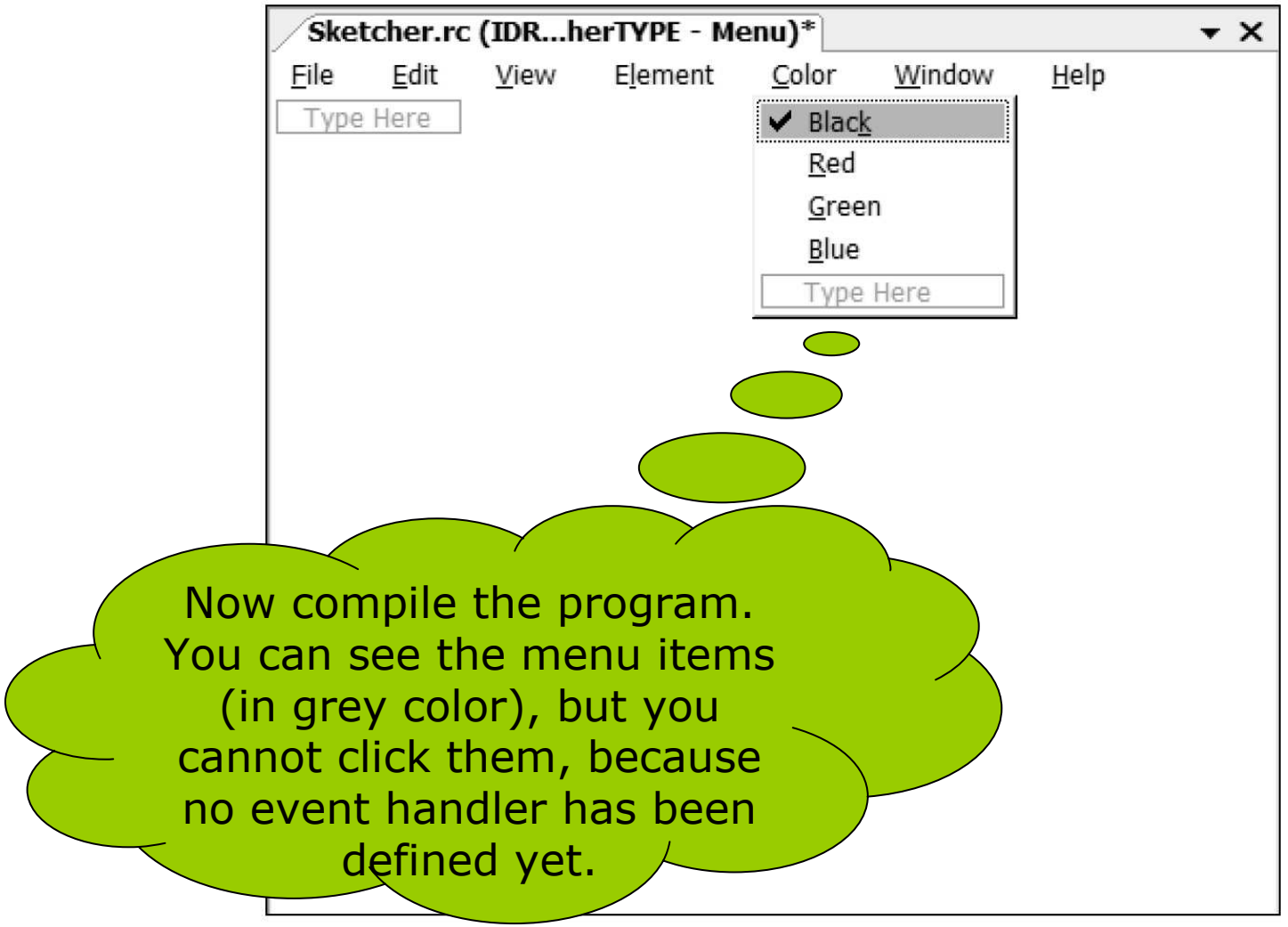
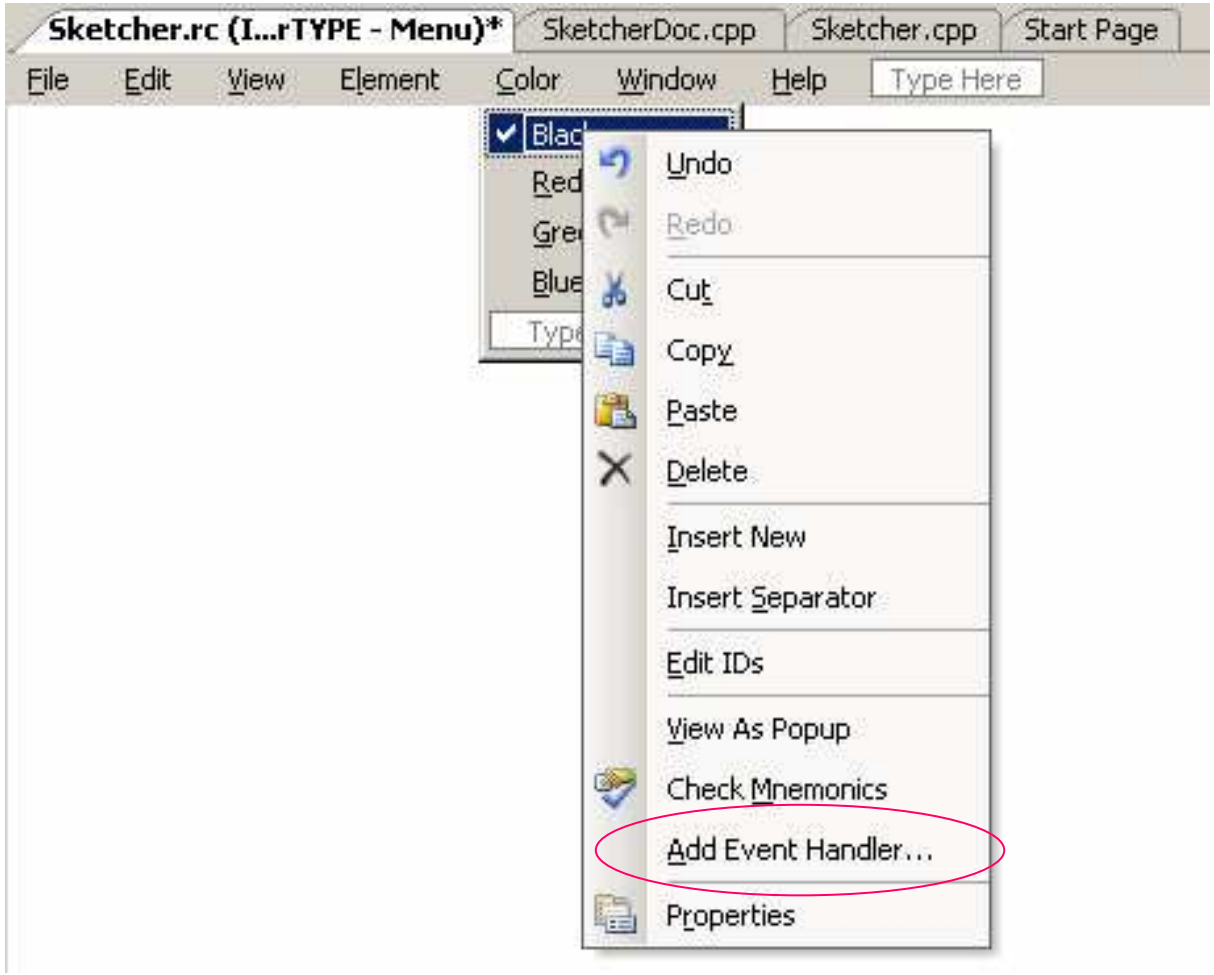


Figure 13-6

Adding Handlers for Menu Messages

- Right-click the item and select Add Event Handler



Event Handler Wizard

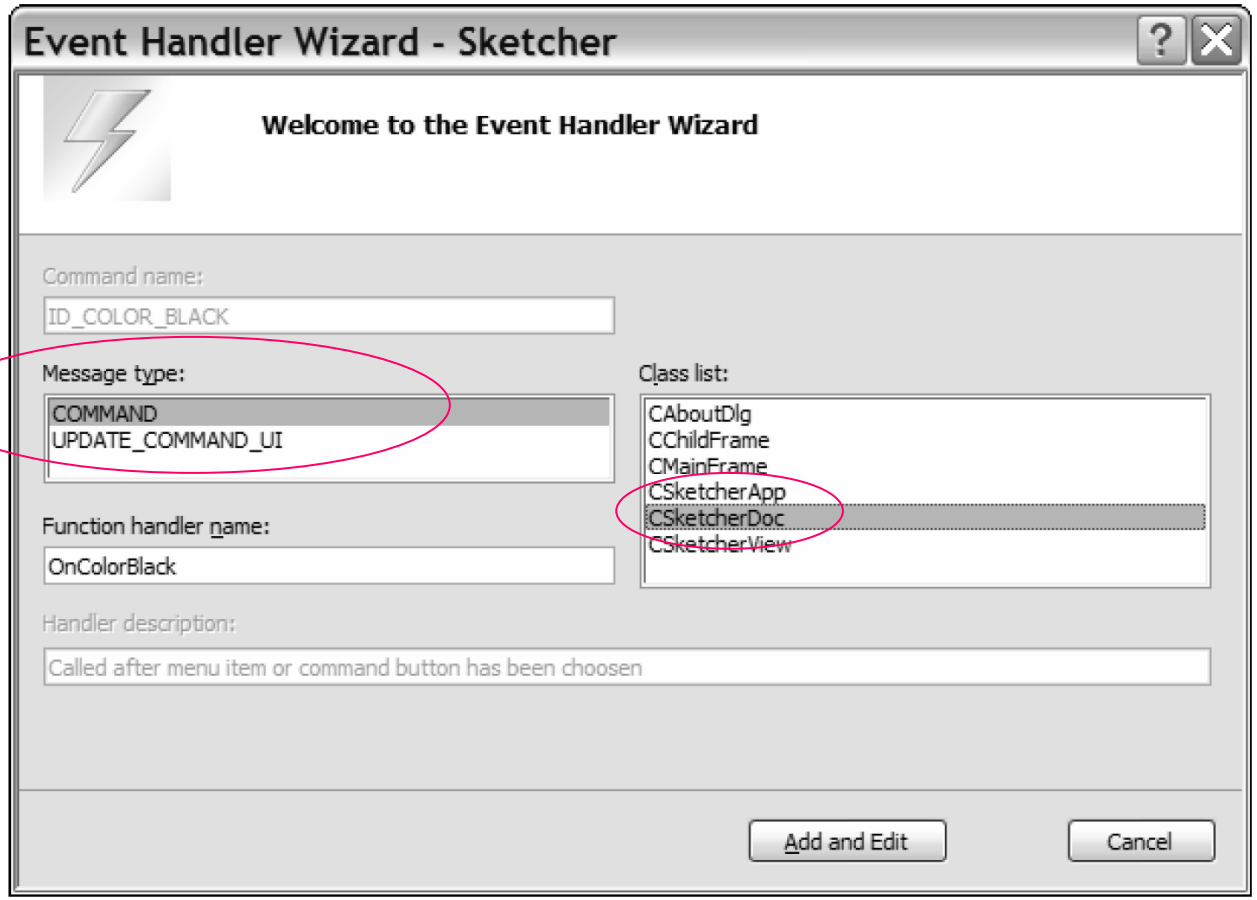


Figure 13-7

Message Type

- COMMAND
  - When you click one of the items in the menu, a COMMAND message for that menu item is sent.
- UPDATE_COMMAND_UI
  - Before a pop-up menu is displayed, an UPDATE_COMMAND_UI message is sent for each item in that menu.

Creating Menu Message Functions

- Highlight the `CSketchDoc` class name in the Event Handler Wizard dialog box.
- Click the `COMMAND` message type.
- Click the `Add and Edit` button.

- The wizard updated the CSketcherDoc class definition (CSketcherDoc.h) by adding
  - afx_msg void OnColorBlack(); // P.818
- It also updated the CSketcherDoc class implementation (CSketcherDoc.cpp) by adding
  - void CSketcherDoc::OnColorBlack()

Creating Menu Message Functions (2)

- Add COMMAND message handlers for the other Color menu IDs and all the Element menu IDs.
- Now the wizard should have added the following handlers to the CSketcherDoc class definition (SketcherDoc.h):

```
// Generated message map functions
protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnColorBlack();
    afx_msg void OnColorRed();
    afx_msg void OnColorGreen();
    afx_msg void OnColorBlue();
    afx_msg void OnElementRectangle();
    afx_msg void OnElementCircle();
    afx_msg void OnElementCurve();
    afx_msg void OnElementLine();
```

Creating Menu Message Functions (3)

- It also automatically updates the message map in your CSketcherDoc class implementation (SketcherDoc.cpp):

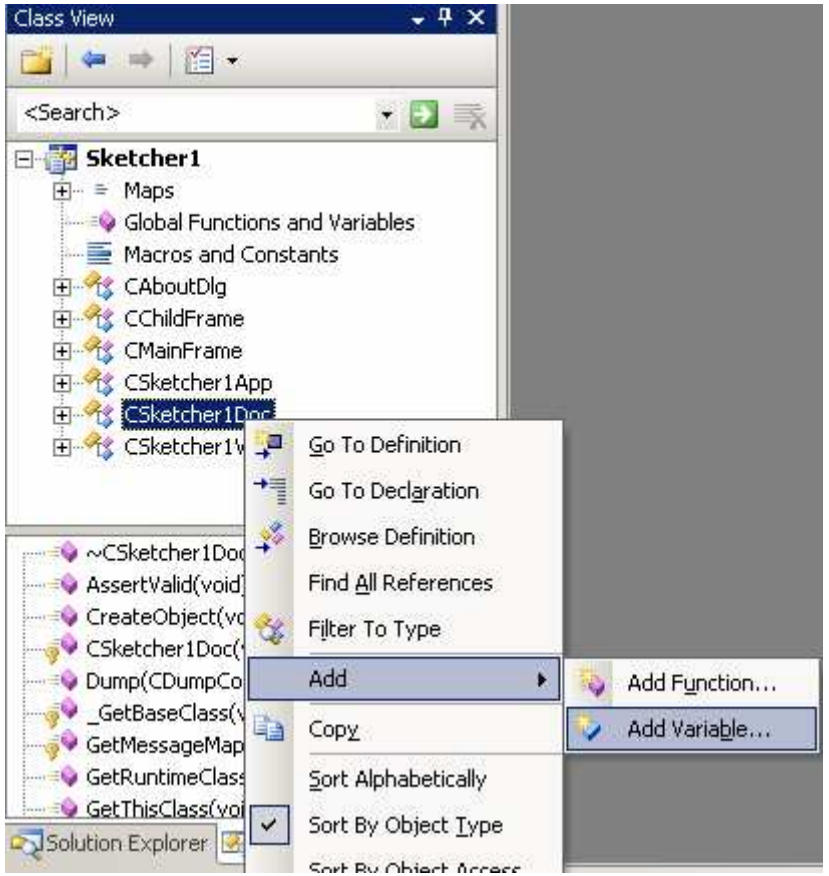
```
BEGIN_MESSAGE_MAP(CSketcherDoc, CDocument)
   ON_COMMAND(ID_COLOR_BLACK, OnColorBlack)
   ON_COMMAND(ID_COL     RED, OnColorRed)
   ON_COMMAND(ID_COL      U  OnColorGreen)
   ON_COMMAND
   ON_COMMAND      a COMMAND message for the menu item
   ON_COMMAND
   ON_COMMAN
   ON_
END_M
```

The member function `OnColorBlack()` is called to with the ID `ID_COLOR_BLACK`.

Now compile the program. You can click the menu items, but nothing will happen.

Coding Menu Message Functions

- We want to record the current element and color in the document.
  - Let us add some data members to the CSketcherDoc class.
  - In the Class View, right-click the CSketcherDoc class name, and then select Add > Add Variable



Adding a Member to Store Element Mode (P.820)

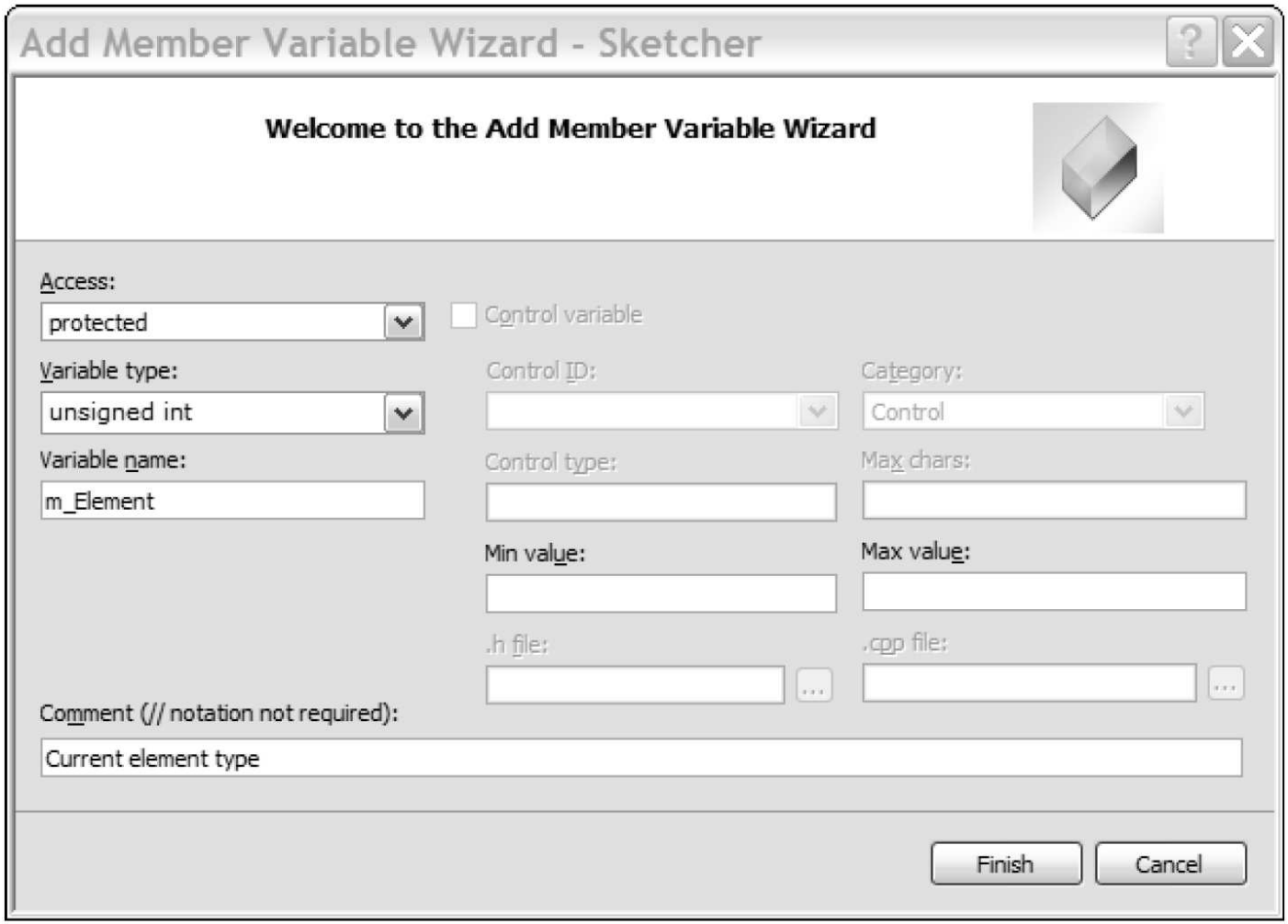


Figure 13-8

Adding a Member to Store Color

- You may also modify SketcherDoc.h manually to add a data member to store the color.

```
protected:
  // Current element type
  unsigned int m_Element;
  COLORREF m_Color;  // Current drawing color
```

- COLORREF is a type defined by the Windows API for representing a color as a 32-bit integer.

Define Some Constants

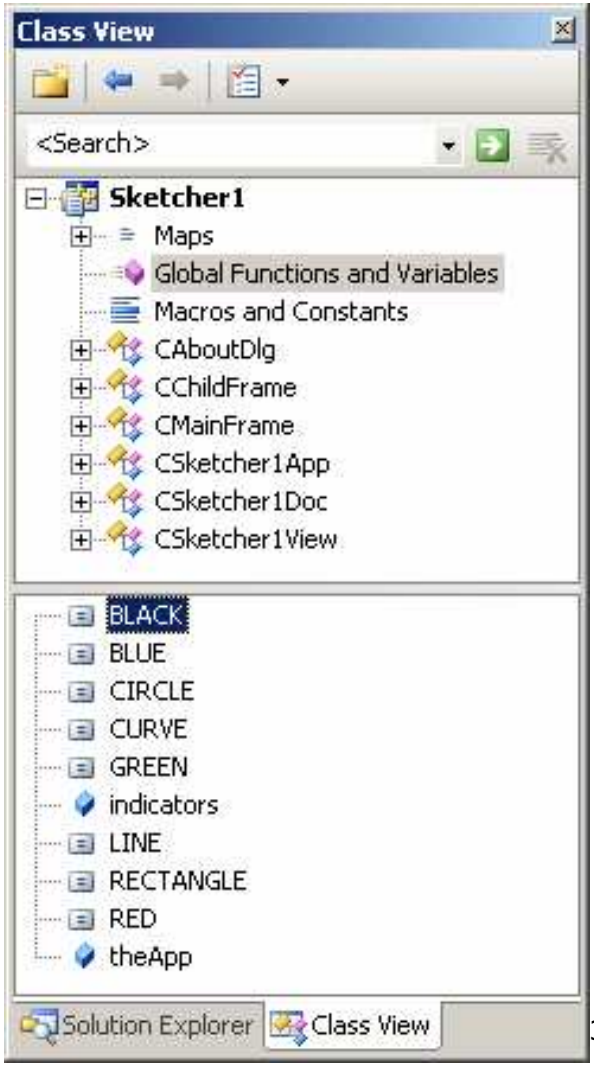
- You should prevent introducing “magic numbers” into your program.

```
#pragma once

    // Element type definitions
    // Each type value must be unique
    const unsigned int LINE = 101U;
    const unsigned int RECTANGLE = 102U;
    const unsigned int CIRCLE = 103U;
    const unsigned int CURVE = 104U;
    ////////////////////////////////////

    // Color values for drawing
    const COLORREF BLACK = RGB(0,0,0);
    const COLORREF RED = RGB(255,0,0);
    const COLORREF GREEN = RGB(0,255,0);
    const COLORREF BLUE = RGB(0,0,255);
    ////////////////////////////////////
```

  - RGB() is a standard macro defined in the Wingdi.h header file.

Define Some Constants (2)

- Save them in a new header file `OurConstants.h`
- At the beginning of Sketcher.h, add
  - `#include "OurConstants.h"`
- Now you can see these new constants in the `Class View` by expanding `Global Functions and Variables`.



Modifying the Class Constructor

- Make sure that the data members m_Element and m_Color are initialized appropriately.
  - Add the following code to SketcherDoc.cpp

```
CSketcherDoc::CSketcherDoc()
: m_Element(LINE), m_Color(BLACK)
{
   // TODO: add one-time construction code here
   // P.823
}
```

Add Code for the Handler Functions

- Click the first handler function OnColorBlack() from the Class View.

```
void CSketcherDoc::OnColorBlack()
{
  m_Color = BLACK; // Set the drawing color to black
}
```

- Do the same for other colors, and elements:

```
void CSketcherDoc::OnElementLine()
{
  m_Element = LINE;  // Set element type as a line
}
```

Running the Extended Example

- If you provided the `Prompt` message, it will be displayed in the status bar when the mouse cursor is over a menu item.
- `Alt+C` and `Alt+L` also works well.

- The check marks for the currently selected color and element are not updated, even if you choose another element or color!

Adding Message Handlers to Update the User Interface (P.824)

Figure 13-10

Message Handlers to Update the User Interface

- A skeleton function definition will be generated in SketcherDoc.cpp

```
void CSketcherDoc::OnUpdateElementLine(CCmdUI *pCmdUI)
{
  // TODO: Add your command update UI handler code here
}
```

- An entry is made in the message map
  - `ON_UPDATE_COMMAND_UI(ID_ELEMENT_LINE, OnUpdateElementLine)`

Coding a Command Update Handler

- Add the code for OnUpdateColorBlack() handler

```
void CSketcherDoc::OnUpdateColorBlack(CCmdUI *pCmdUI)
{// Set menu item Checked if the current color is black
   pCmdUI->SetCheck(m_Color==BLACK);
}
```

- CCmdUI is an MFC class type that is only used with update handlers, but it applies to toolbar buttons as well as menu items.
- It points to an object that originated the update message.

- The CCmdUI class has five member functions
  - ContinueRouting()
  - Enable()
  - SetCheck(int nCheck=1)
    - 1 - checked
    - 0 - unchecked
  - SetRadio()
  - SetText()

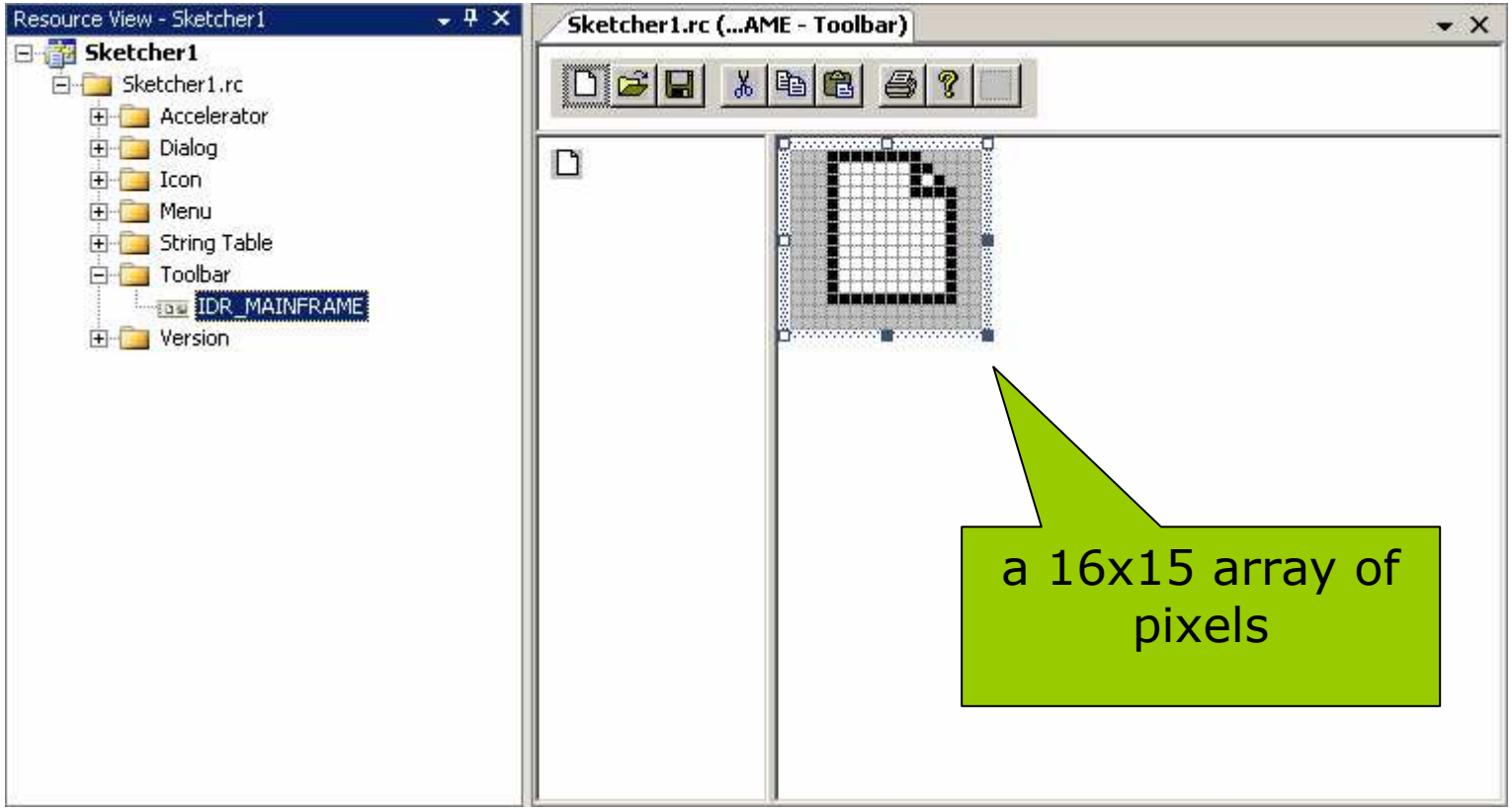
Exercising the Update Handlers

- Code the handlers for
  - OnUpdateColorBlack() P.826
  - OnUpdateColorRed()
  - OnUpdateColorGreen()
  - OnUpdateColorBlue()
- And also for
  - OnUpdateElementLine()
  - OnUpdateElementCurve()
  - OnUpdateElementCircle()
  - OnUpdateElementRectangle()

1. Run the program and see the check marks are changed correctly.
2. Remark the code inside OnUpdateColorBlue(). How will the program be different?
3. Add the following code into OnUpdateColorRed() and observe the difference:
   - if (m_Color==BLACK) pCmdUI->Enable(0);
   - else pCmdUI->Enable();

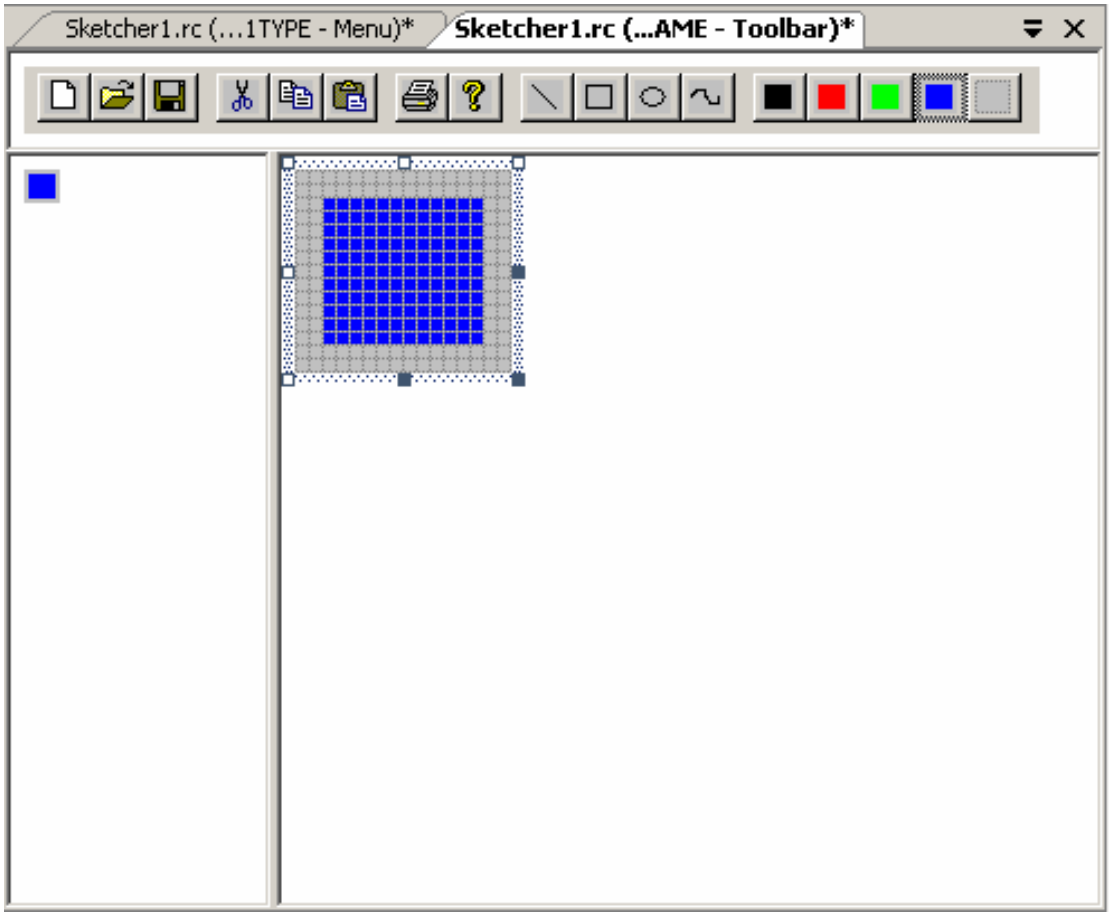
Adding Toolbar Buttons

- Select Resource View and extend the Toolbar resource.
- Double-click IDR_MAINFRAME, and you see an Editor window.



Adding Toolbar Buttons (2)

- To start a new block, drag the new button about half a button width to the right.



Editing Toolbar button Properties (P.829)

- Double-click your new button in the toolbar.
  - It shows a default ID
  - But you want to associate this button with the menu item `Element > Line`
  - Select `ID_ELEMENT_LINE` from the drop-down box.
    - You see the same prompt to appear in the status bar because the prompt is recorded along with the ID.
- Associate each button with the ID corresponding to the equivalent menu item.

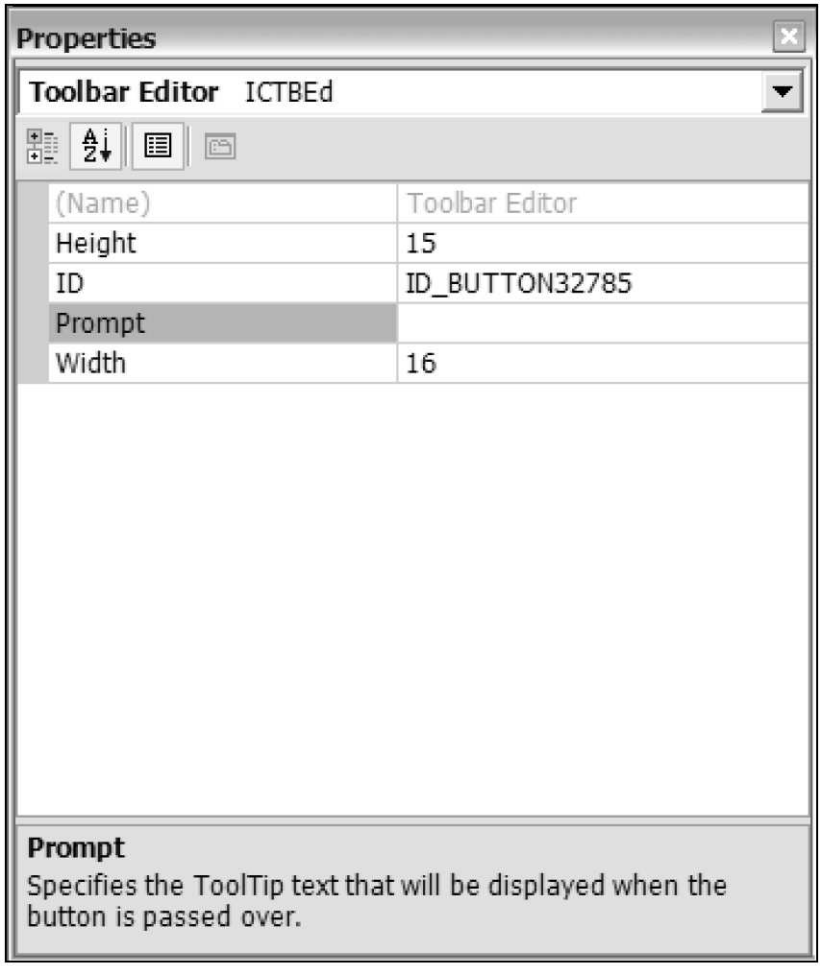
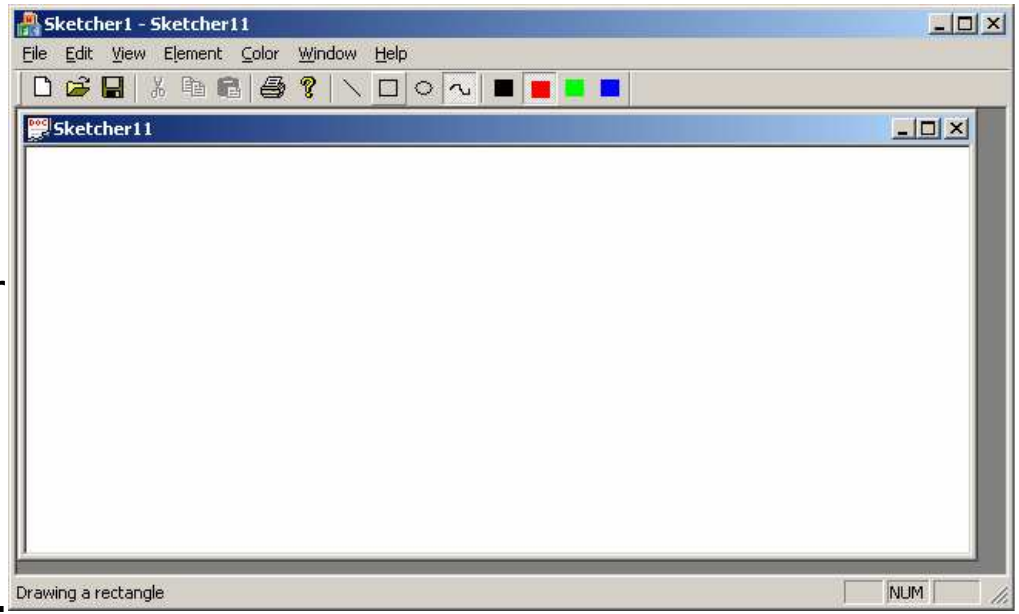


Figure 13-13

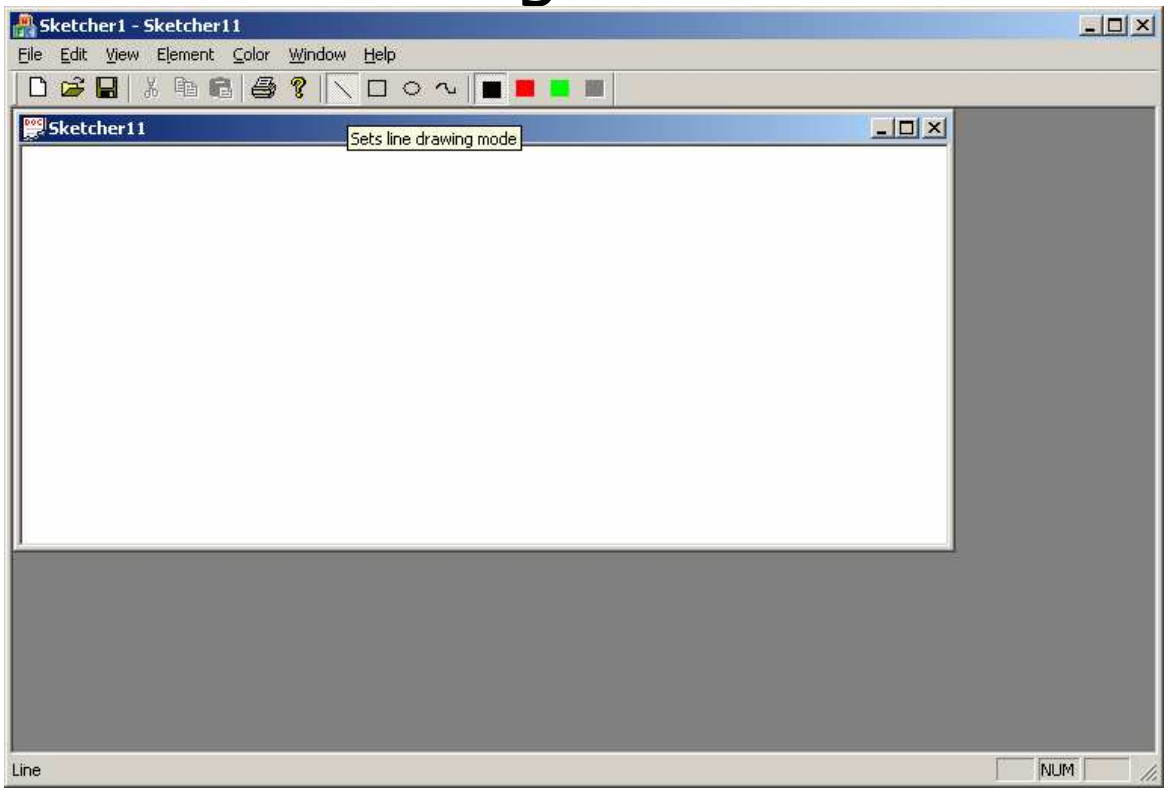
Exercising the Toolbar Buttons

- Build the application again and execute it.
- There are some amazing things:
  - The toolbar buttons will reflect the default settings for menu items.
  - If you move the cursor over one of the new buttons, the prompt for the button appears in the status bar.
  - If you choose a menu item, the toolbar is pressed as well.
- Close the document Sketcher1, you'll see that our toolbar buttons are automatically grayed and disabled.



Tooltips

- A tooltip is a small box that appears adjacent to the toolbar button when you let the cursor linger on the button.



Adding Tooltips

- Select the `Resource View` tab, and expand the resource list.
- Click the `String Table` folder and double-click the resource.
- This contains the IDs and prompt strings associated with menu items and toolbar buttons.
- Change the existing caption for the `ID_ELEMENT_LINE` ID from `Line` to `Line\nSets line drawing mode`.
  - The first part – the prompt that appears in the status bar
  - The second part – the tooltip text