2

Visual Basic, Controls, and Events

2.1 An Introduction to Visual Basic 2015

Visual Basic 2015 is the latest generation of Visual Basic, a language used by many software developers. Visual Basic was designed to make user-friendly programs easier to develop. Prior to the creation of Visual Basic, developing a friendly user interface usually required a programmer to use a language such as C or C++, often requiring hundreds of lines of code just to get a window to appear on the screen. Now the same program can be created in much less time with fewer instructions.

Why Windows and Why Visual Basic?

What people call **graphical user interfaces**, or GUIs, have revolutionized the software industry. Instead of the confusing textual prompts that earlier users once saw, today's users are presented with such devices as icons, buttons, and drop-down lists that respond to mouse clicks. Accompanying the revolution in how programs look was a revolution in how they feel. Consider a program that requests information for a database. Figure 2.1 shows how a program written before the advent of GUIs got its information. The program requests the six pieces of data one at a time, with no opportunity to go back and alter previously entered information. Then the screen clears and the six inputs are again requested one at a time.

```
Enter name (Enter EOD to terminate): Mr. President
Enter Address: 1600 Pennsylvania Avenue
Enter City: Washington
Enter State: DC
Enter Zip code: 20500
Enter Phone Number: 202-456-1414
```

FIGURE 2.1 Input screen of a pre-Visual Basic program to fill a database.

Figure 2.2 shows how an equivalent Visual Basic program gets its information. The boxes may be filled in any order. When the user clicks on a box with the mouse, the cursor moves to that box. The user can either type in new information or edit the existing information. When satisfied that all the information is correct, the user clicks on the *Write to Database* button. The boxes will clear, and the data for another person can be entered. After all names have been entered, the user clicks on the *Exit* button. In Fig. 2.1, the program is in control; in Fig. 2.2, the user is in control!

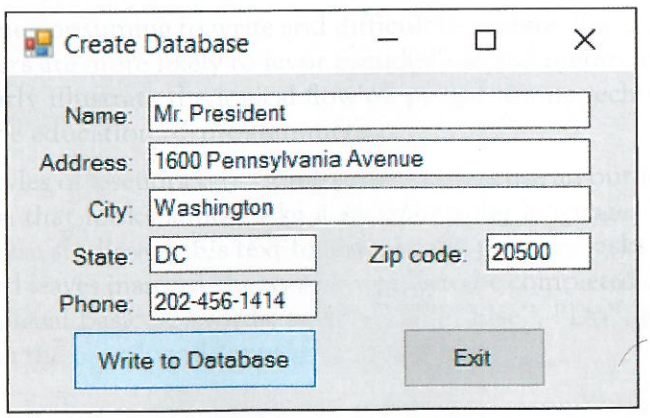


FIGURE 2.2 Input screen of a Visual Basic program to fill a database.

How You Develop a Visual Basic Program

A key element of planning a Visual Basic program is deciding what the user sees—in other words, designing the user interface. What data will he or she be entering? How large a window should the program use? Where will you place the buttons the user clicks on to activate actions in the program? Will the program have places to enter text (text boxes) and places to display output? What kind of warning boxes (message boxes) should the program use? In Visual Basic, the responsive objects a program designer places on windows are called *controls*. Two features make Visual Basic different from traditional programming tools:

1. You literally draw the user interface, much like using a paint program.
2. Perhaps more important, when you're done drawing the interface, the buttons, text boxes, and other objects that you have placed in a blank window will automatically recognize user actions such as mouse movements and button clicks. That is, the sequence of procedures executed in your program is controlled by "events" that the user initiates rather than by a predetermined sequence of procedures in your program.

In any case, only after you design the interface does anything like traditional programming occur. Objects in Visual Basic recognize events like mouse clicks. How the objects respond to them depends on the instructions you write. You always need to write instructions in order to make controls respond to events. This makes Visual Basic programming fundamentally different from traditional programming. Programs in traditional programming languages ran from the top down. For these programming languages, execution started from the first line and moved with the flow of the program to different parts as needed. A Visual Basic program works differently. Its core is a set of independent groups of instructions that are activated by the events they have been told to recognize. This event-driven methodology is a fundamental shift. The user decides the order in which things happen, not the programmer.

Most of the programming instructions in Visual Basic that tell your program how to respond to events like mouse clicks occur in what Visual Basic calls *event procedures*. Essentially, anything executable in a Visual Basic program either is in an event procedure or is used by an event procedure to help the procedure carry out its job. In fact, to stress that Visual Basic is fundamentally different from traditional programming languages, Microsoft uses the term *project* or *application*, rather than *program*, to refer to the combination of programming instructions and user interface that makes a Visual Basic program possible. Here is a summary of the steps you take to design a Visual Basic program:

1. Design the appearance of the window that the user sees.
2. Determine the events that the controls on the window should respond to.
3. Write the event procedures for those events.

Now here is what happens when the program is running:

1. Visual Basic monitors the controls in the window to detect any event that a control can recognize (mouse movements, clicks, keystrokes, and so on).
2. When Visual Basic detects an event, it examines the program to see if you've written an event procedure for that event.
3. If you have written an event procedure, Visual Basic executes the instructions that make up that event procedure and goes back to Step 1.
4. If you have not written an event procedure, Visual Basic ignores the event and goes back to Step 1.

These steps cycle continuously until the program ends. Usually, an event must happen before Visual Basic will do anything. Event-driven programs are more reactive than active—and that makes them more user friendly.

2.2 Visual Basic Controls

Visual Basic programs display a Windows-style screen (called a **form**) with boxes into which users type (and in which users edit) information and buttons that they click on to initiate actions. The boxes and buttons are referred to as **controls**. In this section, we examine forms and four of the most useful Visual Basic controls.

Starting a New Visual Basic Program

Each program is saved (as several files and subfolders) in its own folder. Before writing your first program, you should use File Explorer (with Windows 8 or 10) or Windows Explorer (with Windows 7) to create a folder to hold your programs.

The process for starting Visual Basic varies slightly with the version of Windows and the edition of Visual Studio installed on the computer. Some possible sequences of steps are shown below.

Windows 7 Click the Windows *Start* button, click *All Programs*, and then click on "Microsoft Visual Studio 2015."

Windows 8 and 8.1 Click the tile labeled "Visual Studio 2015." If there is no such tile, click on *Search* in the Charms bar, select the Apps category, type "Visual Studio" into the Search box in the upper-right part of the screen, and click on the rectangle labeled "Visual Studio 2015" that appears on the left side of the screen.

Windows 10 Click the Windows **Start** button, click *All apps*, and then click on *Visual Studio 2015*. Or, click on the tile labeled *Visual Studio 2015*.

Figure 2.3 shows the top part of the screen after Visual Basic is started. A Menu bar and a Toolbar are at the top of the screen. These two bars, with minor variations, are always

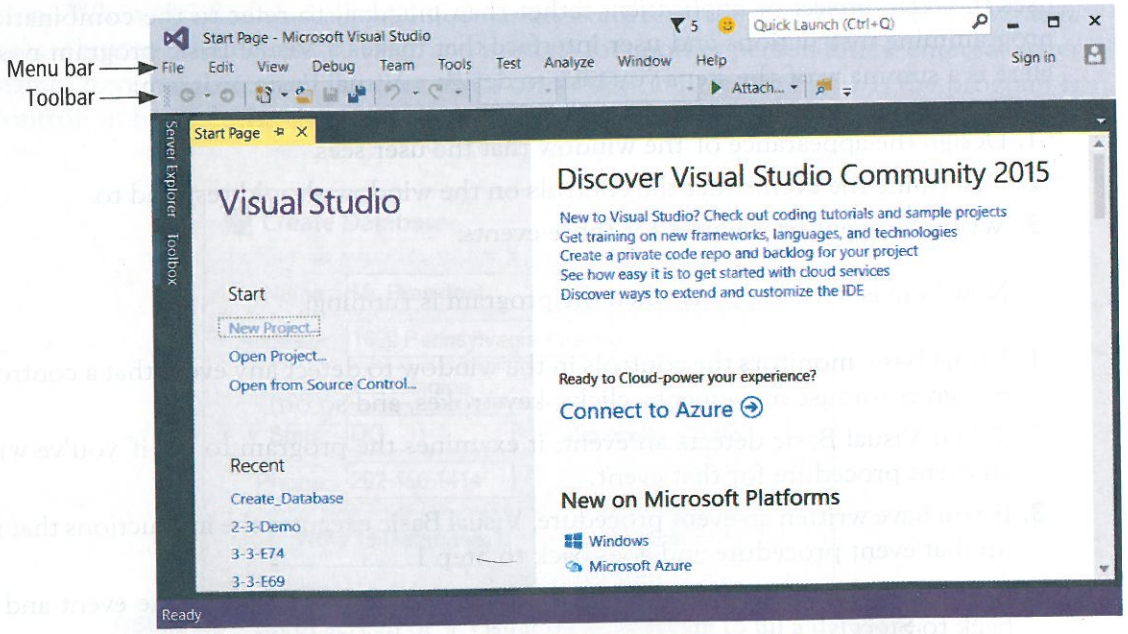


FIGURE 2.3 Visual Basic opening screen.

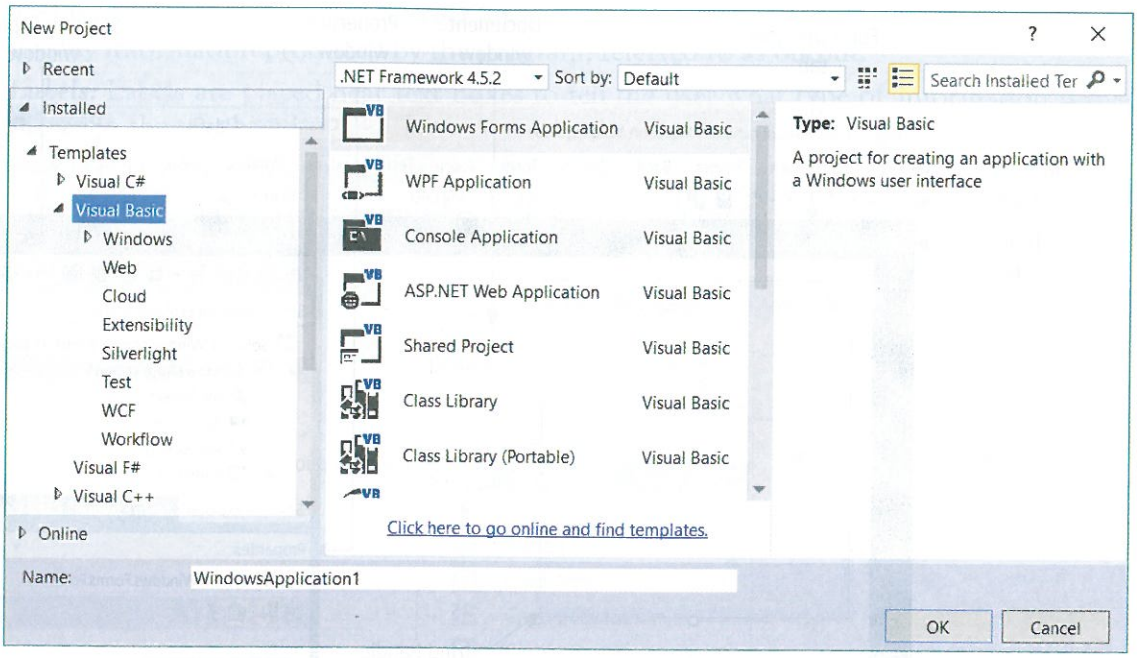


FIGURE 2.4 **The Visual Basic New Project dialog box.**

present while you are working with Visual Basic. The remainder of the screen is called the Start Page. Some tasks can be initiated from the Menu bar, the Toolbar, and the Start Page. We will usually initiate them from the Menu bar or the Toolbar.

The first item on the Menu bar is File. Click on File, hover over (or click on) *New*, and then click on *Project* to produce a New Project dialog box. (Alternately, press Alt/F/N/P or Ctrl+Shift+N.) Figure 2.4 shows a New Project dialog box produced by the Visual Basic Community 2015 edition. Your screen might look somewhat different than Fig. 2.4 even if you are using the Visual Basic Community 2015 edition.

Select *Visual Basic* in the *Templates* list on the left side of Fig. 2.4, and select *Windows Forms Application* in the center list. ***Note:*** The number of items in the center list will vary depending on the edition of Visual Studio you are using.

The name of the program, initially set to WindowsApplication1, can be specified at this time. Since we will have a chance to change it later, let's just use the name WindowsApplication1 for now. Click on the OK button to invoke the Visual Basic programming environment. See Fig. 2.5 on the next page. The Visual Basic programming environment is referred to as the **Integrated Development Environment** or **IDE**. The IDE contains the tools for writing, running, and debugging programs.

It is possible that your screen will look different than Fig. 2.5. The IDE is extremely configurable. Each window in Fig. 2.5 can have its location and size altered. New windows can be displayed in the IDE, and any window can be closed or hidden behind a tab. For instance, in Fig. 2.5 the Toolbox window is hidden behind a tab. The View menu is used to add additional windows to the IDE. If you would like your screen to look similar to Fig. 2.5, click on *Reset Windows Layout* in the Window menu, and then click on the *Yes* button.

The **Menu bar** of the IDE displays the menus of commands you use to work with Visual Basic. Some of the menus, like File, Edit, View, and Window, are common to most Windows applications. Others, such as Project, Debug, and Data, provide commands specific to programming in Visual Basic.

The **Toolbar** holds a collection of buttons that carry out standard operations when clicked. For example, you use the sixth button, which looks like two diskettes, to save the

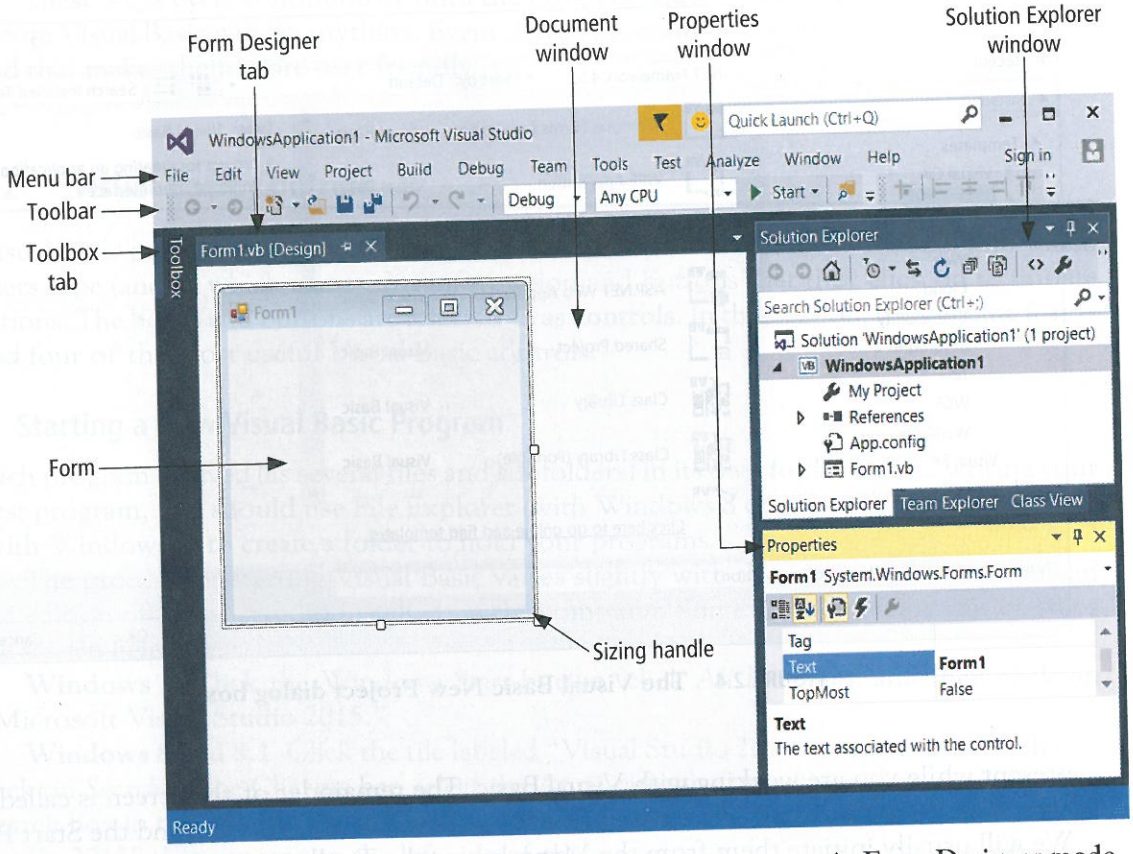


FIGURE 2.5 **The Visual Basic Integrated Development Environment in Form Designer mode.**

files associated with the current program. To reveal the purpose of a Toolbar button, hover the mouse pointer over it. The little information rectangle that pops up is called a **tooltip**.

The **Document window** currently holds the rectangular **Form window**, or **form** for short. (The Form window is also known as the **form designer window** or the **design window**.) The form becomes a Windows window when a program is executed. Most information displayed by the program appears on the form. The information usually is displayed in controls that the programmer has placed on the form. ***Note:*** You can change the size of the form by dragging one of its sizing handles.

The **Properties window** is used to change the initial appearance and behavior of objects on the form. Some (but not all) properties and appearances can be changed by code.

The **Solution Explorer** window displays the files associated with the program and provides access to the commands that pertain to them. (***Note:*** If the Solution Explorer or the Properties window is not visible, click on it in the View menu.)

The **Toolbox** holds icons representing objects (called controls) that can be placed on the form. If your screen does not show the Toolbox, hover the mouse over the Toolbox tab at the left side of the screen. The Toolbox will come into view. Then click on the pushpin icon in the title bar at the top of the Toolbox to keep the Toolbox permanently displayed in the IDE. (***Note:*** If there is no tab marked Toolbox, click on *Toolbox* in the View menu.)

The controls in the Toolbox are grouped into categories such as *All Windows Forms* and *Common Controls*. Figure 2.6 shows the Toolbox after the *Common Controls* group has been expanded. Most of the controls discussed in this text can be found in the list of common controls. (You can obtain a description of a control by hovering the mouse over the control.) The four controls discussed in this chapter are text boxes, labels, buttons, and list boxes. In order to see all the group names, collapse each of the groups.

Text boxes: Text boxes are used to get information from the user, referred to as **input**, or to display information produced by the program, referred to as **output**.

Labels: Labels are placed near text boxes to tell the user what type of information is displayed in the text boxes.

Buttons: The user clicks on a button to initiate an action.

List boxes: In the first part of this book, list boxes are used to display output. Later, they are used to make selections.

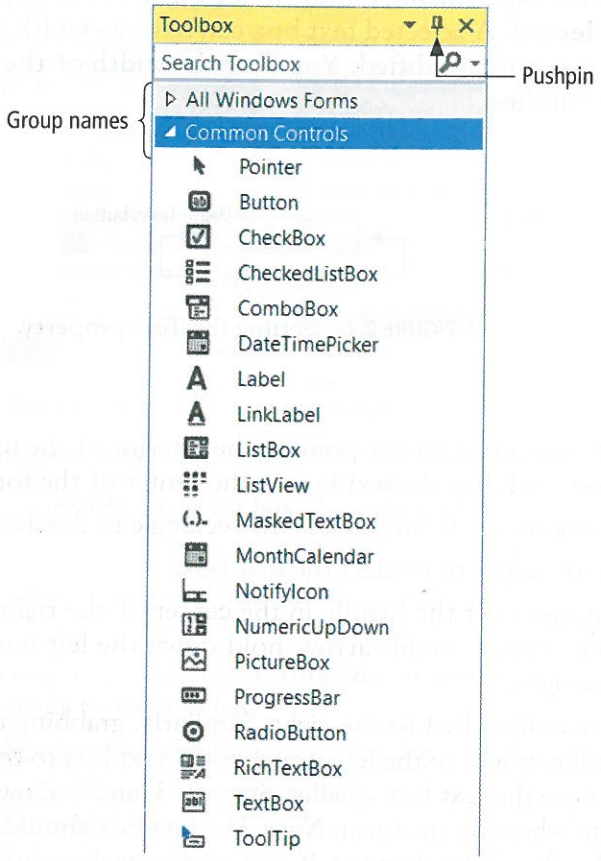


FIGURE 2.6 **The Toolbox's common controls.**

An Important Setting

The process of naming and saving programs can proceed in two different ways. In this book, we do not require that a program be given a name until it is saved. The following steps guarantee that Visual Basic will follow that practice.

1. Click on *Options* from the Tools menu to display an Options dialog box.
2. Click on the *Projects and Solutions* item in the left pane of the Options dialog box.
3. If the box labeled "Save new projects when created" is checked, uncheck it.
4. Click on the *OK* button.
5. Open the File menu in the Toolbar and click on *Close Solution*. ***Note:*** If a dialog box appears and asks you if you want to save or discard changes to the current project, click on the *Discard* button.

VideoNote
Textbox Walkthrough

A Text Box Walkthrough

Place a text box on a form

1. Start a new Visual Basic program.
2. Double-click on the TextBox control (TextBox) in the *Common Controls* group of the Toolbox.

   A rectangle with three small squares appears at the upper-left corner of the form. The square on the top of the text box, called the **Tasks button**, can be used to set the MultiLine property of the text box. The squares on the left and right sides of the text box are called **sizing handles**. See Fig. 2.7. An object showing its handles is said to be **selected**. A selected text box can have its width altered, location changed, and other properties modified. You alter the width of the text box by dragging one of its sizing handles.

FIGURE 2.7 **Setting the Text property.**

3. Move the mouse cursor to any point in the interior of the text box, hold down the left mouse button, and drag the text box to the center of the form.
4. Click anywhere on the form outside the rectangle to deselect the text box.
5. Click on the rectangle to reselect the text box.
6. Hover the mouse over the handle in the center of the right side of the text box until the cursor becomes a double-arrow, hold down the left mouse button, and move the mouse to the right.

   The text box is stretched to the right. Similarly, grabbing the handle on the left side and moving the mouse to the left stretches the text box to the left. You also can use the handles to make the text box smaller. Steps 2, 3, and 6 allow you to place a text box of any width anywhere on the form. ***Note:*** The text box should now be selected; that is, its sizing handles should be showing. If not, click anywhere inside the text box to select it.
7. Press the Delete key to remove the text box from the form.

   Step 8 gives an alternative way to place a text box of any width at any location on the form.
8. Click on the text box icon in the Toolbox, move the mouse pointer to any place on the form, hold down the left mouse button, drag the mouse on a diagonal, and release the mouse button to create a selected text box.

   You can now alter the width and location as before. ***Note:*** The text box should now be selected. If not, click anywhere inside the text box to select it.

Activate, move, and resize the Properties window

9. Press F4 to activate the Properties window.

   You also can activate the Properties window by clicking on it, clicking on *Properties Window* from the View menu, or right-clicking on the text box with the mouse button and selecting *Properties* from the context menu that appears. See Fig. 2.8. The first line of the Properties window (called the **Object box**) reads "TextBox1", etc.

TextBox1 is the current name of the text box. The third button in the row of buttons below the Object box, the *Properties* button ( ), is normally highlighted. If not, click on it. The left column of the Properties window gives the available properties, and the right column gives the current settings of the properties. The first two buttons ( ) in the row of buttons below the Object box permit you to view the list of properties either grouped into categories or alphabetically. You can use the up- and down-arrow keys (or the scroll arrows, scroll box, or the mouse scroll wheel) to move through the list of properties.

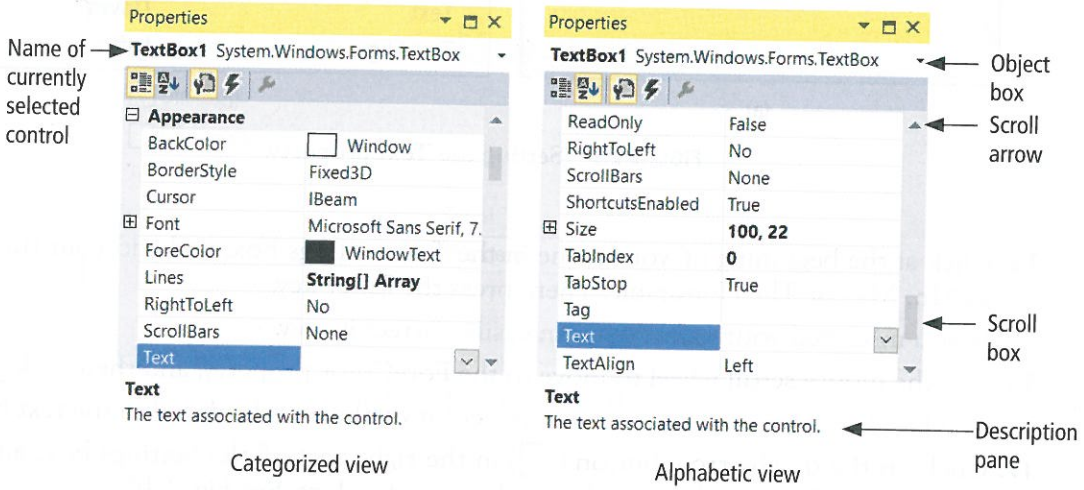


FIGURE 2.8 **Text box Properties window.**

10. Click on the Properties window's title bar and drag the window to the center of the screen.

    The Properties window is said to be **floating** or **undocked**. Some people find a floating window easier to work with.

11. Drag the lower-right corner of the Properties window to change the size of the Properties window.

    An enlarged window will show more properties at once.

12. Hold down the Ctrl key and double-click on the title bar.

    The Properties window will return to its original docked location. We now will discuss four properties in this walkthrough.

Set four properties of the text box

Assume that the text box is selected and its Properties window activated.

Note 1: The third and fourth buttons below the Object box, the *Properties* button and the *Events* button, determine whether properties or events are displayed in the Properties window. Normally the *Properties* button is highlighted. If not, click on it.

Note 2: If the Description pane is not visible, right-click on the Properties window, then click on *Description*. The Description pane describes the currently highlighted property.

13. Move to the Text property with the up- and down-arrow keys (alternatively, scroll until the Text property is visible, and click on the property).

    The Text property, which determines the words displayed in the text box, is now highlighted. Currently, there is no text displayed in the Text property's Settings box on its right.

14. Type your first name, and then press the Enter key or click on another property. Your name now appears in both the Settings box and the text box. See Fig. 2.9.

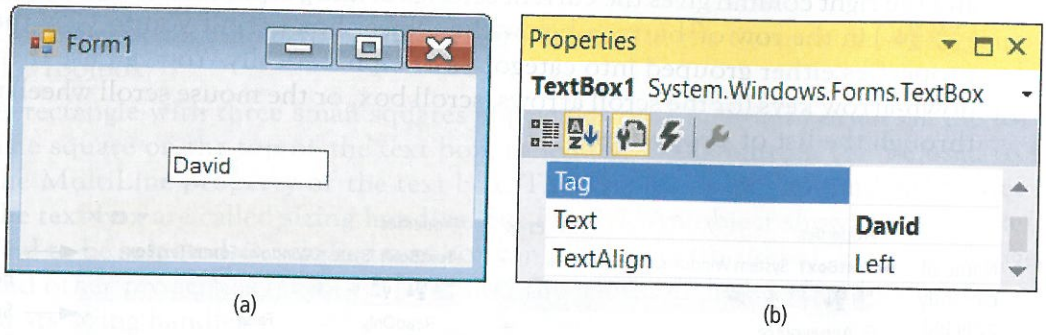


FIGURE 2.9 **Setting the Text property.**

15. Click at the beginning of your name in the Text Settings box, and add your title, such as Mr., Ms., or The Honorable. Then, press the Enter key.

    If you mistyped your name, you can easily correct it now.

16. Use the mouse scroll wheel to move to the ForeColor property, and then click on it.

    The ForeColor property determines the color of the text displayed in the text box.

17. Click on the down-arrow button ( ) in the right part of the Settings box, and then click on the Custom tab to display a selection of colors. See Fig. 2.10.

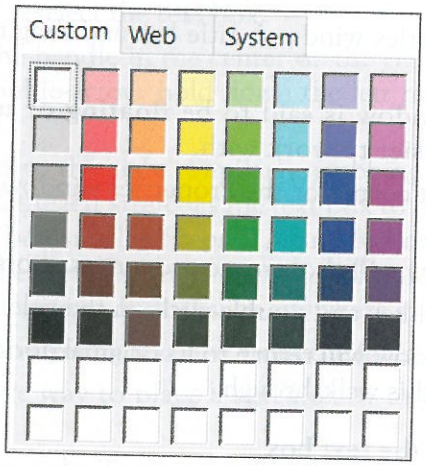


FIGURE 2.10 **Setting the ForeColor property.**

18. Click on one of the colors, such as *blue* or *red*.

    Notice the change in the color of your name.

19. Select the Font property with a single click of the mouse, and click on the ellipsis button ( ) in the right part of its Settings box.

    The Font dialog box in Fig. 2.11 is displayed. The three lists give the current name (Microsoft Sans Serif), current style (Regular), and current size (8 point) of the font. You can change any of these attributes by clicking on an item in its list or by typing into the box at the top of the list.

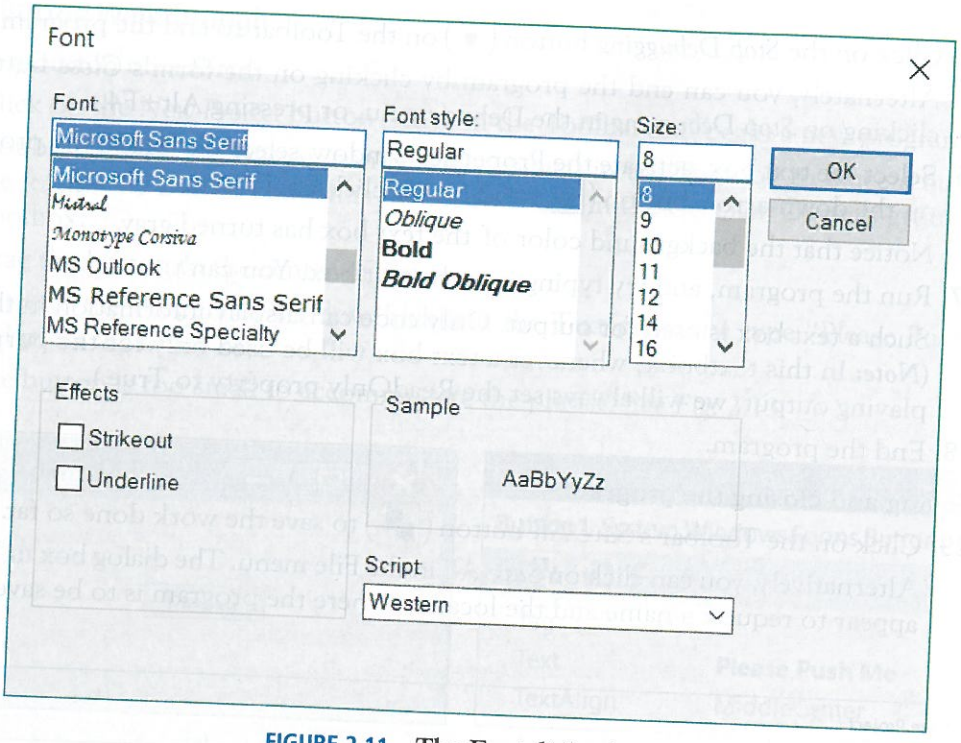


FIGURE 2.11 The Font dialog box.

20. Click on *Bold* in the *Font style* list, click on *12* in the *Size* list, and click on the *OK* button.

Your name is now displayed in a larger bold font. The text box will expand so that it can accommodate the larger font.

21. Click on the text box and resize it to be about 3 inches wide.

Visual Basic programs consist of three parts: interface, values of properties, and code. Our interface consists of a form with a single object—a text box. We have set a few properties for the text box—the text (namely, your name), the foreground color, the font style, and the font size. In Section 2.3, we discuss how to place code into a program. Visual Basic endows certain capabilities to programs that are independent of any code we write. We will now run the current program without adding any code and experience these capabilities.

Run and end the program

22. Click on the *Start* button (▶) on the Toolbar to run the program.

Alternatively, you can press F5 to run the program or can click on *Start Debugging* in the Debug menu. After a brief delay, a copy of the form appears with your name highlighted.

23. Press the End key to move the cursor to the end of your name, type in your last name, and then keep typing.

Eventually, the words will scroll to the left.

24. Press the Home key to return to the beginning of your name.

The text box functions like a miniature word processor. You can place the cursor anywhere you like in order to add or delete text. You can drag the cursor across text to select a block, place a copy of the block in the Clipboard with Ctrl+C, and then duplicate it elsewhere with Ctrl+V.

25. Click on the *Stop Debugging* button ( ) on the Toolbar to end the program.
    Alternately, you can end the program by clicking on the form's *Close* button ( ), clicking on *Stop Debugging* in the Debug menu, or pressing Alt+F4.
26. Select the text box, activate the Properties window, select the ReadOnly property, click on the down-arrow button ( ), and finally click on True.
    Notice that the background color of the text box has turned gray.
27. Run the program, and try typing into the text box. You can't.
    Such a text box is used for output. Only code can display information in the text box. (**Note:** In this textbook, whenever a text box will be used only for the purpose of displaying output, we will always set the ReadOnly property to True.)
28. End the program.

Saving and closing the program

29. Click on the Toolbar's *Save All* button ( ) to save the work done so far.
    Alternatively, you can click on *Save All* in the File menu. The dialog box in Fig. 2.12 will appear to request a name and the location where the program is to be saved.

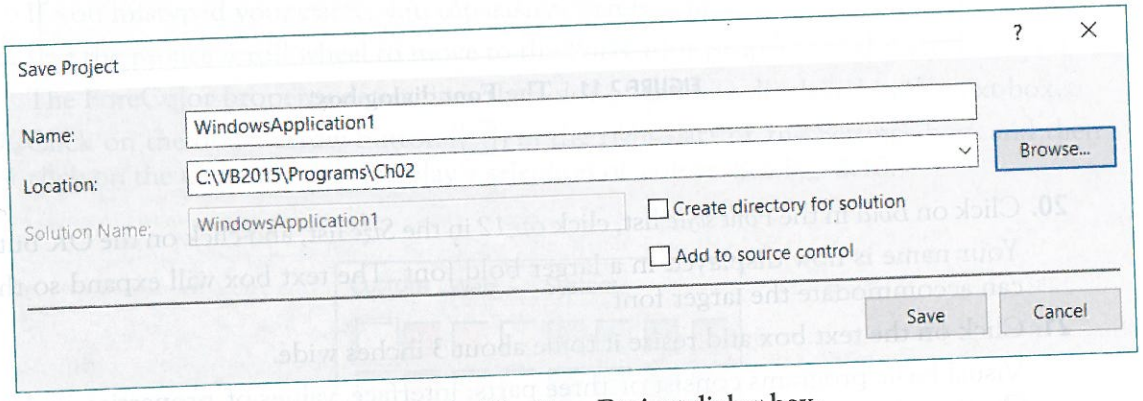


FIGURE 2.12 The Save Project dialog box.

30. Type a name for the program, such as "VBdemo".
    Use Browse to locate a folder. (This folder will automatically be used the next time you click on the *Save All* button.) The files for the program will be saved in a subfolder of the selected folder.
    Important: If the "Create directory for solution" check box is checked, then click on the check box to uncheck it.
31. Click on the *Save* button.
32. Click on *Close Solution* in the File menu.
    In the next step we reload the program.
33. Hover over (or click on) *Open* in the File menu and then click on *Project/Solution* in the context menu that drops down. Navigate to the folder corresponding to the program you just saved, double-click on the folder, and double-click on the file with extension *sln*.
    If you do not see the Form Designer for the program, double-click on Form1.vb in the Solution Explorer. The program now exists just as it did after Step 28. You can now modify the program and/or run it. (**Note:** You can also carry out the task in the first sentence by pressing Alt/F/O/P or Ctrl+Shift+O.)
34. Click on *Close Solution* in the File menu to close the program.

A Button Walkthrough

VideoNote
Button Walkthrough

Place a button on a form

1. Click on the *New Project* button ( ) on the Toolbar and begin a new program.
2. Double-click on the Button control (Button) in the Toolbox to place a button on the form. The Button control is the second item in the *Common Controls* group of the Toolbox.
3. Drag the button to the center of the form.
4. Activate the Properties window, highlight the Text property, type "Please Push Me", and press the Enter key.

   The button is too small to accommodate the phrase. See Fig. 2.13.

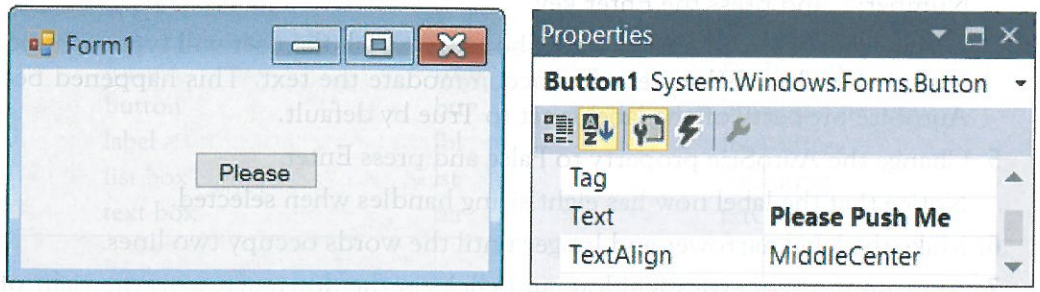


FIGURE 2.13 Setting the Text property.

5. Click on the button to select it, and then drag the right-hand sizing handle to widen the button so that it can accommodate the phrase "Please Push Me" on one line.

   Alternately, you can drag the bottom sizing handle down and have the phrase displayed on two lines.
6. Run the program, and click on the button.

   The color of the button turns blue when the mouse hovers over it. In Section 2.3, we will write code that is executed when a button is clicked on.
7. End the program and select the button.
8. From the Properties window, edit the Text setting by inserting an ampersand (&) before the first letter *P*, and then press the Enter key.

   Notice that the first letter *P* on the button is now underlined. See Fig. 2.14. Pressing Alt+P while the program is running causes the same event to occur as does clicking the button. Here, *P* is referred to as the **access key** for the button. (The access key is always the character following the ampersand.)

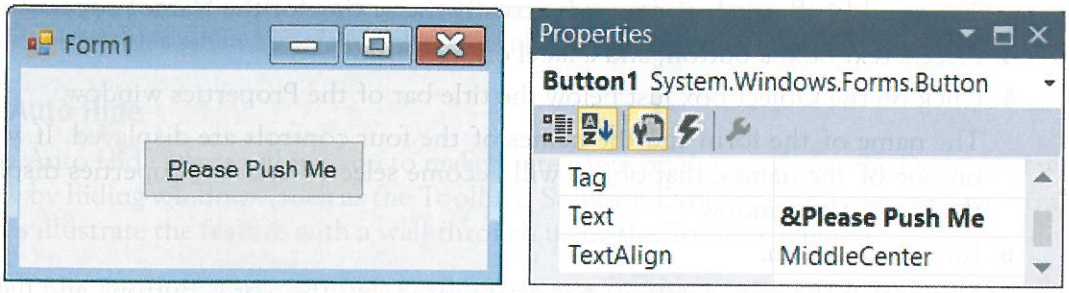


FIGURE 2.14 Designating P as an access key.

9. Click on *Close Solution* in the File menu to close the program.
   There is no need to save this program, so click on the *Discard* button.

A Label Walkthrough

1. Click on the *New Project* button on the Toolbar and begin a new program.
   Feel free to keep the default name, such as WindowsApplication1.
2. Double-click on the Label control (A Label) in the Toolbox to place a label on the form.
3. Drag the label to the center of the form.
4. Activate the Properties window, highlight the Text property, type "Enter Your Phone Number:", and press the Enter key.
   Such a label is placed next to a text box into which the user will type a phone number. Notice that the label widened to accommodate the text. This happened because the AutoSize property of the label is set to True by default.
5. Change the AutoSize property to False and press Enter.
   Notice that the label now has eight sizing handles when selected.
6. Make the label narrower and longer until the words occupy two lines.
7. Activate the Properties window, and click on the down arrow to the right of the setting for the TextAlign property. Experiment by clicking on the various rectangles and observing their effects.
   The combination of sizing and alignment permits you to design a label easily.
8. Run the program.
   Nothing happens, even if you click on the label. Labels just sit there. The user cannot change what a label displays unless you write code to make the change.
9. End the program.
10. Click on *Close Solution* in the File menu to close the program.
    There is no need to save this program, so click on the *Discard* button.

A List Box Walkthrough

1. Click on the *New Project* button on the Toolbar and begin a new program.
   Feel free to keep the default name, such as WindowsApplication1.
2. Place a ListBox control (ListBox) on the form.
3. Press F4 to activate the Properties window and notice that the list box does not have a Text property.
   The word ListBox1 that appears is actually the setting for the Name property.
4. Place a text box, a button, and a label on the form.
5. Click on the Object box just below the title bar of the Properties window.
   The name of the form and the names of the four controls are displayed. If you click on one of the names, that object will become selected and its properties displayed in the Properties window.
6. Run the program.
   Notice that the word ListBox1 has disappeared, but the words Button1 and Label1 are still visible. The list box is completely blank. In subsequent sections, we will write code to place information into the list box.
7. End the program.

8. Click on *Close Solution* in the File menu to close the program.

   There is no need to save this program, so click on the *Discard* button.

The Name Property

The form and each control on it has a Name property. By default, the form is given the name Form1 and controls are given names such as TextBox1 and TextBox2. These names can (and should) be changed to descriptive ones that reflect the purpose of the form or control. Also, it is a good programming practice to have each name begin with a three-letter prefix that identifies the type of the object. See Table 2.1.

TABLE 2.1 Some three-letter prefixes.

Object	Prefix	Example
form	frm	frmPayroll
button	btn	btnComputeTotal
label	lbl	lblAddress
list box	lst	lstOutput
text box	txt	txtCity

The Solution Explorer window contains a file named Form1.vb that holds information about the form. Form1 is also the setting of the form's Name property in the Properties window. If you change the base name of the file Form1.vb, the setting of the Name property will automatically change to the new name. To make the change, right-click on Form1.vb in the Solution Explorer window, click on *Rename* in the context menu that appears, type in a new name (such as frmPayroll.vb), and press the Enter key. ***Important:*** Make sure that the new filename keeps the extension *vb*.

The name of a control placed on a form is changed from the control's Properties window. (The Name property is always the third property in the alphabetized list of properties.) Names of controls and forms must begin with a letter and can include numbers, letters, and underscore (_) characters, but cannot include punctuation marks or spaces.

The Name and Text properties of a button are both initially set to something like Button1. However, changing one of these properties does not affect the setting of the other properties, and similarly for the Name and Text properties of forms, text boxes, and labels. The Text property of a form specifies the words appearing in the form's title bar.

Fonts

The default font for controls is Microsoft Sans Serif. Courier New is another commonly used font. Courier New is a fixed-width font; that is, each character has the same width. With such a font, the letter i occupies the same space as the letter m. Fixed-width fonts are used with tables when information is to be aligned in columns.

Auto Hide

The Auto Hide feature allows you to make more room on the screen for the Document window by hiding windows (such as the Toolbox, Solution Explorer, and Properties windows). Let's illustrate the feature with a walkthrough using the Toolbox window.

1. If the Toolbox window is not visible, click on *Toolbox* in the Menu bar's View menu to see the window.

   Auto Hide is enabled when the pushpin icon is horizontal ( ). When the Auto Hide feature is enabled, the Toolbox window will move out of view when not needed.

2. If the pushpin icon is vertical ( ), then click on the icon to make it horizontal.
   The Auto Hide feature is now enabled.
3. Press Ctrl+Alt+X to display the Toolbox, and then move the mouse cursor somewhere outside the Toolbox window and click the left mouse button.
   The window becomes a tab captioned Toolbox on the left side of the screen.
4. Click on the tab.
   The window comes into view and is ready for use. After you click outside the window, it will return back into the tab.
5. Click on the pushpin icon to make it vertical.
   The Auto Hide feature is now *disabled*.
6. Click the mouse cursor somewhere outside the Toolbox window.
   The Toolbox window stays fixed. ***Note:*** We recommend keeping Auto Hide disabled for the Toolbox, Solution Explorer, and Properties windows unless you are creating a program with a very large form and need extra space.

Positioning and Aligning Controls

Visual Basic provides several tools for positioning and aligning controls on a form. **Proximity lines** are short line segments that help you place controls a comfortable distance from each other and from the sides of the form. **Snap lines** are horizontal and vertical line segments that help you align controls. The Format menu is used to align controls, center controls horizontally and vertically in a form, and make a group of selected controls the same size.

A Positioning and Aligning Walkthrough

1. Begin a new program.
2. Place a button near the center of the form.
3. Drag the button toward the upper-right corner of the form until two short line segments appear. The line segments are called **proximity lines**. See Fig. 2.15(a). The button is now a comfortable distance from each of the two sides of the form.

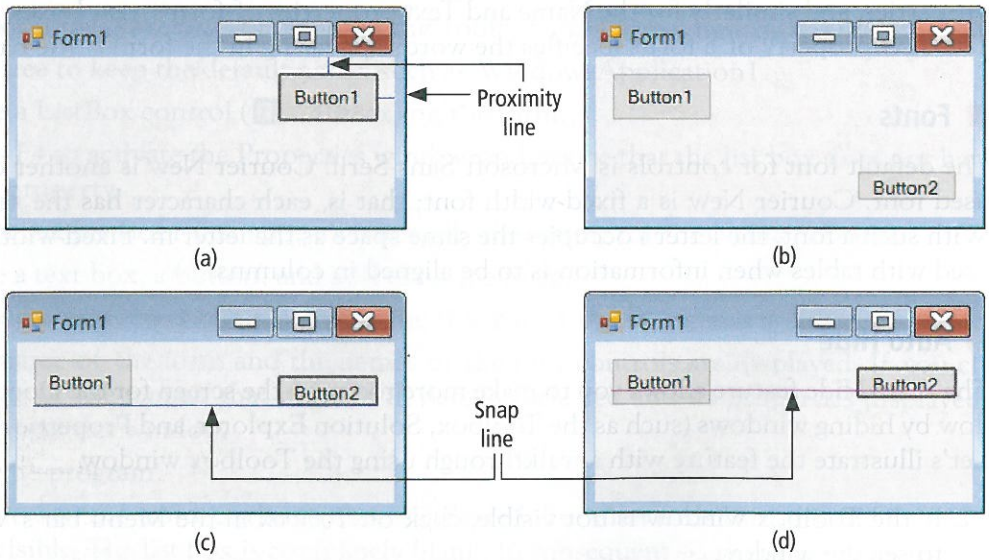


FIGURE 2.15 Positioning controls.

4. Place a second button below the first button and drag it upward until a proximity line appears between the two buttons.

   The buttons are now a comfortable distance apart.

5. Resize and position the two buttons as shown in Fig. 2.15(b).

6. Drag Button2 upward until a blue line appears along the bottoms of the two buttons.

   See Fig. 2.15(c). This blue line is called a **snap line**. The bottoms of the two buttons are now aligned.

7. Continue dragging Button2 upward until a purple snap line appears just underneath the words Button1 and Button2.

   See Fig. 2.15(d). The texts in the two buttons are now aligned. If we were to continue dragging Button2 upward, a blue snap line would tell us when the tops were aligned. Steps 8 and 9 present another way to align the tops of the controls.

8. Click on Button1 and then hold down the Ctrl key and click on Button2.

   After the mouse button is released, both buttons will be selected. ***Note:*** This process (called **selection of multiple controls**) can be repeated to select a group of any number of controls.

9. With the two buttons still selected, open the Format menu in the Menu bar, hover over *Align*, and click on *Tops*.

   The tops of the two buttons are now aligned. Precisely, Button1 (the first button selected) will stay fixed, and Button2 will move up so that its top is aligned with the top of Button1. The *Align* submenu also is used to align middles or corresponding sides of a group of selected controls. Some other useful submenus of the Format menu are as follows:

   Make Same Size: Equalize the width and/or height of the controls in a group of selected controls.

   Center in Form: Center a selected control either horizontally or vertically in a form.

   Vertical Spacing: Equalize the vertical spacing between a column of three or more selected controls.

   Horizontal Spacing: Equalize the horizontal spacing between a row of three or more selected controls.

10. With the two buttons still selected, open the Properties window and set the ForeColor property to blue.

    Notice that the ForeColor property has been altered for both buttons. Actually, any property that is common to every control in a group of selected multiple controls can be set simultaneously for the entire group of controls.

Multiple Controls

When a group of controls are selected with the Ctrl key, the first control selected (called the **dominant control** of the group) will have white sizing handles, while the other controls will have black sizing handles. All alignment and sizing statements initiated from the Format menu will keep the dominant control fixed and will align (or size) the other controls with respect to the dominant control. You can designate a different control to be the dominant control by clicking on it.

After multiple controls have been selected, they can be dragged, deleted, and have properties set as a group. The arrow keys also can be used to move and size multiple controls as a group.

A group of multiple controls also can be selected by clicking the mouse outside the controls, dragging it across the controls, and releasing it. The *Select All* command from the Edit menu (or the key combination Ctrl+A) causes all the controls on the form to be selected.

Setting Tab Order

At any time, only one control can receive user input through the keyboard. That control is said to have the ***focus***. When a text box has the focus, there is a blinking cursor inside it.

Whenever the Tab key is pressed while a program is running, the focus moves from one control to another. The following walkthrough explains how to determine the order in which the focus moves and how that order can be changed.

1. Start a new program.
2. Place a button, a text box, and a list box on the form.
3. Run the program, and successively press the Tab key.

   Notice that the controls receive the focus in the order they were placed on the form.
4. End the program.
5. Click on *Tab Order* in the View menu.

   The screen appears as in Fig. 2.16(a). The controls are numbered from 0 to 2 in the order they were created. Each of the numbers is referred to as a **tab index**.

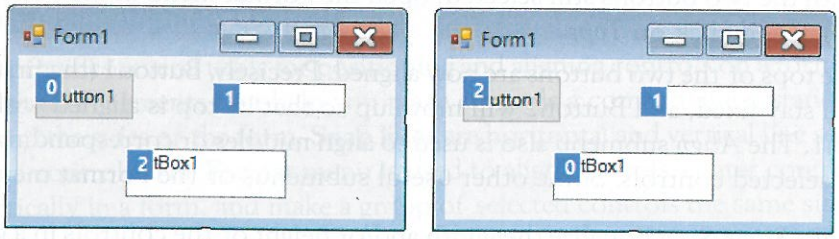


FIGURE 2.16 **Tab order.**

6. Click on the list box, then the text box, and finally the button.

   Notice that the tab indexes change as shown in Fig. 2.16(b).
7. Click again on *Tab Order* in the View menu to set the new tab order.
8. Run the program again, and successively press the Tab key.

   Notice that the controls receive the focus according to the new tab order.
9. End the program.
10. Add a label to the form, rerun the program, and successively press the Tab key.

    Notice that the label does not receive the focus. Whether or not a control can receive the focus is determined by the setting of its TabStop property. By default, the setting of the TabStop property is True for buttons, text boxes, and list boxes, and False for labels. In this book we always use these default settings. ***Note:*** Even though labels do not receive the focus while tabbing, they are still assigned a tab index.

Comments

1. While you are working on a program, the program resides in memory. Removing a program from memory is referred to as **closing** the program. A program is automatically closed when you begin a new program. Also, it can be closed directly with the *Close Solution* command from the File menu.
2. Three useful properties that have not been discussed are the following:

   (a) ***BackColor:*** This property specifies the background color for the form or a control.

(b) ***Visible:*** Setting the Visible property to False causes an object to disappear when the program is run. The object can be made to reappear with code.

(c) ***Enabled:*** Setting the Enabled property of a control to False restricts its use. It appears grayed and cannot receive the focus. Controls sometimes are disabled temporarily when they are not needed in the current state of the program.

3. Most properties can be set or altered with code as the program is running instead of being preset from the Properties window. For instance, a button can be made to disappear with a line such as `Button1.Visible = False`. The details are presented in Section 2.3.

4. If you inadvertently double-click on a form, a window containing text will appear. (The first line is Public Class Form1.) This is the Code Editor, which is discussed in the next section. To return to the Form Designer, click on the tab at the top of the Document window labeled "Form1.vb [Design]."

5. We have seen two ways to place a control onto a form. Another way is to just click on the control in the Toolbox and then drag the control from the Toolbox to the location in the form.

6. There is a small down-arrow button on the right side of the Text property setting box. When you click on that button, a rectangular box appears. The setting for the Text property can be typed into this box instead of into the Settings box. This method of specifying the setting is especially useful when you want a button to have a multiline caption.

7. We recommend setting the StartPosition property of the form to CenterScreen. With this setting the form will appear in the center of the screen when the program is run.

8. Refer to the properties windows in Fig. 2.8. If you click on the button at the right side of the Properties window's Object box, a list showing all the controls on the form will drop down. You can then click on one of the controls to make it the selected control.

9. Exercises 35 through 47 develop additional techniques for manipulating and accessing controls placed on a form. We recommend that you work these exercises whether or not they are assigned by your instructor.

Practice Problems 2.2

1. What is the difference between the Text and the Name properties of a button?

2. The first two group names in the Toolbox are *All Windows Forms* and *Common Controls*. How many groups are there?

EXERCISES 2.2

1. Create a form with two buttons, run the program, and click on each button. What do you notice different about a button after it has been clicked?

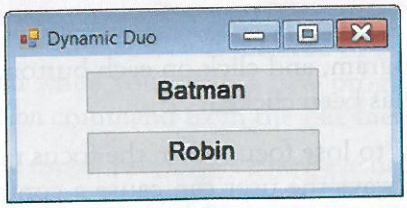
2. While a program is running, a control is said to lose focus when the focus moves from that control to another control. Give three ways the user can cause a control to lose focus.

In Exercises 3 through 24, carry out the task.

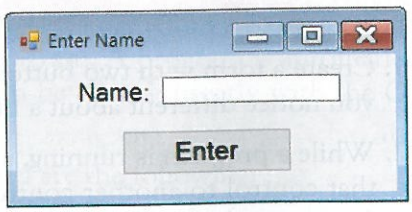
3. Place "CHECKING ACCOUNT" in the title bar of a form.
4. Create a text box containing the words "PLAY IT, SAM" in blue letters.
5. Create a text box with a yellow background.
6. Create a text box named txtGreeting and containing the word "HELLO" in large italic letters.
7. Create a label containing the sentence "After all is said and done, more is said than done." The sentence should occupy three lines, and each line should be centered horizontally in the label.
8. Create a read-only text box containing the words "Visual Basic" in bold white letters on a red background.
9. Create a text box named txtLanguage containing the words "Visual Basic 2015" in Courier New font.
10. Create a yellow button named btnPush containing the word "PUSH".
11. Create a white button containing the word "PUSH" in large italic letters.
12. Create a button containing the word "PUSH" in bold letters with the letter P underlined.
13. Create a button containing the word "PUSH" with the letter H as the access key.
14. Create a label containing the word "ALIAS" in white on a blue background.
15. Create a label named lblAKA containing the centered italicized word "ALIAS".
16. Place "BALANCE SHEET" in the title bar of a form having a yellow background.
17. Create a label containing "VISUAL" on the first line and "BASIC" on the second line. Each word should be right-justified.
18. Create a form named frmHello whose title bar reads "Hello World".
19. Create a label containing the underlined word "PROGRAM" in italics.
20. Create a label containing the bold word "ALIAS" in the Courier New font.
21. Create a list box with a yellow background.
22. Create a list box that will be invisible when the program is run.
23. Create a form named frmYellow with a yellow background.
24. Create a button containing the bold underlined word "BUTTON".

In Exercises 25 through 30, create the form shown in the figure. (These exercises give you practice creating controls and assigning properties. The interfaces do not necessarily correspond to actual programs.)

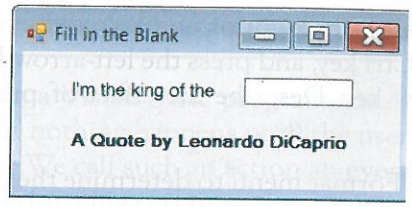
25.



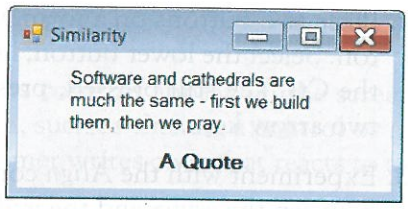
26.



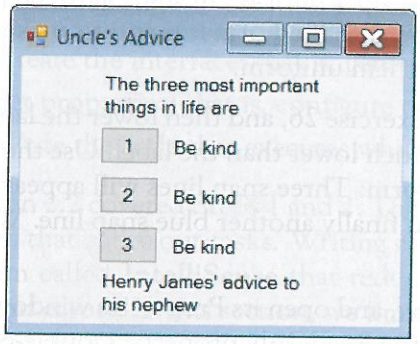
27.



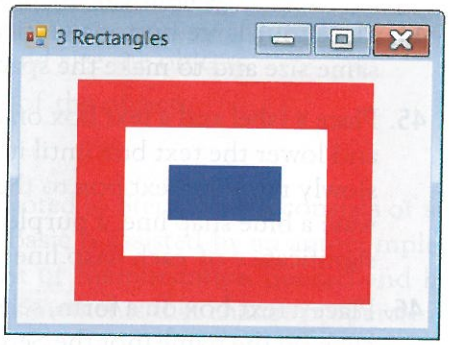
28.



29.



30.



31. Create a replica of your bank check on a form. Words common to all checks, such as "PAY TO THE ORDER OF", should be contained in labels. Items specific to your checks, such as your name at the top left, should be contained in text boxes. Make the check on the screen resemble your personal check as much as possible. ***Note:*** Omit the account number.

32. Create a replica of your campus ID on a form. Words that are on all student IDs, such as the name of the college, should be contained in labels. Information specific to your ID, such as your name and student ID number, should be contained in text boxes.

33. Consider the form shown in Exercise 25. Assume the *Batman* button was added to the form before the *Robin* button. What is the tab index of the *Robin* button?

34. Consider the form shown in Exercise 26. Assume the first control added to the form was the label. What is the tab index of the label?

The following hands-on exercises develop additional techniques for manipulating and accessing controls placed on a form.

35. Place a text box on a form and select the text box. What is the effect of pressing the various arrow keys?

36. Place a text box on a form and select the text box. What is the effect of pressing the various arrow keys while holding down the Shift key?

37. Repeat Exercise 36 for selected multiple controls.

38. Repeat Exercise 35 for selected multiple controls.

39. Place a label and a list box on a form and change their font sizes to 12 at the same time.

40. Place a button in the center of a form and select it. Hold down the Ctrl key and press an arrow key. Repeat this process for each of the other arrow keys. Describe what happens.

41. Place a label and a text box on a form with the label to the left of and above the text box. Select the label. Hold down the Ctrl key and press the down-arrow key twice. With the Ctrl key still pressed, press the right-arrow key. Describe what happens.

42. Place two buttons on a form with one button to the right of and below the other button. Select the lower button, hold down the Ctrl key, and press the left-arrow key. With the Ctrl key still pressed, press the up-arrow key. Describe the effect of pressing the two arrow keys.

43. Experiment with the *Align* command on the Format menu to determine the difference between the *center* and the *middle* of a control.

44. Place four large buttons vertically on a form. Use the Format menu to make them the same size and to make the spacing between them uniform.

45. Place a label and a text box on a form as in Exercise 26, and then lower the label slightly and lower the text box until it is about one inch lower than the label. Use the mouse to slowly raise the text box to the top of the form. Three snap lines will appear along the way: a blue snap line, a purple snap line, and finally another blue snap line. What is the significance of each snap line?

46. Place a text box on a form, select the text box, and open its Properties window. Double-click on the name (not the Settings box) of the ReadOnly property. Double-click again. What is the effect of double-clicking on a property whose possible settings are True and False?

47. Place a button on a form, select the button, and open its Properties window. Double-click on the name (not the Settings box) of the ForeColor property. Double-click repeatedly. Describe what is happening.

Solutions to Practice Problems 2.2

1. The text is the words appearing on the button, whereas the name is the designation used to refer to the button in code. Initially, they have the same value, such as Button1. However, each can be changed independently of the other.
2. The Toolbox in the Community Edition of Visual Basic contains 10 groups. Figure 2.17 shows the Toolbox after each group has been collapsed. ***Note:*** In some other editions of Visual Basic the Toolbox contains more groups.

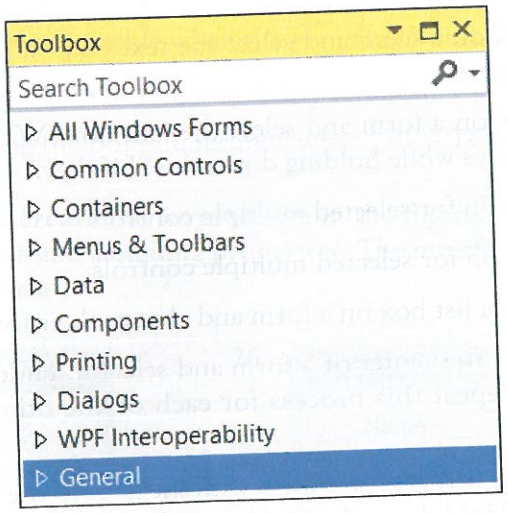


FIGURE 2.17 **Toolbox group names.**

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 certain tasks.

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

   ```
   txtBox.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 as 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(sender As System.Object,
            e As System.EventArgs) Handles objectName.event
```

Since we rarely 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

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

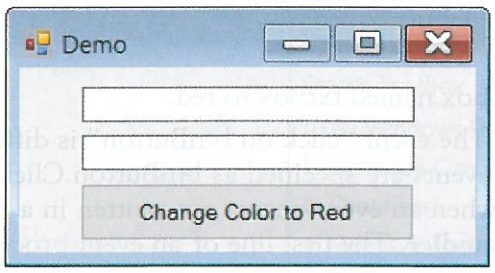
Words such as "Private," "As," "Sub," "Handles," and "End" have special meanings in Visual Basic and are referred to as **keywords** or **reserved words**. The Code Editor automatically capitalizes the first letter of a keyword and displays the word in blue. The word "Sub" in the first line signals the 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** (or to **invoke**, **fire**, or **trigger**) the event, and the event procedure is said to **handle** the event.

An Event Procedure Walkthrough

The form in Fig. 2.18 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.



OBJECT	PROPERTY	SETTING
frmDemo	Text	Demo
txtFirst		
txtSecond		
btnRed	Text	Change Color to Red

FIGURE 2.18 The interface for the event procedure walkthrough.

1. Create the interface in Fig. 2.18 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 Demo, 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** (also known as the *Code view* or the *Code window*). See Fig. 2.19.

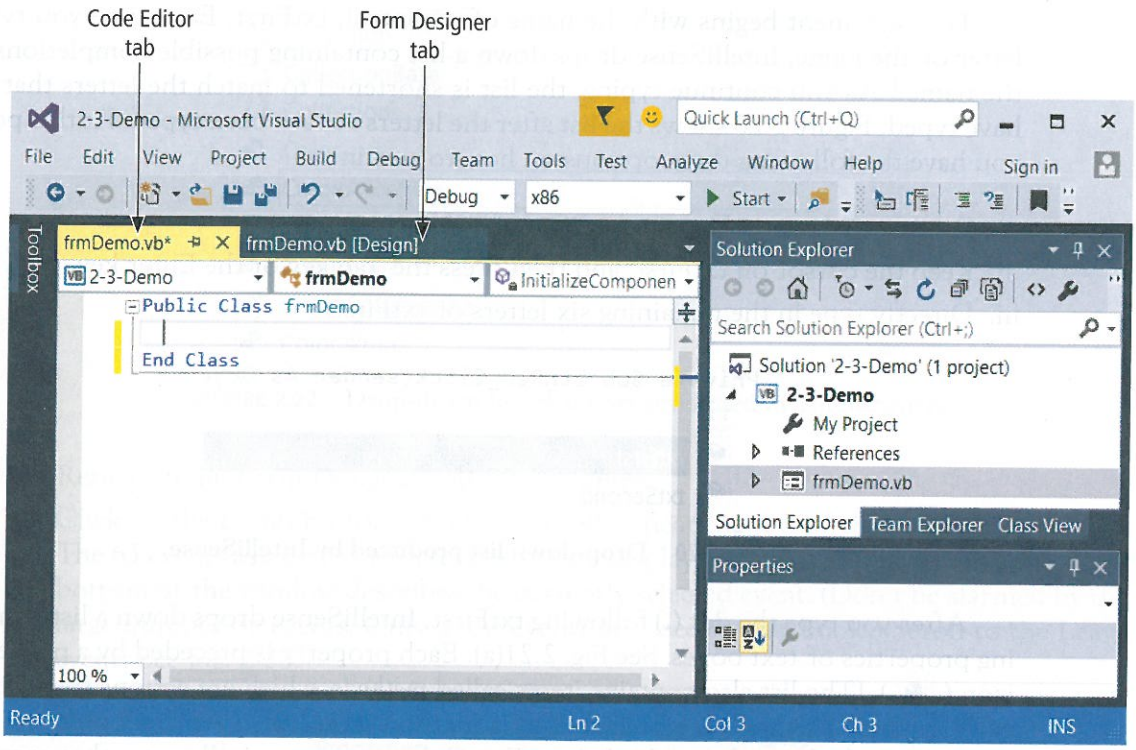


FIGURE 2.19 **The Visual Basic IDE in Code Editor mode.**

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.

Figure 2.19 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 that 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 on *Designer* in the View menu, or by right-clicking the Code Editor and clicking on *View Designer*.)
4. Double-click on the button. The Code Editor reappears, but now the following two lines of code have been added to it 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 an 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 2.20 shows the list after the letters *tx* have been typed. At this point, you have the following three options on how to continue:

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.

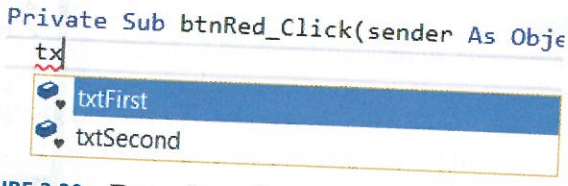


FIGURE 2.20 Drop-down list produced by IntelliSense.

After you type the dot (.) following txtFirst, IntelliSense drops down a list containing properties of text boxes. See Fig. 2.21(a). Each property is preceded by a properties icon ( ). [The list also contains items called *methods*, which are actions the text box can perform. Methods are preceded by a method icon ( ).] At this point, you can scroll down the list and double-click on ForeColor to automatically enter that property. See Fig. 2.21(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. 2.22. You have the option of scrolling to Color.Red and double-clicking on it, or typing Color.Red into the statement.

6. 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 it. 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
```

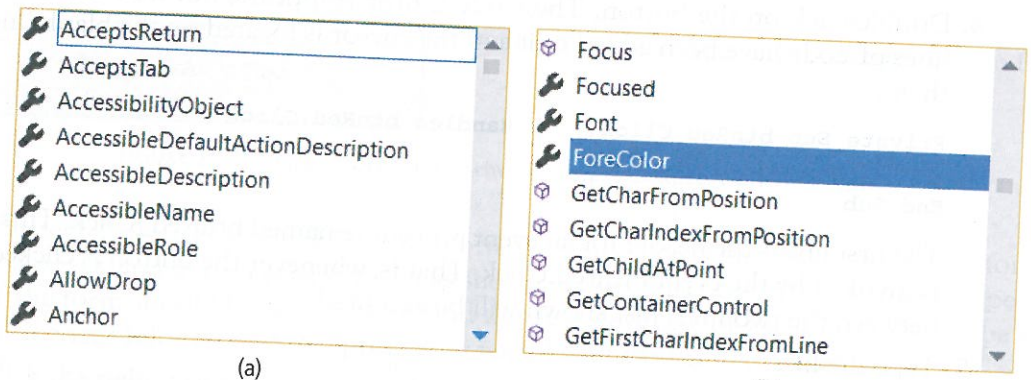


FIGURE 2.21 Drop-down list produced by IntelliSense.

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

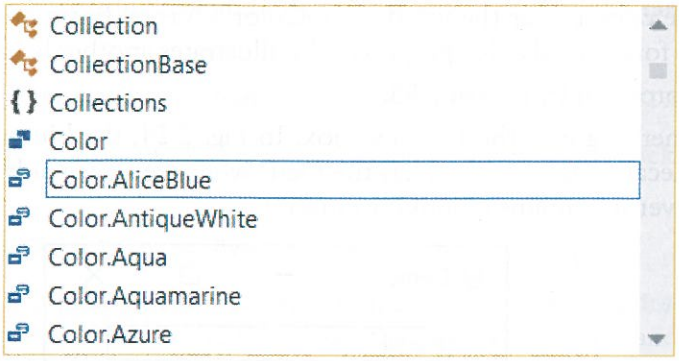


FIGURE 2.22 Drop-down list of colors produced by IntelliSense.

7. Return to the Form Designer and select txtFirst.
8. Click on the *Events* button ( ) in the toolbar near the top of the Properties window. The 63 events associated with text boxes are displayed, and the Description pane at the bottom of the window 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. 2.23.

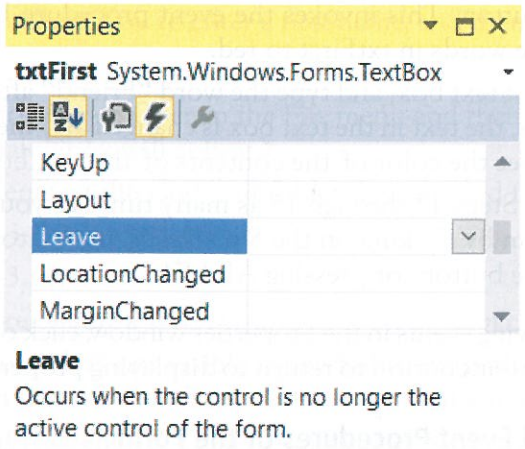


FIGURE 2.23 Events displayed in the Properties window.

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 the 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. 2.24, 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.)

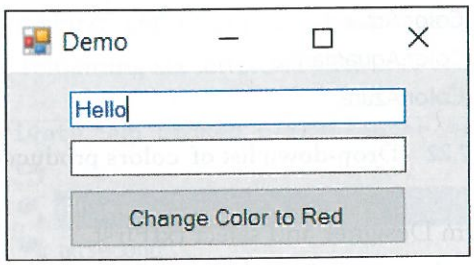


FIGURE 2.24 **Text box containing input.**

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 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 then 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

```
sender As System.Object, 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 make use of "sender" and/or "*e*" in Section 9.4 only when sending output to the printer. You can delete the entire phrase from any other type of program in this book and just leave the blank set of parentheses.

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 in 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, hover over *Open* in the File menu and then click on *Project/Solution*. (An Open Project dialog box will appear.)
2. Navigate to 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 menu.
6. If the file frmPresident.vb is not visible in the Solution Explorer, click on the symbol to the left of 7-2-3 in the Solution Explorer in order to display the file.
7. Double-click on frmPresident.vb. The Form Designer for the program will be revealed and the Solution Explorer window will appear as in Fig. 2.25. (You can click on the *View Code* button to reveal the Code Editor. The *Show All Files* and *Refresh* buttons, which

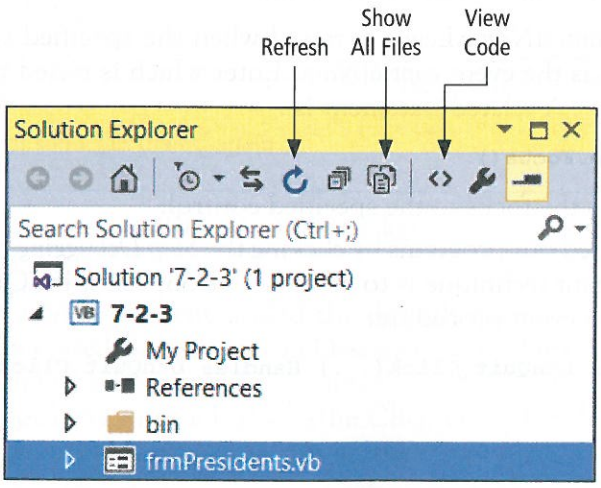


FIGURE 2.25 Solution Explorer window.

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.)

8. Press F5 to run the program.
9. Type in a name (such as James or William), and click the *Display Presidents* button. (See Fig. 2.26.) You can repeat this process as many times as desired.

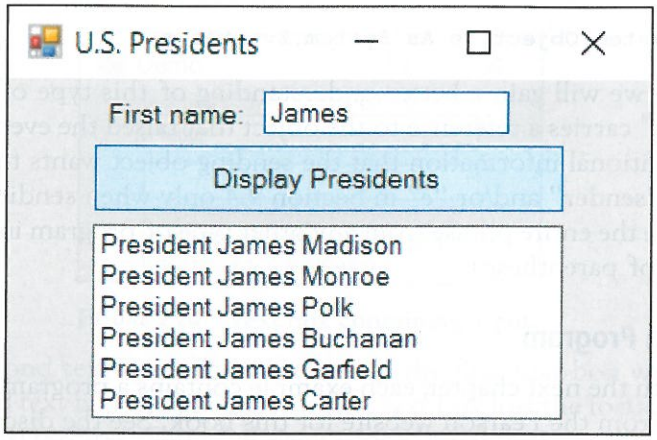


FIGURE 2.26 Possible output for program 7-2-3.

10. End the program.

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 ( ) 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, click on *Options* in the Tools menu to display an Options dialog box, expand the "Text Editor" item in the left pane, expand the "Basic" item, click on "Tabs", enter 2 into the "Indent size:" box, and click on the OK button.
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, the statement might be 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 = "Push 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 = "Push 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 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 2.27 will appear.

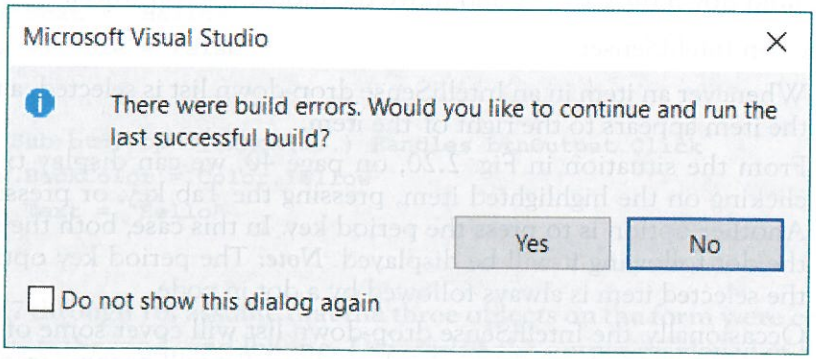


FIGURE 2.27 Error dialog box.

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 on the next page 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.

TABLE 2.2 Some default events.

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

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.

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

```
lstBox.Items.Add(txtBox.Font.Name)
lstBox.Items.Add(txtBox.Font.Bold)
lstBox.Items.Add(txtBox.Font.Size)
```

However, a font's name, style, and size properties cannot be altered in code with statements such as

```
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 is new. The asterisks disappear when the program is saved or run.

Note: When a 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. 2.20, on page 40, 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) 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.

13. While working in the design window, you can obtain information about a control or a keyword by placing the cursor on it and pressing the F1 key. Visual Basic will connect to the Microsoft Website through your browser and display information about the selected item. This feature is called **context-sensitive help**.

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 red 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.

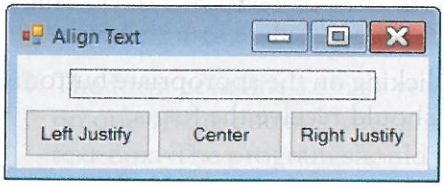
30. Describe the Leave event in your own words.

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.

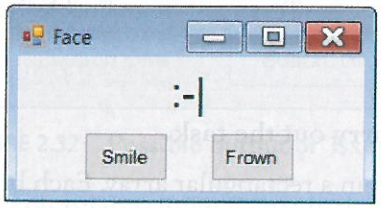
In Exercises 33 through 38, the interface and initial properties are specified. Write a program to carry out the stated task.

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.



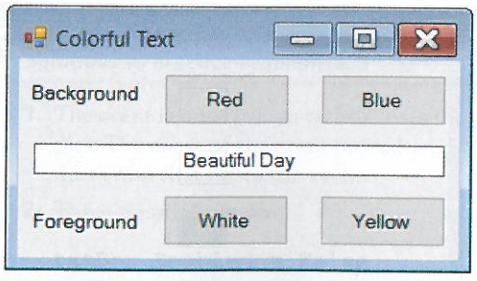
OBJECT	PROPERTY	SETTING
frmAlign	Text	Align Text
txtBox	ReadOnly	True
btnLeft	Text	Left Justify
btnCenter	Text	Center
btnRight	Text	Right Justify

34. When one of the buttons is pressed, the face changes to a smiling face [emoticon :-)] or a frowning face [emoticon :-( ].



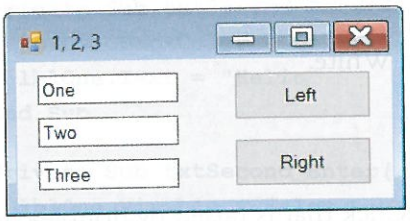
OBJECT	PROPERTY	SETTING
frmFace	Text	Face
lblFace	Font Size	18
	Text	:-|
btnSmile	Text	Smile
btnFrown	Text	Frown

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



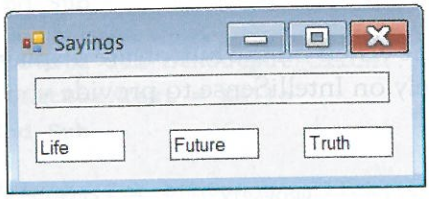
OBJECT	PROPERTY	SETTING
frmColors	Text	Colorful Text
lblBack	Text	Background
btnRed	Text	Red
btnBlue	Text	Blue
txtBox	Text	Beautiful Day
	TextAlign	Center
lblFore	Text	Foreground
btnWhite	Text	White
btnYellow	Text	Yellow

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.



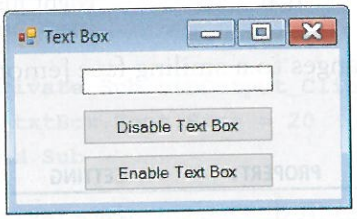
OBJECT	PROPERTY	SETTING
frm123	Text	1, 2, 3
txtOne	Text	One
txtTwo	Text	Two
txtThree	Text	Three
btnLeft	Text	Left
btnRight	Text	Right

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. 2.28.

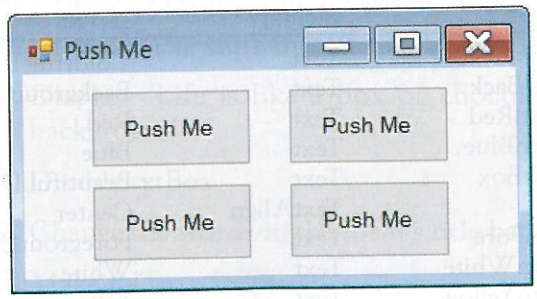


FIGURE 2.28 Form for Exercise 39.

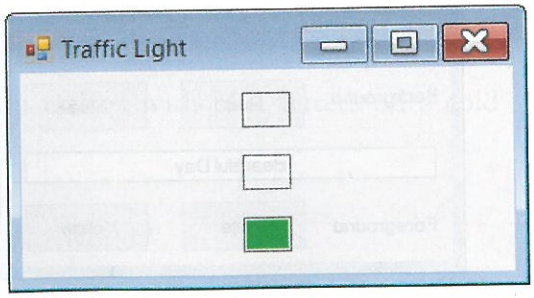


FIGURE 2.29 Form for Exercise 40.

40. Simulate a traffic light with three small square text boxes placed vertically on a form. See Fig. 2.29. 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.

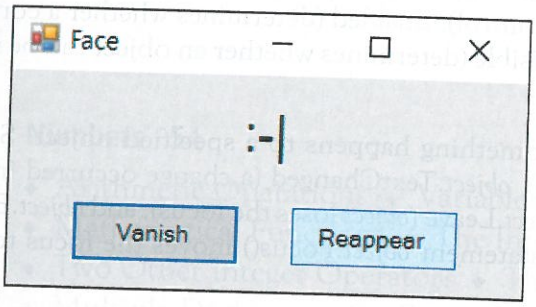


FIGURE 2.30 Possible output of Exercise 41.

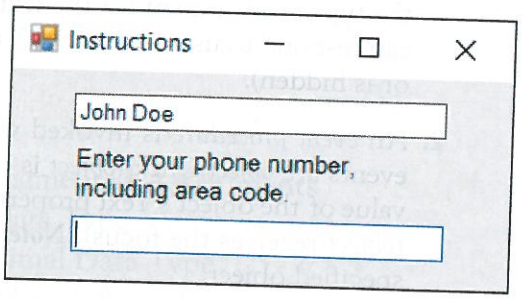


FIGURE 2.31 Possible output of Exercise 42.

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." See Fig 2.31.

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. See Fig 2.32.

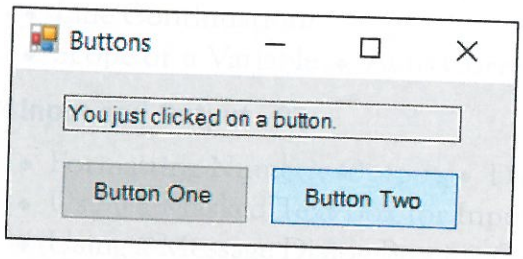


FIGURE 2.32 Possible output of Exercise 43.

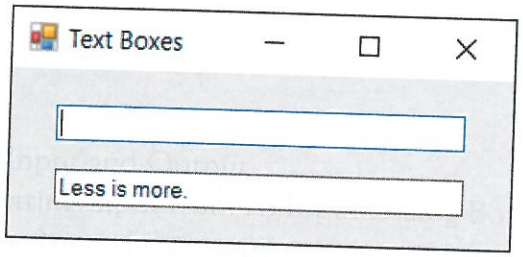


FIGURE 2.33 Possible output of Exercise 44.

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 `txtBox.Clear()` or the statement `txtBox.Text = ""`. See Fig 2.33.

Solutions to Practice Problems 2.3

1. The event is raised when txtBox loses the focus since txtBox.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

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

CHAPTER 2 SUMMARY

1. The Visual Basic Form Designer displays a *form* that can hold a collection of *controls* for which various attributes (called *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.