

Visual Basic .NET

Programming a computer is a lot like teaching a child to tie his shoes. Until you find the correct way of giving the instructions, not much is accomplished. Visual Basic 2010 is a language you can use to tell your computer how to do things; but, like a child, the computer will understand only if you explain things very clearly.

Visual Basic 2010 offers an easy-to-use language to explain some complex tasks. Although it never hurts to have an understanding of what is happening at the lowest levels

A Windows program is quite different from yesteryear's MS-DOS program. A DOS program follows a

relatively strict path from beginning to end. Although this does not necessarily limit the functionality

of the program, it does limit the road the user has to take to get to it. A DOS program is like walking down a hallway; to get to the end you have to walk down the entire hallway, passing any obstacles that you may encounter

Windows, on the other hand, opened up the world of *event-driven programming*. *Events in this context* include clicking a button, resizing a window, or changing an entry in a text box. The code that you write responds to these events. In terms of the hallway analogy: In a Windows program, to get to the end of the hall you just click the end of the hall. The hallway itself can be ignored. If you get to the end and realize that is not where you wanted to be, you can just set off for the new destination without returning to your starting point. The program reacts to your movements and takes the necessary actions to complete your desired tasks.

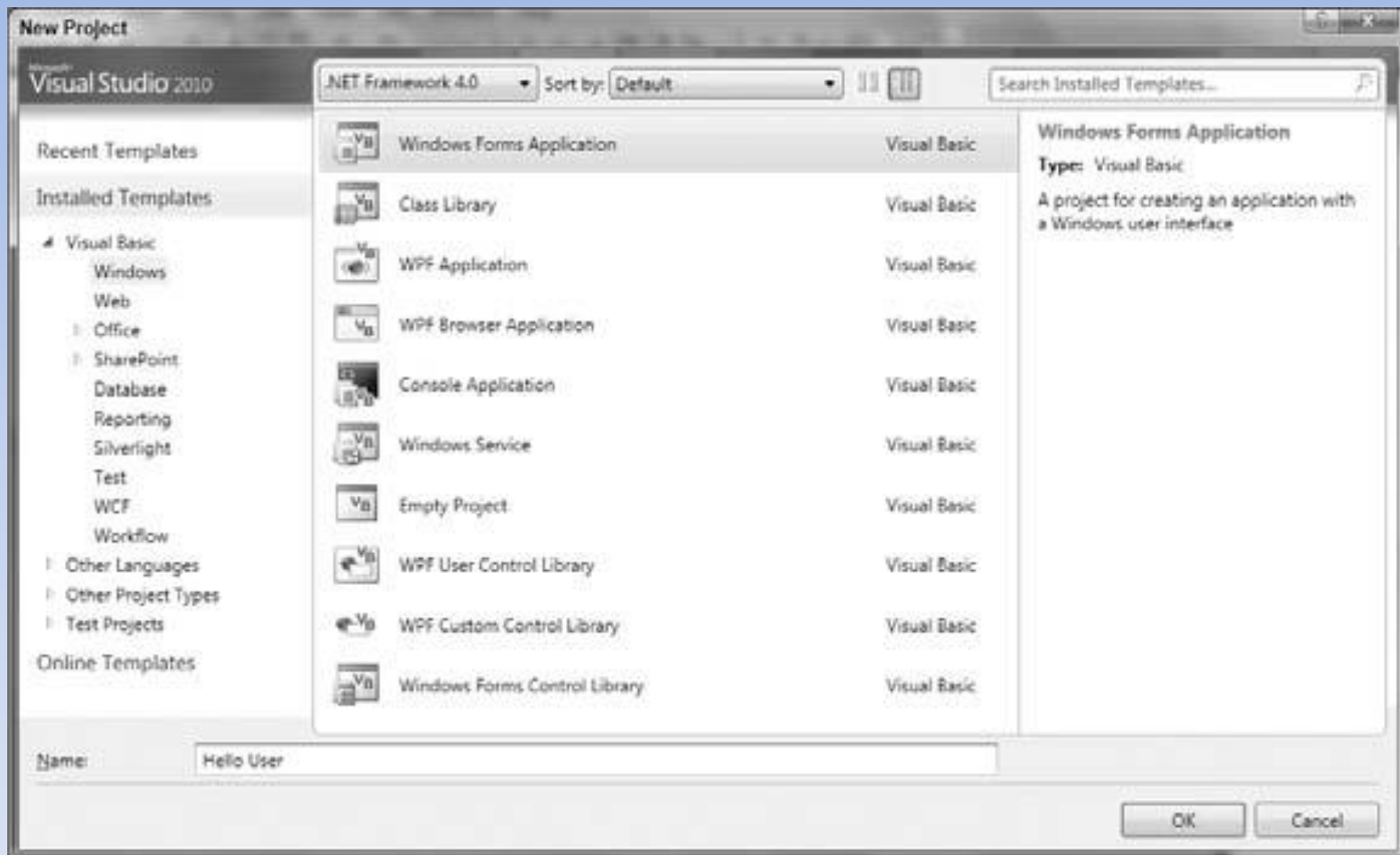
Another big advantage in a Windows program is the *abstraction of the hardware, which means that*

Windows takes care of communicating with the hardware for you. You do not need to know the inner workings of every laser printer on the market just to create output.

The generic routines are referred to as the Windows *application programming interface (API)*, and most of the classes in the .NET Framework take care of communicating with those APIs.

Visual Basic changed the face of Windows programming by removing the complex burden of writing code for the user interface (UI). By allowing programmers to *draw their own UI*, it freed them to concentrate on the business problems they were trying to solve. When the UI is drawn, the programmer can then add the code to react to events.

- 1. Click the New Project button on the toolbar.**
- 2. In the New Project dialog, select Visual Basic in the Installed Templates tree-view box to the left and then select Windows beneath it. The Templates pane on the right will display all of the available templates for the project type chosen. Select the Windows Forms Application template. Finally, type Hello User in the Name text box and click OK. Your New Project dialog should look like below figure.**



Visual Studio 2010 allows you to target your application to a specific version of the Microsoft .NET

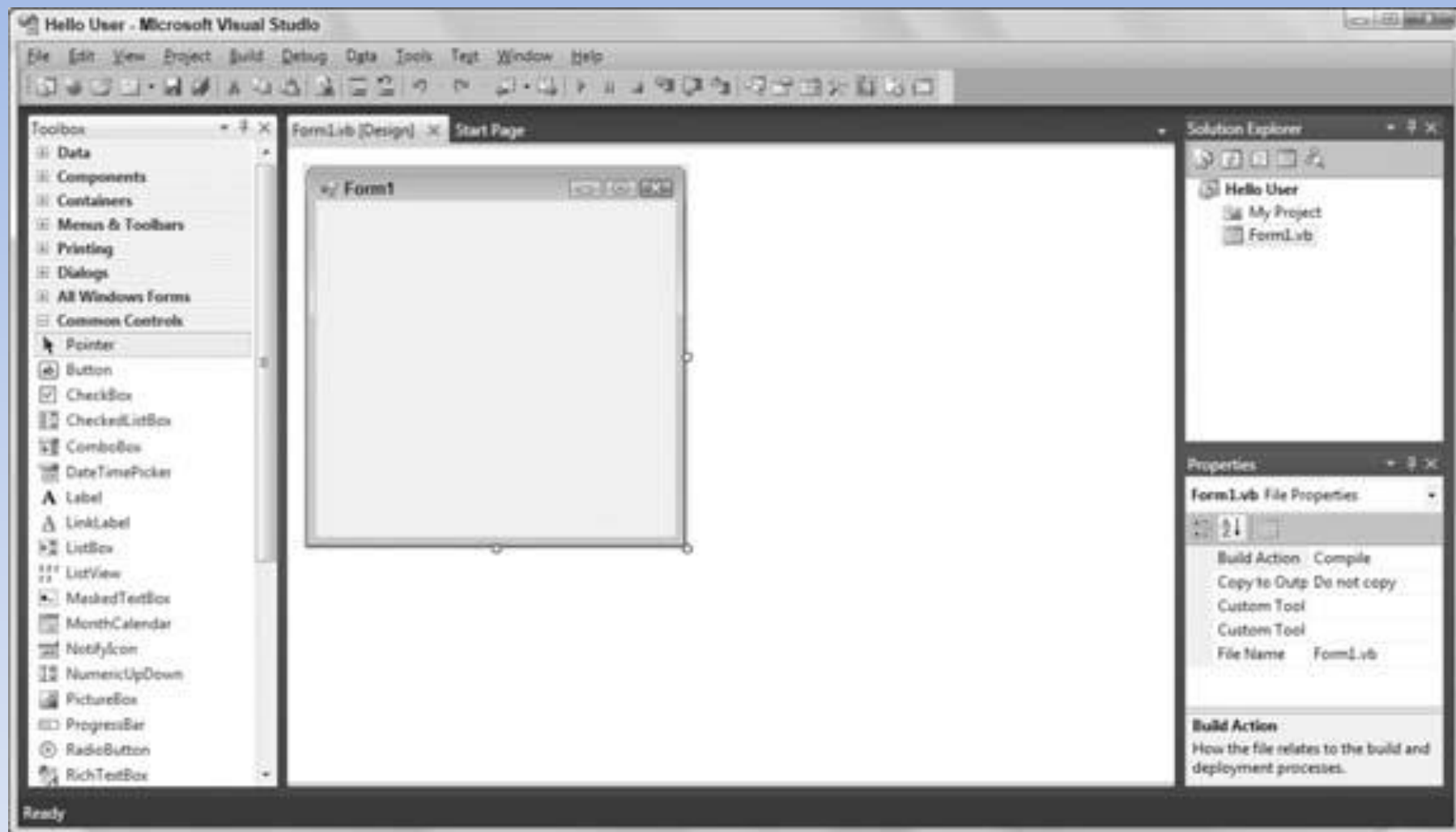
Framework. The combo box at the top of the Templates pane in the New Project dialog has version 4.0 selected, but you can target your application to version 3.5, version 3.0, or even version 2.0 of the .NET Framework.

The IDE will then create an empty Windows application for you. So far, your Hello User program

consists of one blank window, called a Windows Form (or sometimes just a form), with the default

name of

Form1.vb



The most common window

Toolbox: The Toolbox contains reusable controls and components that can be added to your

application. These range from buttons to data connectors to customized controls that you have either purchased or developed.

Design window: The Design window is where a lot of the action takes place. This is where you will draw your user interface on your forms. This window is sometimes referred to as *the Designer*.

Solution Explorer: The Solution Explorer window contains a hierarchical view of your solution.

A solution can contain many projects, whereas a project contains forms, classes, modules, and components that solve a particular problem.

Properties: The Properties window shows what *properties the selected object makes available*. Although you can set these properties in your code, sometimes it is much easier to set them while you are designing your application (for example, drawing the controls on your form). You will notice that the File Name property has the value Form1.vb. This is the physical filename for the form's code and layout information.

Start Application

1. Stop the project if it is still running, because you now want to add some controls to your form.

The simplest way to stop your project is to click the close (X) button in the top-right corner of the form. Alternatively, you can click the blue square on the toolbar (which displays a ToolTip that says “Stop Debugging” if you hover over it with your mouse pointer).

2. Add a Label control to the form. Click Label in the Toolbox, drag it over to the form's Designer

and drop it in the desired location. (You can also place controls on your form by double-clicking the required control in the Toolbox or clicking the control in the Toolbox and then drawing it on the form.)

3. If the Label control you have just drawn is not in the desired location, no problem. When the control is on the form, you can resize it or move it around. Figure 1-11 shows what the control looks like after you place it on the form. To move it, click the control and drag it to the desired location. The label will automatically resize itself to fit the text that you enter in the Text figure property.



4. After drawing a control on the form, you should at least configure its name and the text that it will display. You will see that the Properties window to the right of the Designer has changed to figure. Label1, telling you that you are currently examining the properties for the label. In the Properties window, set your new label's

Text property to **Enter Your Name**. Note that after you press Enter or click on another property, the label on the form has automatically resized itself to fit the text in the Text property. Now set the Name property to **lblName**.

Hello from Visual Basic 2...

Enter Your Name

5. Directly beneath the label, you want to add a text box so that you can enter a name. You are going to repeat the procedure you followed for adding the label, but this time make sure you select the TextBox control from the toolbar. After you have dragged and dropped (or double-clicked) the control into the appropriate position as shown in Figure , use the Properties window to set its Name property to **txtName**. **Notice the sizing** handles on the left and right side of the control. You can use these handles to resize the text box horizontally.



6. In the bottom left corner of the form, add a Button control in exactly the same manner as you added the label and text box. Set its Name property to **btnOK** and its Text property to **&OK**. **Your form should now look similar to the one shown in Figure.** The ampersand (&) is used in the Text property of buttons to figure create a keyboard shortcut (known as a *hot key*). *The letter with* the &sign placed in front of it will become underlined to signal users that they can select that button by pressing the Alt+letter key combination, instead of using the mouse (on some configurations the underline doesn't appear until the user presses Alt). In this particular instance, pressing Alt+O would be the same as clicking the OK button. There is no need to write code to accomplish this.



7. Now add a second Button control to the bottom right corner of the form by dragging the Button control from the Toolbox onto your form. Notice that as you get close to the bottom right of the form, a blue snap line appears, as shown in Figure . This snap line enables you to align this new Button control with the existing Button control on the form. The snap lines assist you in aligning controls to the left, right, top, or bottom of each other, depending on where you are trying to position the new control. The light blue line provides you with a consistent margin between the edge of your control and the edge of the form. Set the Name property to **btnExit** and the Text property to **E&xit**. Your form should look similar to Figure

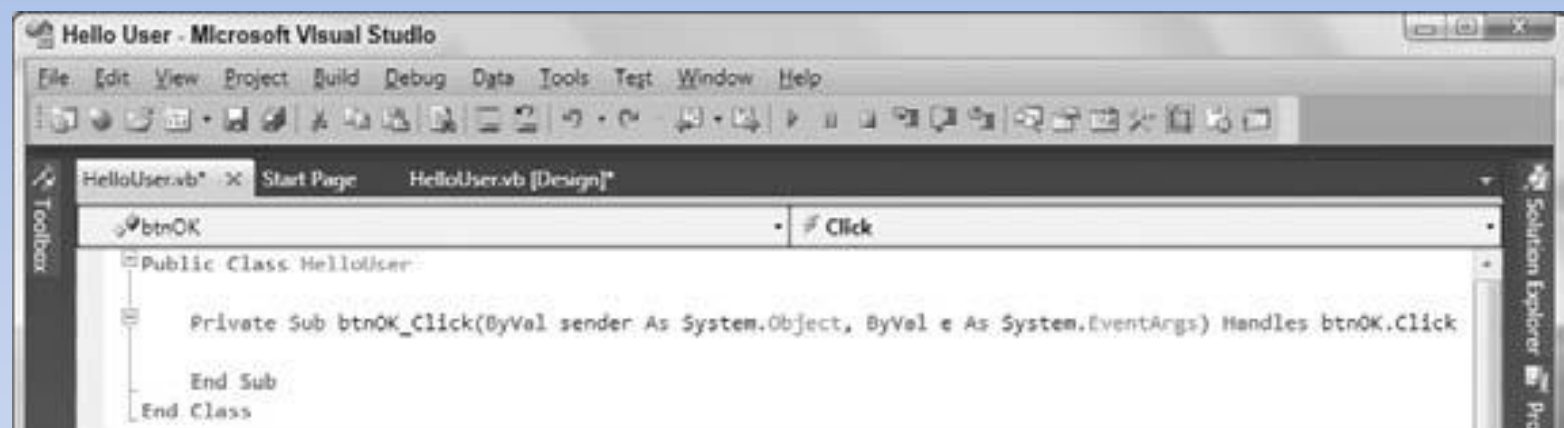
Each name is prefixed with a shorthand identifier describing the type of control it is. This makes it much easier to understand what type of control you are working with as you look through the code. For example, say you had a control called simply Name, without a prefix of lbl or txt. You would not know whether you were working with a text box that accepted a name or with a label that displayed a name.

Common Prefixes

CONTROL	PREFIX
Button	btn
ComboBox	cbo
CheckBox	chk
Label	lbl
ListBox	lst
MainMenu	mnu
RadioButton	rdn
PictureBox	pic
TextBox	txt

Code Editor

Now that you have the HelloUser form defined, you have to add some code to make it actually do something interesting. You have already seen how easy it is to add controls to a form. Providing the functionality behind those on-screen elements is no more difficult. To add the code for a control, you just double-click the control in question. This opens the Code Editor in the main window, shown in Figure .



Note that an additional tab has been created in the main window. Now you have the Design tab and the Code tab, each containing the name of the form you are working on. You draw the controls on

your form in the Design tab, and you write code for your form in the Code tab. One thing to note here is that Visual Studio 2010 has created a separate file for the code. The visual definition and the code behind it exist in separate files:

HelloUser.Designer.vb and HelloUser.vb. This is actually the reason why building applications with Visual Basic 2010 is so slick and easy. Using the design mode you can visually lay out your application; then, using Code view, you add just the bits of code to implement your desired functionality.

Adding Code

1. To begin adding the necessary code, click the Design tab to show the form again. Then doubleclick

the OK button. The code window will open with the following code.

This is the shell of

the button's

Click event and the place where you enter the code that you want to run when you

click the button. This code is known as an *event handler, sometimes also referred to as an event*

procedure:

```
Private Sub btnOK_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles btnOK.Click  
End Sub
```

As a result of the typographic constraints in publishing, it is not possible to put the Sub declaration on one line. Visual Basic 2010 allows you to break up lines of code by using the underscore character (_) to signify a line continuation. The space before the underscore is required. Any whitespace preceding the code on the following line is ignored.

Sub is an example of a *keyword*. *In programming terms, a keyword is a special word that is used to tell Visual Basic 2010 to do something special.* In this case, it tells Visual Basic 2010 that this a *subroutine, a procedure that does not return a value.* Anything that you type between the lines Private Sub and End Sub will make up the event procedure for the OK button.

2. Now add the bolded code to the procedure:

```
Private Sub btnOK_Click(ByVal sender As  
System.Object, _  
ByVal e As System.EventArgs) Handles btnOK.Click  
'Display a message box greeting to the user  
MessageBox.Show("Hello, " & txtName.Text & _  
"! Welcome to Visual Basic 2010.", _  
"Hello User Message")  
End Sub
```

3. After you have added the preceding code, go back to the Design tab and double-click the Exit button. Add the following bolded code to the btnExit_Click event procedure:

```
Private Sub btnExit_Click(ByVal sender As  
System.Object, _  
ByVal e As System.EventArgs) Handles btnExit.Click  
'End the program and close the form  
Me.Close()  
End Sub
```

4. Now that the code is finished, the moment of truth has arrived and you can see your creation.

First, however, save your work by using File

⇒ Save All from the menu or by clicking the Save All

button on the toolbar. The Save Project dialog is displayed, as shown in Figure 1-17, prompting you for a name and location for saving the project.

Save Project

Name: Hello User

Location: C:\Users\thearon\documents\visual studio 10\Projects Browse...

Solution Name: Hello User ☐ Create directory for solution

Save Cancel

By default, a project is saved in a folder with the project name; in this case Hello User. Since this is the only project in the solution, there is no need to create a separate folder for the solution, which contains the same name as the project, thus the “Create directory for solution” check box is unselected.

5. Now click the Start button on the toolbar. At this point Visual

Studio 2010 will *compile the code*. *Compiling is the activity of* taking the Visual Basic 2010 source code that you have written and translating it into a form that the computer understands. After the compilation is complete, Visual Studio 2010 *runs (also* known as *executes) the program, and you'll be able to see the results.*

Hello from Visual Basic 2010

Enter Your Name

Wendy

OK

Exit

Hello User Message

Hello, Wendy! Welcome to Visual Basic 2010.

OK

How it works

The code that you added to the Click event for the OK button will take the name that was entered in the text box and use it as part of the message that was displayed in Figure .

The first line of text you entered in this procedure ('Display a message box greeting to the user)is actually

a comment, text that is meant to be read by the human programmer who is writing or maintaining the

code, not by the computer. Comments in Visual Basic 2010 begin with a single quote ('), and everything

following on that line is considered a comment and ignored by the compiler.

The `MessageBox.Show` method displays a message box that accepts various parameters. As used in your code, you have passed the string text to be displayed in the message box. This is accomplished through the *concatenation of string constants defined by text enclosed in quotes*. Concatenation of strings into one long string is performed through the use of the ampersand (&) character.

the entire code fragment in the Code Editor without having to scroll the Code Editor window to the right to view the entire line of code.

The code that follows concatenates a string constant of “Hello,” followed by the value contained in the Text property of the txtName text box control, followed by a string constant of “! Welcome to Visual Basic 2010.” The second parameter passed to the MessageBox.Show method is the caption to be used in the title bar of the Message Box dialog.

Finally, the underscore (`_`) character used at the end of the lines in the following code enables you to split your code onto separate lines. This tells the compiler that the rest of the code for the parameter is

continued on the next line. This is very useful when building long strings because it enables you to view

```
Private Sub btnOK_Click(ByVal sender As  
System.Object, _  
ByVal e As System.EventArgs) Handles btnOK.Click  
    'Display a message box greeting to the user  
    MessageBox.Show("Hello, " & txtName.Text & _  
        "! Welcome to Visual Basic 2010.", _  
        "Hello User Message")  
End Sub
```

The next procedure that you added code for was the Exit button's Click event. Here you simply enter the code:

Me.Close().The Me keyword refers to the form itself. The Close method of the form closes the form and releases all resources associated with it, thus ending the program:

```
Private Sub btnExit_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles btnExit.Click  
    'End the program and close the form  
    Me.Close()  
End Sub
```