

Try

' Convert quantity to numeric variable.

C H A P T E R Integer.Parse(QuantityTe

1

Introduction to Visual Basic 2008

at the completion of this chapter, you will be able to . . .

- 1.** Describe the process of visual program design and development.
- 2.** Explain the term *object-oriented programming*.
- 3.** Explain the concepts of classes, objects, properties, methods, and events.
- 4.** List and describe the three steps for writing a Visual Basic project.
- 5.** Describe the various files that make up a Visual Basic project.
- 6.** Identify the elements in the Visual Studio environment.
- 7.** Define design time, run time, and debug time.
- 8.** Write, run, save, print, and modify your first Visual Basic project.
- 9.** Identify syntax errors, run-time errors, and logic errors.
- 10.** Use AutoCorrect to correct syntax errors.
- 11.** Look up Visual Basic topics in Help.

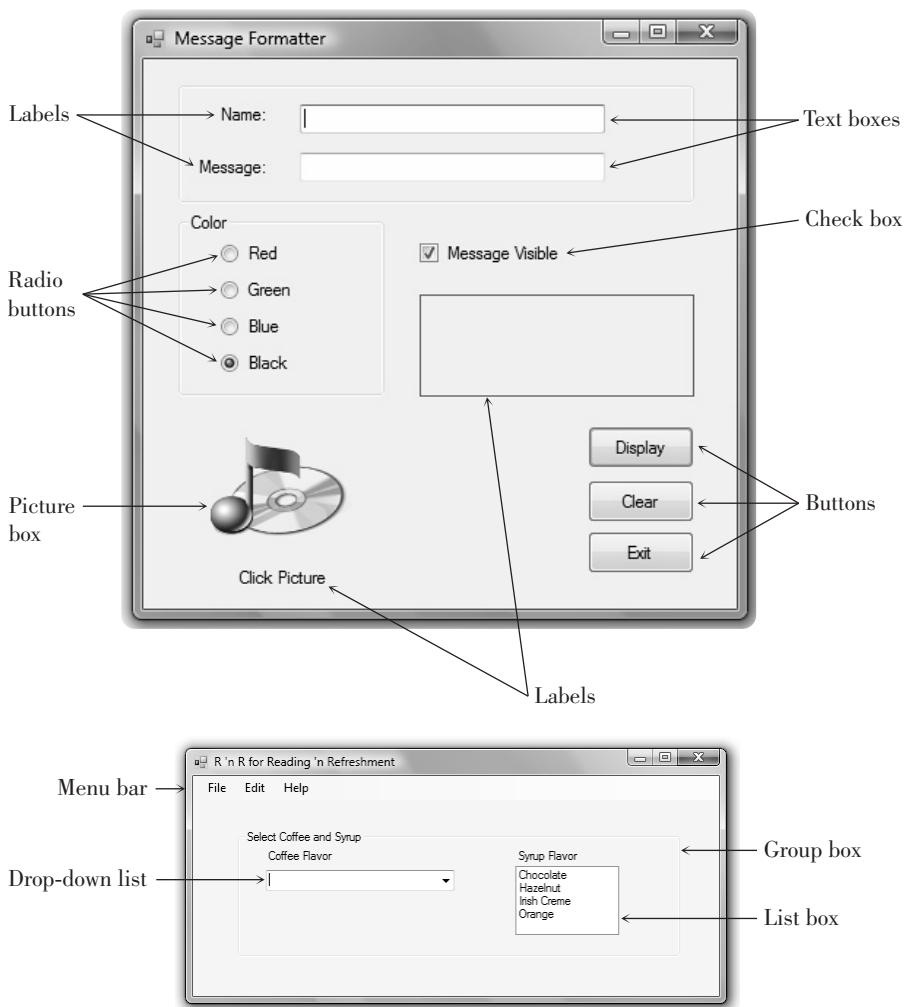
Writing Windows Applications with Visual Basic

Using this text, you will learn to write computer programs that run in the Microsoft Windows environment. Your projects will look and act like standard Windows programs. You will use the tools in Visual Basic (VB) and Windows Forms to create windows with familiar elements such as labels, text boxes, buttons, radio buttons, check boxes, list boxes, menus, and scroll bars. Figure 1.1 shows some sample Windows user interfaces.

Beginning in Chapter 9, you will create programs using Web Forms and Visual Web Developer. You can run Web applications in a browser such as

Figure 1.1

Graphical user interfaces for application programs designed with Visual Basic and Windows Forms.

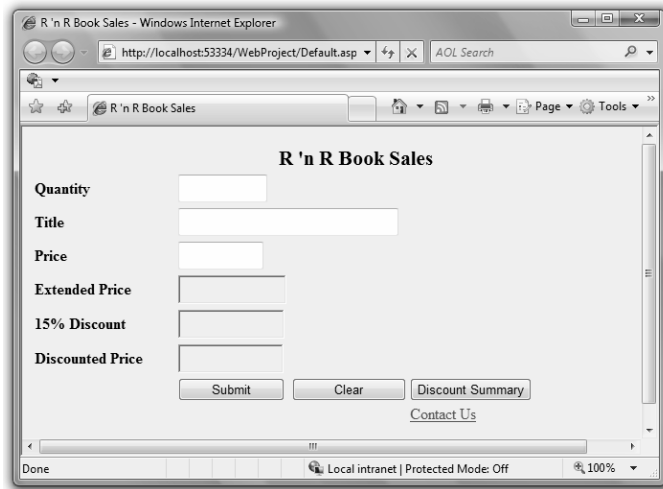


Internet Explorer or Mozilla FireFox, on the Internet, or on a company intranet. Figure 1.2 shows a Web Form application.

You also will become acquainted with Microsoft's new screen design technology, Windows Presentation Foundation (WPF), which is covered in Chapter 14. WPF uses its own designer and design elements, which are different from those used for Windows forms.

Figure 1.2

A Web Form application created with Visual Web Developer, running in a browser.



The Windows Graphical User Interface

Microsoft Windows uses a **graphical user interface**, or **GUI** (pronounced “gooey”). The Windows GUI defines how the various elements look and function. As a Visual Basic programmer, you have available a toolbox of these elements. You will create new windows, called **forms**. Then you will use the toolbox to add the various elements, called **controls**. The projects that you will write follow a programming technique called **object-oriented programming (OOP)**.

Programming Languages—Procedural, Event Driven, and Object Oriented

There are literally hundreds of programming languages. Each was developed to solve a particular type of problem. Most traditional languages, such as BASIC, C, COBOL, FORTRAN, PL/I, and Pascal, are considered *procedural* languages. That is, the program specifies the exact sequence of all operations. Program logic determines the next instruction to execute in response to conditions and user requests.

The newer programming languages, such as Visual Basic, C#, and Java, use a different approach: object-oriented programming. As a stepping stone between procedural programming and object-oriented programming, the early versions of Visual Basic provided many (but not all) elements of an object-oriented language. For that reason, Microsoft referred to Visual Basic (version 6 and earlier) as an event-driven programming language rather than an object-oriented language. But with Visual Studio, which includes Visual Basic, C#, and F#, we have programming languages that are truly object oriented. (Another language, C++, has elements of OOP and of procedural programming and doesn't conform fully to either paradigm.) F#, introduced in 2007, applies the object-oriented paradigm to scripting languages for cross-platform development.

In the OOP model, programs are no longer procedural. They do not follow a sequential logic. You, as the programmer, do not take control and determine the sequence of execution. Instead, the user can press keys and click various buttons and boxes in a window. Each user action can cause an event to occur, which triggers a Basic procedure that you have written. For example, the user clicks on a button labeled Calculate. The clicking causes the button's Click event to occur, and the program automatically jumps to a procedure you have written to do the calculation.

The Object Model

In Visual Basic you will work with **objects**, which have **properties**, **methods**, and **events**. Each object is based on a **class**.

Objects

Think of an object as a thing, or a noun. Examples of objects are forms and controls. *Forms* are the windows and dialog boxes you place on the screen; *controls* are the components you place inside a form, such as text boxes, buttons, and list boxes.

Properties

Properties tell something about or control the behavior of an object, such as its name, color, size, or location. You can think of properties as adjectives that describe objects.

When you refer to a property, you first name the object, add a period, and then name the property. For example, refer to the Text property of a form called SalesForm as SalesForm.Text (pronounced "sales form dot text").

Methods

Actions associated with objects are called *methods*. Methods are the verbs of object-oriented programming. Some typical methods are `Close`, `Show`, and `Clear`. Each of the predefined objects has a set of methods that you can use. You will learn to write additional methods to perform actions in your programs.

You refer to methods as Object.Method ("object dot method"). For example, a `Show` method can apply to different objects: `BillingForm.Show` shows the form object called `BillingForm`; `ExitButton.Show` shows the button object called `ExitButton`.

Events

You can write procedures that execute when a particular event occurs. An event occurs when the user takes an action, such as clicking a button, pressing a key, scrolling, or closing a window. Events also can be triggered by actions of other objects, such as repainting a form or a timer reaching a preset point.

Classes

A class is a template or blueprint used to create a new object. Classes contain the definition of all available properties, methods, and events.

Each time that you create a new object, it must be based on a class. For example, you may decide to place three buttons on your form. Each button is based on the `Button` class and is considered one object, called an *instance* of the class. Each button (or instance) has its own set of properties, methods, and events. One



The term *members* is used to refer to both properties and methods. ■

button may be labeled “OK”, one “Cancel”, and one “Exit”. When the user clicks the OK button, that button’s Click event occurs; if the user clicks on the Exit button, that button’s Click event occurs. And, of course, you have written different program instructions for each of the buttons’ Click events.

An Analogy

If the concepts of classes, objects, properties, methods, and events are still a little unclear, maybe an analogy will help. Consider an Automobile class. When we say *automobile*, we are not referring to a particular auto, but we know that an automobile has a make and model, a color, an engine, and a number of doors. These elements are the *properties* of the Automobile class.

Each individual auto is an object, or an instance of the Automobile class. Each Automobile object has its own settings for the available properties. For example, each object has a Color property, such as `MyAuto.Color = Blue` and `YourAuto.Color = Red`.

The methods, or actions, of the Automobile class might be `Start`, `SpeedUp`, `SlowDown`, and `Stop`. To refer to the methods of a specific object of the class, use `MyAuto.Start` and `YourAuto.Stop`.

The events of an Automobile class could be `Arrive` or `Crash`. In a VB program you write procedures that specify the actions you want to take when a particular event occurs for an object. For example, you might write a procedure for the `YourAuto.Crash` event.

Note: Chapter 12 presents object-oriented programming in greater depth.

Microsoft’s Visual Studio

The latest version of Microsoft’s Visual Studio, called Visual Studio 2008, includes Visual Basic, Visual C++, Visual C# (C sharp), and the .NET 3.5 Framework.

The .NET Framework

The programming languages in Visual Studio run in the .NET Framework. The Framework provides for easier development of Web-based and Windows-based applications, allows objects from different languages to operate together, and standardizes how the languages refer to data and objects. Several third-party vendors have announced or have released versions of other programming languages to run in the .NET Framework, including .NET versions of APL by Dyalog, FORTRAN by Lahey Computer Systems, COBOL by Fujitsu Software Corporation, Pascal by the Queensland University of Technology (free), PERL by ActiveState, RPG by ASNA, and Java, known as IKVM.NET.

The .NET languages all compile to (are translated to) a common machine language, called Microsoft Intermediate Language (MSIL). The MSIL code, called *managed code*, runs in the Common Language Runtime (CLR), which is part of the .NET Framework.

Visual Basic

Microsoft Visual Basic comes with Visual Studio. You also can purchase VB by itself (without the other languages but *with* the .NET Framework). VB is available in an **Express Edition**, a **Standard Edition**, a **Professional Edition**, and a **Team System Edition**. Anyone planning to do professional

application development that includes the advanced features of database management should use the Professional Edition or the Team System Edition. You can find a matrix showing the features of each edition in Help. The Professional Edition is available to educational institutions through the Microsoft Academic Alliance program and is the best possible deal. When a campus department purchases the Academic Alliance, the school can install Visual Studio on all classroom and lab computers and provide the software to all students and faculty at no additional charge.

Microsoft provides an Express Edition of each of the programming languages, which you can download for free (www.microsoft.com/express/download/). You can use Visual Basic Express for Windows development and Visual Web Developer Express for the Web applications in Chapter 9. This text is based on the Professional Edition of Visual Studio 2008. However, you can do the projects using Visual Basic 2008 Express Edition and Visual Web Developer 2008 Express Edition, all of which are the current versions. This version of Visual Basic is called both Visual Basic 2008 and Visual Basic 9. You cannot run the projects in this text in any earlier version of VB.

Writing Visual Basic Projects

When you write a Visual Basic application, you follow a three-step process for planning the project and then repeat the three-step process for creating the project. The three steps involve setting up the user interface, defining the properties, and then creating the code.

The Three-Step Process

Planning

1. *Design the user interface.* When you plan the **user interface**, you draw a sketch of the screens the user will see when running your project. On your sketch, show the forms and all the controls that you plan to use. Indicate the names that you plan to give the form and each of the objects on the form. Refer to Figure 1.1 for examples of user interfaces.

Before you proceed with any more steps, consult with your user and make sure that you both agree on the look and feel of the project.

2. *Plan the properties.* For each object, write down the properties that you plan to set or change during the design of the form.
3. *Plan the Basic code.* In this step, you plan the classes and procedures that will execute when your project runs. You will determine which events require action to be taken and then make a step-by-step plan for those actions.

Later, when you actually write the Visual Basic **code**, you must follow the language syntax rules. But during the planning stage, you will write out the actions using **pseudocode**, which is an English expression or comment that describes the action. For example, you must plan for the event that occurs when the user clicks on the Exit button. The pseudocode for the event could be *Terminate the project* or *Quit*.





Programming



After you have completed the planning steps and have approval from your user, you are ready to begin the actual construction of the project. Use the same three-step process that you used for planning.

1. *Define the user interface.* When you define the user interface, you create the forms and controls that you designed in the planning stage.
Think of this step as defining the objects you will use in your application.
2. *Set the properties.* When you set the properties of the objects, you give each object a name and define such attributes as the contents of a label, the size of the text, and the words that appear on top of a button and in the form's title bar.
You might think of this step as describing each object.
3. *Write the Basic code.* You will use Basic programming statements (called *Basic code*) to carry out the actions needed by your program. You will be surprised and pleased by how few statements you need to create a powerful Windows program.
You can think of this third step as defining the actions of your program.

Visual Basic Application Files

A Visual Basic application, called a **solution**, can consist of one or more projects. Since all of the solutions in this text have only one project, you can think of one solution = one project. Each project can contain one or more form files. In Chapters 1 through 5, all projects have only one form, so you can think of one project = one form. Starting in Chapter 6, your projects will contain multiple forms and additional files. As an example, the HelloWorld application that you will create later in this chapter creates the following files:

File Name	File Icon	Description
HelloWorld.sln		The solution file. A text file that holds information about the solution and the projects it contains. This is the primary file for the solution—the one that you open to work on or run your project. Note the “9” on the icon, which refers to VB version 9.
HelloWorld.suo		Solution user options file. Stores information about the state of the integrated development environment (IDE) so that all customizations can be restored each time you open the solution.
HelloForm.vb		A .vb file that holds the code procedures that you write. This is a text file that you can open in any editor. Warning: You should not modify this file unless you are using the editor in the Visual Studio environment.
HelloForm.resx		A resource file for the form. This text file defines all resources used by the form, including strings of text, numbers, and any graphics.

File Name	File Icon	Description
HelloForm.Designer.vb		A file created by the Form Designer that holds the definition of the form and its controls. You should not modify this file directly, but make changes in the Designer and allow it to update the file.
HelloWorld.vbproj.user		The project user option file. This text file holds IDE option settings so that the next time you open the project, all customizations will be restored.

Note: You can display file extensions. In Windows Vista, open the Explorer and select *Organize / Folders and Search Options*, click on the *View* tab, and deselect the check box for *Hide extensions for known file types*. In Windows XP, in the My Computer *Tools* menu, select *Folder Options* and the *View* tab, Deselect the check box for *Hide extensions for known file types*. If you do not display the extensions, you can identify the file types by their icons.

After you run your project, you will find several more files created by the system. These include the *AssemblyInfo.vb*, *MyApplication.myapp*, *MyEvents.vb*, *Resources.resx*, and *Resources.vb*. The only file that you open directly is the *.sln*, or solution file.

The Visual Studio Environment

The **Visual Studio environment** is where you create and test your projects. A development environment, such as Visual Studio, is called an **integrated development environment (IDE)**. The IDE consists of various tools, including a form designer, which allows you to visually create a form; an editor, for entering and modifying program code; a compiler, for translating the Visual Basic statements into the intermediate machine code; a debugger, to help locate and correct program errors; an object browser, to view available classes, objects, properties, methods, and events; and a Help facility.

In versions of Visual Studio prior to .NET, each language had its own IDE. For example, to create a VB project you would use the VB IDE, and to create a C++ project you would use the C++ IDE. But in Visual Studio, you use one IDE to create projects in any of the supported languages.

Note that this text is based on the Express Edition of Visual Studio. If you are using the Professional Edition, the screens differ somewhat from those that you see.

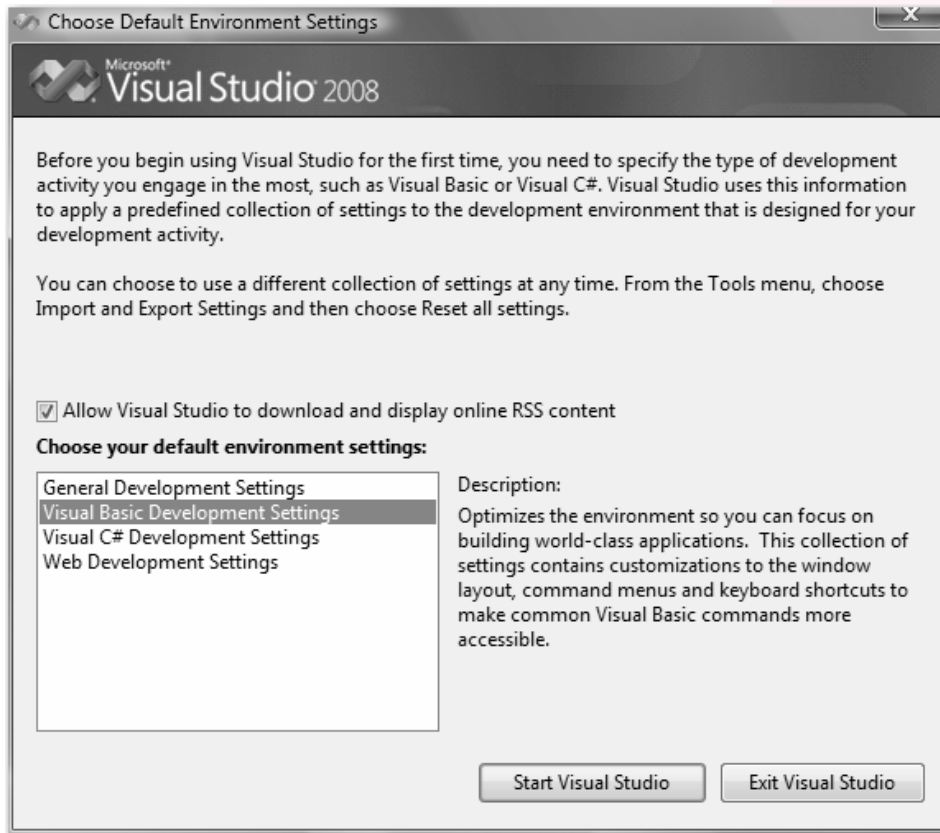
Default Environment Settings

The full version of Visual Studio 2008 provides an option to allow the programmer to select the default profile for the IDE. The first time you open Visual Studio, you are presented with the *Choose Default Environment Settings* dialog box (Figure 1.3), where you can choose *Visual Basic Development Settings*. Notice the instructions in the dialog box: you can make a different selection later from the *Tools* menu.

Note: If you are using the Express Edition of Visual Basic, you won't see this dialog box.

Figure 1.3

The first time you open the Visual Studio IDE, you must select the default environment settings for a Visual Basic developer.



The IDE Initial Screen

When you open the Visual Studio IDE, you generally see an empty environment with a Start Page (Figure 1.4). However, it's easy to customize the environment, so you may see a different view. In the step-by-step exercise later in this chapter, you will learn to reset the IDE layout to its default view.

The contents of the Start Page vary, depending on whether you are connected to the Internet. Microsoft has added links that can be updated, so you may find new and interesting information on the Start Page each time you open it. To display or hide the Start Page, select *View / Other Windows / Start Page*.

You can open an existing project or begin a new project using the Start Page or the *File* menu.

The New Project Dialog

You will create your first Visual Basic projects by selecting *File / New Project* on the menu bar or clicking *Create: Project* on the Start Page, either of which opens the *New Project* dialog (Figure 1.5). In the *New Project* dialog, select *Windows*

Figure 1.4

The Visual Studio IDE with the Start Page open, as it first appears in Windows Vista, without an open project.

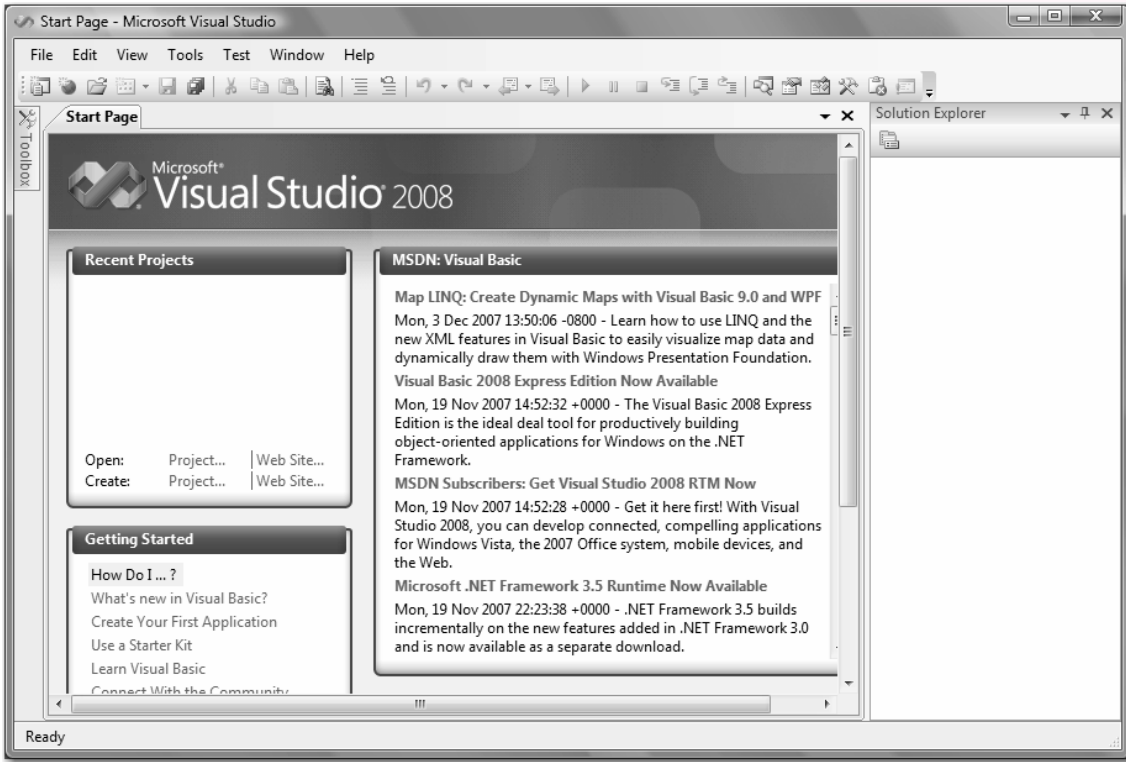
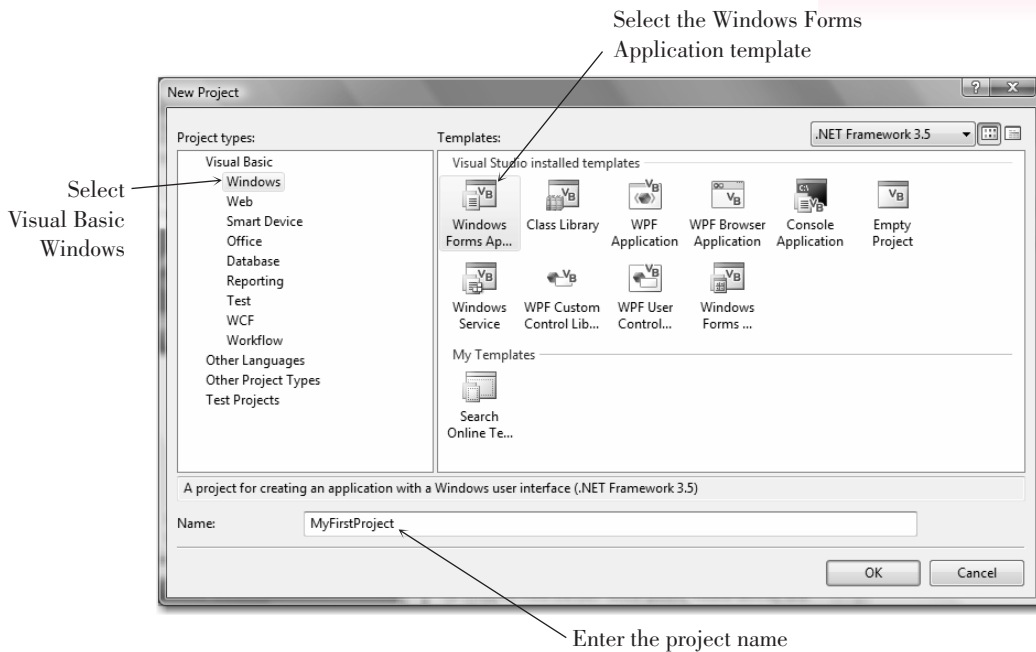


Figure 1.5

Begin a new VB Windows project using the Windows Forms Application template.



Forms Application if you are using the VB Express Edition. In the Professional Edition, first select *Visual Basic* and *Windows* in the *Project Types* box and *Windows Application* in the *Templates* box. You also give the project a name on this dialog box.

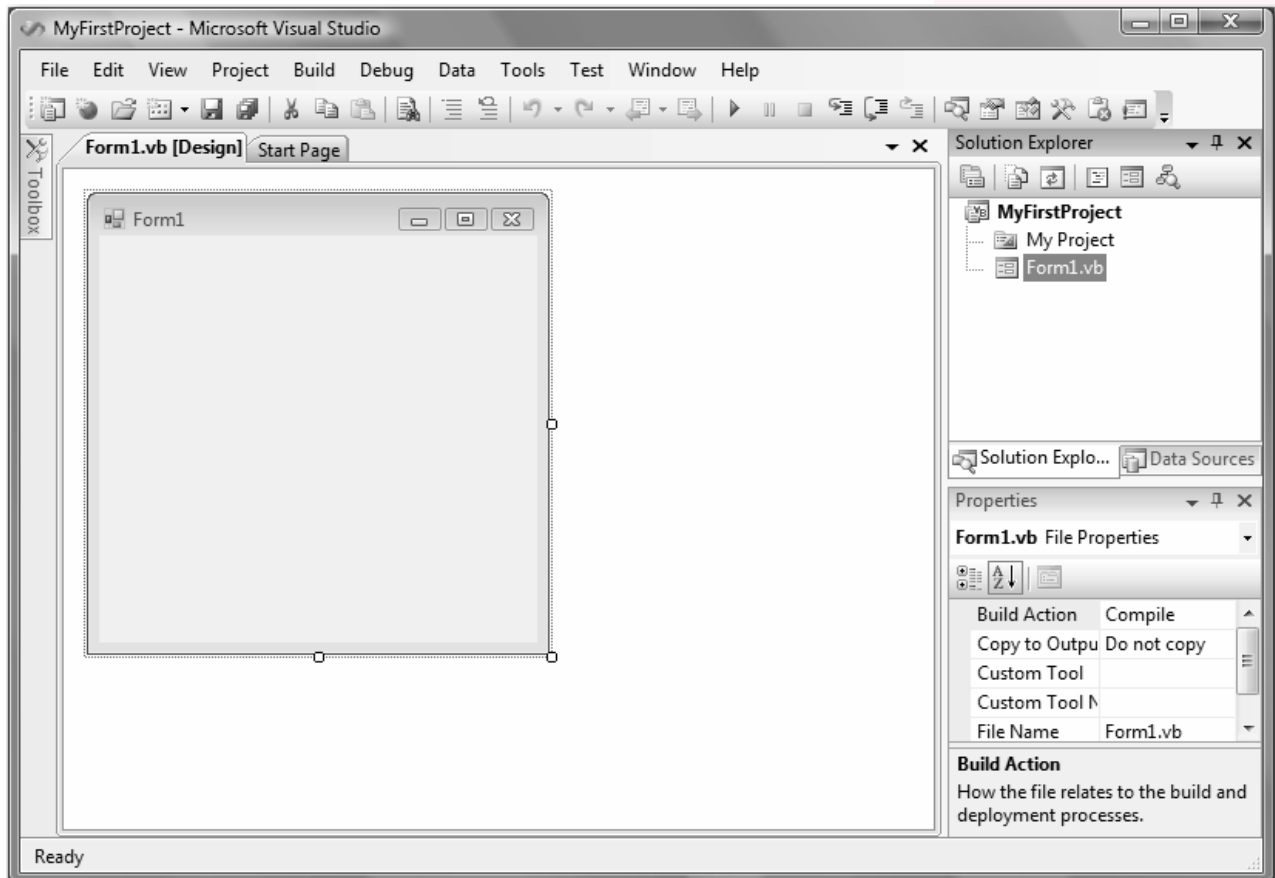
The IDE Main Window

Figure 1.6 shows the Visual Studio environment's main window and its various child windows. Note that each window can be moved, resized, opened, closed, and customized. Some windows have tabs that allow you to display different contents. Your screen may not look exactly like Figure 1.6; in all likelihood you will want to customize the placement of the various windows. The windows in the IDE are considered either document windows or tool windows. The Designer and Editor windows are generally displayed in tabs in the center of the screen (the Document window), and the various tool windows are docked along the edges and bottom of the IDE, but the locations and docking behavior are all customizable.

The IDE main window holds the Visual Studio menu bar and the toolbars. You can display or hide the various windows from the *View* menu.

Figure 1.6

The Visual Studio environment. Each window can be moved, resized, closed, or customized.

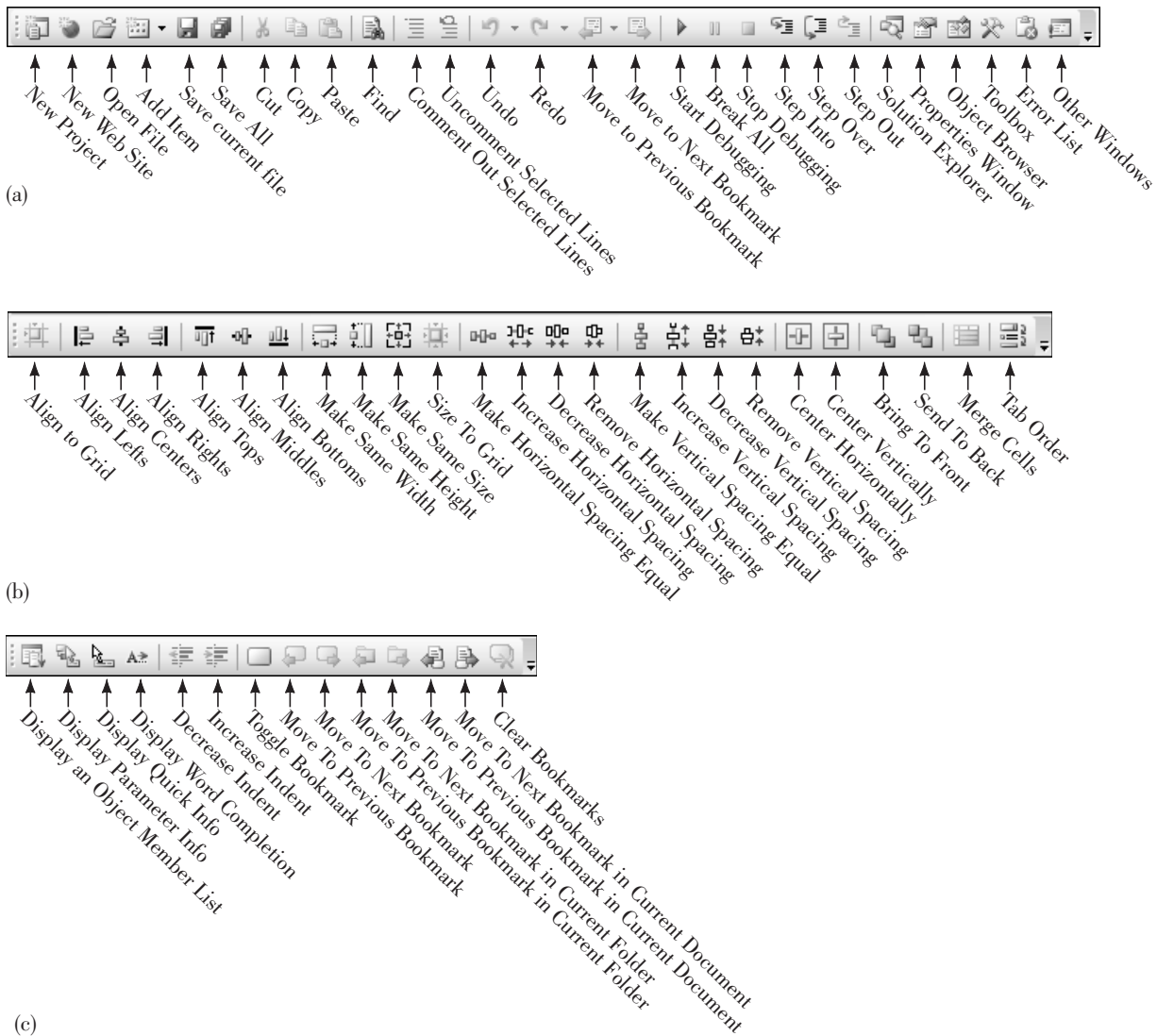


The Toolbars

You can use the buttons on the **toolbars** as shortcuts for frequently used operations. Each button represents a command that also can be selected from a menu. Figure 1.7a shows the toolbar buttons on the Standard toolbar for the Professional Edition, which displays in the main window of the IDE; Figure 1.7b shows the Layout toolbar, which is useful for designing forms in the Form Designer; and Figure 1.7c shows the Text Editor toolbar, which contains buttons to use in the Editor window. Select *View / Toolbars* to display or hide these and other toolbars.

Figure 1.7

The Visual Studio toolbars contain buttons that are shortcuts for menu commands. You can display or hide each of the toolbars: a. the Standard toolbar; b. the Layout toolbar; and c. the Text Editor toolbar.



The Document Window

The largest window in the center of the screen is the **Document window**. Notice the tabs across the top of the window, which allow you to switch between

open documents. The items that display in the Document window include the Form Designer, the Code Editor, the Database Designer, and the Object Browser.

You can switch from one tab to another, or close any of the documents using its Close button.

The Form Designer

The **Form Designer** is where you design a form that makes up your user interface. In Figure 1.6, the Form Designer for Form1 is currently displaying. You can drag the form's sizing **handle** or selection border to change the size of the form.

When you begin a new Visual Basic Windows application, a new form is added to the project with the default name Form1. In the step-by-step exercise later in the chapter, you will learn to change the form's name.

The Solution Explorer Window

The **Solution Explorer window** holds the filenames for the files included in your project and a list of the classes it references. The Solution Explorer window and the Window's title bar hold the name of your solution (.sln) file, which is WindowsApplication1 by default unless you give it a new value in the *New Project* dialog box. In Figure 1.6, the name of the solution is MyFirstProject.

The Properties Window

You use the **Properties window** to set the properties for the objects in your project. See "Set Properties" later in this chapter for instructions on changing properties.

The Toolbox

The **toolbox** holds the tools you use to place controls on a form. You may have more or different tools in your toolbox, depending on the edition of Visual Basic you are using (Express, Standard, Professional, or Team System). Figure 1.8 shows the Express Edition toolbox.

Help

Visual Studio has an extensive **Help** feature, which includes the Microsoft Developer Network library (MSDN). You can find reference materials for Visual Basic, C++, C#, and Visual Studio; several books; technical articles; and the Microsoft Knowledge Base, a database of frequently asked questions and their answers.

Help includes the entire reference manual, as well as many coding examples. See the topic "Visual Studio Help" later in this chapter for help on Help.

When you make a selection from the *Help* menu, the requested item appears in a new window that floats on top of the IDE window (Figure 1.9), so you can keep both open at the same time. It's a good idea to set the *Filtered By* entry to *Visual Basic* or *Visual Basic Express Edition*, depending on the edition you are using.



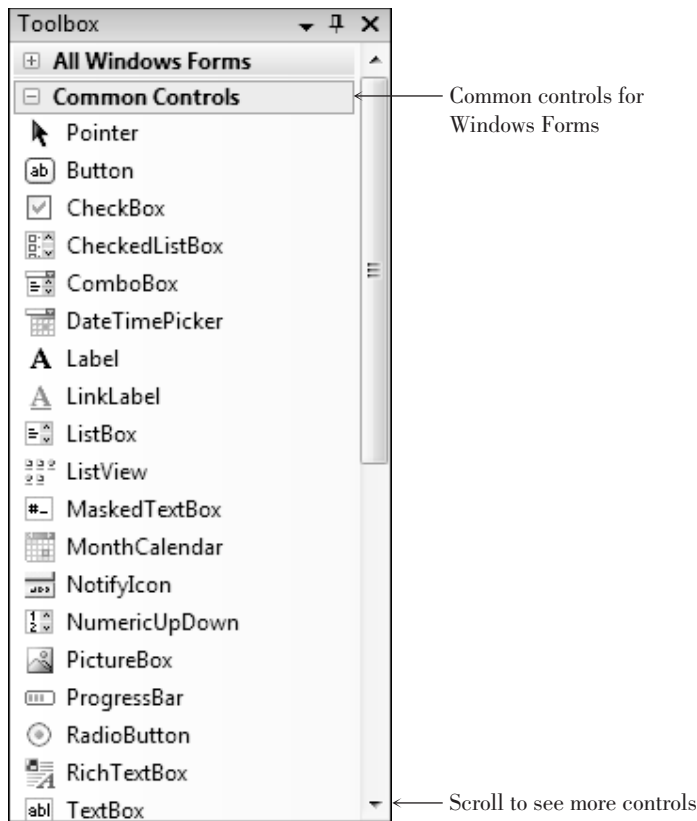
TIP

Use Ctrl + Tab to switch to another open document in the Document window. ■



TIP

You can sort the properties in the window either *alphabetically* or *by categories*. Use the buttons on the Properties window. ■

Figure 1.8

The toolbox for Visual Studio Windows Forms. Your toolbox may have more or fewer tools, depending on the edition you are using.

✓ TIP

You can sort the tools in the toolbox: Right-click the toolbox and select *Sort Items Alphabetically* from the context menu (the shortcut menu). ■

Design Time, Run Time, and Debug Time

Visual Basic has three distinct modes. While you are designing the user interface and writing code, you are in **design time**. When you are testing and running your project, you are in **run time**. If you get a run-time error or pause program execution, you are in **debug time**. The IDE window title bar indicates (Running) or (Debugging) to indicate that a project is no longer in design time.

Writing Your First Visual Basic Project

For your first VB project, you will create a form with three controls (see Figure 1.10). This simple project will display the message “Hello World” in a label when the user clicks the *Push Me* button and will terminate when the user clicks the *Exit* button.

Set Up Your Workspace

Before you can begin a project, you must run the Visual Studio IDE. You also may need to customize your workspace.

Figure 1.9

Help displays in a new window, independent of the Visual Studio IDE window.

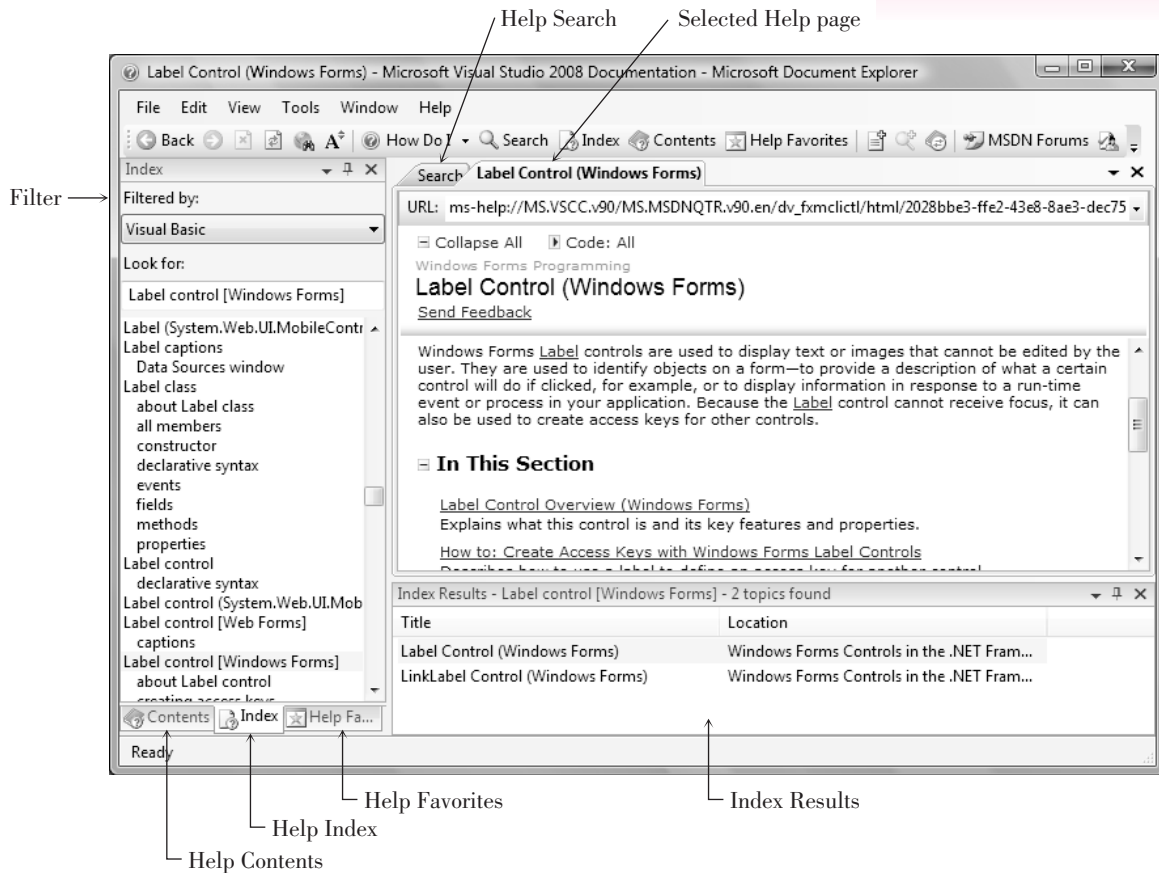


Figure 1.10

The Hello World form. The “Hello World” message will appear in a label when the user clicks on the Push Me button. The label does not appear until the button is pressed.



Run Visual Studio

These instructions assume that Visual Studio is installed in the default location. If you are running in a classroom or lab, the program may be installed in an alternate location, such as directly on the desktop.

- STEP 1:** Click the Windows *Start* button and move the mouse pointer to *All Programs*.
- STEP 2:** Locate *Microsoft Visual Basic 2008 Express Edition* or *Microsoft Visual Studio 2008*.
- STEP 3:** If a submenu appears, select *Microsoft Visual Studio 2008*. Visual Studio (VS) will start and display the Start Page (refer to Figure 1.4). If you are using Visual Studio Professional Edition and this is the first time that VS has been opened on this computer, you may need to select *Visual Basic Development Settings* from the *Choose Default Environment Setting* dialog box (refer to Figure 1.3).

Note: The VS IDE can be customized to not show the Start Page when it opens.

Start a New Project

- STEP 1:** Select *File / New Project*. The *New Project* dialog box opens (refer to Figure 1.5). Make sure that *Visual Basic* and *Windows* are selected for *Project types* and *Windows Forms Application* is selected for the template. If you are using Visual Basic Express, the dialog box differs slightly and you don't have to choose the language, but you can still choose a Windows Forms Application.
- STEP 2:** Enter "HelloWorld" (without the quotes) for the name of the new project (Figure 1.11) and click the *OK* button. The new project opens (Figure 1.12). At this point, your project is stored in a temporary directory. You specify the location for the project later when you save it.

Note: Your screen may look significantly different from the figure since the environment can be customized.

Figure 1.11

Enter the name for the new project.

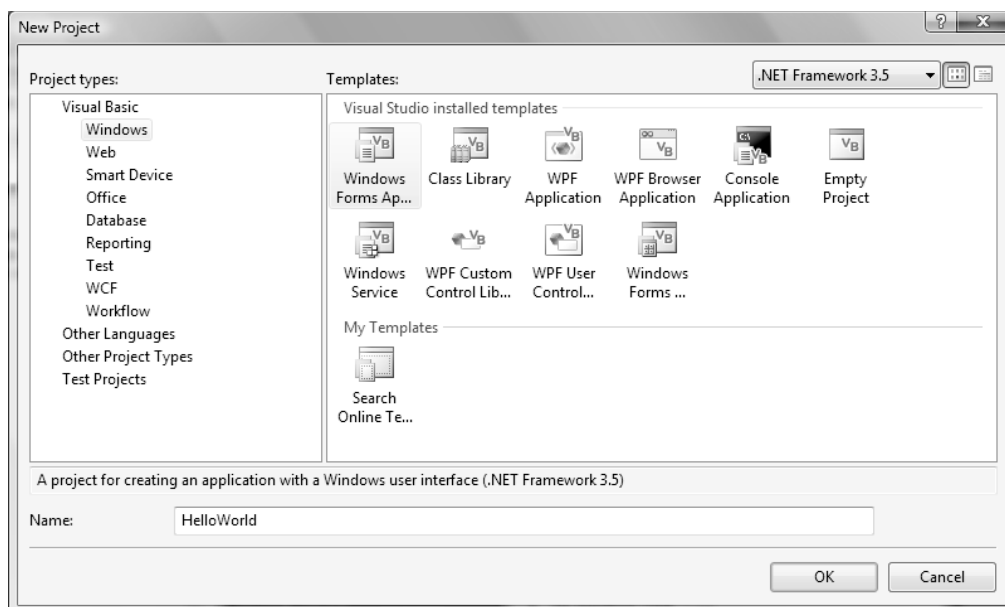
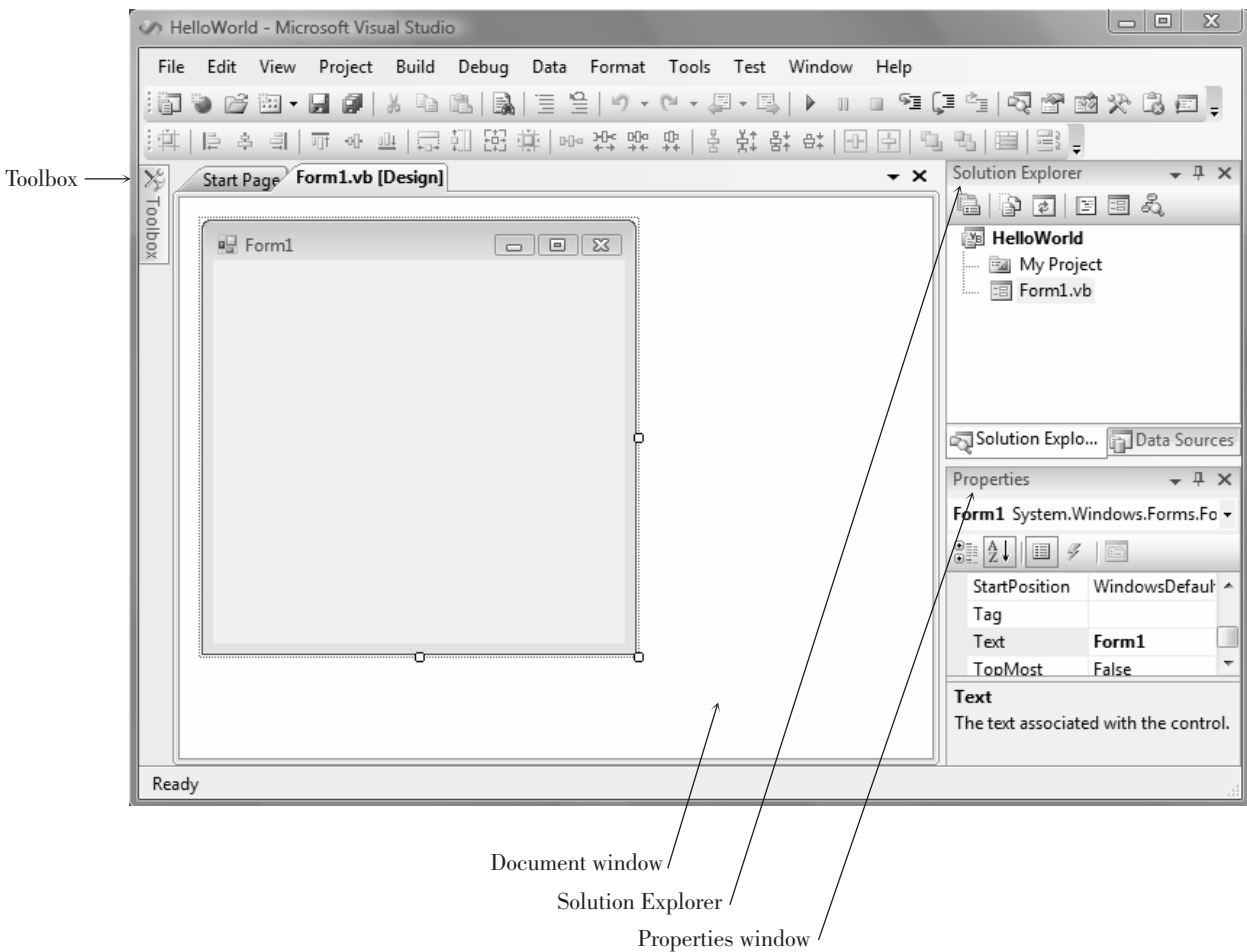


Figure 1.12

The Visual Studio IDE with the new HelloWorld project.



Set Up Your Environment

In this section, you will customize the environment. For more information on customizing windows, floating and docking windows, and altering the location and contents of the various windows, see Appendix C.

STEP 1: Reset the IDE's default layout by choosing *Window / Reset Window Layout* and respond Yes to the confirmation. The IDE should now match Figure 1.12.

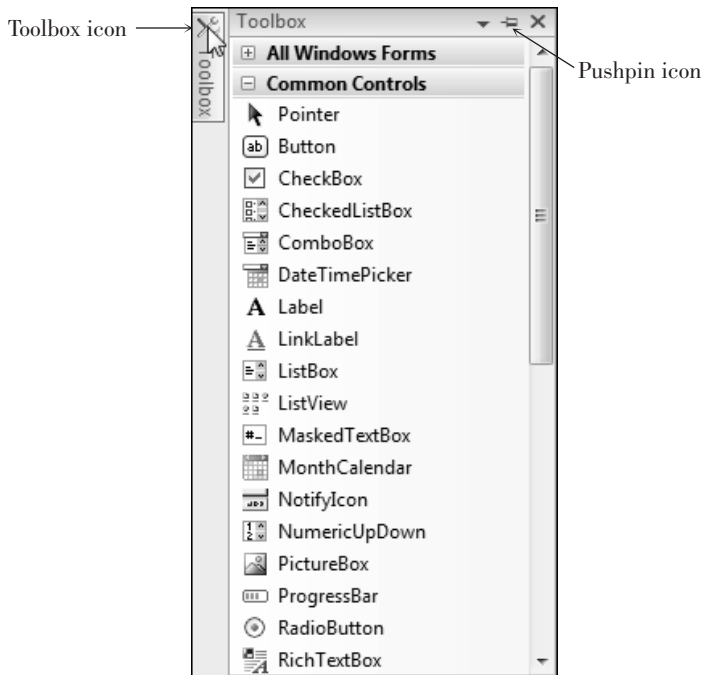
Note: If the Data Sources window appears on top of the Solution Explorer window, click on the Solution Explorer tab to make it appear on top.

STEP 2: Point to the icon for the toolbox at the left of the IDE window. The Toolbox window pops open. Notice the pushpin icon at the top of the window (Figure 1.13); clicking this icon pins the window open rather than allowing it to AutoHide.

STEP 3: Click the AutoHide pushpin icon for the Toolbox window; the toolbox will remain open.

Figure 1.13

The Toolbox window.

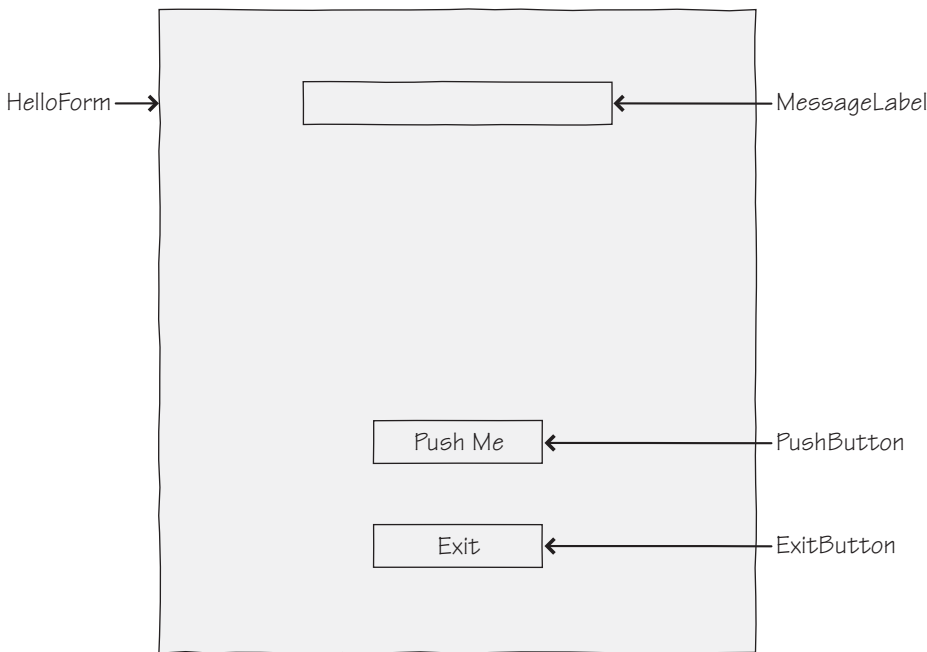


Plan the Project

The first step in planning is to design the user interface. Figure 1.14 shows a sketch of the form that includes a label and two buttons. You will refer to the sketch as you create the project.

Figure 1.14

A sketch of the HelloWorld form for planning.



The next two steps, planning the properties and the code, have already been done for this first sample project. You will be given the values in the steps that follow.

Define the User Interface

Set Up the Form

Notice that the new form in the Document window has all the standard Windows features, such as a title bar, maximize and minimize buttons, and a close button.

STEP 1: Resize the form in the Document window: Drag the handle in the lower-right corner down and to the right (Figure 1.15).

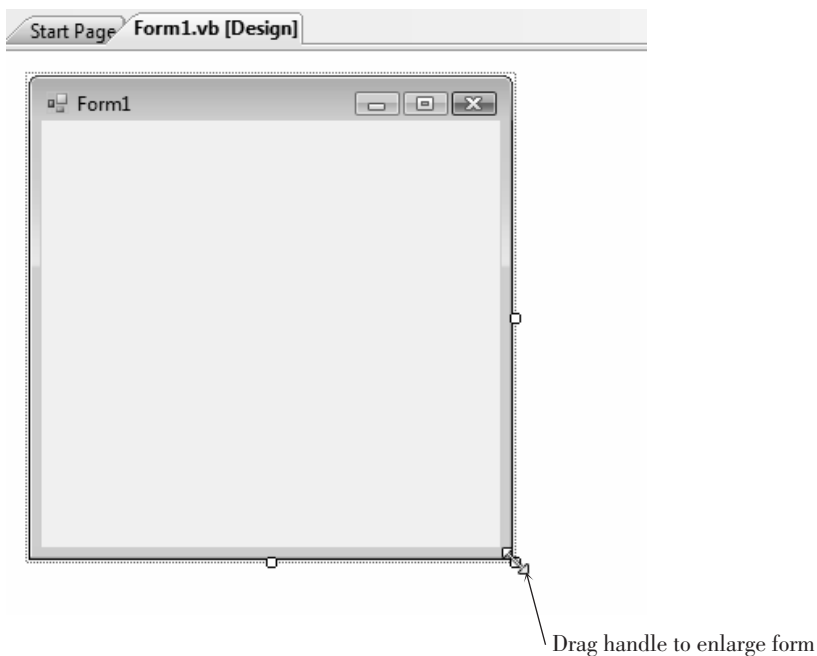


Figure 1.15

Make the form larger by dragging its lower-right handle diagonally. The handles disappear as you drag the corner of the form.

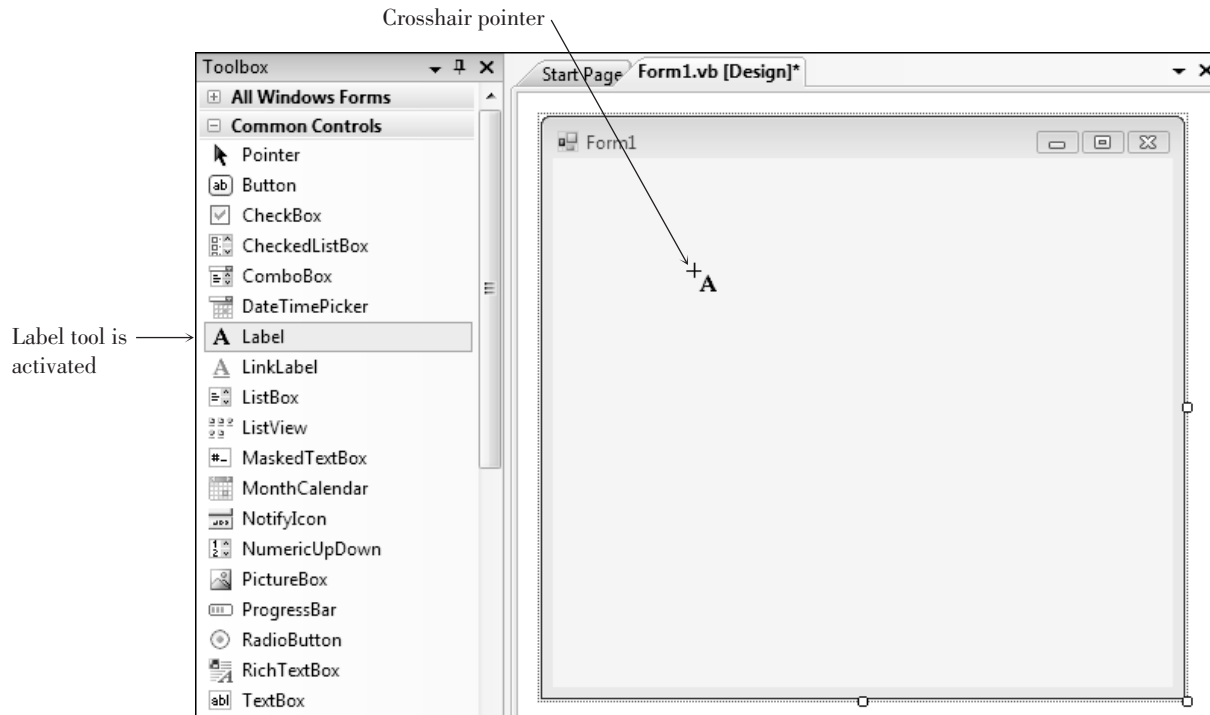
Place Controls on the Form

You are going to place three controls on the form: a **Label** and two **Buttons**.

STEP 1: Point to the Label tool in the toolbox and click. Then move the pointer over the form. Notice that the pointer becomes a crosshair with a big A and the Label tool appears selected, indicating it is the active tool (Figure 1.16).

Figure 1.16

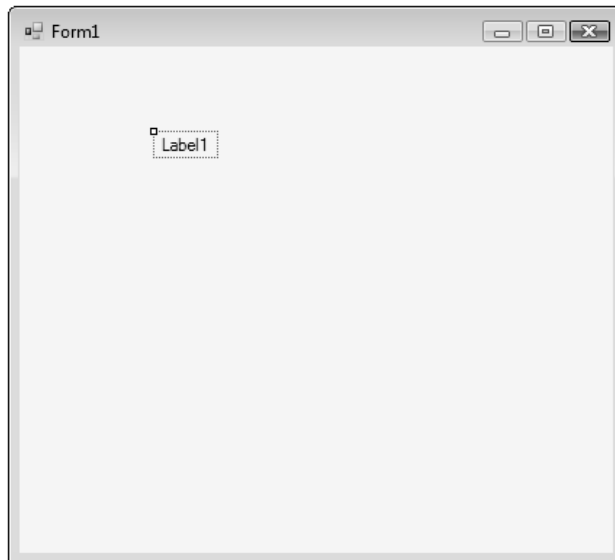
When you click on the Label tool in the toolbox, the tool's button is activated and the mouse pointer becomes a crosshair.



STEP 2: Point to a spot where you want the left edge of the label and click. The label and its default contents (Label1) will appear (Figure 1.17).

Figure 1.17

The newly created label appears outlined, indicating that it is selected. Notice that the contents of the label are set to Label1 by default.



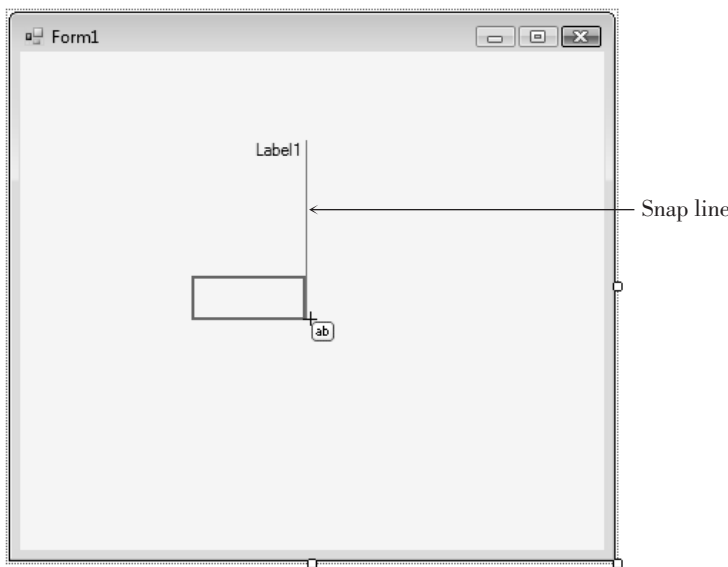
As long as the label is selected, you can press the Delete key to delete it, or drag it to a new location.

You can tell that a label is selected; it has a dotted border as shown in Figure 1.17 when the `AutoSize` property is `True` [the default] or sizing handles if you set the `AutoSize` property to `False`.

STEP 3: Place a button on the form using one of two techniques: (1) You can click on the `Button` tool in the toolbox, position the crosshair pointer for one corner of the button, and drag to the diagonally opposite corner (Figure 1.18); or (2) you can drag and drop the tool from the toolbox, which creates a button of the default size. The new button should appear selected and have **resizing handles**. The blue lines that appear are called **snap lines**, which can help you align your controls.

Figure 1.18

Select the `Button` tool and drag diagonally to create a new `Button` control. The blue snap lines help to align controls.



While a control is selected, you can delete it or move it. If it has resizing handles, you can also resize it. Refer to Table 1.1 for instructions for selecting, deleting, resizing, and moving controls. Click outside of a control to deselect it.

Selecting, Deleting, Moving, and Resizing Controls on a Form

Table 1.1

Select a control	Click on the control.
Delete a control	Select the control and then press the <code>Delete</code> key on the keyboard.
Move a control	Select the control, point inside the control (not on a handle), press the mouse button, and drag it to a new location.
Resize a control	Make sure the control is selected and has resizing handles; then either point to one of the handles, press the mouse button, and drag the handle; or drag the form's bottom border to change the height or the side border to change the width. Note that the default format for Labels does not allow resizing.

STEP 4: Create another button using another technique: While the first button is still selected, point to the Button tool in the toolbox and double-click. A new button of the default size will appear on top of the last-drawn control (Figure 1.19).

STEP 5: Keep the new button selected, point anywhere inside the button (not on a handle), and drag the button below your first button (Figure 1.20).

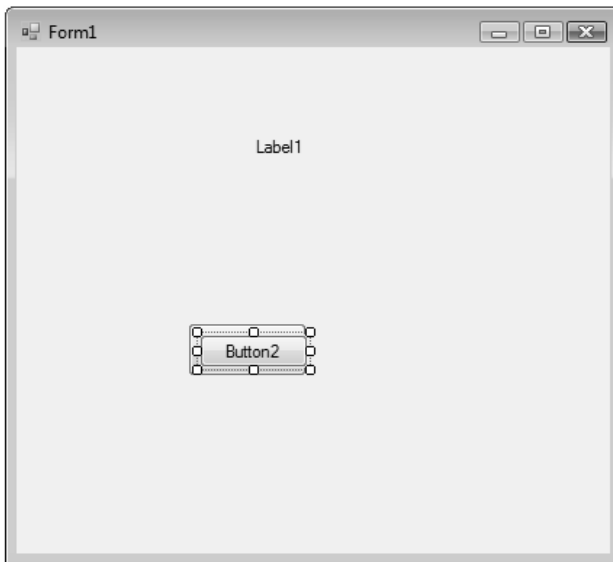


Figure 1.19

Place a new button on the form by double-clicking the Button tool in the toolbox. The new button appears on top of the previously selected control.

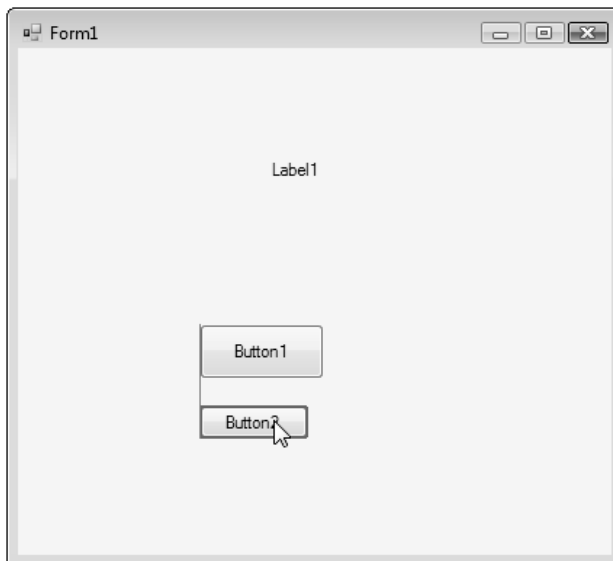


Figure 1.20

Drag the new button (Button2) below Button1.

TIP

If no control is selected when you double-click a tool, the new control is added to the upper-left corner of the form. ■

STEP 6: Select each control and move and resize the controls as necessary. Make the two buttons the same size and line them up. Use the snap lines to help with the size and alignment. Note that you can move but not resize the label.

STEP 7: Point to one of the controls and click the right mouse button to display a **context menu**. On the context menu, select *Lock Controls* (Figure 1.21). Locking prevents you from accidentally moving the controls. When your controls are locked, a selected control has no handles, but instead has a small lock symbol in the upper-left corner.

Note: You can unlock the controls at any time if you wish to redesign the form. Just click again on *Lock Controls* on the context menu to deselect it.

At this point you have designed the user interface and are ready to set the properties.

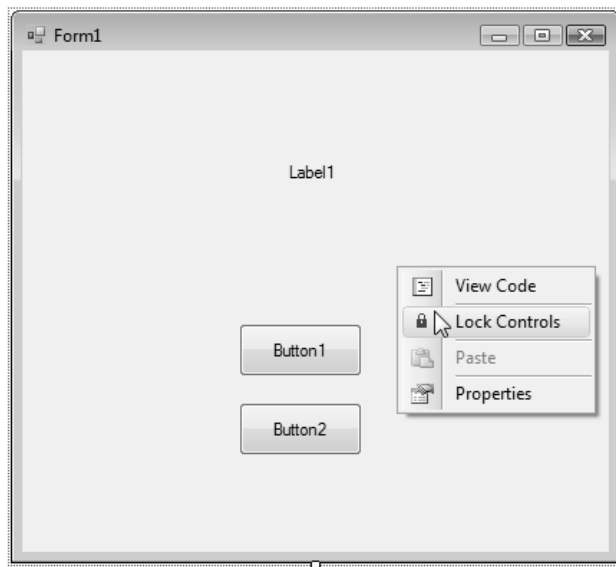


Figure 1.21

*After the controls are placed into the desired location, lock them in place by selecting *Lock Controls* from the context menu. Remember that context menus differ depending on the current operation and system setup.*

Set Properties

Set the Name and Text Properties for the Label

STEP 1: Click on the label you placed on the form; an outline appears around the control. Next click on the title bar of the Properties window to make it the active window (Figure 1.22). *Note:* If the Properties window is not displaying, select *View / Properties Window*.

Notice that the Object box at the top of the Properties window is showing *Label1* (the name of the object) and *System.Windows.Forms.Label* as the class of the object. The actual class is *Label*; *System.Windows.Forms* is called the **namespace**, or the hierarchy used to locate the class.

STEP 2: In the Properties window, click on the Alphabetic button to make sure the properties are sorted in alphabetic order. Then select the Name property, which appears near the top of the list. Click on *(Name)* and notice that the Settings box shows *Label1*, the default name of the label (Figure 1.23).



If the Properties window is not visible, you can choose *View / Properties Window* from the menu or press the F4 shortcut key to show it. ■

Figure 1.22

The currently selected control is shown in the Properties window.

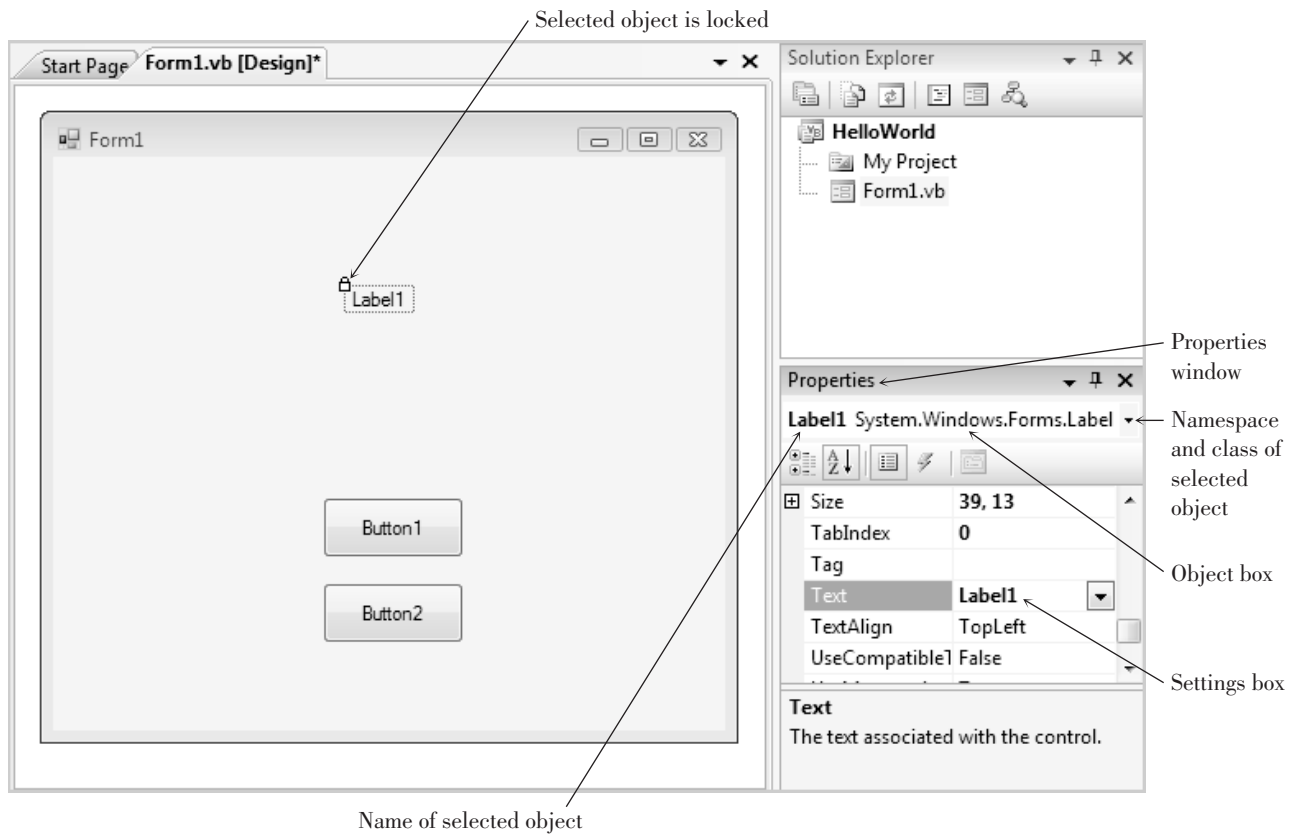
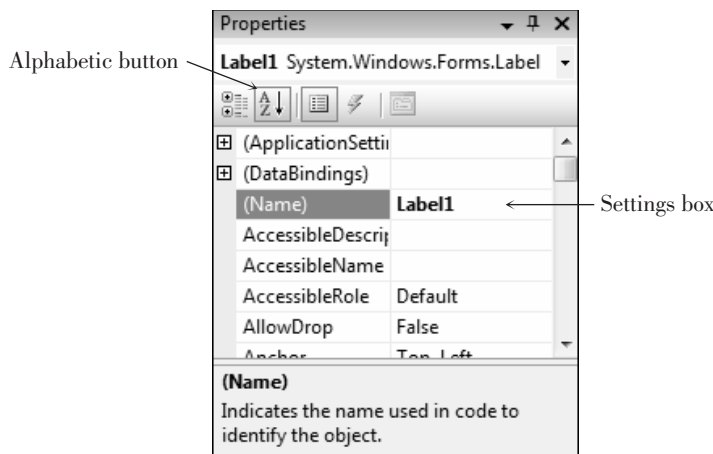


Figure 1.23

The Properties window. Click on the Name property to change the value in the Settings box.

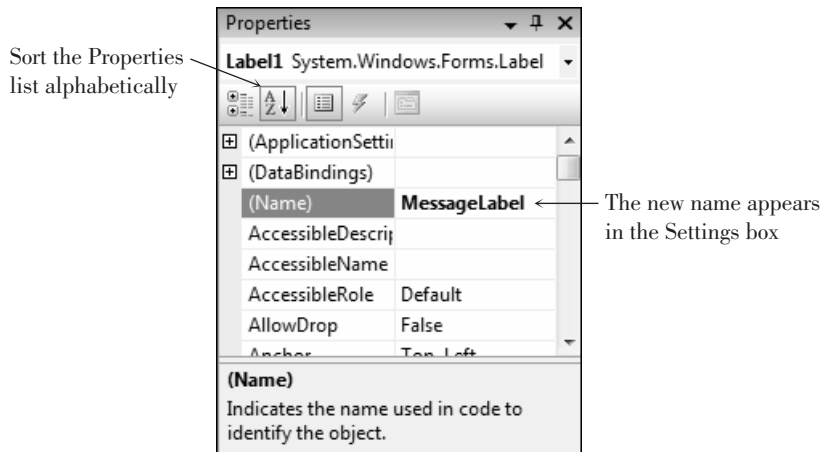


STEP 3: Type “MessageLabel” (without the quotation marks). See Figure 1.24. As a shortcut, you may wish to delete the 1 from the end of Label1, press the Home key to get to the beginning of the word, and then type “Message”.

After you change the name of the control and press Enter or Tab, you can see the new name in the Object box’s drop-down list.

Figure 1.24

Type “MessageLabel” into the Settings box for the Name property.



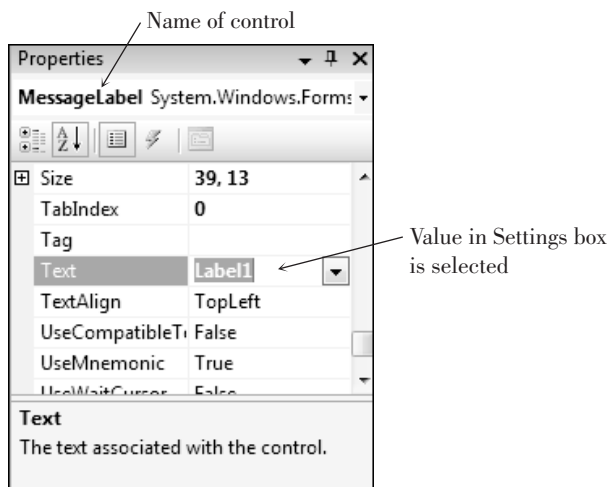
STEP 4: Click on the Text property to select it. (Scroll the Properties list if necessary.)

The **Text property** of a control determines what will be displayed on the form. Because nothing should display when the program begins, you must delete the value of the Text property (as described in the next two steps).

STEP 5: Double-click on *Label1* in the Settings box; the entry should appear selected (highlighted). See Figure 1.25.

Figure 1.25

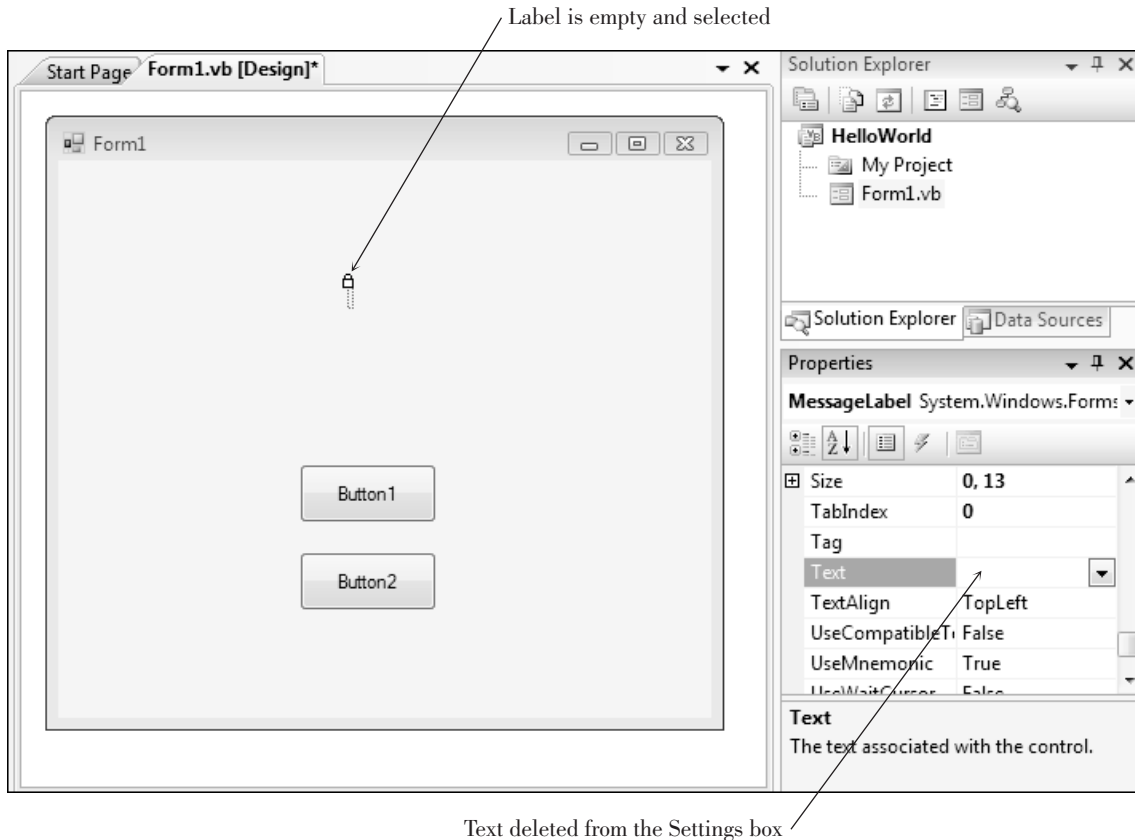
Double-click in the Settings box to select the entry.



STEP 6: Press the Delete key to delete the value of the Text property. Then press Enter and notice that the label on the form nearly disappears. All you see is the lock symbol and a very small dotted line (Figure 1.26), and if you click anywhere else on the form, which deselects the label, you cannot see it at all.

Figure 1.26

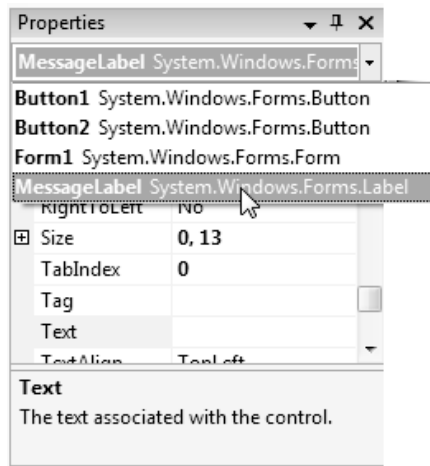
Delete the value for the Text property from the Settings box; the label on the form also appears empty and the control shrinks in size because the *AutoSize* property is set to *True*.



Labels have an *AutoSize* property, which is set to *True* by default. Labels shrink or grow to adjust for the *Text* property. You can set the *AutoSize* property to *False* if you want to specify the Label's size, in which case you can see the outline of the Label on the form.

Note that if you want text to flow to multiple lines in a label, you should set its *AutoSize* property to *False* and use the sizing handle to make the label large enough.

If you need to select the label after deselecting it, use the Properties window: Drop down the Object list at the top of the window; you can see a list of all controls on the form and can make a selection (Figure 1.27).

**Figure 1.27**

Drop down the Object box in the Properties window to select any control on the form.



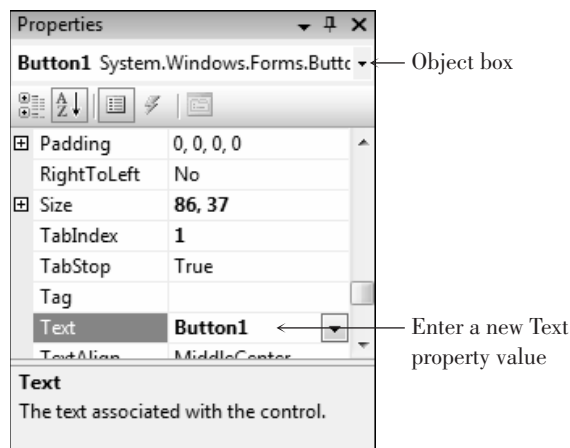
TIP Don't confuse the Name property with the Text property. You will use the Name property to refer to the control in your Basic code. The Text property determines what the user will see on the form. Visual Basic sets both of these properties to the same value by default, and it is easy to confuse them. ■

Note: As an alternate technique for deleting a property, you can double-click on the property name, which automatically selects the entry in the Settings box. Then you can press the Delete key or just begin typing to change the entry.

Set the Name and Text Properties for the First Button

STEP 1: Click on the first button (Button1) to select it and then look at the Properties window. The Object box should show the name (*Button1*) and class (*System.Windows.Forms.Button*) of the button. See Figure 1.28.

Problem? If you should double-click and code appears in the Document window, simply click on the *Form1.vb [Design]* tab at the top of the window.

Figure 1.28

Change the properties of the first button.

STEP 2: Change the Name property of the button to “PushButton” (without the quotation marks).

Although the project would work fine without this step, we prefer to give this button a meaningful name, rather than use Button1, its

default name. The guidelines for naming controls appear later in this chapter in the section “Naming Rules and Conventions for Objects.”

STEP 3: Change the Text property to “Push Me” (without the quotation marks). This step changes the words that appear on top of the button.

Set the Name and Text Properties for the Second Button

STEP 1: Select Button2 and change its Name property to “ExitButton”.

STEP 2: Change the Text property to “Exit”.

Change Properties of the Form

STEP 1: Click anywhere on the form, except on a control. The Properties window Object box should now show the form as the selected object (*Form1* as the object’s name and *System.Windows.Forms.Form* as its class).

STEP 2: Change the Text property to “Hello World by Your Name” (again, no quotation marks and use your own name).

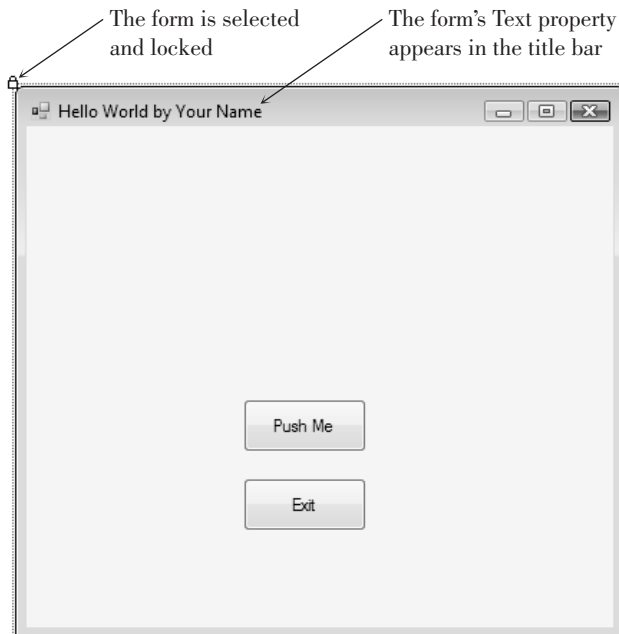
The Text property of a form determines the text to appear in the title bar. Your screen should now look like Figure 1.29.



Always set the Name property of controls before writing code. Although the program will still work if you reverse the order, the method names won’t match the control names, which can cause confusion. ■

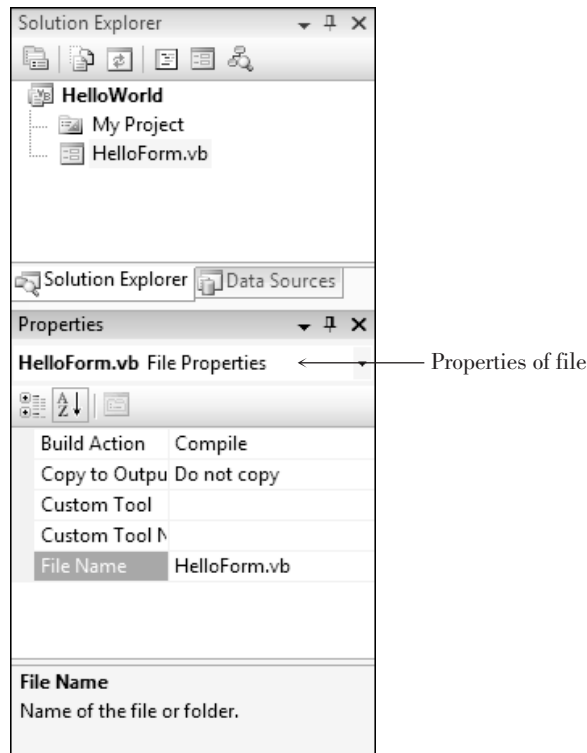
Figure 1.29

Change the form’s Text property to set the text that appears in the form’s title bar.



STEP 3: In the Properties window, click on the StartPosition property and notice the arrow on the property setting, indicating a drop-down list. Drop down the list and select *CenterScreen*. This will make your form appear in the center of the screen when the program runs.

STEP 4: In the Solution Explorer, right-click on Form1.vb and choose *Rename* from the context menu. Change the filename to “HelloForm.vb”, making sure to retain the .vb extension. Press Enter when finished and click *Yes* on the confirmation dialog box. This changes the name of the file that saves to disk as well as the name of the class. (See Figure 1.30.)

**Figure 1.30**

The Properties window shows the file's properties with the new name for the file. You can change the filename in the Properties window or the Solution Explorer.

STEP 5: Click on the form in the Document window, anywhere except on a control. The name of the file appears on the tab at the top of the Designer window, and the Properties window shows properties for the form's class, not the file. The VB designer changed the name of the form's class to match the name of the file (Figure 1.31).

✓ TIP

If you change the form's filename before changing the form's class name, the IDE automatically changes the form's class name to match the filename. It does not make the change if you have changed the form's class name yourself. ■

Write Code

Visual Basic Events

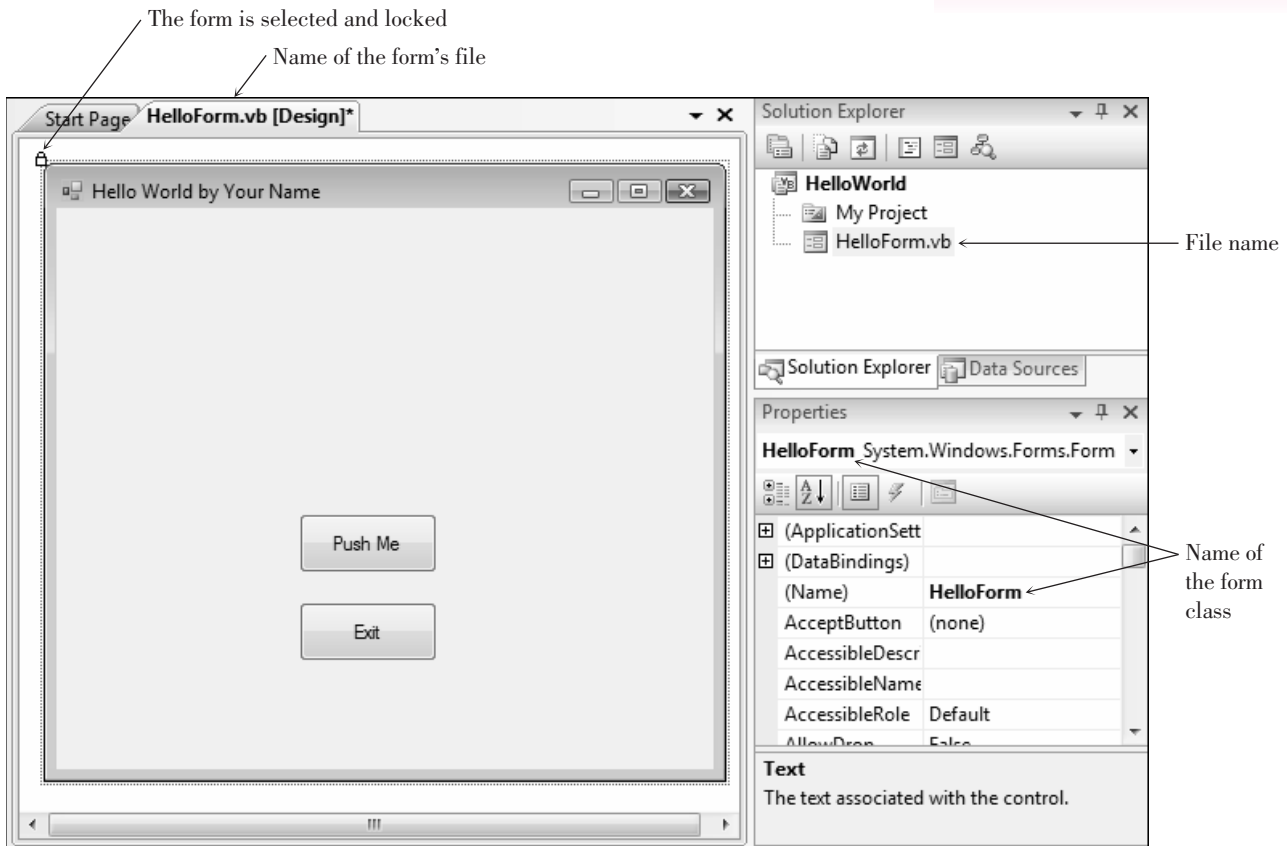
While your project is running, the user can do many things, such as move the mouse around; click on either button; move, resize, or close your form's window; or jump to another application. Each action by the user causes an event to occur in your Visual Basic project. Some events (like clicking on a button) you care about, and some events (like moving the mouse and resizing the window) you do not care about. If you write Basic code for a particular event, then Visual Basic will respond to the event and automatically execute your procedure. *VB ignores events for which no procedures are written.*

Visual Basic Event Procedures

You write code in Visual Basic in **procedures**. For now, each of your procedures will be a **sub procedure**, which begins with the words `Private Sub` and ends with `End Sub`. (Later you will also learn about other types of procedures.) Note that many programmers refer to sub procedures as subprograms or

Figure 1.31

The Properties window for the form. The form's class name now matches the name of the form's file.



subroutines. Subprogram is acceptable; subroutine is not because Basic actually has a different statement for a subroutine, which is not the same as a sub procedure. You also can refer to procedures as methods.

Visual Basic automatically names your **event procedures**. The name consists of the object name, an underscore (_), and the name of the event. For example, the Click event for your button called PushButton will be PushButton_Click. For the sample project you are writing, you will have a PushButton_Click procedure and an ExitButton_Click procedure. Note that another way to refer to these methods is to call them “event-handling methods,” such as the PushButton_Click event-handling method.

Visual Basic Code Statements

This first project requires two Visual Basic statements: the **remark** and the **assignment statement**. You also will execute a method of an object.

The Remark Statement

Remark statements, sometimes called *comments*, are used for project documentation only. They are not considered “executable” and have no effect when the project runs. The purpose of remarks is to make the project more readable and understandable by the people who read it.

Good programming practices dictate that programmers include remarks to clarify their projects. Every procedure should begin with a remark that describes its purpose. Every project should have remarks that explain the purpose of the program and provide identifying information such as the name of the programmer and the date the program was written and/or modified. In addition, it is a good idea to place remarks within the logic of a project, especially if the purpose of any statements might be unclear.

When you try to read someone else's code, or your own after a period of time, you will appreciate the generous use of remarks.

Visual Basic remarks begin with an apostrophe. Most of the time your remarks will be on a separate line that starts with an apostrophe. You can also add an apostrophe and a remark to the right end of a line of code.

The Remark Statement—Examples

Examples

```
' This project was written by Jonathon Edwards.
' Exit the project.
MessageLabel.Text = "Hello World" ' Assign the message to the Text property.
```

The Assignment Statement

The assignment statement assigns a value to a property or variable (you learn about variables in Chapter 3). Assignment statements operate from right to left; that is, the value appearing on the right side of the equal sign is assigned to the property named on the left of the equal sign. It is often helpful to read the equal sign as “is replaced by.” For example, the following assignment statement would read “MessageLabel.Text is replaced by Hello World.”

```
MessageLabel.Text = "Hello World"
```

The Assignment Statement—General Form

General Form

```
Object.Property = value
```

The value named on the right side of the equal sign is assigned to (or placed into) the property named on the left.

The Assignment Statement—Examples

Examples

```
TitleLabel.Text = "A Snazzy Program"
AddressLabel.Text = "1234 South North Street"
MessageLabel.AutoSize = True
NumberInteger = 12
```

Notice that when the value to assign is some actual text (called a *literal*), it is enclosed in quotation marks. This convention allows you to type any combination of alpha and numeric characters. If the value is numeric, do not enclose it in quotation marks. And do not place quotation marks around the terms True and False, which Visual Basic recognizes as special key terms.

Ending a Program by Executing a Method

To execute a method of an object, you write:

```
Object.Method()
```

Notice that methods always have parentheses. Although this might seem like a bother, it's helpful to distinguish between properties and methods: Methods always have parentheses; properties don't.

Examples

```
HelloButton.Hide()
MessageLabel.Show()
```

To execute a method of the current object (the form itself), you use the `Me` keyword for the object. And the method that closes the form and terminates the project execution is `Close`.

```
Me.Close()
```

In most cases, you will include `Me.Close()` in the sub procedure for an Exit button or an *Exit* menu choice.

Note: The keyword `Me` refers to the current object. You can omit `Me` since a method without an object reference defaults to the current object.



TIP

If you don't type the parentheses after a method, the editor adds it for you, for most (but not all) methods. ■

Code the Event Procedures for Hello World

Code the Click Event for the Push Me Button

STEP 1: Double-click the *Push Me* button. The Visual Studio editor opens with the first and last lines of your sub procedure already in place, with the insertion point indented inside the sub procedure (Figure 1.32).

Figure 1.32

The Editor window, showing the first and last lines of the `PushButton_Click` event procedure.



STEP 2: Type this remark statement:

```
' Display the Hello World message.
```

Notice that the editor automatically displays remarks in green (unless you or someone else has changed the color with an Environment option).

Follow good coding conventions and indent all lines between `Private Sub` and `End Sub`. The smart editor attempts to help you follow this convention. Also, always leave a blank line after the remarks at the top of a sub procedure.

STEP 3: Press Enter twice and then type this assignment statement:

```
MessageLabel.Text = "Hello World"
```

Note: When you type the names of objects and properties, allow IntelliSense to help you. When you type the first character of a name, such as the “M” of “MessageLabel”, IntelliSense pops up a list of possible object names from your program (Figure 1.33). When several items match the first letter, you can type additional characters until you get a match, or you can use your keyboard down arrow or the mouse to highlight the correct item. To accept the correct item when it is highlighted, press the punctuation character that should follow the item, such as the period, spacebar, equal sign, Tab key, or Enter key, or double-click the item with your mouse. For example, accept “MessageLabel” by pressing the period and accept “Text” by pressing the spacebar, because those are the characters that follow the selected items.



Figure 1.33

IntelliSense pops up to help you. Select the correct item from the list and press the period, spacebar, Tab key, or Enter key to accept the text.

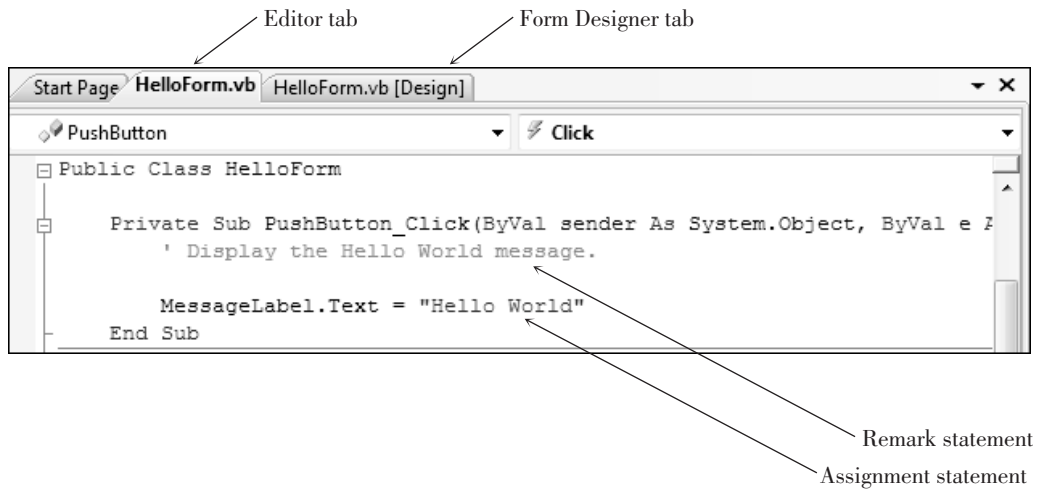
The assignment statement

```
MessageLabel.Text = "Hello World"
```

assigns the literal “Hello World” to the Text property of the control called MessageLabel. Compare your screen to Figure 1.34.

Figure 1.34

Type the remark and assignment statement for the `PushButton_Click` event procedure.



STEP 4: Return to the form's design view (Figure 1.29) by clicking on the `HelloForm.vb [Design]` form designer tab on the Document window (refer to Figure 1.34).

Code the Click Event for the Exit Button

STEP 1: Double-click the `Exit` button to open the editor for the `ExitButton_Click` event.

STEP 2: Type this remark:

```
' Exit the project.
```

STEP 3: Press Enter twice and type this Basic statement:

```
Me.Close()
```

Note: You can omit the parentheses; the smart editor will add them for you.

STEP 4: Make sure your code looks like the code shown in Figure 1.35.

✓ TIP

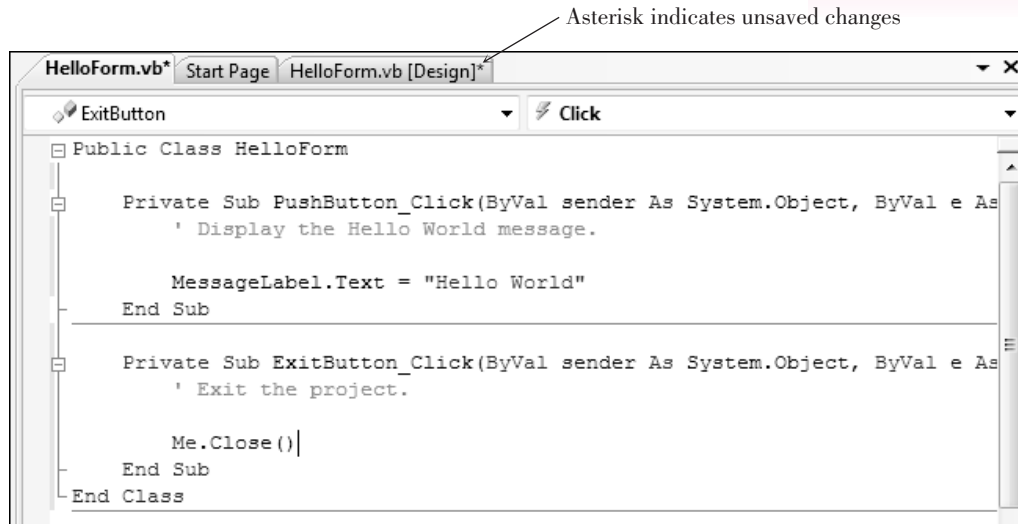
Allow the Editor and IntelliSense to help you. If the IntelliSense list does not pop up, you probably misspelled the name of the control. And don't worry about capitalization when you type the name of an object; if the name matches a defined object, the Editor fixes the capitalization. ■

✓ TIP

Accept an entry from the IntelliSense popup list by typing the punctuation that follows the entry or by pressing the Enter key. You can also scroll the list and select with your mouse. ■

Figure 1.35

Type the code for the `ExitButton_Click` event procedure. Notice that an asterisk appears on the tab at the top of the window, indicating that there are unsaved changes in the file.



Run the Project

After you have finished writing the code, you are ready to run the project. Use one of these three techniques:

1. Open the *Debug* menu and choose *Start Debugging*.
2. Press the *Start Debugging* button on the toolbar.
3. Press F5, the shortcut key for the *Start Debugging* command.

Start the Project Running

STEP 1: Choose one of the three methods previously listed to start your project running (Figure 1.36).

Problems? See “Finding and Fixing Errors” later in this chapter. You must correct any errors and restart the program.

If all went well, the Visual Studio title bar now indicates that you are running.

Click the Push Me Button

STEP 1: Click the *Push Me* button. Your “Hello World” message appears in the label (Figure 1.37).

Click the Exit Button

STEP 1: Click the *Exit* button. Your project terminates, and you return to design time.



TIP

If your form disappears during run time, click its button on the Windows task bar. ■

Figure 1.36

The form of the running application.

**Figure 1.37**

Click the Push Me button and "Hello World" appears in the label.



Save Your Work

Of course, you must always save your work often. Except for a very small project like this one, you will usually save your work as you go along. Unless you (or someone else) has changed the setting in the IDE's *Options* dialog box, your files are automatically saved each time you build (compile) or execute (run) your project *after* your initial save. You also can save the files as you work.

TIP

Click the *Save All* toolbar button to quickly save all of your work. ■

Save the Files

STEP 1: Open the Visual Studio *File* menu and choose *Save All*. This option saves the current form, project, and solution files. You already selected the name for the project when you first created the project. Make sure to set the location to the folder in which you want to store the project. Press the *Browse* button to select a location other than the one specified.

Clear the check box for *Create directory for solution*. The IDE automatically creates a new folder for the solution; checking this check box creates a folder within a folder.

When you have set the name and location and cleared the check box, click *Save*.

Note: Do not attempt to choose a new name for the project to save a modified version. If you want to move or rename a project, close it first. See Appendix C for help.

Close the Project

STEP 1: Open the *File* menu and choose *Close Project*. If you haven't saved since your last change, you will be prompted to save.

Open the Project

Now is the time to test your save operation by opening the project from disk. You can choose one of three ways to open a saved project:

- Select *Open Project* from the Visual Studio *File* menu and browse to find your *.sln* file.
- Choose the project from the *Files / Recent Projects* menu item.
- Choose the project from the Start Page.

Open the Project File

STEP 1: Open your project by choosing one of the previously listed methods. Remember that the file to open is the solution (*.sln*) file.

If you do not see your form on the screen, check the Solution Explorer window—it should say *HelloWorld* for the project. Select the icon for your form: *HelloForm.vb*. You can double-click the icon or single-click and click on the *View Designer* button at the top of the Solution Explorer (Figure 1.38); your form will appear in the Designer window. Notice that you also can click on the *View Code* button to display your form's code in the Editor window.

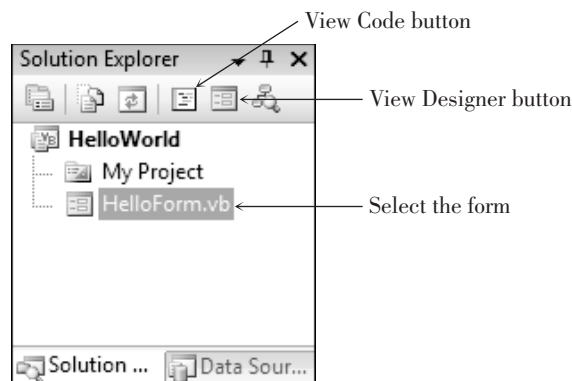


Figure 1.38

To display the form layout, select the form name and click on the *View Designer* button, or double-click on the form name. Click on the *View Code* button to display the code in the editor.

Modify the Project

Now it's time to make some changes to the project. We'll change the size of the "Hello World" message, display the message in two different languages, add a *Print* button, and display the programmer name (that's you) on the form.

Change the Size and Alignment of the Message

- STEP 1:** Right-click one of the form's controls to display the context menu. If your controls are currently locked, select *Lock Controls* to unlock the controls so that you can make changes.
- STEP 2:** Drop down the Object list at the top of the Properties window and select *MessageLabel*, which will make the tiny label appear selected.
- STEP 3:** Scroll to the *Font* property in the Properties window. The *Font* property is actually a *Font* object that has a number of properties. To see the *Font* properties, click on the small plus sign on the left (Figure 1.39); the *Font* properties will appear showing the current values (Figure 1.40).

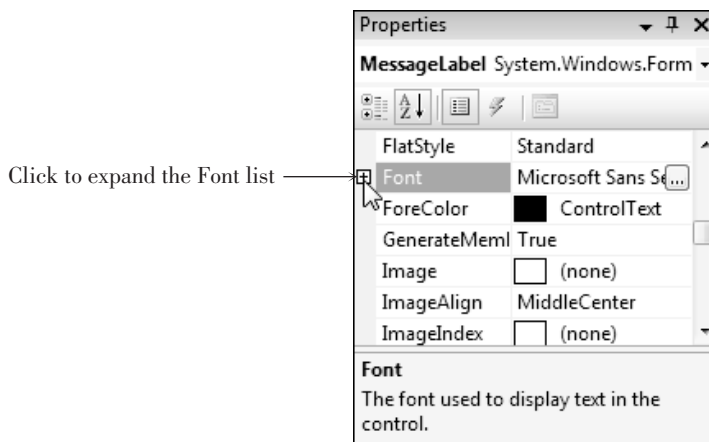


Figure 1.39

Click on the *Font*'s plus sign to view the properties of the *Font* object.

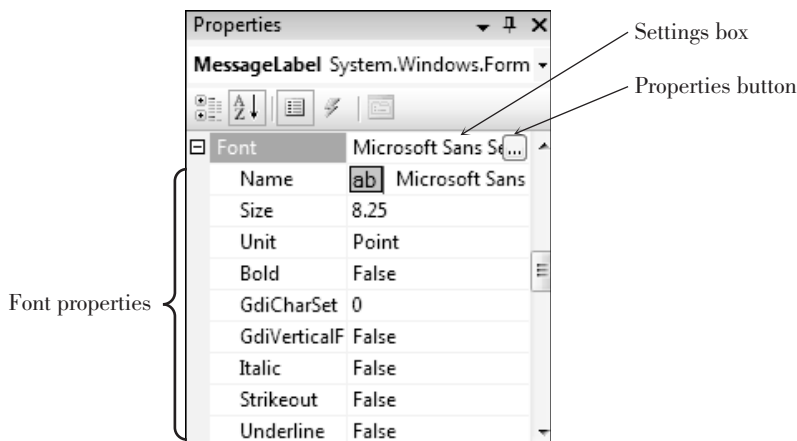


Figure 1.40

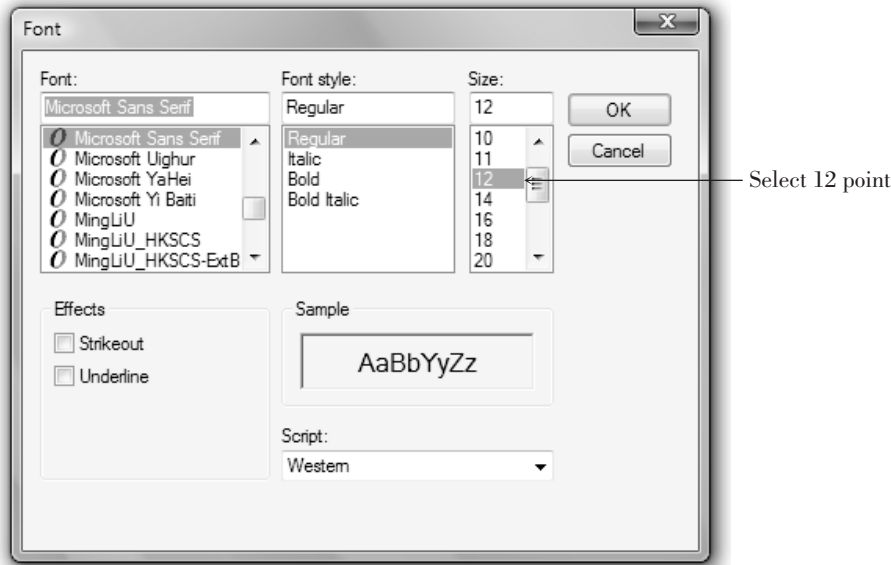
You can change the individual properties of the *Font* object.

You can change any of the Font properties in the Properties window, such as setting the Font's Size, Bold, or Italic properties. You also can display the *Font* dialog box and make changes there.

STEP 4: Click the Properties button for the font (the button with the ellipsis on top) to display the *Font* dialog box (Figure 1.41). Select 12 point if it is available. (If it isn't available, choose another number larger than the current setting.) Click OK to close the *Font* dialog box.

Figure 1.41

Choose 12 point on the *Font* dialog box.



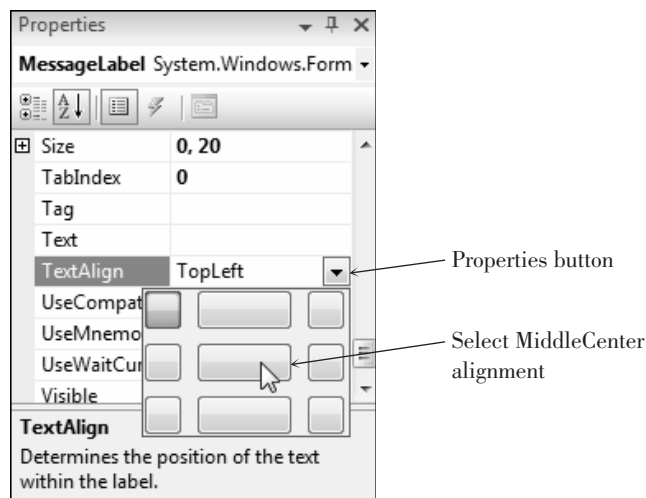
TIP

When you change a property from its default value, the property name appears bolded; you can scan down the property list and easily identify the properties that are changed from their default value. ■

STEP 5: Select the *TextAlign* property. The Properties button that appears with the down-pointing arrow indicates a drop-down list of choices. Drop down the list (Figure 1.42) and choose the center box; the alignment property changes to *MiddleCenter*.

Figure 1.42

Select the center box for the *TextAlign* property.



TIP

You can change the Font property of the form, which sets the default font for all objects on the form. ■

Add a New Label for Your Name

STEP 1: Click on the Label tool in the toolbox and create a new label along the bottom edge of your form (Figure 1.43). (You can resize the form if necessary, but you must unlock the controls first.)

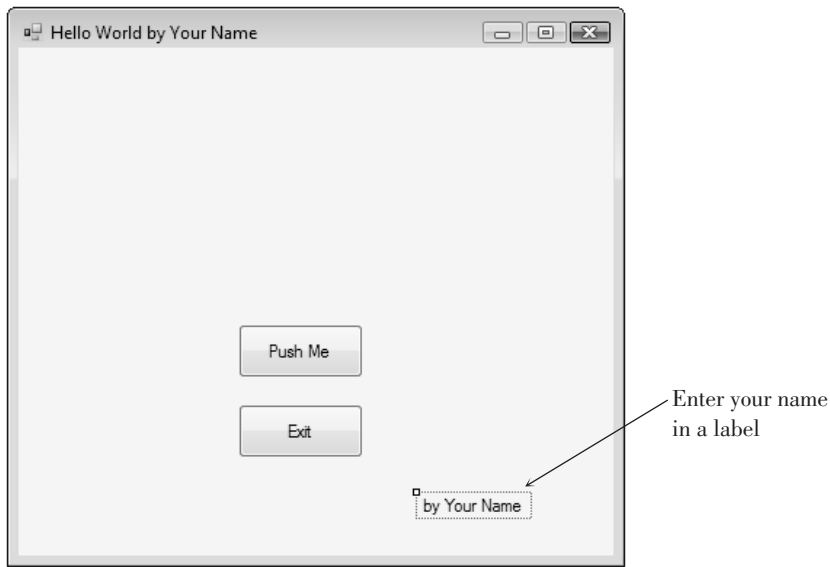


Figure 1.43

Add a new label for your name at the bottom of the form.

STEP 2: Change the label's Text property to "by Your Name". (Use your name and omit the quotation marks.)

Note: You do not need to change the name of this label because it will never be referred to in the code.

Change the Location and Text of the Push Me Button

Because we plan to display the message in one of two languages, we'll change the text on the *Push Me* button to "English" and move the button to allow for a second button.

STEP 1: Select the *Push Me* button and change its Text property to English.

STEP 2: Move the *English* button to the left to make room for a *Spanish* button (see Figure 1.44).

Note: If you cannot move the button, check your *Lock Controls* setting in the context menu.

Figure 1.44

Move the English button to the left and add a Spanish button.



Add a Spanish Button

STEP 1: Add a new button. Move and resize the buttons as necessary, referring to Figure 1.44.

STEP 2: Change the Name property of the new button to SpanishButton.

STEP 3: Change the Text property of the new button to Spanish.

Add an Event Procedure for the Spanish Button

STEP 1: Double-click on the *Spanish* button to open the editor for SpanishButton_Click.

STEP 2: Add a remark:

```
' Display the Hello World message in Spanish.
```

STEP 3: Press Enter twice and type the following Basic code line:

```
MessageLabel.Text = "Hola Mundo"
```

STEP 4: Return to design view.

Add a Print Button

STEP 1: Add another button and name it PrintButton.

STEP 2: Change the button's Text property to "Print".

STEP 3: Move and resize the buttons as necessary.

Add a PrintForm Component

Visual Basic 2008 includes a **PrintForm** component. This component was not included in previous versions of Visual Basic but was available as a separate download. You will add the PrintForm component to your form, but it is not a visible element, such as the controls you have already added.

TIP

An easy way to create multiple similar controls is to copy an existing control and paste it on the form. You can paste multiple times to create multiple controls. ■

You can choose to send the printer output to the printer or to the Print Preview window, which saves paper while you are testing your program.

- STEP 1:** Scroll down to the bottom of the toolbox and find the header for *Visual Basic PowerPacks*.
- STEP 2:** Click on the plus sign to open the section of the toolbox. You should see the PrintForm component listed (Figure 1.45).
- STEP 3:** Double-click on the PrintForm component. Or you can drag and drop the component on the form. In either case, the component appears in a new pane that opens at the bottom of the Form Designer (Figure 1.46). This pane, called the **component tray**, holds components that do not have a visual representation at run time. You will see more controls that use the component tray later in this text.

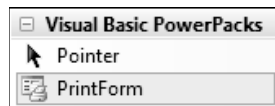


Figure 1.45

Open the Visual Basic PowerPacks section of the toolbox and double-click on the PrintForm component.

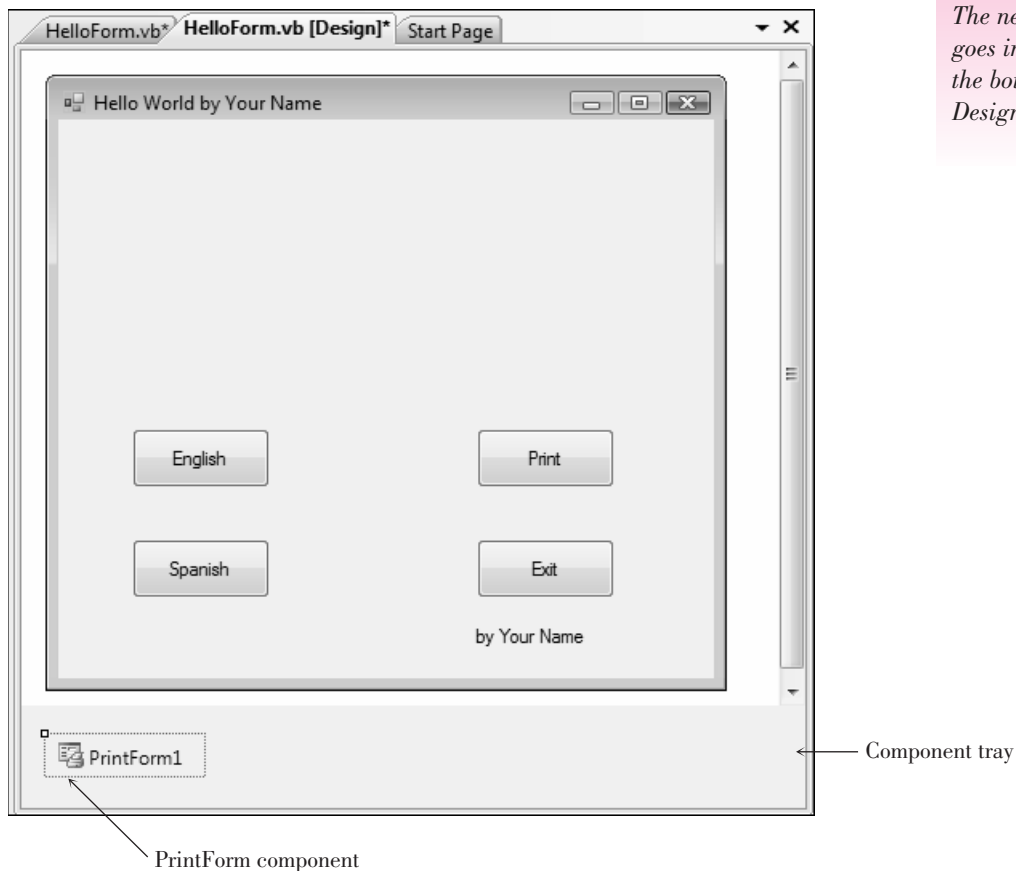


Figure 1.46

The new PrintForm component goes in the component tray at the bottom of the Form Designer window.

Add an Event Procedure for the Print Button

STEP 1: Double-click the *Print* button to open the editor.

STEP 2: Add the remark and press Enter twice.

```
' Print the form on the printer.
```

STEP 3: Allow IntelliSense to help you set the `PrintAction` property of the `PrintForm` component: Type “Printf” and the `PrintForm1` item is selected in the IntelliSense list. Then press the period to accept the name, and the list of possible properties and methods for that object pops up. Select `PrintAction` by using the keyboard down arrow, by typing enough letters to automatically select the property, or by clicking with the mouse.

Then accept `PrintAction` by pressing the spacebar or the equal sign. If you pressed the spacebar, type an equal sign, and the list of possible choices will pop up. Choose `Printing.PrintAction.PrintToPreview` and press the spacebar or the Enter key to accept it. Your line of code should look like this:

```
PrintForm1.PrintAction = Printing.PrintAction.PrintToPreview
```

STEP 4: Add the last line of code, which starts the print operation. The `Print` method sends the output to the device specified in the `PrintAction` property, which defaults to the printer.

```
PrintForm1.Print()
```

STEP 5: Return to design view.

Lock the Controls

STEP 1: When you are satisfied with the placement of the controls on the form, display the context menu and select *Lock Controls* again.

Save and Run the Project

STEP 1: Save your project again. You can use the *File / Save All* menu command or the *Save All* toolbar button.

STEP 2: Run your project again. Try clicking on the *English* button and the *Spanish* button.

Problems? See “Finding and Fixing Errors” later in this chapter.

STEP 3: Click on the *Print* button to test the Print Preview function.

STEP 4: Click the *Exit* button to end program execution.

Add Remarks

Good documentation guidelines require some more remarks in the project. Always begin each procedure with remarks that tell the purpose of the procedure. In addition, each project file needs identifying remarks at the top.

The **Declarations section** at the top of the file is a good location for these remarks.

STEP 1: Display the code in the editor and click in front of the first line (`Public Class HelloForm`). Make sure that you have an insertion point; if the entire first line is selected, press the left arrow to set the insertion point.

STEP 2: Press Enter to create a blank line.

Warning: If you accidentally deleted the first line, click Undo (or press Ctrl + Z) and try again.

STEP 3: Move the insertion point up to the blank line and type the following remarks, one per line (Figure 1.47):

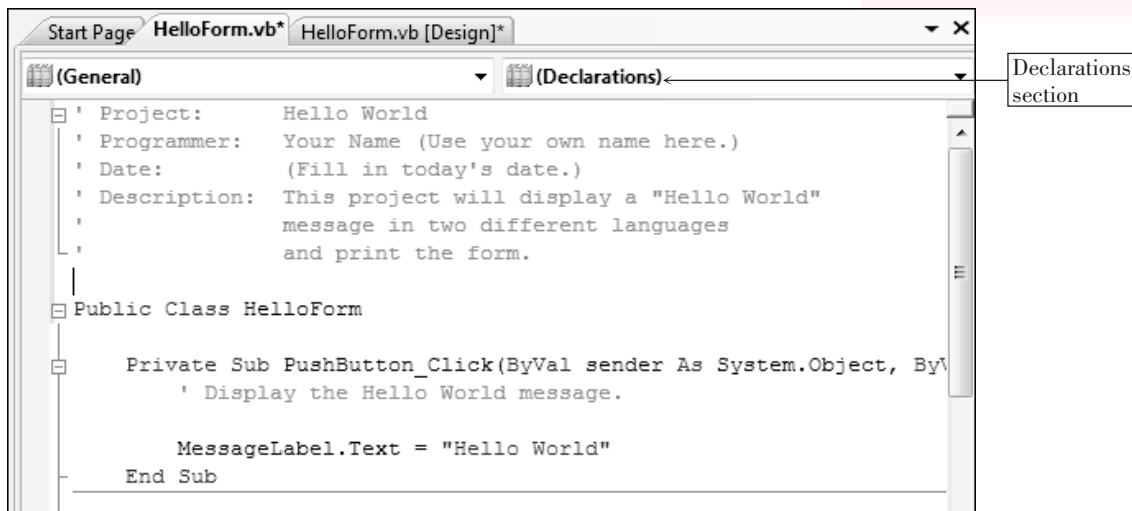
```
' Project:      Hello World
' Programmer:   Your Name (Use your own name here.)
' Date:         (Fill in today's date.)
' Description:  This project will display a "Hello World"
               message in two different languages
               and print the form.
```



TIP
Press Ctrl + Home to quickly move the insertion point to the top of the file. ■

Figure 1.47

Enter remarks at the top of the form file.



Explore the Editor Window

STEP 1: Notice the two drop-down list boxes at the top of the Editor window, called the *Class Name list* and the *Method Name list*. You can use these lists to move to any procedure in your code.

STEP 2: Click on the left down-pointing arrow to view the Class Name list. Notice that every object in your form is listed there (Figure 1.48). At the top of the list, you see the name of your form: *HelloForm*.

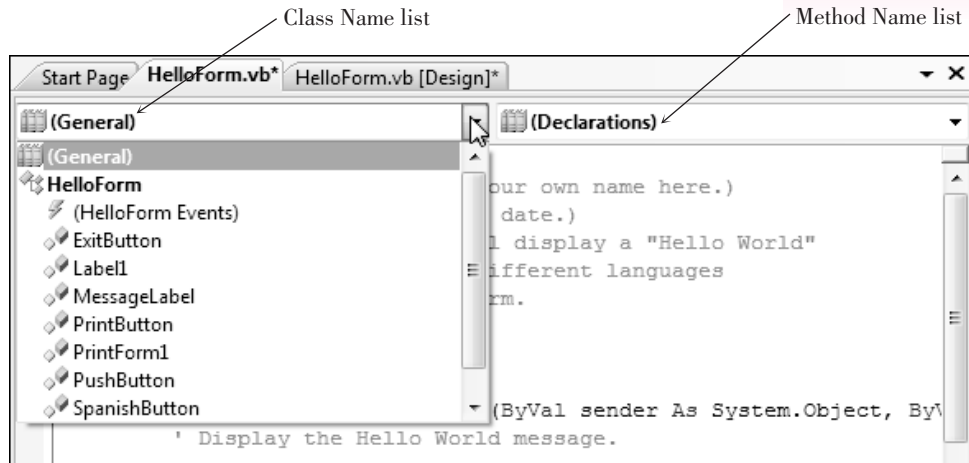
STEP 3: Select SpanishButton from the Class Name list. The insertion point jumps to the first line within the SpanishButton_Click event procedure.

STEP 4: Drop down the Method Name list (the right list); it shows all possible events for a Button control. Notice that the Click event is bold and the rest are not. Any event for which you have written an event procedure appears in bold.

To write code for more than one event for an object, you can use the Method Name drop-down list. When you select a new event from the Method Name list, the editor generates the `Private Sub` and `End Sub` lines for that procedure and moves the insertion point to the new procedure.

Figure 1.48

View the list of objects in this form by dropping down the Class Name list. Select an object from the list to display the sub procedures for that object.



Finish Up

STEP 1: Save the project again.

Print the Code

Select the Printing Options

STEP 1: Make sure that the Editor window is open, showing your form's code. The *File / Print* command is disabled unless the code is displaying and its window selected.

Note: If the *File / Print* command is disabled, click an insertion point in the Editor window.

STEP 2: Open the *File* menu and choose *Print*. Click OK.

A Sample Printout

This output is produced when you print the form's code. Notice the `↵` symbol used to continue long lines on the printout. On the screen, those long lines are not split, but scroll off the right side of the screen.

C:\Users\ . . . \HelloWorld\HelloForm.vb

1

```

'Project:           Hello World
'Programmer:       Your Name
'Date:             Today's Date
'Description:      This project will display a "Hello World"
'                  message in two different languages and print the form.

```

```

Public Class HelloForm

```

```
Private Sub PushButton_Click(ByVal sender As System.Object, ByVal e As System.
EventArgs) Handles PushButton.Click
    ' Display the Hello World Message.
```

```
        MessageLabel.Text = "Hello World"
End Sub
```

```
Private Sub ExitButton_Click(ByVal sender As System.Object, ByVal e As System.
EventArgs) Handles ExitButton.Click
    ' Exit the project.
```

```
        Me.Close()
End Sub
```

```
Private Sub SpanishButton_Click(ByVal sender As System.Object, ByVal e As System.
EventArgs) Handles SpanishButton.Click
    ' Display the Hello World message in Spanish.
```

```
        MessageLabel.Text = "Hola Mundo"
End Sub
```

```
Private Sub PrintButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles PrintButton.Click
    ' Print the form.
```

```
        PrintForm1.PrintAction = Printing.PrintAction.PrintToPreview
        PrintForm1.Print()
End Sub
```

```
End Class
```

Finding and Fixing Errors

You already may have seen some errors as you entered the first sample project. Programming errors come in three varieties: **syntax errors**, **run-time errors**, and **logic errors**.

Syntax Errors

When you break VB's rules for punctuation, format, or spelling, you generate a syntax error. Fortunately, the smart editor finds most syntax errors and even corrects many of them for you. The syntax errors that the editor cannot identify are found and reported by the compiler as it attempts to convert the code into intermediate machine language. A compiler-reported syntax error may be referred to as a *compile error*.

The editor can correct some syntax errors by making assumptions and not even report the error to you. For example, a string of characters must have opening and closing quotes, such as "Hello World". But if you type the opening quote and forget the closing quote, the editor automatically adds the closing quote when you move to the next line. And if you forget the opening and closing parentheses after a method name, such as `Close()`, again the editor will add them for you when you move off the line. Of course, sometimes the editor will make a wrong assumption, but you will be watching, right?

The editor identifies syntax errors as you move off the offending line. A blue squiggly line appears under the part of the line that the editor cannot interpret. You can view the error message by pausing the mouse pointer over the error, which pops up a box that describes the error (Figure 1.49). You also can display an Error List window (*View / Error List*), which appears at the bottom of the Editor window and shows all error messages along with the line number of the statement that caused the error. You can display line numbers on the source code (Figure 1.50) with *Tools / Options*. If *Show All Settings* is selected in the *Options* dialog box, choose *Text Editor / Basic* and check *Line Numbers*; if *Show All Settings* is not checked, then choose *Text Editor Basic / Editor* and check *Line Numbers*.

Figure 1.49

The editor identifies a syntax error with a squiggly blue line, and you can point to an error to pop up the error message.

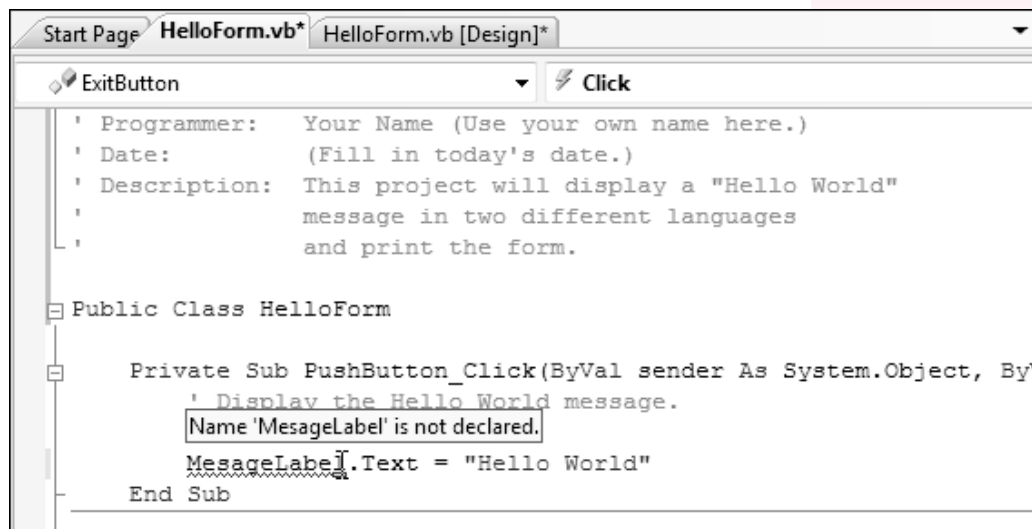
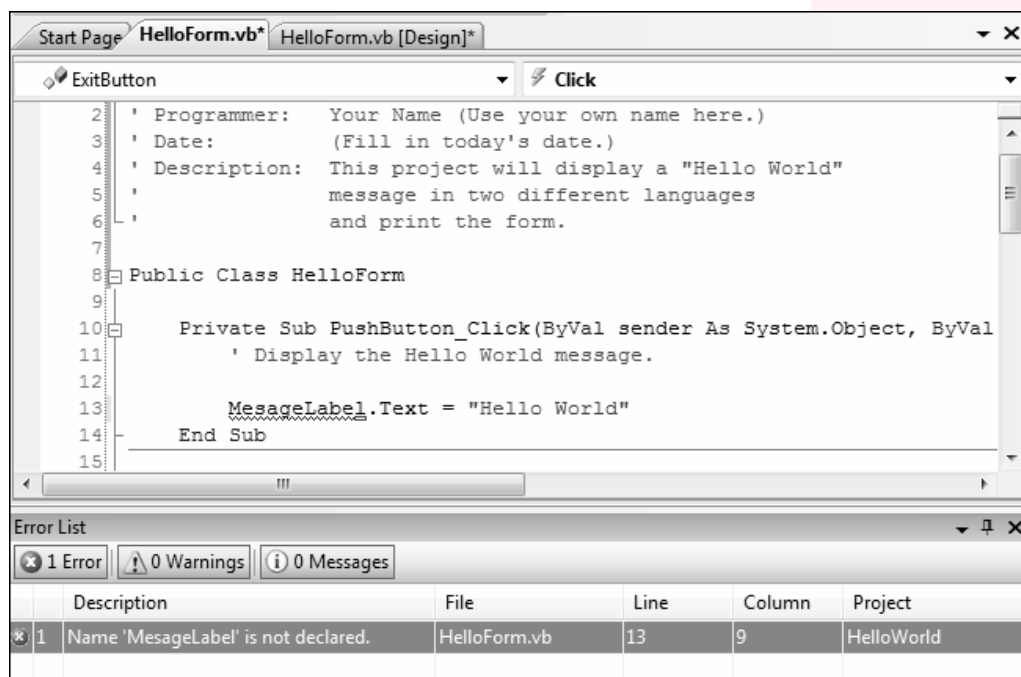


Figure 1.50

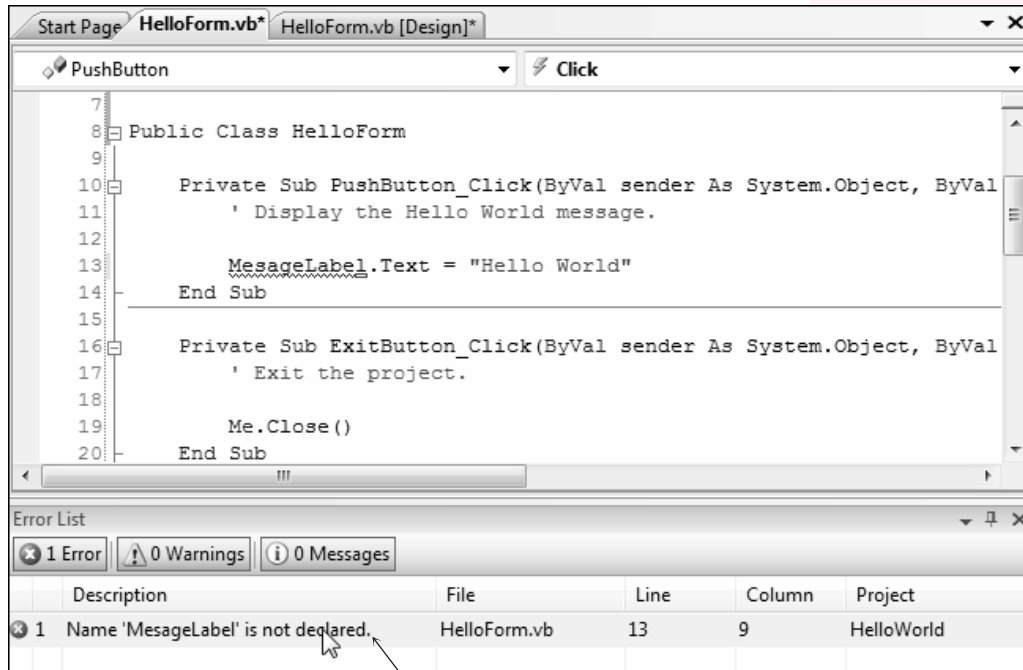
You can display the Error List window and line numbers in the source code to help locate the error lines.



The quickest way to jump to an error line is to point to a message in the Error List window and double-click. The line in error will display in the Editor window with the insertion point in the line (Figure 1.51).

Figure 1.51

Quickly jump to the line in error by double-clicking on the error message in the Error List window.



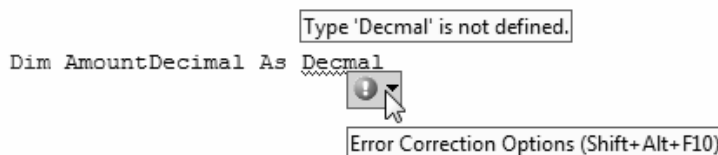
Double-click anywhere on this line to jump to the error

At times the editor can recognize errors and offer suggested solutions. This is more likely to occur in later chapters as you begin to use new keywords. In Chapter 3 you learn to declare elements that can use a data type called *Decimal*. If you accidentally mistype the word *Decimal*, a small red line appears at the end of the word. Point to the line and an **AutoCorrect** box appears, offering to change the word to the correct spelling. Figures 1.52 and 1.53 show AutoCorrect in action.

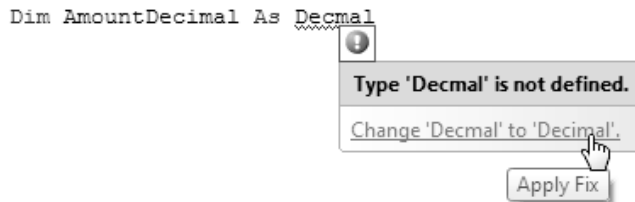


TIP The Visual Basic AutoCorrect feature can suggest corrections for common syntax errors. ■

Figure 1.52

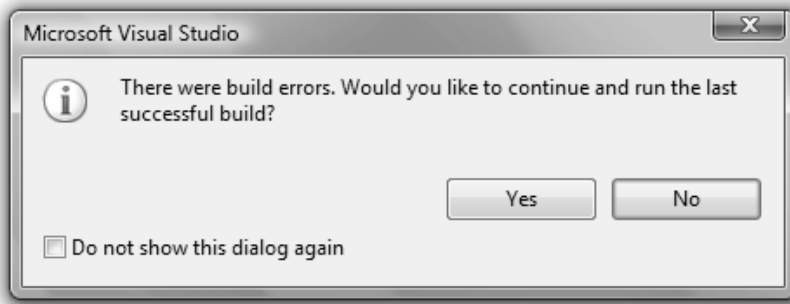


Point to the small red line and the AutoCorrect feature pops up a message and a box with a down arrow. Display the suggestions by clicking the down arrow or pressing Shift + Alt + F10, as suggested in the popup message.

**Figure 1.53**

The Error Corrections Options box displays suggested corrections. You can make a selection from the list.

If a syntax error is found by the compiler, you will see the dialog box shown in Figure 1.54. Click No and return to the editor, correct your errors, and run the program again.

**Figure 1.54**

When the compiler identifies syntax errors, it cannot continue. Click No to return to the editor and correct the error.

Run-Time Errors

If your project halts during execution, it is called a *run-time error* or an *exception*. Visual Basic displays a dialog box and highlights the statement causing the problem.

Statements that cannot execute correctly cause run-time errors. The statements are correctly formed Basic statements that pass the syntax checking; however, the statements fail to execute due to some serious issue. You can cause run-time errors by attempting to do impossible arithmetic operations, such as calculate with nonnumeric data, divide by zero, or find the square root of a negative number.

In Chapter 3 you will learn to catch exceptions so that the program does not come to a halt when an error occurs.

Logic Errors

When your program contains logic errors, your project runs but produces incorrect results. Perhaps the results of a calculation are incorrect or the wrong text appears or the text is okay but appears in the wrong location.

Beginning programmers often overlook their logic errors. If the project runs, it must be right—right? All too often, that statement is not correct. You may need to use a calculator to check the output. Check all aspects of the project output: computations, text, and spacing.

For example, the Hello World project in this chapter has event procedures for displaying “Hello World” in English and in Spanish. If the contents of the two procedures were switched, the program would work but the results would be incorrect.

The following code does not give the proper instructions to display the message in Spanish:

```
Private Sub SpanishButton_Click
    ' Display the Hello World Message in Spanish.

    MessageLabel.Text = "Hello World"
End Sub
```

Project Debugging

If you talk to any computer programmer, you will learn that programs don't have errors, but that programs get “bugs” in them. Finding and fixing these bugs is called *debugging*.

For syntax errors and run-time errors, your job is easier. Visual Basic displays the Editor window with the offending line highlighted. However, you must identify and locate logic errors yourself.

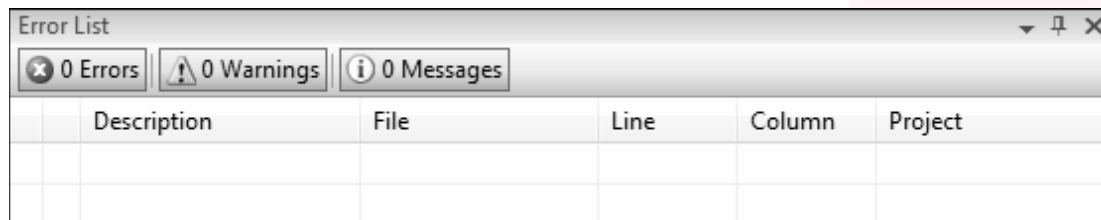
VB 2008 has a very popular feature: edit-and-continue. If you are able to identify the run-time error and fix it, you can continue project execution from that location by clicking on the *Start Debugging* button, pressing F5, or choosing *Debug / Start Debugging*. You also can correct the error and restart from the beginning.

The Visual Studio IDE has some very helpful tools to aid in debugging your projects. The debugging tools are covered in Chapter 4.

A Clean Compile

When you start executing your program, the first step is called *compiling*, which means that the VB statements are converted to Microsoft Intermediate Language (MSIL). Your goal is to have no errors during the compile process: a **clean compile**. Figure 1.55 shows the Error List window for a clean compile: 0 Errors; 0 Warnings; 0 Messages.

Zero errors, warnings, and messages means that you have a clean compile.



TIP
If you get the message “There were build errors. Continue?” always say No. If you say Yes, the last cleanly compiled version runs, rather than the current version. ■

Figure 1.55

Naming Rules and Conventions for Objects

Using good consistent names for objects can make a project easier to read and understand, as well as easier to debug. You *must* follow the Visual Basic rules for naming objects, procedures, and variables. In addition, conscientious programmers also follow certain naming conventions.

Most professional programming shops have a set of standards that their programmers must use. Those standards may differ from the ones you find in this book, but the most important point is this: *Good programmers follow standards. You should have a set of standards and always follow them.*

The Naming Rules

When you select a name for an object, Visual Basic requires the name to begin with a letter or an underscore. The name can contain letters, digits, and underscores. An object name cannot include a space or punctuation mark and cannot be a reserved word, such as Button or Close, but can contain one. For example, ExitButton and CloseButton are legal.

The Naming Conventions

This text follows standard naming conventions, which help make projects more understandable. When naming controls, use **Pascal casing**, which means that you begin the name with an uppercase character and capitalize each additional word in the name. Make up a meaningful name and append the full name of the control's class. Do not use an abbreviation unless it is a commonly used term that everyone will understand. All names must be meaningful and indicate the purpose of the object.

Examples

MessageLabel
ExitButton
DataEntryForm
DiscountRateLabel

Do not keep the default names assigned by Visual Basic, such as Button1 and Label3. Also, do not name your objects with numbers. The exception to this rule is for labels or other controls that never change during project execution. Labels usually hold items such as titles, instructions, and labels for other controls. Leaving these labels with their default names is perfectly acceptable and is practiced in this text.

Refer to Table 1.2 for sample object names.

Visual Studio Help

Visual Studio has an extensive Help facility, which contains much more information than you will ever use. You can look up any Basic statement, class, property, method, or programming concept. Many coding examples are available, and you can copy and paste the examples into your own project, modifying them if you wish.

The VS Help facility includes the Microsoft Developer Network library (MSDN), which contains several books, technical articles, and the Microsoft Knowledge Base, a database of frequently asked questions and their answers. MSDN includes reference materials for the VS IDE, the .NET Framework, Visual Basic, C#, and C++. You will want to filter the information to display only the VB information.

Recommended Naming Conventions for Visual Basic Objects

Table 1.2

Object Class	Example
Form	DataEntryForm
Button	ExitButton
Label	TotalLabel
TextBox	PaymentAmountTextBox
Radio button	BoldRadioButton
CheckBox	PrintSummaryCheckBox
Horizontal scroll bar	RateHorizontalScrollBar
Vertical scroll bar	TemperatureVerticalScrollBar
PictureBox	LandscapePictureBox
ComboBox	BookListComboBox
Listbox	IngredientsListBox
SoundPlayer	IntroPageSoundPlayer

Installing and Running MSDN

You can run MSDN from a local hard drive or from the Web. Of course, if you plan to access MSDN from the Web, you must have a live Internet connection as you work.

Depending on how you install Visual Basic, you are given the option to refer first to local, first to online, or only to local. You can change this setting later in the *Options* dialog box. Select *Tools / Options* and check the *Show All Settings* check box. Then expand the *Environment* node and the *Help* node and click on *Online*. You can choose the options to *Try online first, then local*; *Try local first, then online*; or *Try local only, not online*. Notice also that you can select sites to include in Help topics.

The extensive Help is a two-edged sword: You have available a wealth of materials, but it may take some time to find the topic you want.

Viewing Help Topics

The Help system in Visual Studio 2008 allows you to view the Help topics in a separate window from the VS IDE, so you can have both windows open at the same time. When you choose *How Do I*, *Search*, *Contents*, *Index*, or *Help Favorites* from the *Help* menu, a new window opens on top of the IDE window (Figure 1.56). You can switch from one window to the other, or resize the windows to view both on the screen if your screen is large enough.

You can choose to filter the Help topics so that you don't have to view topics for all of the languages when you search for a particular topic. In the *Index* or *Contents* window, drop down the *Filtered By* list and choose *Visual Basic Express Edition* for the Express Edition or *Visual Basic* for the Professional Edition (Figure 1.57).

Figure 1.56

The Help window. Search appears in a tabbed window in the main Document window; Contents, Index, and Help Favorites appear in tabbed windows docked at the left of the main window.

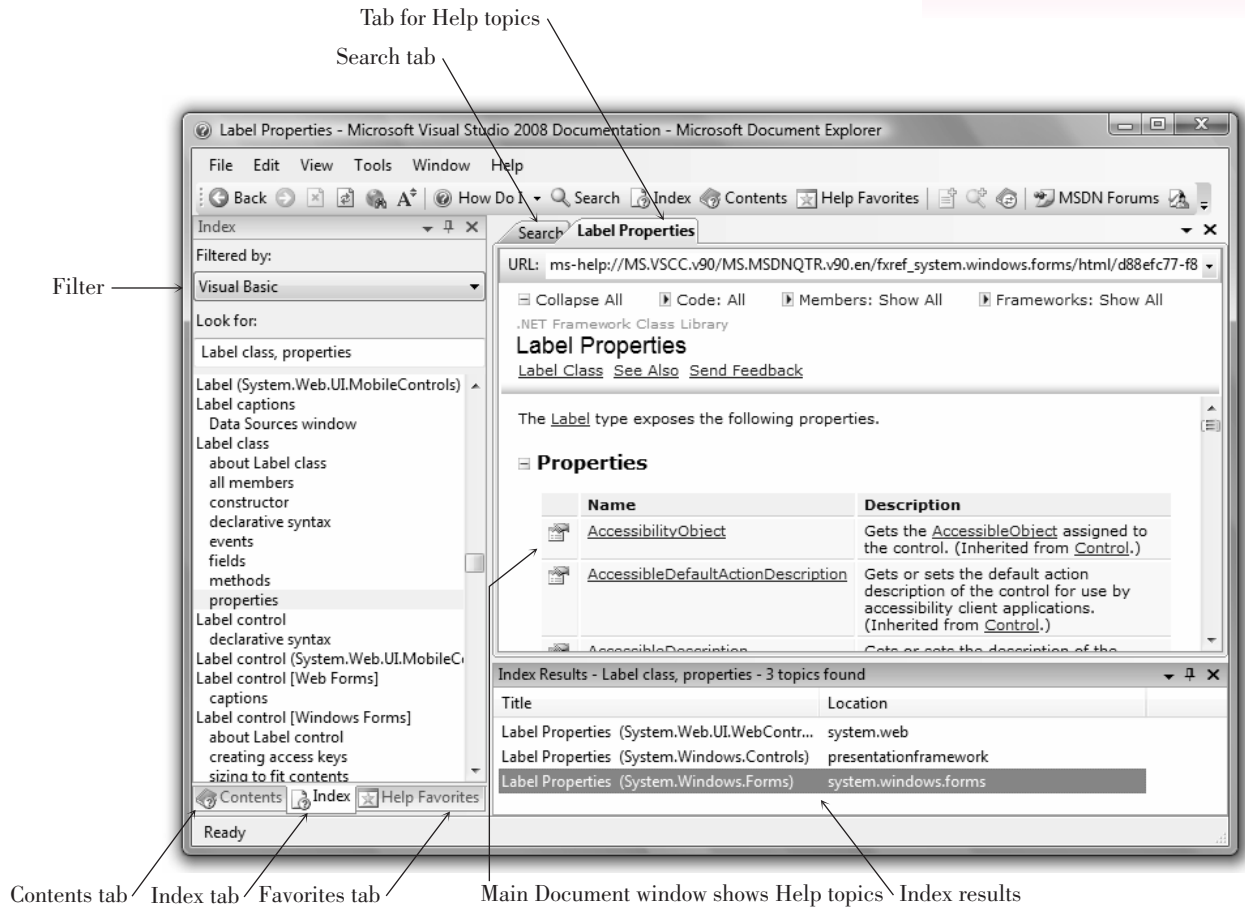
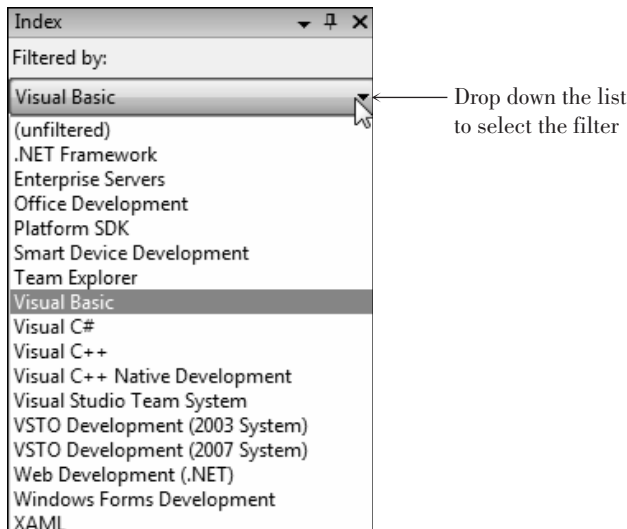


Figure 1.57

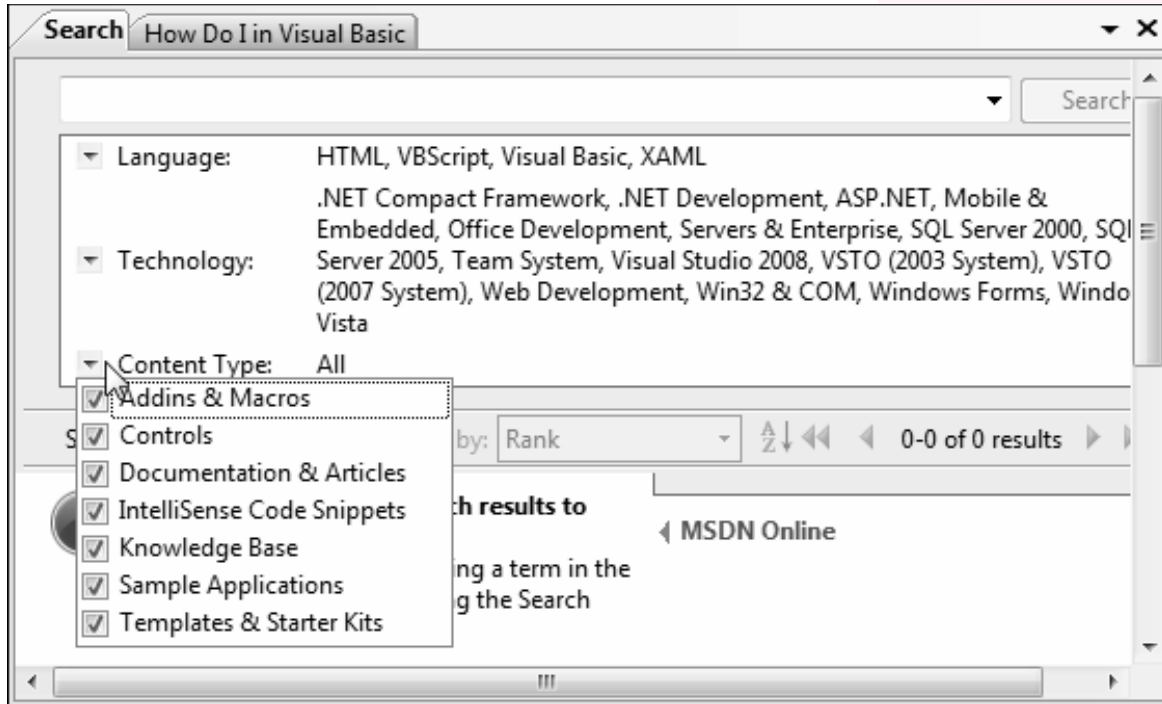
Filter the Help topics so that only the Visual Basic topics appear.



In the Search window, you can choose additional filter options, such as the technology and topic type. Drop down a list and select any desired options (Figure 1.58).

Figure 1.58

Drop down the Content Type list to make selections for the Search window.



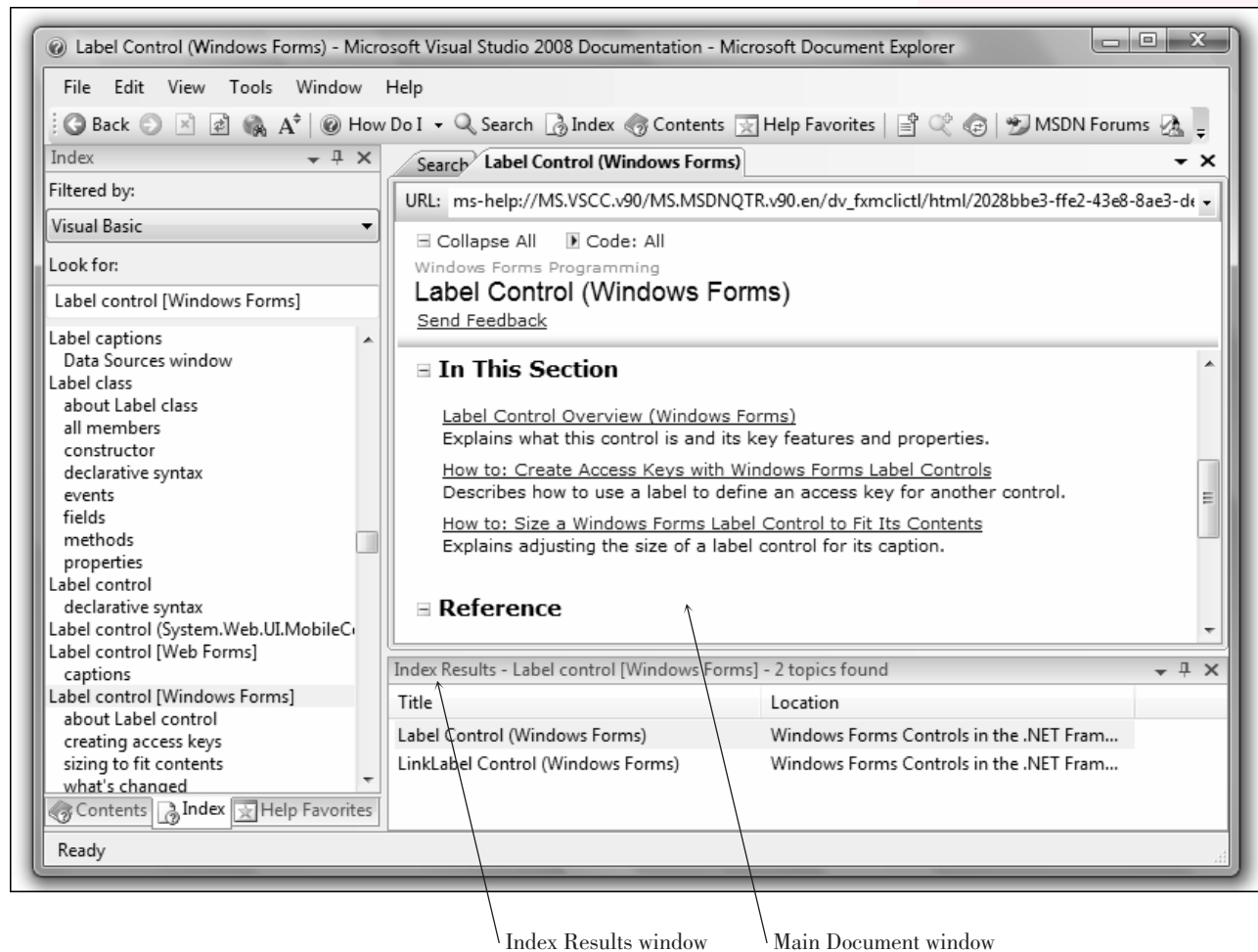
In the Help Index window, you see main topics and subtopics (indented beneath the main topics). All main topics and some subtopics have multiple entries available. When you choose a topic that has more than one possible entry, the choices appear in the Index Results window (Figure 1.59). Double-click on the entry of your choice to display its Help topic in the main Document window.

Many Help topics have entries for both Windows Forms and Web Forms (and some even Mobile Forms). For now, always choose Windows Forms. Chapters 1 to 8 deal with Windows Forms exclusively; Web Forms are introduced in Chapter 9.

A good way to start using Help is to view the topics that demonstrate how to look up topics in Help. On the *Help Contents* tab, select *Help on Help (Microsoft Document Explorer Help)*. Then choose *Microsoft Document Explorer Overview* and *What's New in Document Explorer*. Make sure to visit *Managing Help Topics* and *Windows*, which has subtopics describing how to copy topics and print topics.

Figure 1.59

When an Index topic has a choice of subtopics, the choices appear in an Index Results pane at the bottom of the screen. Double-click the desired subtopic; the selected topic appears in the main Document window.



Context-Sensitive Help

A quick way to view Help on any topic is to use **context-sensitive Help**. Select a VB object, such as a form or control, or place the insertion point in a word in the editor and press F1. The Help window pops up with the corresponding Help topic displayed, if possible, saving you a search. You can display context-sensitive Help about the environment by clicking in an area of the screen and pressing Shift + F1.

Managing Windows

At times you may have more windows and tabs open than you want. You can hide or close any window, or switch to a different window.

To close a window that is a part of a tabbed window, click the window's Close button. Only the top window will close.

To switch to another window that is part of a tabbed window, click on its tab.

For additional help with the environment, see Appendix C, “Tips and Shortcuts for Mastering the Visual Studio Environment.”

Feedback 1.1

Note: Answers for Feedback questions appear in Appendix A.

1. Display the Help Index, filter by *Visual Basic Express Edition* (or *Visual Basic*), and type “button control”. In the Index list, notice multiple entries for button controls. Depending on the edition of Visual Basic, you may see entries for HTML, Web Forms, and Windows Forms. Locate the main topic *Button control [Windows Forms]* and click on the entry for *about Button control*. The topics included for the Professional Edition are more extensive than those for the Express Edition. In the Express Edition, only one page matches the selection and it appears in the main Document window. In the Professional Edition, several topics appear in the Index Results list. Click on a title in the Index Results to display the corresponding page in the Document window. Notice that additional links appear in the text in the Document window. You can click on a link to view another topic.
2. Display the Editor window of your Hello World project. Click on the `Close` method to place the insertion point. Press the F1 key to view context-sensitive help.
3. Select each of the options from the VS IDE’s *Help* menu to see how they respond.

Your Hands-On Programming Example

Write a program for *R’nR--For Reading and Refreshment* to display the current special promotions. Include labels to display the current special and the promotion code and buttons for each of the following departments: Books, Music, Periodicals, and Coffee Bar.

The user interface should also have buttons for *Print* and *Exit* and a label with the programmer’s name. Change the name of the form to `MainForm`, and place “*R’nR--For Reading and Refreshment*” in the title bar of the form.

The *Print* button should display the form in a Print Preview window.

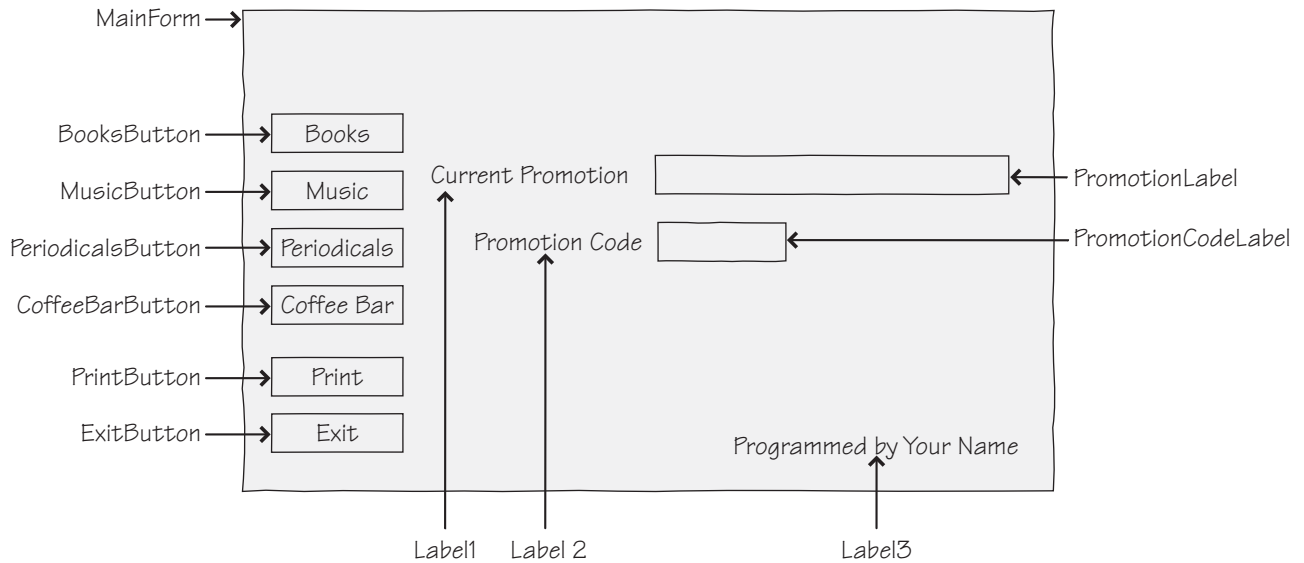
Planning the Project

Sketch a form (Figure 1.60), which your users sign off on as meeting their needs.

Note: Although this step may seem unnecessary, having your users sign off is standard programming practice and documents that your users have been involved and have approved the design.

Figure 1.60

A Planning sketch of the form for the hands-on programming example.



Plan the Objects and Properties Plan the property settings for the form and for each control.

Object	Property	Setting
MainForm	Name	MainForm
	Text	R 'n R--For Reading and Refreshment
	StartPosition	CenterScreen
BooksButton	Name	BooksButton
	Text	Books
MusicButton	Name	MusicButton
	Text	Music
PeriodicalsButton	Name	PeriodicalsButton
	Text	Periodicals
CoffeeBarButton	Name	CoffeeBarButton
	Text	Coffee Bar
PrintButton	Name	PrintButton
	Text	Print
ExitButton	Name	ExitButton
	Text	Exit
Label1	Text	Current Promotion
Label2	Text	Promotion Code
Label3	Text	Programmed by Your Name
PromotionLabel	Name	PromotionLabel
	AutoSize	False
	BorderStyle	FixedSingle
	Text	(blank)
	TextAlign	MiddleLeft

PromotionCodeLabel	Name	PromotionCodeLabel
	AutoSize	False
	BorderStyle	FixedSingle
	Text	(blank)
	TextAlign	MiddleLeft
PrintForm1	Name	PrintForm1

Plan the Event Procedures You will need event procedures for each button. Write the actions in pseudocode, which is English language that is “like code.”

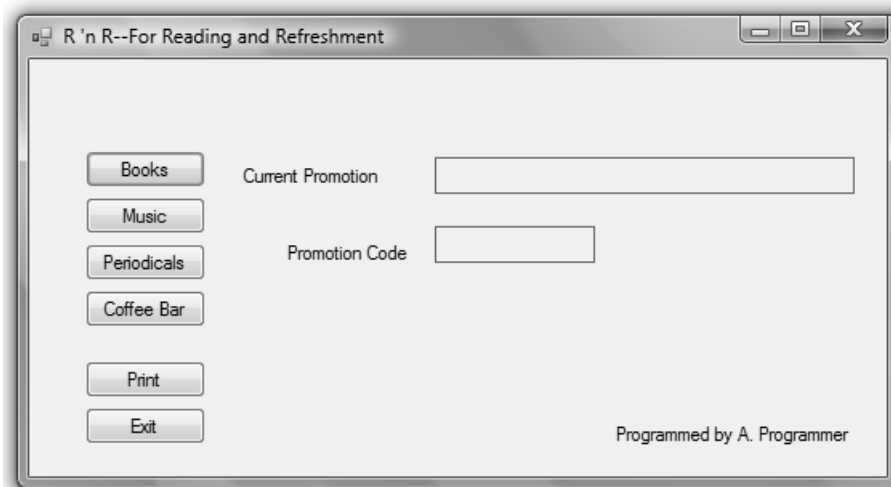
Procedure	Actions—Pseudocode
BooksButton_Click	Display “Buy two, get the second one for half price” and code B608.
MusicButton_Click	Display “Get a free MP3 download with purchase of a CD” and code M608.
PeriodicalsButton_Click	Display “Elite members receive 10% off every purchase” and code P608.
CoffeeBarButton_Click	Display “Celebrate the season with 20% off specialty drinks” and code C608.
PrintButton_Click	Set the print action to Print Preview. Call the Print method.
ExitButton_Click	End the project.

Write the Project Follow the sketch in Figure 1.60 to create the form. Figure 1.61 shows the completed form.

- Set the properties of each object, as you have planned.
- Working from the pseudocode, write each event procedure.
- When you complete the code, thoroughly test the project.

Figure 1.61

The form for the hands-on programming example



The Project Coding Solution

```
'Project:           Ch01HandsOn
'Programmer:        Bradley/Millspaugh
'Date:              June 2008
'Description:       This project displays current promotions for each department.
```

```
Public Class MainForm
```

```
Private Sub BooksButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles BooksButton.Click
    ' Display the current promotion.

    PromotionLabel.Text = "Buy two, get the second one for half price."
    PromotionCodeLabel.Text = "B608"
End Sub
```

```
Private Sub MusicButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MusicButton.Click
    ' Display the current promotion.

    PromotionLabel.Text = "Get a free MP3 download with purchase of a CD."
    PromotionCodeLabel.Text = "M608"
End Sub
```

```
Private Sub PeriodicalsButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles PeriodicalsButton.Click
    ' Display the current promotion.

    PromotionLabel.Text = "Elite members receive 10% off every purchase."
    PromotionCodeLabel.Text = "P608"
End Sub
```

```
Private Sub CoffeeBarButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles CoffeeBarButton.Click
    ' Display the current promotion.

    PromotionLabel.Text = "Celebrate the season with 20% off specialty drinks."
    PromotionCodeLabel.Text = "C608"
End Sub
```

```
Private Sub PrintButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles PrintButton.Click
    ' Print the form.

    PrintForm1.PrintAction = Printing.PrintAction.PrintToPreview
    PrintForm1.Print()
End Sub
```

```
Private Sub ExitButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles ExitButton.Click
    ' End the program.

    Me.Close()
End Sub
```

```
End Class
```

Summary

1. Visual Basic is an object-oriented language primarily used to write application programs that run in Windows or on the Internet using a graphical user interface (GUI).
2. In the OOP object model, classes are used to create objects that have properties, methods, and events.
3. The current release of Visual Basic is called 2008. Visual Basic is part of Visual Studio. VB 2008 has an Express Edition, a Standard Edition, a Professional Edition, and a Team System Edition.
4. The .NET Framework provides an environment for the objects from many languages to interoperate. Each language compiles to Microsoft Intermediate Language (MSIL) and runs in the Common Language Runtime (CLR).
5. To plan a project, first sketch the user interface and then list the objects and properties needed. Then plan the necessary event procedures.
6. The three steps to creating a Visual Basic project are (1) define the user interface, (2) set the properties, and (3) write the Basic code.
7. A Visual Basic application is called a *solution*. Each solution can contain multiple projects, and each project may contain multiple forms and additional files. The solution file has an extension of .sln, a project file has an extension of .vbproj, and form files and additional VB files have an extension of .vb. In addition, the Visual Studio environment and the VB compiler both create several more files.
8. The Visual Studio integrated development environment (IDE) consists of several tools, including a form designer, an editor, a compiler, a debugger, an object browser, and a Help facility.
9. VB has three modes: design time, run time, and debug time.
10. You can customize the Visual Studio IDE and reset all customizations back to their default state.
11. You create the user interface for an application by adding controls from the toolbox to a form. You can move, resize, and delete the controls.
12. The Name property of a control is used to refer to the control in code. The Text property holds the words that the user sees on the screen.
13. Visual Basic code is written in procedures. Sub procedures begin with the word Sub and end with End Sub.
14. Project remarks are used for documentation. Good programming practice requires remarks in every procedure and in the Declarations section of a file.
15. Assignment statements assign a value to a property or a variable. Assignment statements work from right to left, assigning the value on the right side of the equal sign to the property or variable named on the left side of the equal sign.
16. The Me.Close() method terminates program execution.
17. Each event to which you want to respond requires an event procedure.
18. You can print out the Visual Basic code for documentation. You also can use the PrintForm component in an application to print the current form, either to the printer or to the Print Preview window.
19. Three types of errors can occur in a Visual Basic project: syntax errors, which violate the syntax rules of the Basic language; run-time errors, which

contain a statement that cannot execute properly; and logic errors, which produce erroneous results.

20. Finding and fixing program errors is called *debugging*.
21. You should have a clean compile before you run the program.
22. Following good naming conventions can help make a project easier to debug.
23. Visual Basic Help has very complete descriptions of all project elements and their uses. You can use the *How Do I*, *Contents*, *Index*, *Search*, *Help Favorites*, or context-sensitive Help.

Key Terms

- assignment statement 30
- AutoCorrect 48
- Button 19
- class 4
- clean compile 50
- code 6
- component tray 42
- context menu 23
- context-sensitive Help 55
- control 3
- debug time 14
- debugging 50
- Declarations section 43
- design time 14
- Document window 12
- event 4
- event procedure 30
- Express Edition 5
- form 3
- Form Designer 13
- graphical user interface (GUI) 3
- handle 13
- Help 13
- integrated development environment (IDE) 8
- Label 19
- logic error 46
- method 4
- namespace 23
- object 4
- object-oriented programming (OOP) 3
- Pascal casing 51
- PrintForm 41
- procedure 29
- Professional Edition 5
- Properties window 13
- property 4
- pseudocode 6
- remark 30
- resizing handle 21
- run time 14
- run-time error 46
- snap lines 21
- solution 7
- Solution Explorer window 13
- solution file 7
- Standard Edition 5
- sub procedure 29
- syntax error 46
- Team System Edition 5
- Text property 25
- toolbar 12
- toolbox 13
- user interface 6
- Visual Studio environment 8

Review Questions

1. What are objects and properties? How are they related to each other?
2. What are the three steps for planning and creating Visual Basic projects? Describe what happens in each step.
3. What is the purpose of these Visual Basic file types: .sln, .suo, and .vb?
4. When is Visual Basic in design time? run time? debug time?
5. What is the purpose of the Name property of a control?
6. Which property determines what appears on the form for a Label control?
7. What is the purpose of the Text property of a button? the Text property of a form?
8. What does PushButton_Click mean? To what does PushButton refer? To what does Click refer?
9. What is a Visual Basic event? Give some examples of events.
10. What property must be set to center text in a label? What should be the value of the property?
11. What is the Declarations section of a file? What belongs there?
12. What is meant by the term *debugging*?
13. What is a syntax error, when does it occur, and what might cause it?
14. What is a run-time error, when does it occur, and what might cause it?
15. What is a logic error, when does it occur, and what might cause it?
16. Tell the class of control and the likely purpose of each of these object names:
 - AddressLabel
 - ExitButton
 - NameTextBox
 - TextBlueRadioButton
17. What does context-sensitive Help mean? How can you use it to see the Help page for a button?

Programming Exercises

- 1.1 For your first Visual Basic exercise, you must first complete the Hello World project. Then add buttons and event procedures to display the “Hello World” message in two more languages. You may substitute any other languages for those shown. Feel free to modify the user interface to suit yourself (or your instructor).

Make sure to use meaningful names for your new buttons, following the naming conventions in Table 1.2. Include remarks at the top of every procedure and at the top of the file.

“Hello World” in French:	Bonjour tout le monde
“Hello World” in Italian:	Ciao Mondo
 - 1.2 Write a new Visual Basic project that displays a different greeting, or make it display the name of your school or your company. Include at least three buttons to display the greeting, print, and exit the project.

Include a label that holds your name at the bottom of the form and change the Text property of the form to something meaningful.

Follow good naming conventions for object names; include remarks at the top of every procedure and at the top of the file.

Select a different font name and font size for the greeting label. If you wish, you also can select a different color for the font. Select each font attribute from the *Font* dialog box from the Properties window.

- 1.3 Write a project that displays four sayings, such as “The early bird gets the worm” or “A penny saved is a penny earned.” (You will want to keep the sayings short, as each must be entered on one line.) When the saying displays on your form, long lines will run off the form if the label’s *AutoSize* property is set to *True*. To wrap text within the label, change the *AutoSize* property to *False* and use the sizing handles to make the label large enough.

Make a button for each saying with a descriptive *Text* property for each, a button to print, and a button to exit the project.

Include a label that holds your name at the bottom of the form. Also, make sure to change the form’s title bar to something meaningful.

You may change the *Font* properties of the large label to the font and size of your choice.

Make sure the buttons are large enough to hold their entire *Text* properties.

Follow good naming conventions for object names; include remarks at the top of every procedure and at the top of the file.

- 1.4 Write a project to display company contact information. Include buttons and labels for the contact person, department, and phone. When the user clicks on one of the buttons, display the contact information in the corresponding label. Include a button to print and another to exit.

Include a label that holds your name at the bottom of the form and change the title bar of the form to something meaningful.

You may change the *Font* properties of the labels to the font and size of your choice.

Follow good naming conventions for object names; include remarks at the top of every procedure and at the top of the file.

- 1.5 Create a project to display the daily specials for “your” diner. Make up a name for your diner and display it in a label at the top of the form. Add a label to display the appropriate special depending on the button that is pressed. The buttons should be

- Soup of the Day
- Chef’s Special
- Daily Fish

Also include a *Print* button and an *Exit* button.

Sample Data: Dorothy’s Diner is offering Tortilla Soup, a California Cobb Salad, and Hazelnut-Coated Mahi Mahi.

Case Studies

Very Busy (VB) Mail Order

If you don't have the time to look for all those hard-to-find items, tell us what you're looking for. We'll send you a catalog from the appropriate company or order for you.

We can place an order and ship it to you. We also help with shopping for gifts; your order can be gift wrapped and sent anywhere you wish.

The company title will be shortened to VB Mail Order. Include this name on the title bar of the first form of each project that you create for this case study.

Your first job is to create a project that will display the name and telephone number for the contact person for the customer relations, marketing, order processing, and shipping departments.

Include a button for each department. When the user clicks on the button for a department, display the name and telephone number for the contact person in

two labels. Also include identifying labels with Text "Department Contact" and "Telephone Number".

Be sure to include a button for *Print* and one for *Exit*.

Include a label at the bottom of the form that holds your name.

Test Data

Department	Department Contact	Telephone Number
Customer Relations	Tricia Mills	500-1111
Marketing	Michelle Rigner	500-2222
Order Processing	Kenna DeVoss	500-3333
Shipping	Eric Andrews	500-4444

Valley Boulevard (VB) Auto Center

Valley Boulevard Auto Center will meet all of your automobile needs. The center has facilities with everything for your vehicles including sales and leasing for new and used cars and RVs, auto service and repair, detail shop, car wash, and auto parts.

The company title will be shortened to VB Auto Center. This name should appear as the title bar on the first form of every project that you create throughout the text for this case study.

Your first job is to create a project that will display current notices.

Include four buttons labeled "Auto Sales", "Service Center", "Detail Shop", and "Employment Opportunities". One label will be used to display the information when the buttons are clicked. Be sure to include a *Print* button and an *Exit* button.

Include your name in a label at the bottom of the form.

Test Data

Button	Label Text
Auto Sales	Family wagon, immaculate condition \$12,995
Service Center	Lube, oil, filter \$25.99
Detail Shop	Complete detail \$79.95 for most cars
Employment Opportunities	Sales position, contact Mr. Mann 551-2134 x475

Video Bonanza

This neighborhood store is an independently owned video rental business. The owners would like to allow their customers to use the computer to look up the aisle number for movies by category.

Create a form with a button for each category. When the user clicks on a button, display the corresponding aisle number in a label. Include a button to print and one to exit.

Include a label that holds your name at the bottom of the form and change the title bar of the form to Video Bonanza.

You may change the font properties of the labels to the font and size of your choice. Include additional categories, if you wish.

Follow good programming conventions for object names; include remarks at the top of every procedure and at the top of the file.

Test Data

Button	Location
Comedy	Aisle 1
Drama	Aisle 2
Action	Aisle 3
Sci-Fi	Aisle 4
Horror	Aisle 5
New Releases	Back Wall

Very Very Boards

This chain of stores features a full line of clothing and equipment for snowboard and skateboard enthusiasts. Management wants a computer application to allow their employees to display the address and hours for each of their branches.

Create a form with a button for each store branch. When the user clicks on a button, display the correct address and hours.

Include a label that holds your name at the bottom of the form and change the title bar of the form to Very Very Boards.

You may change the font properties of the labels to the font and size of your choice.

Include a *Print* button and an *Exit* button. Follow good programming conventions for object names; include remarks at the top of every procedure and at the top of the file.

Store Branches: The three branches are Downtown, Mall, and Suburbs. Make up hours and locations for each.

