Chapter 3

Working with Toolbox Controls

After completing this chapter, you will be able to:

- Use *TextBox* and *Button* controls to create a Hello World program.
- Use the *DateTimePicker* control to display your birth date.
- Use *CheckBox*, *RadioButton*, and *ListBox* controls to process user input.
- Use the *LinkLabel* control and the *Process.Start* method to display a Web page by using your system's default browser.

As you learned in earlier chapters, Microsoft Visual Studio 2010 controls are the graphical tools you use to build the user interface of a Microsoft Visual Basic program. Controls are located in the development environment's Toolbox, and you use them to create objects on a form with a simple series of mouse clicks and dragging motions.

Windows Forms controls are specifically designed for building Windows applications, and you'll find them organized on the All Windows Forms tab of the Toolbox, although many of the controls are also accessible on tabs such as Common Controls, Containers, and Printing. (You used a few of these controls in the previous chapter.) Among the Common Controls, there are few changes between Visual Basic 2008 and Visual Basic 2010, so if you're really experienced with the last version of Visual Basic, you may simply want to move on to the database and Web application chapters of this book (Part IV), or the detailed material about programming techniques in Parts II and III. However, for most casual Visual Basic users, there is a lot still to learn about the language's extensive collection of Windows Forms Toolbox controls, and we'll work with several of them here.

In this chapter, you'll learn how to display information in a text box; work with date and time information on your system; process user input with *CheckBox*, *RadioButton*, and *ListBox* controls; and display a Web page within a Visual Basic program. The exercises in this chapter will help you design your own Visual Basic applications and will teach you more about objects, properties, and program code. If you are new to Visual Studio and Visual Basic, this chapter will be especially useful.

The Basic Use of Controls: The Hello World Program

A great tradition in introductory programming books is the Hello World program, which demonstrates how the simplest utility can be built and run in a given programming language. In the days of character-based programming, Hello World was usually a two-line or three-line program typed in a program editor and assembled with a stand-alone compiler.

With the advent of complex operating systems and graphical programming tools, however, the typical Hello World has grown into a more sophisticated program containing dozens of lines and requiring several programming tools for its construction. Fortunately, creating a Hello World program is still quite simple with Visual Studio 2010 and Visual Basic. You can construct a complete user interface by creating two objects, setting two properties, and entering one line of code. Give it a try.

Create a Hello World program

1. Start Visual Studio 2010 if it isn't already open.
2. On the File menu, click New Project.

   Visual Studio displays the New Project dialog box, which prompts you for the name of your project and for the template that you want to use.

Note Use the following instructions each time you want to create a new project on your hard disk.

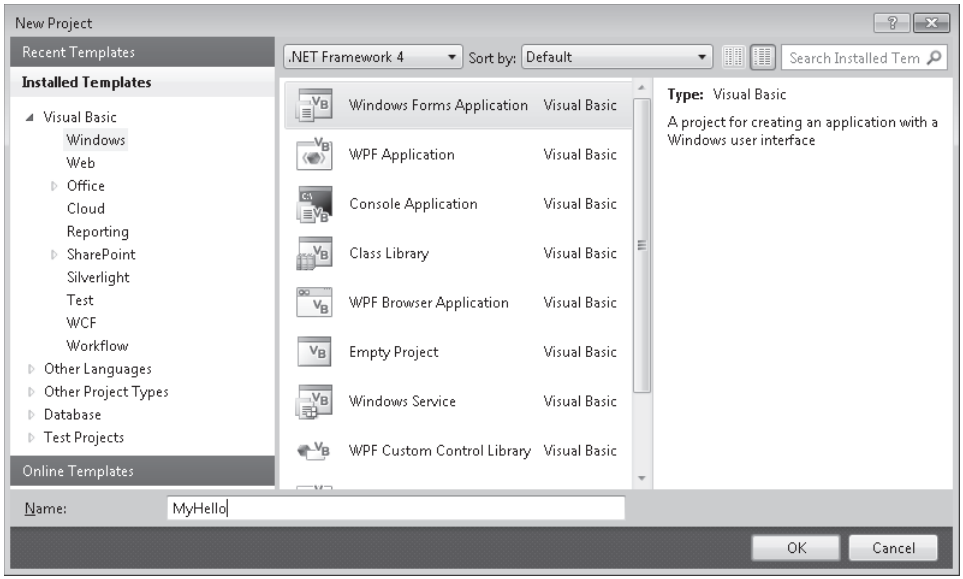
3. Ensure that the Visual Basic Windows category is selected on the left side of the dialog box, and that Windows Forms Application template is also selected in the middle of the dialog box.

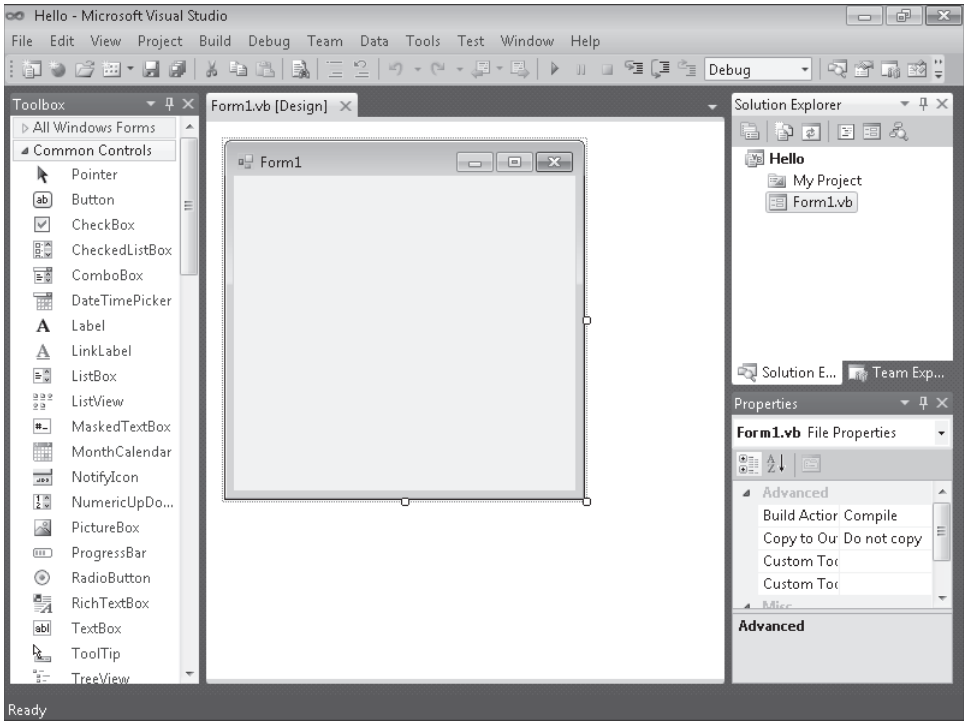
   These selections indicate that you'll be building a stand-alone Visual Basic application that will run under Windows.
4. Remove the default project name (WindowsApplication1) from the Name text box, and then type **MyHello**.

Note Throughout this book, I ask you to create sample projects with the "My" prefix, to distinguish your own work from the practice files I include on the companion CD-ROM. However, I'll usually show projects in the Solution Explorer without the "My" prefix (because I've built the projects without it).

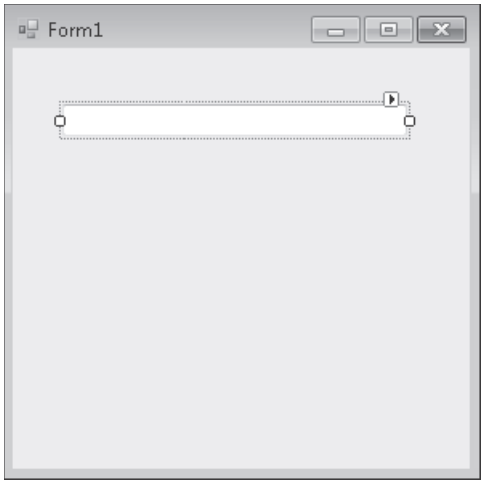
   The New Project dialog box now looks like the screen shot at the top of page 69. If you are using Visual Basic 2010 Express, you will just see a Visual Basic category on the left.
5. Click OK to create your new project.

   The new project is created, and a blank form appears in the Designer, as shown in the screen shot on the bottom of page 69. The two controls you'll use in this exercise, *Button* and *TextBox*, are visible in the Toolbox, which appears in the screen shot as a docked window. If your programming tools are configured differently, take a few moments to organize them, as shown in the screen shot. (Chapter 1, "Exploring the Visual Studio Integrated Development Environment," describes how to configure the IDE if you need a refresher course.)


New Project
Recent Templates
Installed Templates
Visual Basic
Windows
Web
Office
Cloud
Reporting
SharePoint
Silverlight
Test
WCF
Workflow
Other Languages
Other Project Types
Database
Test Projects
Online Templates
.NET Framework 4
Sort by: Default
Search Installed Tem
Windows Forms Application Visual Basic
WPF Application Visual Basic
Console Application Visual Basic
Class Library Visual Basic
WPF Browser Application Visual Basic
Empty Project Visual Basic
Windows Service Visual Basic
WPF Custom Control Library Visual Basic
Type: Visual Basic
A project for creating an application with a Windows user interface
Name: MyHello
OK
Cancel


Hello - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Test Window Help
Debug
Toolbox
All Windows Forms
Common Controls
Pointer
Button
CheckBox
CheckedListBox
ComboBox
DateTimePicker
Label
LinkLabel
ListBox
ListView
MaskedTextBox
MonthCalendar
NotifyIcon
NumericUpDo...
PictureBox
ProgressBar
RadioButton
RichTextBox
TextBox
ToolTip
TreeView
Form1.vb [Design]
Form1
Solution Explorer
Hello
My Project
Form1.vb
Solution E...
Team Exp...
Properties
Form1.vb File Properties
Advanced
Build Actior Compile
Copy to Ou Do not copy
Custom Toc
Custom Toc
Advanced
Ready

6. Click the *TextBox* control on the Common Controls tab of the Toolbox.
7. Draw a text box similar to this:

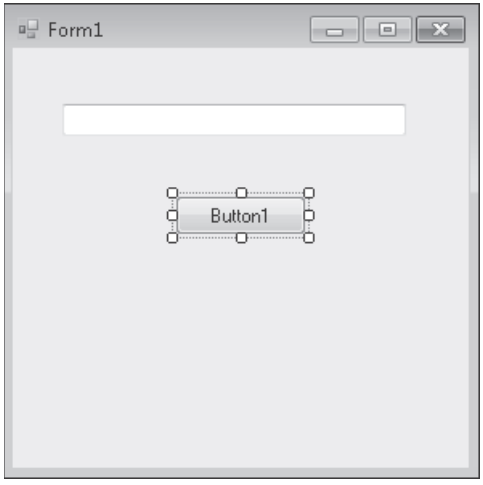


Text boxes are used to display text on a form or to get user input while a program is running. How a text box works depends on how you set its properties and how you reference the text box in the program code. In this program, a text box object will be used to display the message "Hello, world!" when you click a button object on the form.

You'll add a button to the form now.

8. Click the *Button* control in the Toolbox.
9. Draw a button below the text box on the form.

Your form looks something like this:



As you learned in Chapter 2, "Writing Your First Program," buttons are used to get the most basic input from a user. When a user clicks a button, he or she is requesting that the program perform a specific action immediately. In Visual Basic terms, the user is using the button to create an *event* that needs to be processed in the program. Typical buttons in a program are the OK button, which a user clicks to accept a list of options and to indicate that he or she is ready to proceed; the Cancel button, which a user clicks to discard a list of options; and the Quit button, which a user clicks to exit the program. In each case, you should use these buttons in the standard way so that they work as expected when the user clicks them. A button's characteristics (like those of all objects) can be modified with property settings and references to the object in program code.

10. Set the following property for the button object by using the Properties window:

Object	Property	Setting
Button1	Text	"OK"

For more information about setting properties and reading them in tables, see the section entitled "The Properties Window" in Chapter 1.

11. Double-click the OK button, and type the following program statement between the *Private Sub Button1_Click* and *End Sub* statements in the Code Editor:

```
TextBox1.Text = "Hello, world!"
```

Note As you type statements, Visual Studio displays a list box containing all valid items that match your text. After you type the *TextBox1* object name and a period, Visual Studio displays a list box containing all the valid properties and methods for text box objects, to jog your memory if you've forgotten the complete list. This list box is called Microsoft IntelliSense and can be very helpful when you are writing code. If you click an item in the list box, you will typically get a tooltip that provides a short description of the selected item. You can add the property from the list to your code by double-clicking it or by using the arrow keys to select it and then pressing TAB. You can also continue typing to enter the property yourself. (I usually just keep typing, unless I'm exploring new features.)

The statement you've entered changes the *Text* property of the text box to "Hello, world!" when the user clicks the button at run time. (The equal sign (=) assigns everything between the quotation marks to the *Text* property of the *TextBox1* object.) This example changes a property at run time—one of the most common uses of program code in a Visual Basic program.

Now you're ready to run the Hello program.

Run the Hello program

Tip The complete Hello program is located in the C:\Vb10sbs\Chap03\Hello folder.

1. Click the Start Debugging button on the Standard toolbar.

   The Hello program compiles and, after a few seconds, runs in the Visual Studio IDE.

2. Click OK.

   The program displays the greeting "Hello, world!" in the text box, as shown here:

   When you clicked the OK button, the program code changed the *Text* property of the empty *TextBox1* text box to "Hello, world!" and displayed this text in the box. If you didn't get this result, repeat the steps in the previous section, and build the program again. You might have set a property incorrectly or made a typing mistake in the program code. (Syntax errors appear with a jagged underline in the Code Editor.)

3. Click the Close button in the upper-right corner of the Hello World program window to stop the program.

Note To stop a program running in Visual Studio, you can also click the Stop Debugging button on the Standard toolbar to close the program.

4. Click the Save All button on the Standard toolbar to save your new project to disk.

   Visual Studio now prompts you for a name and a location for the project.

5. Click the Browse button.

   The Project Location dialog box opens. You use this dialog box to specify the location of your project and to create new folders for your projects if necessary. Although you

can save your projects in any location (the Documents\Visual Studio 2010\Projects folder is a common location), in this book I instruct you to save your projects in the C:\Vb10sbs folder, the default location for your *Step by Step* practice files. If you ever want to remove all the files associated with this programming course, you'll know just where the files are, and you'll be able to remove them easily by deleting the entire folder.

6. Browse to the C:\Vb10sbs\Chap03 folder.
7. Click the Select Folder or Open button to open the folder you specified.
8. Clear the check mark from the Create Directory For Solution check box if it is selected.

   Because this solution contains only one project (which is the case for most of the solutions in this book), you don't need to create a separate root folder to hold the solution files for the project. (However, you can create an extra folder if you want.)
9. Click Save to save the project and its files.

Congratulations—you've joined the ranks of programmers who've written a Hello World program. Now let's try another control.

Using the *DateTimePicker* Control

Some Visual Basic controls display information, and others gather information from the user or process data behind the scenes. In this exercise, you'll work with the *DateTimePicker* control, which prompts the user for a date or time by using a graphical calendar with scroll arrows. Although your use of the control will be rudimentary at this point, experimenting with *DateTimePicker* will give you an idea of how much Visual Basic controls can do for you automatically and how you process the information that comes from them.

The Birthday Program

The Birthday program uses a *DateTimePicker* control and a *Button* control to prompt the user for the date of his or her birthday. It then displays that information by using a message box. Give it a try now.

Build the Birthday program

1. On the File menu, click Close Project to close the MyHello project.

   The files associated with the Hello World program close.
2. On the File menu, click New Project.

   The New Project dialog box opens.
3. Create a new Visual Basic Windows Forms Application project named **MyBirthday**.

   The new project is created, and a blank form appears in the Designer.

4. Click the *DateTimePicker* control in the Toolbox.
5. Draw a date/time picker object near the top of the form, as shown in the following screen shot:

The date/time picker object by default displays the current date, but you can adjust the displayed date by changing the object's *Value* property. Displaying the date is a handy design guide—it lets you size the date/time picker object appropriately when you're creating it.

6. Click the *Button* control in the Toolbox, and then add a button object below the date/time picker.

You'll use this button to display your birth date and to verify that the date/time picker works correctly.

7. In the Properties window, change the *Text* property of the button object to **Show My Birthday**.

Now you'll add a few lines of program code to a procedure associated with the button object. This is an event procedure because it runs when an event, such as a mouse click, occurs, or *fires,* in the object.

8. Double-click the button object on the form to display its default event procedure, and then type the following program statements between the *Private Sub* and *End Sub* statements in the *Button1_Click* event procedure:

```
MsgBox("Your birth date was " & DateTimePicker1.Text)
MsgBox("Day of the year: " & _
  DateTimePicker1.Value.DayOfYear.ToString())
```

These program statements display two message boxes (small dialog boxes) with information from the date/time picker object. The first line uses the *Text* property of the date/time picker to display the birth date information that you select when using the object at run time. The *MsgBox* function displays the string value "Your birth date was" in addition to the textual value held in the date/time picker's *Text* property. These two pieces of information are joined together by the string concatenation operator (&). You'll learn more about the *MsgBox* function and the string concatenation operator in Chapter 5, "Visual Basic Variables and Formulas, and the .NET Framework."

The second and third lines collectively form one program statement and have been broken by the line continuation character (_) because the statement was a bit too long to print in this book.

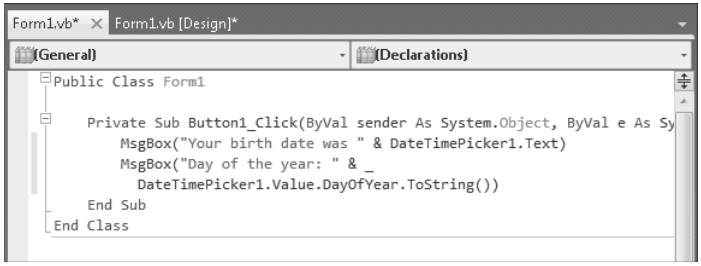
Program lines can be more than 65,000 characters long in the Visual Studio Code Editor, but it's usually easiest to work with lines of 80 or fewer characters. You can divide long program statements among multiple lines by using a space and a line continuation character (_) at the end of each line in the statement except for the last line. (You cannot use a line continuation character to break a string that's in quotation marks, however.) I use the line continuation character in this exercise to break the second line of code into two parts.

Note Starting in Visual Basic 2010, the line continuation character (_) is optional. There are a few instances where the line continuation character is needed, but they are rare. In this book, I still use line continuation characters to make it clear where there are long lines, but you don't have to include them.

The statement DateTimePicker1.Value.DayOfYear.ToString() uses the date/time picker object to calculate the day of the year in which you were born, counting from January 1. This is accomplished by the *DayOfYear* property and the *ToString* method, which converts the numeric result of the date calculation to a textual value that's more easily displayed by the *MsgBox* function.

Methods are special statements that perform an action or a service for a particular object, such as converting a number to a string or adding items to a list box. Methods differ from properties, which contain a value, and event procedures, which execute when a user manipulates an object. Methods can also be shared among objects, so when you learn how to use a particular method, you'll often be able to apply it to several circumstances. We'll discuss several important methods as you work through this book.

After you enter the code for the *Button1_Click* event procedure, the Code Editor looks similar to this:



9. Click the Save All button to save your changes to disk, and specify C:\Vb10sbs\Chap03 as the folder location.

   Now you're ready to run the Birthday program.

Run the Birthday program

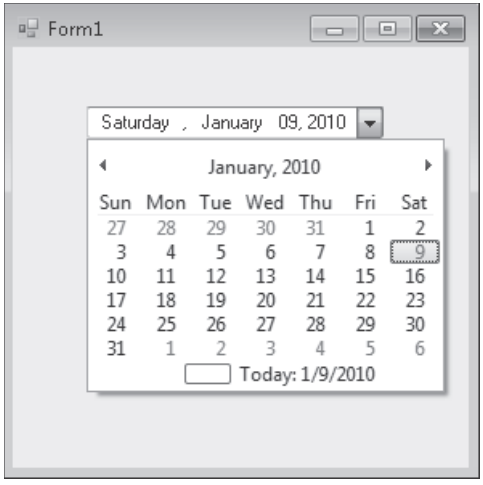
Tip The complete Birthday program is located in the C:\Vb10sbs\Chap03\Birthday folder.

1. Click the Start Debugging button on the Standard toolbar.

   The Birthday program starts to run in the IDE. The current date is displayed in the date/time picker.

2. Click the arrow in the date/time picker to display the object in Calendar view.

   Your form looks like the following screen shot, but with a different date.



3. Click the Left scroll arrow to look at previous months on the calendar.

Notice that the text box portion of the object also changes as you scroll the date. The "today" value at the bottom of the calendar doesn't change, however.

Although you can scroll all the way back to your exact birthday, you might not have the patience to scroll month by month. To move to your birth year faster, select the year value in the date/time picker text box and enter a new year.

4. Select the four-digit year in the date/time picker text box.

   When you select the date, the date/time picker closes.

5. Type your birth year in place of the year that's currently selected, and then click the arrow again.

   The calendar reappears in the year of your birth.

6. Click the scroll arrow again to locate the month in which you were born, and then click the exact day on which you were born.

   If you didn't know the day of the week on which you were born, now you can find out!

   When you select the final date, the date/time picker closes, and your birth date is displayed in the text box. You can click the button object to see how this information is made available to other objects on your form.

7. Click the Show My Birthday button.

   Visual Basic executes your program code and displays a message box containing the day and date of your birth. Notice how the two dates shown in the two boxes match:

8. Click OK in the message box.

   A second message box appears, indicating the day of the year on which you were born—everything seems to work! You'll find this control to be quite capable—not only

does it remember the new date or time information that you enter, but it also keeps track of the current date and time, and it can display this date and time information in a variety of useful formats.

Note To configure the date/time picker object to display times instead of dates, set the object's *Format* property to Time.

9. Click OK to close the message box, and then click the Close button on the form.

   You're finished using the *DateTimePicker* control for now.

Controls for Gathering Input

Visual Basic provides several mechanisms for gathering input in a program. *Text boxes* accept typed input, *menus* present commands that can be clicked or chosen with the keyboard, and *dialog boxes* offer a variety of elements that can be chosen individually or selected in a group. In the next few exercises, you'll learn how to use three important controls that help you gather input in several different situations. You'll learn about the *CheckBox*, *RadioButton*, *GroupBox*, *PictureBox, ListBox* controls. You'll explore each of these objects as you use a Visual Basic program called Input Controls, which is the user interface for a simple, graphics-based ordering system. As you run the program, you'll get some hands-on experience with the input objects. In the next chapter, I'll discuss how these objects can be used along with menus in a full-fledged program.

As a simple experiment, try using the *CheckBox* control now to see how user input is processed on a form and in program code.

Experiment with the *CheckBox* control

1. On the File menu, click Close Project to close the Birthday project.
2. On the File menu, click New Project.

   The New Project dialog box opens.
3. Create a new Visual Basic Windows Forms Application project named **MyCheckBox**.

   The new project is created, and a blank form appears in the Designer.
4. Click the *CheckBox* control in the Toolbox.
5. Draw two check box objects on the form, one above the other.

   Check boxes appear as objects on your form just as other objects do. You'll have to click the *CheckBox* control in the Toolbox a second time for the second check box.
6. Using the *PictureBox* control, draw two square picture box objects beneath the two check boxes.

7. Select the first *PictureBox* control named PictureBox1.
8. Click the *Image* property in the Properties window, and then click the ellipsis button in the second column.

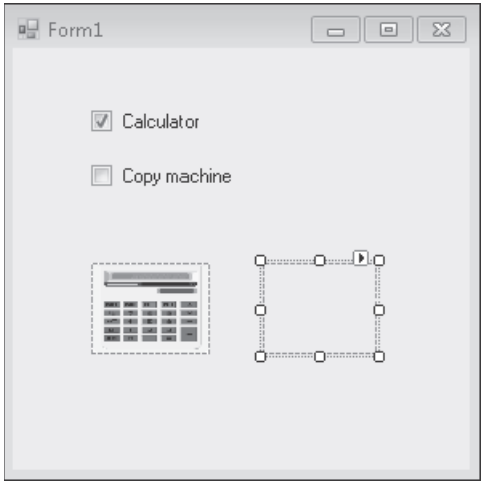
   The Select Resource dialog box appears.
9. Click the Local Resource radio button, and then click the Import button.
10. In the Open dialog box, navigate to the C:\Vb10sbs\Chap03 folder.
11. Select Calcultr.bmp, and then click Open.
12. Click OK in the Select Resource dialog box.

    The calculator appears in the *PictureBox*.
13. Set the *SizeMode* property on the *PictureBox* to StretchImage.
14. Set the following properties for the check box and *PictureBox2* objects:

Object	Property	Setting
CheckBox1	*Checked* *Text*	True "Calculator"
CheckBox2	*Text*	"Copy machine"
PictureBox2	*SizeMode*	StretchImage

In these steps, you'll use the check boxes to display and hide images of a calculator and a copy machine. The *Text* property of the check box object determines the contents of the check box label in the user interface. With the *Checked* property, you can set a default value for the check box. Setting *Checked* to True places a check mark in the box, and setting *Checked* to False (the default setting) removes the check mark. I use the *SizeMode* properties in the picture boxes to size the images so that they stretch to fit in the picture box.

Your form looks something like this:



15. Double-click the first check box object to open the *CheckBox1_CheckedChanged* event procedure in the Code Editor, and then enter the following program code:

```
If CheckBox1.CheckState = 1 Then
    PictureBox1.Image = System.Drawing.Image.FromFile _
      ("c:\vb10sbs\chap03\calcultr.bmp")
    PictureBox1.Visible = True
Else
    PictureBox1.Visible = False
End If
```

The *CheckBox1_CheckedChanged* event procedure runs only if the user clicks in the first check box object. The event procedure uses an If … Then decision structure (described in Chapter 6, "Using Decision Structures") to confirm the current status, or *state*, of the first check box, and it displays a calculator picture from the C:\Vb10sbs\Chap03 folder if a check mark is in the box. The *CheckState* property holds a value of 1 if there's a check mark present and 0 if there's no check mark present. (You can also use the CheckState.Checked enumeration, which appears in IntelliSense when you type, as an alternative to setting the value to 1.) I use the *Visible* property to display the picture if a check mark is present or to hide the picture if a check mark isn't present. Notice that I wrapped the long line that loads the image into the picture box object by using the line continuation character (_).

16. Click the View Designer button in Solution Explorer to display the form again, double-click the second check box, and then add the following code to the *CheckBox2_CheckedChanged* event procedure:

```
If CheckBox2.CheckState = 1 Then
    PictureBox2.Image = System.Drawing.Image.FromFile _
      ("c:\vb10sbs\chap03\copymach.bmp")
    PictureBox2.Visible = True
Else
    PictureBox2.Visible = False
End If
```

This event procedure is almost identical to the one that you just entered; only the names of the image (Copymach.bmp), the check box object (*CheckBox2*), and the picture box object (*PictureBox2*) are different.

17. Click the Save All button on the Standard toolbar to save your changes, specifying the C:\Vb10sbs\Chap03 folder as the location.

Run the CheckBox program

Tip The complete CheckBox program is located in the C:\Vb10sbs\Chap03\Checkbox folder.

1. Click the Start Debugging button on the Standard toolbar.

   Visual Basic runs the program in the IDE. The calculator image appears in a picture box on the form, and the first check box contains a check mark.

2. Select the Copy Machine check box.

   Visual Basic displays the copy machine image, as shown here:

3. Experiment with different combinations of check boxes, selecting or clearing the boxes several times to test the program. The program logic you added with a few short lines of Visual Basic code manages the boxes perfectly. (You'll learn much more about program code in upcoming chapters.)

4. Click the Close button on the form to end the program.

Using Group Boxes and Radio Buttons

The *RadioButton* control is another tool that you can use to receive input in a program, and it is also located on the Common Controls tab of the Toolbox. Radio buttons get their name from the old push-button car radios of the 1950s and 1960s, when people pushed or "selected" one button on the car radio and the rest of the buttons clunked back to the unselected position. Only one button could be selected at a time, because (it was thought) the driver should listen to only one thing at a time. In Visual Studio, you can also offer mutually exclusive options for a user on a form, allowing them to pick one (and only one) option from a group. The procedure is to use the *GroupBox* control to create a frame on the form, and then to use the *RadioButton* control to place the desired number of radio buttons in the frame. (Because the *GroupBox* control is not used that often, it is located on the Containers tab of the Toolbox.) Note also that your form can have more than one group of

radio buttons, each operating independently of one another. For each group that you want to construct, simply create a group box object first and then add radio buttons one by one to the group box.

In the following exercise, you'll create a simple program that uses *GroupBox, RadioButton*, and *PictureBox* controls to present three graphical ordering options to a user. Like the *CheckBox* control, the *RadioButton* control is programmed by using event procedures and program code, with which you'll also experiment. Give it a try now.

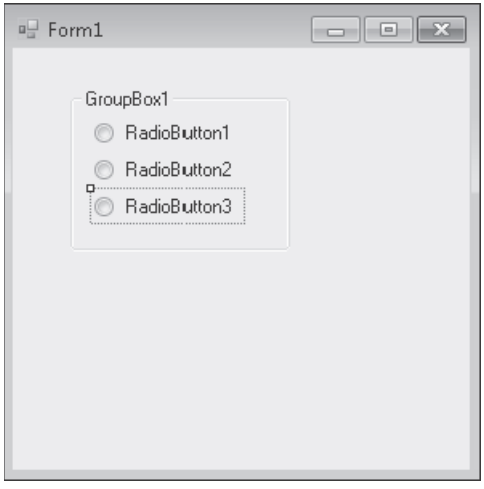
Gather input with the *GroupBox* and *RadioButton* controls

1. On the File menu, click Close Project to close the Check Box project.
2. On the File menu, click New Project.

   The New Project dialog box opens.
3. Create a new Visual Basic Windows Forms Application project named **MyRadioButton**.

   The new project is created, and a blank form appears in the Designer.
4. In the Toolbox, expand to the Containers tab and click the *GroupBox* control.
5. Create a medium-sized group box on the top half of the form.
6. Return to the Toolbox, scroll up to the Common Controls tab, and click the *RadioButton* control.
7. Create three radio button objects in the group box.

   It is handy to double-click the *RadioButton* control to create radio buttons. Notice that each radio button gets its own number, which you can use to set properties. Your form should look about like this:



8. Using the *PictureBox* control, create one square picture box object beneath the group box on the form.

9. Set the following properties for the group box, radio button, and picture box objects:

Object	Property	Setting
GroupBox1	*Text*	"Select a Computer Type"
RadioButton1	*Checked*	True
	Text	"Desktop PC"
RadioButton2	*Text*	"Desktop Mac"
RadioButton3	*Text*	"Laptop"
PictureBox1	*Image*	C:\Vb10sbs\Chap03\Pcomputr.bmp
	SizeMode	StretchImage

   The initial radio button state is controlled by the *Checked* property. Notice that the Desktop PC radio button now appears selected in the IDE. Now you'll add some program code to make the radio buttons operate while the program runs.

10. Double-click the *RadioButton1* object on the form to open the Code Editor.

    The *CheckedChanged* event procedure for the *RadioButton1* object appears in the Code Editor. This procedure is run each time the user clicks the first radio button. Because you want to change the picture box image when this happens, you'll add a line of program code to accomplish that.

11. Type the following program code:

```
PictureBox1.Image = System.Drawing.Image.FromFile _
  ("c:\vb10sbs\chap03\pcomputr.bmp")
```

    This program statement uses the *FromFile* method to load the picture of the PC from the hard disk into the picture box object. You'll use a similar statement for the second and third radio buttons.

12. Switch back to the Designer, double-click the *RadioButton2* object on the form, and type the following program code:

```
PictureBox1.Image = System.Drawing.Image.FromFile _
  ("c:\vb10sbs\chap03\computer.bmp")
```

13. Switch back to the Designer, double-click the *RadioButton3* object on the form, and type the following program code:

```
PictureBox1.Image = System.Drawing.Image.FromFile _
  ("c:\vb10sbs\chap03\laptop1.bmp")
```

14. Click the Save All button on the toolbar to save your changes, specifying the C:\Vb10sbs\Chap03 folder as the location.

Run the Radio Button program

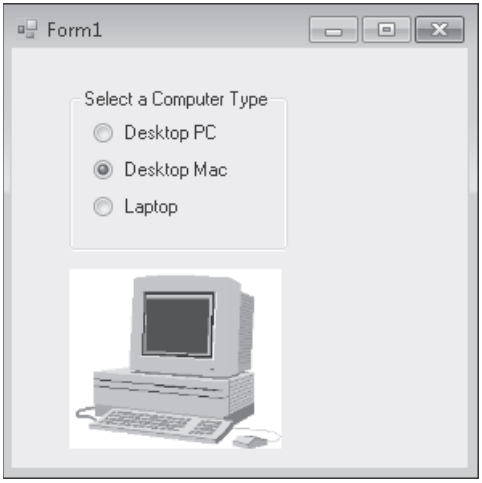
Tip The complete Radio Button program is located in the C:\Vb10sbs\Chap03\Radio Button folder.

1. Click the Start Debugging button on the Standard toolbar.

   Visual Basic runs the program in the IDE. The desktop PC image appears in a picture box on the form, and the first radio button is selected.

2. Click the second radio button (Desktop Mac).

   Visual Basic displays the image, as shown here:



3. Click the third radio button (Laptop).

   The laptop image appears.

4. Click the first radio button (Desktop PC).

   The desktop PC image appears again. It appears that each of the three *CheckedChanged* event procedures is loading the images just fine. Nice work.

5. Click the Close button on the form to end the program.

   Perfect. You're finished working with radio buttons and group boxes for now. But can you imagine how you might use them on your own in a program?

Processing Input with List Boxes

As you well know from your own use of Windows, one of the key mechanisms for getting input from the user—in addition to check boxes and radio buttons—are basic *list boxes*,

those rectangular containers used in dialog boxes or on forms that present a list of items and encourage the user to select one of them. List boxes are created in Visual Studio by using the *ListBox* control, and they are valuable because they can expand to include many items while the program is running. In addition, scroll bars can appear in list boxes if the number of items is larger than will fit in the box as you designed it on the form.

Unlike radio buttons, a list box doesn't require that the user be presented with a default selection. Another difference, from a programmatic standpoint, is that items in a list box can be rearranged while the program is running by adding items to a list, removing items, or sorting items. (You can also add a collection of items to a list box at design time by setting the *Items* property under the Data category with the Properties window.) However, if you prefer to see a list with check marks next to some of or all the items, you should use the *CheckedListBox* control in the Toolbox instead of *ListBox*. As a third option, you can use the handy *ComboBox* control to create a list box on a form that collapses to the size of a text box when not in use.

The key property of the *ListBox* control is *SelectedIndex*, which returns to the program the number of the item selected in the list box. Also important is the *Add* method, which allows you to add items to a list box in an event procedure. In the following exercise, you'll try out both of these features.

Create a list box to determine a user's preferences

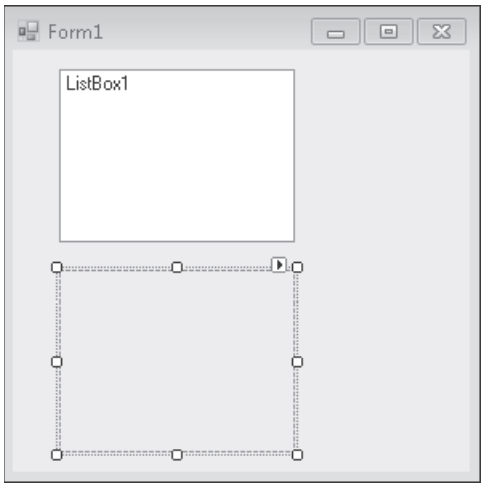
1. On the File menu, click Close Project to close the Radio Button project.
2. On the File menu, click New Project, and create a new Windows Forms Application project named **MyListBox**.

   The new project is created, and a blank form appears in the Designer.
3. In the Toolbox, click the *ListBox* control in the Toolbox, and create a medium-sized list box object on the top half of the form.

   The list box object offers a *Text* property, which (like the *GroupBox* control) allows you to assign a title to your container.
4. Use the *PictureBox* control to create a square picture box object beneath the list box object on the form.
5. Set the following property for the picture box object:

Object	Property	Setting
PictureBox1	*SizeMode*	StretchImage

Your form now will look similar to this:



Now you'll add the necessary program code to fill the list box object with valid selections, and to pick from the selections while the program is running.

6. Double-click the *ListBox1* object on the form to open the Code Editor.

   The *SelectedIndexChanged* event procedure for the *ListBox1* object appears in the Code Editor. This procedure runs each time the user clicks an item in the list box object. We need to update the image in the picture box object when this happens, so you'll add a line of program code to make it happen.

7. Type the following program code:

```
'The list box item selected (0-2) is held in the SelectedIndex property
Select Case ListBox1.SelectedIndex
    Case 0
        PictureBox1.Image = System.Drawing.Image.FromFile _
          ("c:\vb10sbs\chap03\harddisk.bmp")
    Case 1
        PictureBox1.Image = System.Drawing.Image.FromFile _
          ("c:\vb10sbs\chap03\printer.bmp")
    Case 2
        PictureBox1.Image = System.Drawing.Image.FromFile _
          ("c:\vb10sbs\chap03\satedish.bmp")
End Select
```

   As you learned in Chapter 2, the first line of this event procedure is a comment. Comments, which are displayed in green type, are simply notes written by a programmer to describe what's important or interesting about a particular piece of program code. I wrote this comment to explain that the *SelectedIndex* property returns a number to the program corresponding to the placement of the item that the user selected in the list box. There will be three items in the list box in this program,

and they will be numbered 0, 1, and 2 (from top to bottom). One interesting point here is that Visual Studio starts the count at 0, not 1, which is fairly typical among computer programs and something you'll see elsewhere in the book.

The entire block of code that you typed is actually called a *Select Case* decision structure, which explains to the compiler how to process the user's selection in the list box. The important keyword that begins this decision structure is *ListBox1* .*SelectedIndex*, which is read as "the *SelectedIndex* property of the list box object named *ListBox1*." If item 0 is selected, the *Case 0* section of the structure, which uses the *FromFile* method to load a picture of an external hard disk into the picture box object, will be executed. If item 1 is selected, the *Case 1* section will be executed, and a printer will appear in the picture box object. If item 2 is selected, the *Case 2* section will be executed, and a satellite dish will appear. Don't worry too much if this is a little strange—you'll get a more fulsome introduction to decision structures in Chapter 6.

Now you need to enter some program code to add text to the list box object. To do this, we'll do something new—we'll put some program statements in the *Form1_Load* event procedure, which is run when the program first starts.

8. Switch back to the Designer and double-click the form (*Form1*) to display the *Form1_Load* event procedure in the Code Editor.

   The *Form1_Load* event procedure appears. This program code is executed each time the List Box program is loaded into memory. Programmers put program statements in this special procedure when they want them executed every time a form loads. (Your program can display more than one form, or none at all, but the default behavior is that Visual Basic loads and runs the *Form1_Load* event procedure each time the user runs the program.) Often, as in the List Box program, these statements define an aspect of the user interface that couldn't be created easily by using the controls in the Toolbox or the Properties window.

9. Type the following program code:

```
'Add items to a list box like this:
ListBox1.Items.Add("Extra hard disk")
ListBox1.Items.Add("Printer")
ListBox1.Items.Add("Satellite dish")
```

   The first line is simply a comment offering a reminder about what the code accomplishes. The next three lines add items to the list box (*ListBox1*) in the program. The words in quotes will appear in the list box when it appears on the form. The important keyword in these statements is *Add*, a handy method that adds items to list boxes or other items. Remember that in the *ListBox1_SelectedIndexChanged* event procedure, these items will be identified as 0, 1, and 2.

10. Click the Save All button on the toolbar to save your changes, specifying the C:\Vb10sbs\Chap03 folder as the location.

Run the List Box program

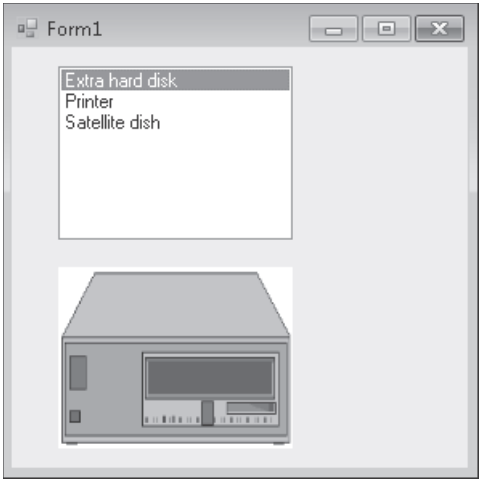
Tip The complete List Box program is located in the C:\Vb10sbs\Chap03\List Box folder.

1. Click the Start Debugging button on the Standard toolbar.

   Visual Basic runs the program in the IDE. The three items appear in the list box, but because no item is currently selected, nothing appears yet in the picture box object.

2. Click the first item in the list box (Extra Hard Disk).

   Visual Basic displays the hard disk image, as shown here:



3. Click the second item in the list box (Printer).

   The printer image appears.

4. Click the third item in the list box (Satellite Dish).

   The satellite dish appears. Perfect—all of the list box code seems to be working correctly, although you should always continue to test these things (that is, check the various user input options) to make sure that nothing unexpected happens. As you'll learn later in the book, you always want to test your programs thoroughly, especially the UI elements that users have access to.

5. Click the Close button on the form to end the program.

   You're finished working with list boxes for now. If you like, you can continue to experiment with the *ComboBox* and *CheckedListBox* controls on your own—they operate similar to the tools you have been using in the last few exercises.

Tip Speaking of building robust programs, you should know that most of the images in this simple example were loaded by using an *absolute path name* in the program code. Absolute path names (that is, exact file location designations that include all the folder names and drive letters) work well enough so long as the item you are referencing actually exists at the specified path. However, in a commercial application, you can't always be sure that your user won't move around the application files, which could cause programs like this one to generate an error when the files they need are no longer located in the expected place. To make your applications more seaworthy, or *robust*, it is usually better to use relative paths when accessing images and other resources. You can also embed images and other resources within your application. For information about this handy technique, see the "How to: Create Embedded Resources" and "Accessing Application Resources" topics in the Visual Studio 2010 Help documentation.

A Word About Terminology

OK—now that this chapter is complete, let's do a quick terminology review. So far in this book, I've used several different terms to describe items in a Visual Basic program. Do you know what most these items are yet? It's worth listing several of them now to clear up any confusion. If they are still unclear to you, bookmark this section and review the chapters that you have just completed for more information. (A few new terms are also mentioned here for the sake of completeness, and I'll describe them more fully later in the book.)

- **Program statement** A line of code in a Visual Basic program; a self-contained instruction executed by the Visual Basic compiler that performs useful work within the application. Program statements can vary in length (some contain only one Visual Basic keyword!), but all program statements must follow syntax rules defined and enforced by the Visual Basic compiler. In Visual Studio 2010, program statements can be composed of keywords, properties, object names, variables, numbers, special symbols, and other values. (See Chapters 2 and 5.)
- **Keyword** A reserved word within the Visual Basic language that is recognized by the Visual Basic compiler and performs useful work. (For example, the *End* keyword stops program execution.) Keywords are one of the basic building blocks of program statements; they work with objects, properties, variables, and other values to form complete lines of code and (therefore) instructions for the compiler and operating system. Most keywords are shown in blue type in the Code Editor. (See Chapter 2.)
- **Variable** A special container used to hold data temporarily in a program. The programmer creates variables by using the *Dim* statement and then uses these variables to store the results of a calculation, file names, input, and other items. Numbers, names, and property values can be stored in variables. (See Chapter 5.)

- **Control** A tool that you use to create objects in a Visual Basic program (most commonly, on a form). You select controls from the Toolbox and use them to draw objects with the mouse on a form. You use most controls to create UI elements such as buttons, picture boxes, and list boxes. (See especially Chapters 2 through 4.)
- **Object** An element that you create in a Visual Basic program with a control in the Toolbox. (In addition, objects are sometimes supplied by other system components, and many of these objects contain data.) In Visual Basic, the form itself is also an object. Technically speaking, objects are instances of a class that supports properties, methods, and events. In addition, objects have what is known as *inherent functionality*—they know how to operate and can respond to certain situations on their own. A list box "knows" how to scroll, for example. (See Chapters 1 through 4.)
- **Class** A blueprint or template for one or more objects that defines what the object does. Accordingly, a class defines what an object can do, but it is not the object itself. In Visual Basic, you can use existing .NET Framework classes (like *System.Math* and *System.Windows.Forms.Form*), and you can build your own classes and inherit properties, methods, and events from them. (*Inheritance* allows one class to acquire the pre-existing interface and behavior characteristics of another class.) Although classes might sound esoteric at this point, they are a key feature of Visual Studio 2010. In this book, you will use them to build user interfaces rapidly and to extend the work that you do to other programming projects. (See Chapters 5 and 16.)
- **Namespace** A hierarchical library of classes organized under a unique name, such as *System.Windows* or *System.Diagnostics*. To access the classes and underlying objects within a namespace, you place an *Imports* statement at the top of your program code. Every project in Visual Studio also has a root namespace, which is set using the project's Properties page. Namespaces are often referred to as *class libraries* in Visual Studio books and documentation. (See Chapter 5.)
- **Property** A value or characteristic held by an object. For example, a button object has a *Text* property, to specify the text that appears on the button, and an *Image* property, to specify the path to an image file that should appear on the button face. In Visual Basic, properties can be set at design time by using the Properties window, or at run time by using statements in the program code. In code, the format for setting a property is

  ```
  Object.Property = Value
  ```

  where *Object* is the name of the object you're customizing, *Property* is the characteristic you want to change, and *Value* is the new property setting. For example,

  ```
  Button1.Text = "Hello"
  ```

  could be used in the program code to set the *Text* property of the *Button1* object to "Hello". (See Chapters 1 through 3.)

- **Event procedure** A block of code that's executed when an object is manipulated in a program. For example, when the *Button1* object is clicked, the *Button1_Click* event procedure is executed. Event procedures typically evaluate and set properties and use other program statements to perform the work of the program. (See Chapters 1 through 3.)
- **Method** A special statement that performs an action or a service for a particular object in a program. In program code. The notation for using a method is

  ```
  Object.Method(Value)
  ```

  where *Object* is the name of the object you want to work with, *Method* is the action you want to perform, and *Value* is zero or more arguments to be used by the method. For example, the statement

  ```
  ListBox1.Items.Add("Check")
  ```

  uses the *Add* method to put the word *Check* in the *ListBox1* list box. Methods and properties are often identified by their position in a collection or class library, so don't be surprised if you see long references such as *System.Drawing.Image.FromFile*, which would be read as "the *FromFile* method, which is a member of the *Image* class, which is a member of the *System.Drawing* namespace." (See Chapters 1 through 5.)

One Step Further: Using the *LinkLabel* Control

Providing access to the Web is now a standard feature of many Windows applications, and with Visual Studio, adding this functionality is easier than ever. You can create a Visual Basic program that runs from a Web server by creating a Web Forms project and using controls in the Toolbox optimized for the Web. Alternatively, you can use Visual Basic to create a Windows application that opens a Web browser within the application, providing access to the Web while remaining a Windows program running on a client computer. We'll postpone writing Web Forms projects for a little while longer in this book, but in the following exercise, you'll learn how to use the *LinkLabel* Toolbox control to create a Web link in a Windows program that provides access to the Internet through Windows Internet Explorer or the default Web browser on your system.

Note To learn more about writing Web-aware Visual Basic 2010 applications, read Chapter 20, "Creating Web Sites and Web Pages Using Visual Web Developer and ASP.NET."

Create the WebLink program

1. On the File menu, click Close Project to close the List Box project.
2. On the File menu, click New Project.

   The New Project dialog box opens.

3. Create a new Visual Basic Windows Forms Application project named **MyWebLink**.

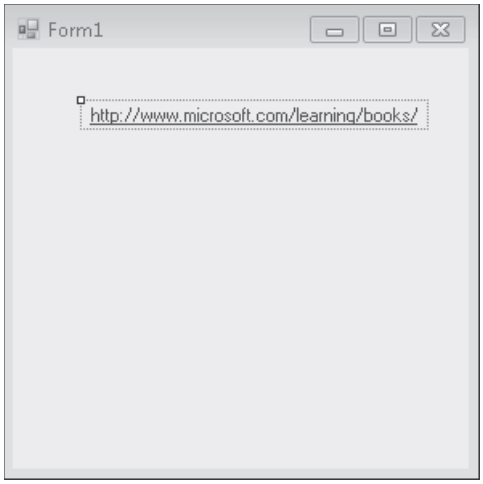
   The new project is created, and a blank form appears in the Designer.

4. Click the *LinkLabel* control in the Toolbox, and draw a rectangular link label object on your form.

   Link label objects look like label objects except that all label text is displayed in blue underlined type on the form.

5. Set the *Text* property of the link label object to the Uniform Resource Locator (URL) for the Microsoft Press home page: *http://www.microsoft.com/learning/books/.*

   Your form looks like this:



6. Click the form in the IDE to select it. (Click the form itself, not the link label object.)

   This is the technique that you use to view the properties of the default form, Form1, in the Properties window. Like other objects in your project, the form also has properties that you can set.

7. Set the *Text* property of the form object to **Web Link Test**.

   The *Text* property for a form specifies what appears on the form's title bar at design time and when the program runs. Although this customization isn't related exclusively to the Web, I thought you'd enjoy picking up that skill now, before we move on to other projects. (We'll customize the title bar in most of the programs we build.)

8. Double-click the link label object, and then type the following program code in the *LinkLabel1_LinkClicked* event procedure:

```
' Change the color of the link by setting LinkVisited to True.
LinkLabel1.LinkVisited = True
' Use the Process.Start method to open the default browser
' using the Microsoft Press URL:
```

```
System.Diagnostics.Process.Start _
  ("http://www.microsoft.com/learning/books/")
```

I've included more comments in the program code to give you some practice entering them. As soon as you enter the single quote character ('), Visual Studio changes the color of the line to green.

The two program statements that aren't comments control how the link works. Setting the *LinkVisited* property to True gives the link that dimmer color of purple, which indicates in many browsers that the Hypertext Markup Language (HTML) document associated with the link has already been viewed. Although setting this property isn't necessary to display a Web page, it's a good programming practice to provide the user with information in a way that's consistent with other applications.

The second program statement (which I have broken into two lines) runs the default Web browser (such as Internet Explorer) if the browser isn't already running. (If the browser is running, the URL just loads immediately.) The *Start* method in the *Process* class performs the important work, by starting a process or executable program session in memory for the browser. The *Process* class, which manages many other aspects of program execution, is a member of the *System.Diagnostics* namespace. By including an Internet address or a URL with the *Start* method, I'm letting Visual Basic know that I want to view a Web site, and Visual Basic is clever enough to know that the default system browser is the tool that would best display that URL, even though I didn't identify the browser by name.

An exciting feature of the *Process.Start* method is that it can be used to run other Windows applications, too. If I did want to identify a particular browser by name to open the URL, I could have specified one using the following syntax. (Here I'll request the Internet Explorer browser.)

```
System.Diagnostics.Process.Start("IExplore.exe", _
  "http://www.microsoft.com/learning/books/")
```

Here, two arguments are used with the *Start* method, separated by a comma. The exact location for the program named IExplore.exe on my system isn't specified, but Visual Basic will search the current system path for it when the program runs.

If I wanted to run a different application with the *Start* method—for example, if I wanted to run the Microsoft Office Word application and open the document C:\Myletter.doc—I could use the following syntax:

```
System.Diagnostics.Process.Start("Winword.exe", _
  "c:\myletter.doc")
```

As you can see, the *Start* method in the *Process* class is very useful.

Now that you've entered your code, you should save your project. (If you experimented with the *Start* syntax as I showed you, restore the original code shown at the beginning of step 8 first.)

9. Click the Save All button on the Standard toolbar to save your changes, and specify C:\Vb10sbs\Chap03 as the location.

   You can now run the program.

Run the WebLink program

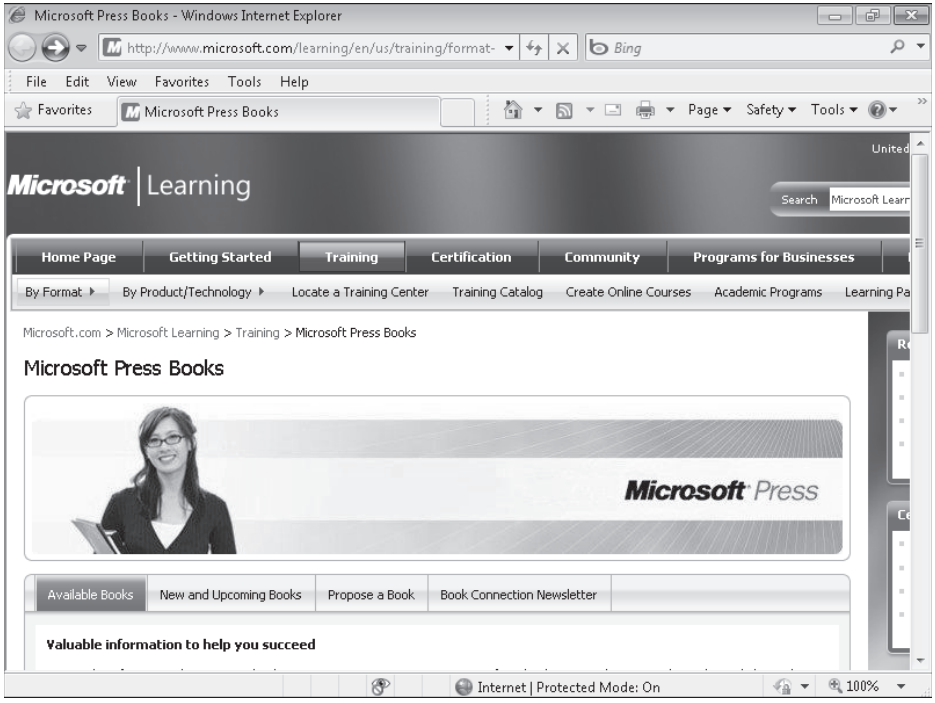
Tip The complete WebLink program is located in the C:\Vb10sbs\Chap03\Weblink folder.

1. Click the Start Debugging button on the Standard toolbar to run the WebLink program.

   The form opens and runs, showing its Web site link and handsome title bar text.

2. Click the link to open the Web site at *http://www.microsoft.com/learning/books/*.

   Recall that it's only a happy coincidence that the link label *Text* property contains the same URL as the site you named in the program code. (It is not necessary that these two items match.) You can enter any text you like in the link label. You can also use the *Image* property for a link label to specify a picture to display in the background of the link label. The following figure shows what the Microsoft Press Web page looks like (in English) when the WebLink program displays it using Internet Explorer.



3. Display the form again. (Click the Web Link Test form icon on the Windows taskbar if the form isn't visible.)

   Notice that the link now appears in a dimmed style. Like a standard Web link, your link label communicates that it's been used (but is still active) by the color and intensity that it appears in.

4. Click the Close button on the form to quit the test utility.

   You're finished writing code in this chapter, and you're gaining valuable experience with some of the Toolbox controls available for creating Windows Forms applications. Let's keep going!

Chapter 3 Quick Reference

To	Do This
Create a text box	Click the *TextBox* control, and draw the box.
Create a button	Click the *Button* control, and draw the button.
Change a property at run time	Change the value of the property by using program code. For example: `Label1.Text = "Hello!"`
Create a radio button	Use the *RadioButton* control. To create multiple radio buttons, place more than one radio button object inside a box that you create by using the *GroupBox* control.
Create a check box	Click the *CheckBox* control, and draw a check box.
Create a list box	Click the *ListBox* control, and draw a list box.
Create a drop-down list box	Click the *ComboBox* control, and draw a drop-down list box.
Add items to a list box	Include statements with the *Add* method in the *Form1_Load* event procedure of your program. For example: `ListBox1.Items.Add("Printer")`
Use a comment in code	Type a single quotation mark (') in the Code Editor, and then type a descriptive comment that will be ignored by the compiler. For example: `' Use the Process.Start method to start IE`
Display a Web page	Create a link to the Web page by using the *LinkLabel* control, and then open the link in a browser by using the *Process.Start* method in program code.