

FIGURE 2.15 Toolbox group names.

2.3 Visual Basic Events



VideoNote
Event
procedures

When a Visual Basic program runs, the form and its controls appear on the screen. Normally, nothing happens until the user takes an action, such as clicking a control or pressing a key. We call such an action an **event**. The programmer writes code that reacts to an event by performing some functionality.

The three steps in creating a Visual Basic program are as follows:

1. Create the interface; that is, generate, position, and size the objects.
2. Set properties; that is, configure the appearance of the objects.
3. Write the code that executes when events occur.

Section 2.2 covered Steps 1 and 2; this section is devoted to Step 3. Code consists of statements that carry out tasks. Writing code in Visual Basic is assisted by an autocompletion system called **IntelliSense** that reduces the amount of memorization needed and helps prevent errors. In this section, we limit ourselves to statements that change properties of a control or the form while a program is running.

Properties of controls are changed in code with statements of the form

```
controlName.property = setting
```

where *controlName* is the name of the control, *property* is one of the properties of the control, and *setting* is a valid setting for that property. Such statements are called **assignment statements**. They assign values to properties. Here are three examples of assignment statements:

1. The statement

```
textBox.Text = "Hello"
```

displays the word *Hello* in the text box.

2. The statement

```
btnButton.Visible = True
```

makes the button visible.

3. The statement

```
txtBox.ForeColor = Color.Red
```

sets the color of the characters in the text box named txtBox to red.

Most events are associated with controls. The event “click on btnButton” is different from the event “click on lstBox”. These two events are specified btnButton.Click and lstBox.Click. The statements to be executed when an event occurs are written in a block of code called an **event procedure** or **event handler**. The first line of an event procedure (called the **header**) has the form

```
Private Sub objectName_event(ByVal sender As System.Object,  
    ByVal e As System.EventArgs) Handles objectName.event
```

Since we do not make any use of the lengthy text inside the parentheses in this book, for the sake of readability we replace it with an ellipsis. However, it will automatically appear in our programs each time Visual Basic creates the header for an event procedure. The structure of an event procedure is

```
Private Sub objectName_event(...) Handles objectName.event  
    statements  
End Sub
```

where the three dots (that is, the ellipsis) represent

```
ByVal sender As System.Object, ByVal e As System.EventArgs
```

Words such as “Private,” “ByVal,” “As,” “Sub,” “Handles,” and “End” have special meanings in Visual Basic and are referred to as **keywords** or **reserved words**. The Code Editor automatically capitalizes the first letter of a keyword and displays the word in blue. The word “Sub” in the first line signals the beginning of the procedure, and the first line identifies the object and the event occurring to that object. The last line signals the termination of the event procedure. The statements to be executed appear between these two lines. These statements are referred to as the **body** of the event procedure. (**Note:** The word “Private” indicates that the event procedure cannot be invoked by another form. This will not concern us until much later in the book. The expression following Handles identifies the object and the event happening to that object. The expression “objectName_event” is the default name of the procedure and can be changed if desired. In this book, we always use the default name. The word “Sub” is an abbreviation of *Subroutine*.) For instance, the event procedure

```
Private Sub btnButton_Click(...) Handles btnButton.Click  
    txtBox.ForeColor = Color.Red  
End Sub
```

changes the color of the words in the text box to red when the button is clicked. The clicking of the button is said to **raise** the event, and the event procedure is said to **handle** the event.

■ An Event Procedure Walkthrough

The form in Fig. 2.16, which contains two text boxes and a button, will be used to demonstrate what event procedures are and how they are created. Three event procedures will be used to alter the appearance of a phrase appearing in a text box. The event procedures are named txtFirst_TextChanged, btnRed_Click, and txtFirst_Leave.

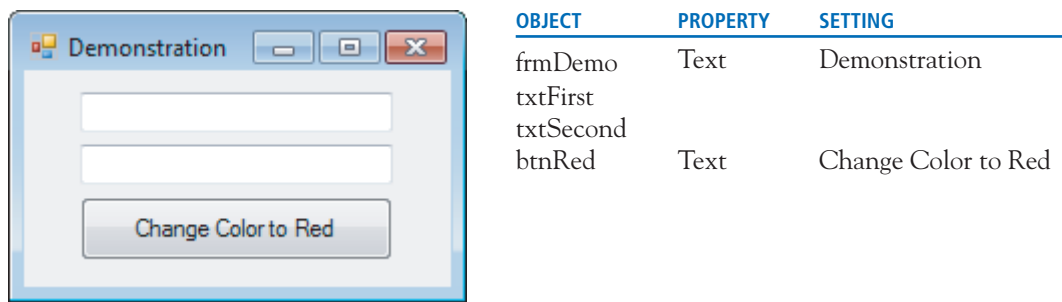


FIGURE 2.16 The interface for the event procedure walkthrough.

1. Create the interface in Fig. 2.16 in the Form Designer. The Name properties of the form, text boxes, and button should be set as shown in the Object column. The Text property of the form should be set to *Demonstration*, and the Text property of the button should be set to *Change Color to Red*. No properties need be set for the text boxes.
2. Click the right mouse button anywhere on the Form Designer, and click on *View Code*. The Form Designer IDE is replaced by the **Code Editor IDE** (also known as the *Code view* or the *Code window*). See Fig. 2.17.

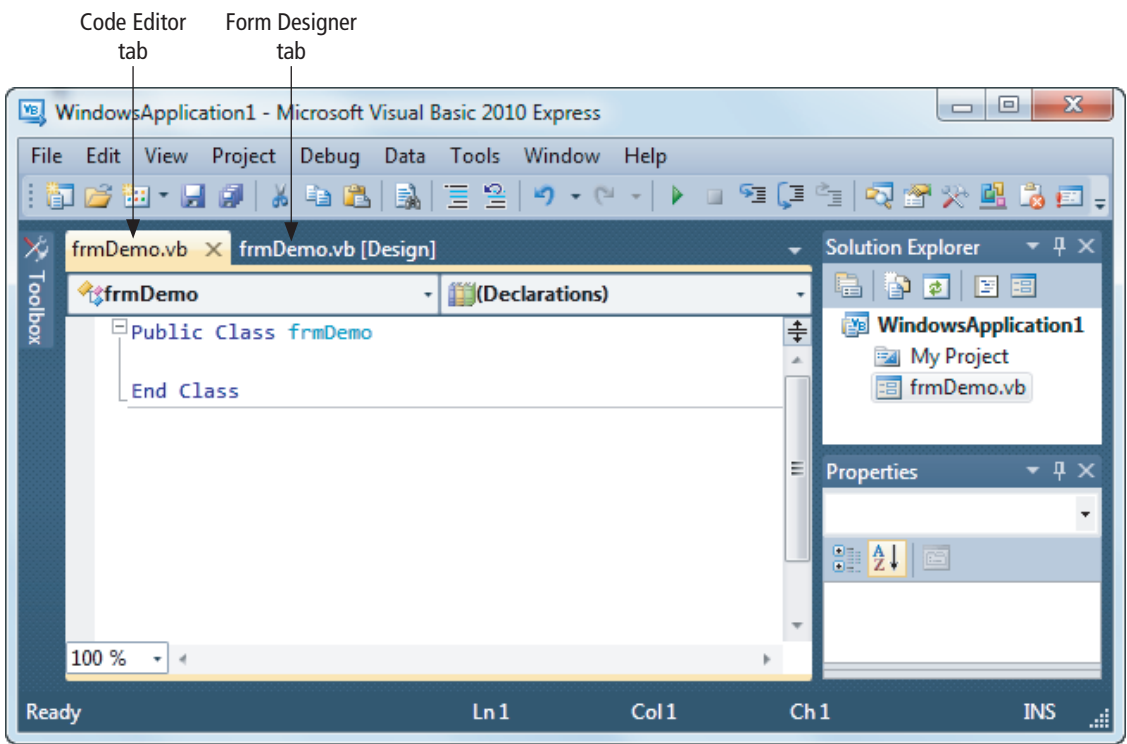


FIGURE 2.17 The Visual Basic IDE in Code Editor mode.

The tab labeled `frmDemo.vb` corresponds to the Code Editor. Click on the tab labeled `frmDemo.vb [Design]`, when you want to return to the Form Designer. We will place our program code between the two lines shown. Let’s refer to this region as the **program region**.

Figure 2.17 shows that the Code Editor IDE has a Toolbox, Solution Explorer, and Properties window that support Auto Hide. The Solution Explorer window for the Code Editor

functions exactly like the one for the Form Designer. The Code Editor's Toolbox has just one group, General, that is used to store code fragments which can be copied into a program when needed. The Code Editor's Properties window will not be used in this textbook.

3. Click on the tab labeled "frmDemo.vb [Design]" to return to the Form Designer. (You also can invoke the Form Designer by clicking *Designer* in the *View* menu, or by right-clicking the Code Editor and clicking *View Designer*.)
4. Double-click on the button. The Code Editor reappears, but now the following two lines of code have been added to the program region and the cursor is located on the blank line between them.

```
Private Sub btnRed_Click(...) Handles btnRed.Click
End Sub
```

The first line is the header for the event procedure named `btnRed_Click`. This procedure is invoked by the event `btnRed.Click`. That is, whenever the button is clicked, the code between the two lines just shown will be executed.

5. Type the line

```
txtFirst.ForeColor = Color.Red
```

at the cursor location.

This statement begins with the name of a control, `txtFirst`. Each time you type a letter of the name, IntelliSense drops down a list containing possible completions for the name.¹ As you continue typing, the list is shortened to match the letters that you have typed. Figure 2.18 shows the list after the letters `tx` have been typed. At this point, you have the following three options on how to continue:

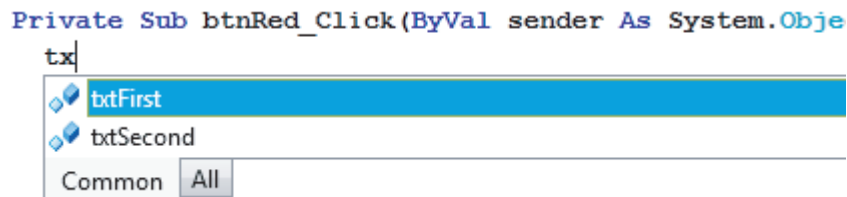


FIGURE 2.18 Drop-down list produced by IntelliSense.

- i. Double-click on `txtFirst` in the list.
- ii. Keep the cursor on `txtFirst`, and then press the Tab key or the Enter key.
- iii. Directly type in the remaining six letters of `txtFirst`.

After you type the dot (.) following `txtFirst`, IntelliSense drops down a list containing properties of text boxes. See Fig. 2.19(a). Each property is preceded by a properties icon (🔧). [The list also contains items called *methods*, which we will discuss later. Methods are preceded by a method icon (🔗).] At this point, you can scroll up the list and double-click on `ForeColor` to automatically enter that property. See Fig. 2.19(b). Or, you can keep typing. After you have typed "For", the list shortens to the single word `ForeColor`. At that point, you can press the Tab key or the Enter key, or keep typing to obtain the word `ForeColor`.

After you type in the equal sign, IntelliSense drops down the list of colors shown in Fig. 2.20. You have the option of scrolling to `Color.Red` and double-clicking on it, or typing `Color.Red` into the statement.

¹This feature of IntelliSense is referred to as **Complete Word**.

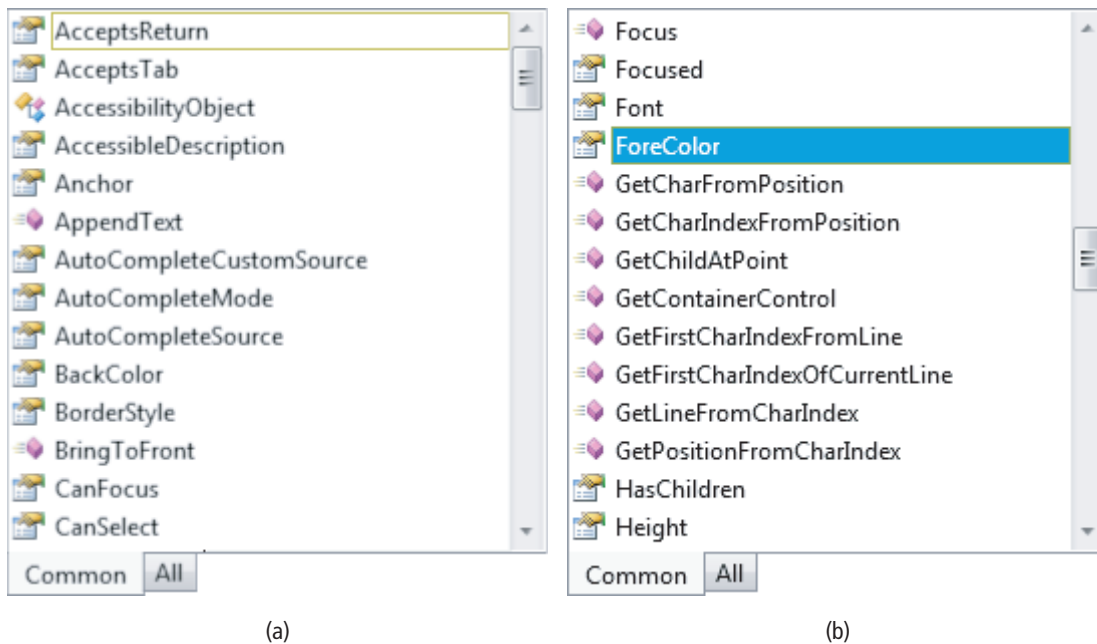


FIGURE 2.19 Drop-down list produced by IntelliSense.

```
Private Sub btnRed_Click(ByVal sender As System.Object)
    txtFirst.ForeColor =
```

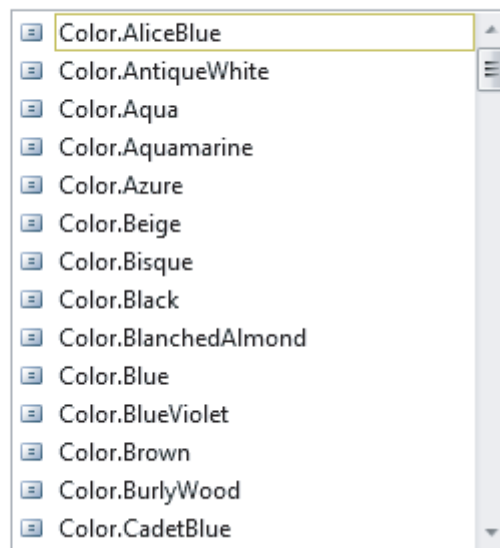


FIGURE 2.20 Drop-down list of colors produced by IntelliSense.

- Return to the Form Designer and double-click on the first text box. The Code Editor reappears, and the first and last lines of the event procedure `txtFirst_TextChanged` appear in the program region. This procedure is raised by the event `txtFirst.TextChanged`—that is, whenever there is a change in the text displayed in the text box `txtFirst`. Type the line that sets the `ForeColor` property of `txtFirst` to `Blue`. The event procedure will now appear as follows:

```
Private Sub txtFirst_TextChanged(...) Handles txtFirst.TextChanged
    txtFirst.ForeColor = Color.Blue
End Sub
```

7. Return to the Form Designer and select txtFirst.
8. Click on the *Events* button (⚡) in the toolbar at the top of the Properties window. The 63 events associated with text boxes are displayed, and the Description pane describes the currently selected event. (Don't be alarmed by the large number of events. Only a few events are used in this book.) Scroll to the *Leave* event. See Fig. 2.21.

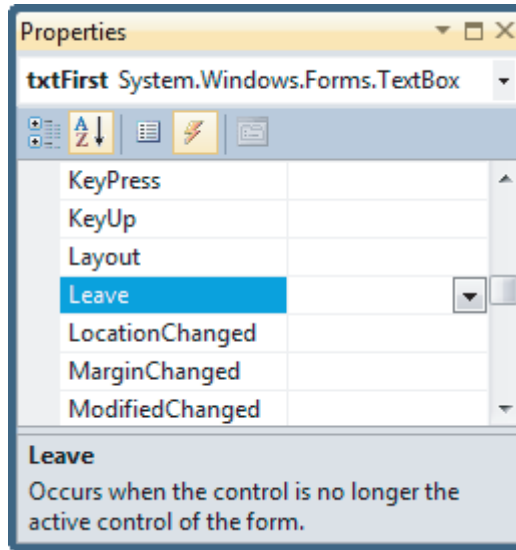


FIGURE 2.21 Events displayed in the Properties window.

9. Double-click on the *Leave* event. (The event txtFirst.Leave is raised when the focus is moved away from the text box.) The header and last line of the event procedure txtFirst_Leave will be displayed. In this procedure, type the line that sets the *ForeColor* property of txtFirst to *Black*. The Code Editor will now look as follows:

```
Public Class frmDemo
    Private Sub btnRed_Click(...) Handles btnRed.Click
        txtFirst.ForeColor = Color.Red
    End Sub

    Private Sub txtFirst_Leave(...) Handles txtFirst.Leave
        txtFirst.ForeColor = Color.Black
    End Sub

    Private Sub txtFirst_TextChanged(...) Handles txtFirst.TextChanged
        txtFirst.ForeColor = Color.Blue
    End Sub
End Class
```

10. Hover the cursor over the word "ForeColor". Visual Basic now displays information about the foreground color property. This illustrates another help feature of Visual Basic.
11. Run the program by pressing F5.
12. Type something into the first text box. In Fig. 2.22, the blue word "Hello" has been typed. (Recall that a text box has the focus whenever it is ready to accept typing—that is, whenever it contains a blinking cursor.)
13. Click on the second text box. The contents of the first text box will become black. When the second text box was clicked, the first text box lost the focus; that is, the event *Leave*

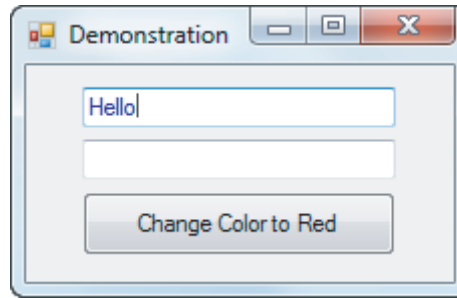



FIGURE 2.22 Text box containing input.

happened to txtFirst. Thus, the event procedure txtFirst_Leave was invoked, and the code inside the procedure was executed.

14. Click on the button. This invokes the event procedure btnRed_Click, which changes the color of the words in txtFirst to Red.
15. Click on the first text box, and type the word “Friend” after the word “Hello”. As soon as typing begins, the text in the text box is changed and the TextChanged event is raised. This event causes the color of the contents of the text box to become blue.
16. You can repeat Steps 12 through 15 as many times as you like. When you are finished, end the program by clicking on the *Stop Debugging* button on the Toolbar, clicking on the form’s Close button, or pressing Alt + F4.

Note: After viewing events in the Properties window, click on the *Properties* button () to the left of the *Events* button to return to displaying properties in the Properties window.

■ Properties and Event Procedures of the Form

You can assign properties to the form itself in code. However, a statement such as

```
frmDemo.Text = "Demonstration"
```

will not work. The form is referred to by the keyword *Me*. Therefore, the proper statement is

```
Me.Text = "Demonstration"
```

To display a list of all the events associated with frmDemo, select the form in the Form Designer and click on the Events button in the Properties window’s toolbar.

■ The Header of an Event Procedure

As mentioned earlier, in the header for an event procedure such as

```
Private Sub btnOne_Click(...) Handles btnOne.Click
```

btnOne_Click is the name of the event procedure, and btnOne.Click identifies the event that invokes the procedure. The name can be changed at will. For instance, the header can be changed to

```
Private Sub ButtonPushed(...) Handles btnOne.Click
```

Also, an event procedure can handle more than one event. For instance, if the previous line is changed to

```
Private Sub ButtonPushed(...) Handles btnOne.Click, btnTwo.Click
```

the event procedure will be invoked if either btnOne or btnTwo is clicked.

We have been using ellipses (...) as place holders for the phrase

```
ByVal sender As System.Object, ByVal e As System.EventArgs
```

In Chapter 5, we will gain a better understanding of this type of phrase. Essentially, the word “sender” carries a reference to the object that raised the event, and the letter “e” carries some additional information that the sending object wants to communicate. We will not make use of either “sender” or “e”.

■ Opening a Program

Beginning with the next chapter, each example contains a program. These programs can be downloaded from the Pearson website for this book. See the discussion on pages xiv–xv of the Preface for details. The process of loading a program stored on a disk into the Visual Basic environment is referred to as **opening** the program. Let’s open the downloaded program 7-2-3 from Chapter 7. That program allows you to enter a first name, and then displays U.S. Presidents having that first name.

1. From Visual Basic, click on *Open Project* in the *File* menu. (A dialog box will appear.)
2. Display the contents of the Ch07 subfolder downloaded from the website.
3. Double-click on 7-2-3.
4. Double-click on 7-2-3.sln.
5. If the Solution Explorer window is not visible, click on *Solution Explorer* in the *View/Other Windows* menu.
6. In the Solution Explorer window, click on frmPresident.vb. (Five buttons will appear at the top of the Solution Explorer. See Fig. 2.23.) At any time you can click on the *View Code* button to invoke the Code Editor, or you can click on the *View Designer* button to invoke the Form Designer. The *Show All Files* and *Refresh* buttons, which allow you to view all the files in a program’s folder and to update certain files, will be used extensively beginning with Chapter 7. The first button will not be used in this textbook.

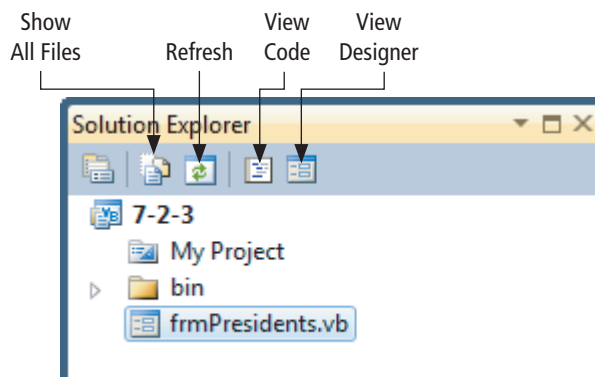


FIGURE 2.23 Solution Explorer window.

7. Press F5 to run the program.
8. Type in a name (such as James or William), and press the *Display Presidents* button. (See Fig. 2.24.) You can repeat this process as many times as desired.
9. End the program.

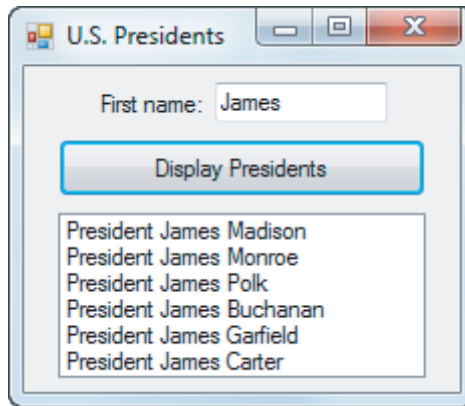


FIGURE 2.24 Output for program 7-2-3.

The program just executed uses a text file named `USPres.txt`. To view the text file, open the folder *bin*, open the subfolder *Debug*, and click on `USPres.txt`. (If the *bin* folder is not visible, click on the *Show All Files* button. If `USPres.txt` is not listed in the *Debug* subfolder, click the *Refresh* button and reopen the folders. After reading Chapter 7, you will understand why text files are placed in the *Debug* subfolder of the *bin* folder.) The first line of the file gives the name of the first president; the second line gives the name of the second president, and so on. To close the text file, click on the close button (X) on the `USPres.txt` tab.

Comments

1. The Visual Basic editor automatically indents the statements inside procedures. In this book, we indent by two spaces. To instruct your editor to indent by two spaces, select *Options* from the *Tools* menu, and uncheck the “Show all settings” box in the Options window that appears. Expand “Text Editor Basic” or “Text Editor”, click on “Editor”, enter 2 into the “Indent size:” box, and click on OK.
2. The event `controlName.Leave` is raised when the specified control loses the focus. Its counterpart is the event `controlName.Enter` which is raised when the specified control gets the focus. A related statement is

```
controlName.Focus()
```

which moves the focus to the specified control.

3. We have ended our programs by clicking the Stop Debugging button or pressing `Alt + F4`. A more elegant technique is to create a button, call it `btnQuit`, with caption `Quit` and the following event procedure:

```
Private Sub btnQuit_Click(...) Handles btnQuit.Click
    Me.Close()
End Sub
```

4. For statements of the form

```
object.Text = setting
```

the expression for *setting* must be surrounded by quotation marks. (For instance, `lblName.Text = “Name”`.) For properties where the proper setting is one of the words `True` or `False`, these words should *not* be surrounded by quotation marks.

5. Names of existing event procedures associated with an object are not automatically changed when you rename the object. You must change them yourself. However, the event

that invokes the procedure (and all other references to the control) will change automatically. For example, suppose an event procedure is

```
Private Sub btnOne_Click(...) Handles btnOne.Click
    btnOne.Text = "Press Me"
End Sub
```

and, in the Form Designer, you change the name of btnOne to btnTwo. Then, when you return to the Code Editor, the procedure will be

```
Private Sub btnOne_Click(...) Handles btnTwo.Click
    btnTwo.Text = "Press Me"
End Sub
```

6. The Code Editor has many features of a word processor. For instance, the operations cut, copy, paste, and find can be carried out with the sixth through ninth buttons on the Toolbar. These operations, and several others, also can be executed from the *Edit* menu.
7. The Code Editor can detect certain types of errors. For instance, if you type

```
txtFirst.Text = hello
```

and then move away from the line, the automatic syntax checker will underline the word “hello” with a blue squiggle to indicate that something is wrong. When the mouse cursor is hovered over the underlined expression, the editor will display a message explaining what is wrong. If you try to run the program without correcting the error, the dialog box in Figure 2.25 will appear.

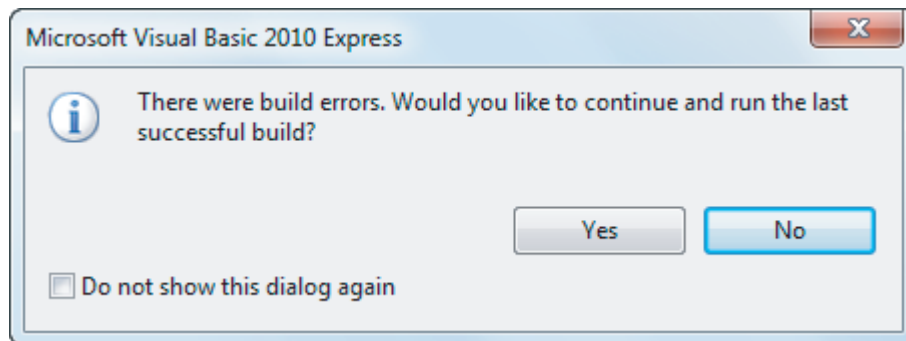


FIGURE 2.25 Error dialog box.

8. Each control has a favored event, called the **default event**, whose event procedure template can be generated from the Form Designer by double-clicking on the control. Table 2.2 shows some controls and their default events. The most common event appearing in this book is the Click event for a button. The TextChanged event for a text box was used in this section. The SelectedIndexChanged event for a list box is introduced in Section 4.4, and the Load event for a form is introduced in Section 7.1. The Click event for a label is never used in this book.

TABLE 2.2 Some default events.

Control	Default Event
form	Load
button	Click
label	Click
list box	SelectedIndexChanged
text box	TextChanged

9. Font properties, such as the name, style, and size, are usually specified at design time. The setting of the properties can be displayed in code with statements such as

```
1stBox.Items.Add(txtBox.Font.Name)
1stBox.Items.Add(txtBox.Font.Bold)
1stBox.Items.Add(txtBox.Font.Size)
```

However, a font's name, style, and size properties cannot be altered in code with statements of the form

```
txtBox.Font.Name = "Courier New"
txtBox.Font.Bold = True
txtBox.Font.Size = 16
```

10. When you make changes to a program, asterisks appear as superscripts on the tabs labeled "frmName.vb [design]" and "frmName.vb" to indicate that some part of the program has not been saved. The asterisks disappear when the program is saved or run.

Note: If the program has been saved to disk, all files for the program will be automatically updated on the disk whenever the program is saved or run.

11. You can easily change the size of the font used in the current program's Code Editor. Just hold down the Ctrl key and move the mouse's scroll wheel.
12. Notes on IntelliSense:

- (a) Whenever an item in an IntelliSense drop-down list is selected, a tooltip describing the item appears to the right of the item.
- (b) From the situation in Fig. 2.18, we can display txtFirst by double-clicking on the highlighted item, pressing the Tab key, or pressing the Enter key. Another option is to press the period key. In this case, both the name txtFirst and the dot following it will be displayed. **Note:** The period key option works only if the selected item is always followed by a dot in code.
- (c) IntelliSense drop-down lists have tabs labeled *Common* and *All*. When the *All* tab is selected, every possible continuation choice appears in the list. When the *Common* tab is selected, only the most frequently used continuation choices appear.
- (d) Occasionally, the IntelliSense drop-down list will cover some of your program. If you hold down the Ctrl key, the drop-down list will become transparent and allow you to see the covered-up code.

Practice Problems 2.3

1. Describe the event that invokes the following event procedure.

```
Private Sub btnCompute_Click() Handles txtBox.Leave
    txtBox.Text = "Hello world"
End Sub
```

2. Give a statement that will prevent the user from typing into txtBox.

EXERCISES 2.3

In Exercises 1 through 6, describe the contents of the text box after the button is clicked.

1.

```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
    txtBox.Text = "Hello"
End Sub
```

2.

```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
    txtBox.ForeColor = Color.Red
    txtBox.Text = "Hello"
End Sub
```
3.

```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
    txtBox.BackColor = Color.Orange
    txtBox.Text = "Hello"
End Sub
```
4.

```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
    txtBox.Text = "Goodbye"
    txtBox.Text = "Hello"
End Sub
```
5.

```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
    txtBox.Text = "Hello"
    txtBox.Visible = False
End Sub
```
6.

```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
    txtBox.BackColor = Color.Yellow
    txtBox.Text = "Hello"
End Sub
```

In Exercises 7 through 10, assume that the three objects on the form were created in the order txtFirst, txtSecond, and lblOne. Determine the output displayed in lblOne when the program is run and the Tab key is pressed. *Note:* Initially, txtFirst has the focus.

7.

```
Private Sub txtFirst_Leave(...) Handles txtFirst.Leave
    lblOne.ForeColor = Color.Green
    lblOne.Text = "Hello"
End Sub
```
8.

```
Private Sub txtFirst_Leave(...) Handles txtFirst.Leave
    lblOne.BackColor = Color.White
    lblOne.Text = "Hello"
End Sub
```
9.

```
Private Sub txtSecond_Enter(...) Handles txtSecond.Enter
    lblOne.BackColor = Color.Gold
    lblOne.Text = "Hello"
End Sub
```
10.

```
Private Sub txtSecond_Enter(...) Handles txtSecond.Enter
    lblOne.Visible = False
    lblOne.Text = "Hello"
End Sub
```

In Exercises 11 through 16, determine the errors.

11.

```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
    Form1.Text = "Hello"
End Sub
```

- ```
12. Private Sub btnOutput_Click(...) Handles btnOutput.Click
 txtBox.Text = Hello
End Sub

13. Private Sub btnOutput_Click(...) Handles btnOutput.Click
 txtFirst.ForeColor = Red
End Sub

14. Private Sub btnOutput_Click(...) Handles btnOutput.Click
 txtBox = "Hello"
End Sub

15. Private Sub btnOutput_Click(...) Handles btnOutput.Click
 txtBox.Font.Size = 20
End Sub

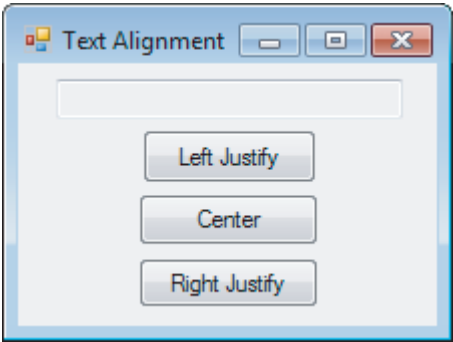
16. Private Sub btnOutput_Click(...) Handles btn1.Click, btn2.Click
 Me.Color = Color.Yellow
End Sub
```

In Exercises 17 through 28, write a line (or lines) of code to carry out the task.

17. Display "E.T. phone home." in lblTwo.
18. Display "Play it, Sam." in lblTwo.
19. Display "The stuff that dreams are made of." in red letters in txtBox.
20. Display "Life is like a box of chocolates." in txtBox with blue letters on a gold background.
21. Disable txtBox.
22. Change the words in the form's title bar to "Hello World."
23. Make lblTwo disappear.
24. Change the color of the letters in lblName to red.
25. Enable the disabled button btnOutcome.
26. Give the focus to btnCompute.
27. Give the focus to txtBoxTwo.
28. Change the background color of the form to White.
29. Describe the Enter event in your own words.
30. Describe the Leave event in your own words.
31. The label control has an event called DoubleClick that is raised by double-clicking the left mouse button. Write a simple program to test this event. Determine whether you can raise the DoubleClick event without also raising the Click event.
32. Write a simple program to demonstrate that a button's Click event is raised when you press the Enter key while the button has the focus.

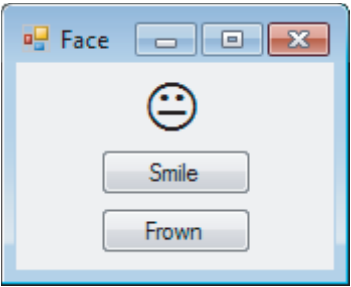
In Exercises 33 through 38, the interface and initial properties are specified. Write a program to carry out the stated task.

33. When one of the three buttons is pressed, the words on the button are displayed in the text box with the stated alignment. **Note:** Rely on IntelliSense to provide you with the proper settings for the TextAlign property.



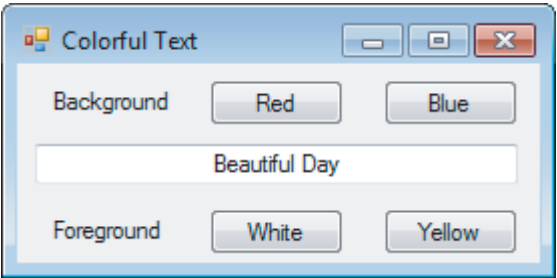
| OBJECT    | PROPERTY | SETTING        |
|-----------|----------|----------------|
| frmAlign  | Text     | Text Alignment |
| txtBox    | ReadOnly | True           |
| btnLeft   | Text     | Left Justify   |
| btnCenter | Text     | Center         |
| btnRight  | Text     | Right Justify  |

34. When one of the buttons is pressed, the face changes to a smiling face (Wingdings character “J”) or a frowning face (Wingdings character “L”).



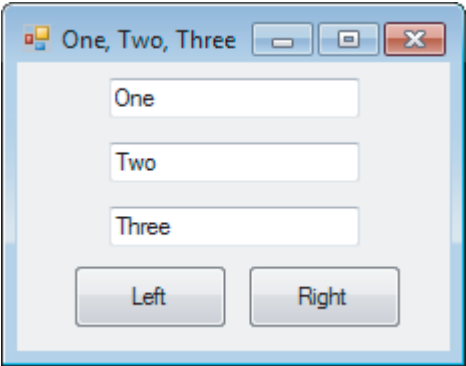
| OBJECT   | PROPERTY  | SETTING   |
|----------|-----------|-----------|
| frmFace  | Text      | Face      |
| lblFace  | Font Name | Wingdings |
|          | Font Size | 24        |
|          | Text      | K         |
| btnSmile | Text      | Smile     |
| btnFrown | Text      | Frown     |

35. Pressing the buttons alters the background and foreground colors in the text box.



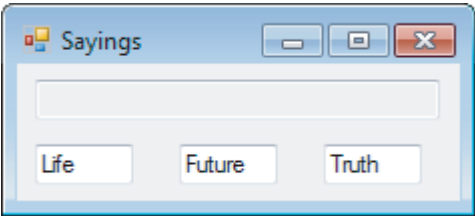
| OBJECT    | PROPERTY  | SETTING       |
|-----------|-----------|---------------|
| frmColors | Text      | Colorful Text |
| lblBack   | Text      | Background    |
| btnRed    | Text      | Red           |
| btnBlue   | Text      | Blue          |
| txtBox    | Text      | Beautiful Day |
|           | TextAlign | Center        |
| lblFore   | Text      | Foreground    |
| btnWhite  | Text      | White         |
| btnYellow | Text      | Yellow        |

36. When one of the three text boxes receives the focus, its text becomes red. When it loses the focus, the text returns to black. The buttons set the alignment in the text boxes to Left or Right. **Note:** Rely on IntelliSense to provide you with the proper settings for the TextAlign property.



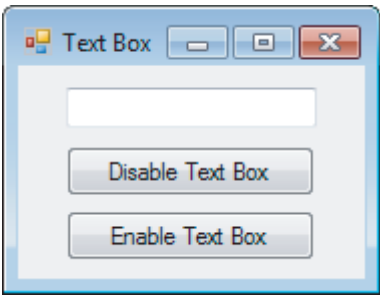
| OBJECT   | PROPERTY | SETTING         |
|----------|----------|-----------------|
| frm123   | Text     | One, Two, Three |
| txtOne   | Text     | One             |
| txtTwo   | Text     | Two             |
| txtThree | Text     | Three           |
| btnLeft  | Text     | Left            |
| btnRight | Text     | Right           |

37. When the user moves the focus to one of the three small text boxes at the bottom of the form, an appropriate saying is displayed in the large text box. Use the sayings “I like life, it’s something to do.”; “The future isn’t what it used to be.”; and “Tell the truth and run.”



| OBJECT    | PROPERTY | SETTING |
|-----------|----------|---------|
| frmQuote  | Text     | Sayings |
| txtQuote  | ReadOnly | True    |
| txtLife   | Text     | Life    |
| txtFuture | Text     | Future  |
| txtTruth  | Text     | Truth   |

38. The user can disable or enable the text box by clicking on the appropriate button. After the user clicks the Enable button, the text box should receive the focus.



| OBJECT     | PROPERTY | SETTING          |
|------------|----------|------------------|
| frmTextBox | Text     | Text Box         |
| txtBox     |          |                  |
| btnDisable | Text     | Disable Text Box |
| btnEnable  | Text     | Enable Text Box  |

In Exercises 39 through 44, write a program to carry out the task.

39. The form contains four square buttons arranged in a rectangular array. Each button has the caption “Push Me”. When the user clicks on a button, the button disappears and the other three become or remain visible. See Fig. 2.26.

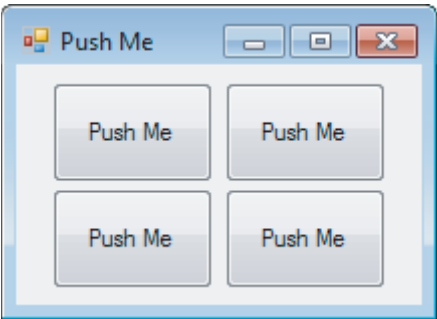


FIGURE 2.26 Form for Exercise 39.

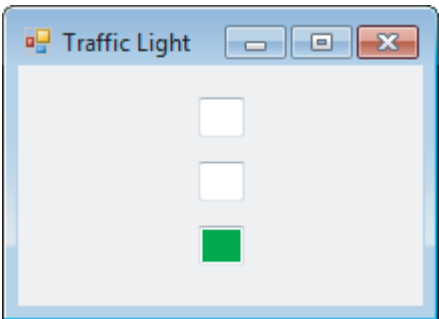


FIGURE 2.27 Form for Exercise 40.

40. Simulate a traffic light with three small square text boxes placed vertically on a form. See Fig. 2.27. Initially, the bottom text box is solid green and the other text boxes are dark gray. When the Tab key is pressed, the middle text box turns yellow and the bottom text box turns dark gray. The next time Tab is pressed, the top text box turns red and the middle text box turns dark gray. Subsequent pressing of the Tab key cycles through the three colors. **Hint:** First place the bottom text box on the form, then the middle text box, and finally the top text box.
41. Use the same form and properties as in Exercise 34, with the captions for the buttons replaced with Vanish and Reappear. Clicking a button should produce the stated result.
42. A form contains two text boxes and one large label between them with no preset caption. When the first text box receives the focus, the label reads “Enter your full name.” When the second text box receives the focus, the label reads “Enter your phone number, including area code.”



43. The form contains a single read-only text box and two buttons. When the user clicks on one of the buttons, the sentence “You just clicked on a button.” is displayed in the text box. The program should consist of a single event procedure.
44. The form contains two text boxes into which the user types information. When the user clicks on one of the text boxes, it becomes blank and its contents are displayed in the other text box. **Note:** A text box can be cleared with the statement `textBox.Clear()` or the statement `textBox.Text = ""`.

#### Solutions to Practice Problem 2.3

1. The event is raised when `textBox` loses the focus since `textBox.Leave` is the event following the keyword `Handles`. The name of the event procedure, `btnCompute_Click`, can be anything; it plays no role in determining the action that raises the event.
2. Three possibilities are

```
textBox.Enabled = False
textBox.ReadOnly = True
textBox.Visible = False
```

## CHAPTER 2 SUMMARY

1. The Visual Basic Form Designer displays a *form* that can hold a collection of *controls* for which various properties can be set. Some examples of controls are text boxes, labels, buttons, and list boxes. Some useful properties are `Text` (sets the text displayed in a control), `Name` (used to give a meaningful name to a control), `Font.Name` (selects the name of the font used), `Font.Size` (sets the size of the text displayed), `Font.Bold` (displays boldface text), `Font.Italic` (displays italics text), `BackColor` (sets the background color), `ForeColor` (sets the color of the text), `ReadOnly` (determines whether text can be typed into a text box when the program is running), `TextAlign` (sets the type of alignment for the text in a control), `Enabled` (determines whether a control can respond to user interaction), and `Visible` (determines whether an object can be seen or is hidden).
2. An *event procedure* is invoked when something happens to a specified object. Some events are `object.Click` (*object* is clicked), `object.TextChanged` (a change occurred in the value of the object's `Text` property), `object.Leave` (*object* loses the focus), and `object.Enter` (*object* receives the focus). **Note:** The statement `object.Focus()` moves the focus to the specified object.
3. *IntelliSense* provides a host of features that help you write code.
4. *Tab order*, the order in which the user moves the focus from one control to another by pressing the `Tab` key while the program is running, can be set from the Properties window.