

# 1

## Introduction to Visual Basic .NET

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

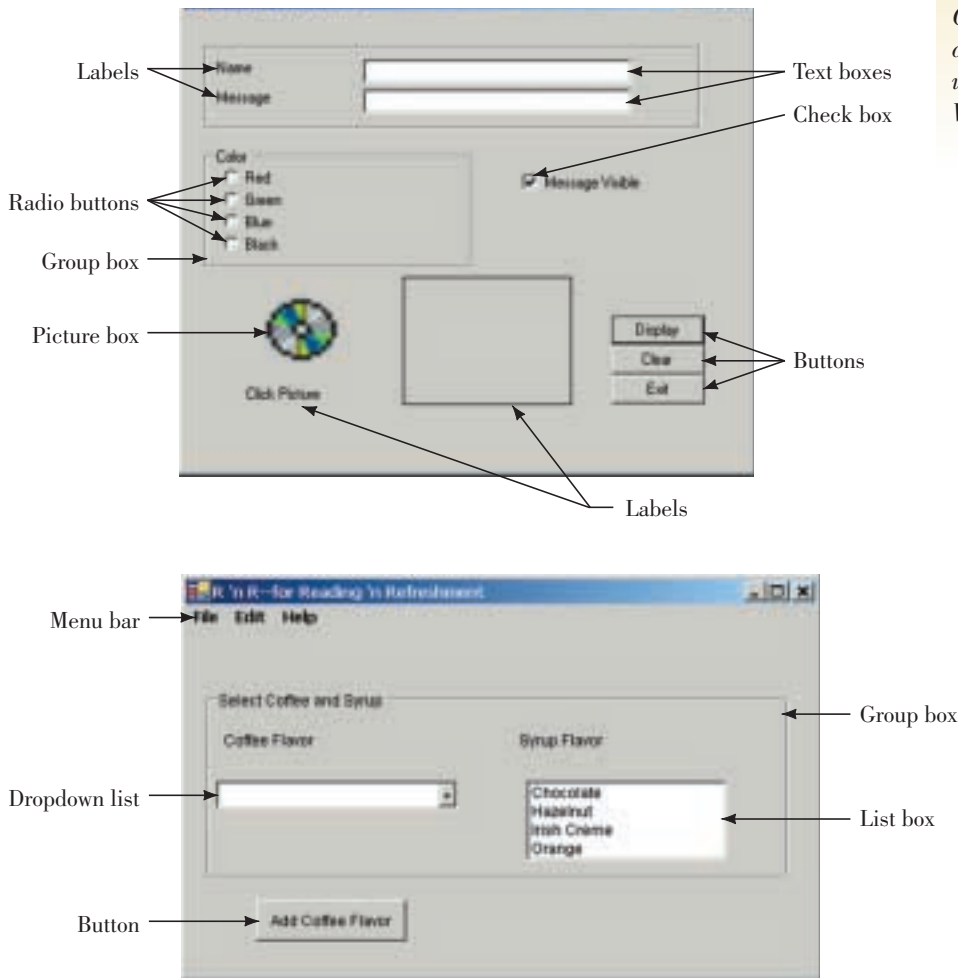
1. Describe the process of visual program design and development.
2. Explain the term *object-oriented programming*.
3. Explain the concepts of classes, objects, properties, methods, and events.
4. List and describe the three steps for writing a Visual Basic project.
5. Describe the various files that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

**Figure 1.1**

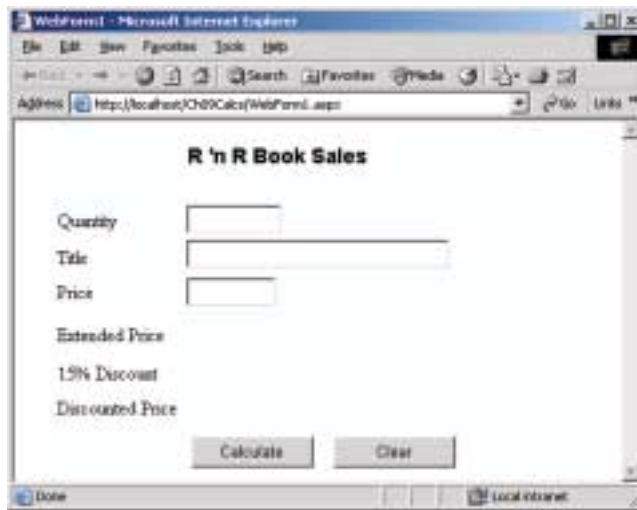
*Graphical user interfaces for application programs designed with Visual Basic .NET and Windows Forms.*



Beginning in Chapter 10, you will create programs using Web Forms. You can run Web Forms applications in a browser, such as Internet Explorer, on the Internet, or on a company intranet. Figure 1.2 shows a Web Form application.

**Figure 1.2**

A Web Forms application created with Visual Basic .NET, 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 relatively new type of programming, called *object-oriented programming*.

## 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 C++, Visual Basic .NET, and Java, use a different approach: **object-oriented programming (OOP)**. Earlier versions of Visual Basic had many (but not all) elements of an object-oriented language. For that reason, Microsoft referred to Visual Basic 6.0 as an event-driven programming language, rather than an object-oriented language. But with the release of Visual Studio .NET, which includes Visual Basic .NET, VB is finally a true object-oriented language.

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 an object, such as its name, color, size, location, or how it will behave. 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 Form1 as Form1.Text (pronounced “form1 dot text”).

### Methods

Actions associated with objects are called *methods*. **Methods** are the verbs of object-oriented programming. Some typical methods are Move, Hide, Show, and Clear.

You refer to methods as Object.Method (“object dot method”). For example, a Show method can apply to different objects. Form1.Show shows the form object called Form1; btnExit.Show shows the button object called btnExit.

### 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 can also 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 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 button's 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 car 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 `MyCar.Color = Blue` and `YourCar.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 `MyCar.Start` and `YourCar.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 `YourCar_Crash` event. Note that you use an underscore between the object name and event, rather than a period.

*Note:* Chapter 6 presents object-oriented programming in greater depth.

## Microsoft's Visual Studio .NET

---

The latest version of Microsoft's Visual Studio, called Visual Studio .NET, includes Visual Basic, Visual C++, the new language, C# ("C sharp"), and the .NET Framework. Visual Studio .NET, sometimes referred to as Version 7, is a major revision of the previous version. In fact, Visual Basic was completely rewritten from Version 6 to Version 7.

### The .NET Framework

The programming languages in Visual Studio .NET run in the new .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 versions of other languages to run in the .NET Framework, including .NET versions of FORTRAN, COBOL, and Java.

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 .NET

Microsoft Visual Basic .NET comes with Visual Studio .NET. You also can purchase a Standard Edition of VB .NET by itself (without the other languages but *with* the .NET Framework). Visual Studio .NET is available in an **Academic Edition**, a **Professional Edition**, an **Enterprise Developer Edition**, and an **Enterprise Architect Edition**. Anyone planning to do professional application development that includes the advanced features of database management should use the Professional Edition or one of the Enterprise editions. You can find a matrix showing the features of each edition in Help.

This text is based on Visual Basic .NET, the current version. 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 process for creating the project.

### The Three-Step Process

The three steps for planning a Visual Basic application involve setting up the user interface, defining the properties, and then creating the code.

#### 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 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*.

#### 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 6, all projects have only one form, so you can think of one project = one form. Starting in Chapter 7, your projects will contain multiple forms and additional files. The HelloWorld application that you will create later in this chapter creates these files:

HelloWorld.sln	The <b>solution file</b> . 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.
HelloWorld.suo	Solution user options file. Stores information about the selected options, so that all customizations can be restored each time you open the solution.
frmHello.vb	A .vb file. Holds the definition of a form, its controls, and code procedures. This is a text file that you can open in any editor. <i>Warning:</i> You should not modify this file unless you are using the editor in the Visual Studio environment.
frmHello.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.
HelloWorld.vbproj	A <b>project file</b> . A text file that describes the project and lists the files that are included in the project.
HelloWorld.vbproj.user	The project user option file. This text file holds project option settings, so that the next time you open the project, all selected options will be restored.

After you run your project, you will find several more files created by the system. The only file that you will open directly is the .sln, or solution file.

- A Solution can contain multiple projects (all ours have only one). A Project can contain multiple forms (ours have only one form in Chapters 1–6). Forms, classes, and code files have an extension of .vb.

## 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 the available classes, objects, properties, methods, and events; and a Help facility.

In earlier versions of Visual Studio, 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 .NET, you use the one IDE to create projects in any of the .NET languages.

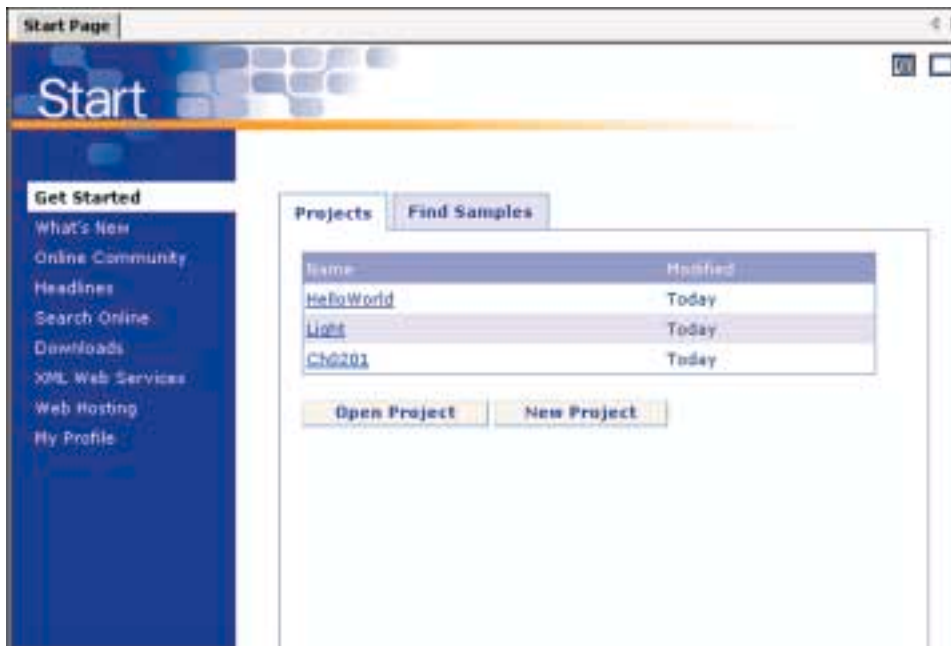
### The IDE Start Page

When you open the Visual Studio IDE, you see its Start Page (Figure 1.3). Recent projects appear on the list, which enable you to open an existing project, or you can select *New Project* to begin a new project.

- Terminology changes:
  - VB IDE becomes the Visual Studio (VS) IDE.
  - Project Explorer becomes the Solution Explorer.
  - Document window has tabs to display various contents:
    - Form window becomes the Form Designer.
    - Code window becomes the Editor window or the VS editor.
    - Note that the tabbed display is the default but can be turned off in *Tools / Options / General / Tabbed Documents*.

**Figure 1.3**

*The Visual Studio IDE Start Page.*





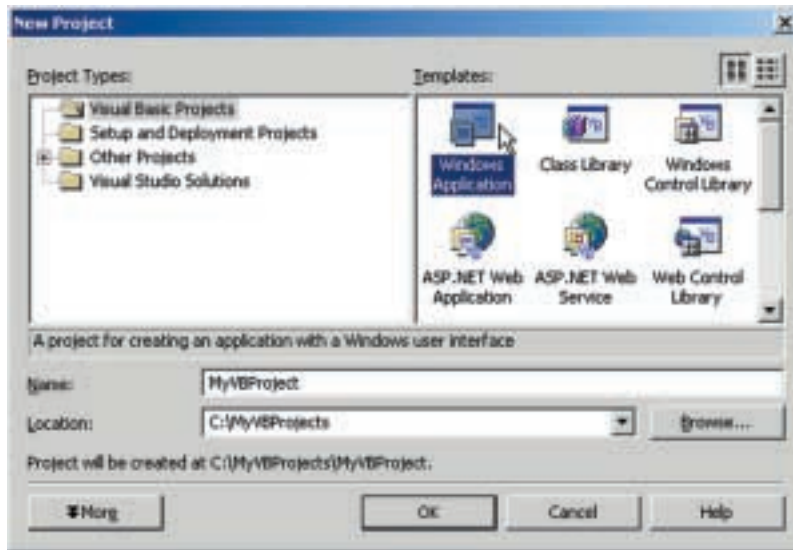
## The New Project Dialog

You will create your first Visual Basic projects based on Windows Forms. In the *New Project* dialog (Figure 1.4) select *Visual Basic Projects* in the *Project Types* box and *Windows Application* in the *Templates* box. You also give the project a name and a path on this dialog box.

- The IDE automatically creates a new folder for each project. Do not create your own, or you will have a folder within a folder.

**Figure 1.4**

*Begin a new VB .NET project using Windows Forms.*



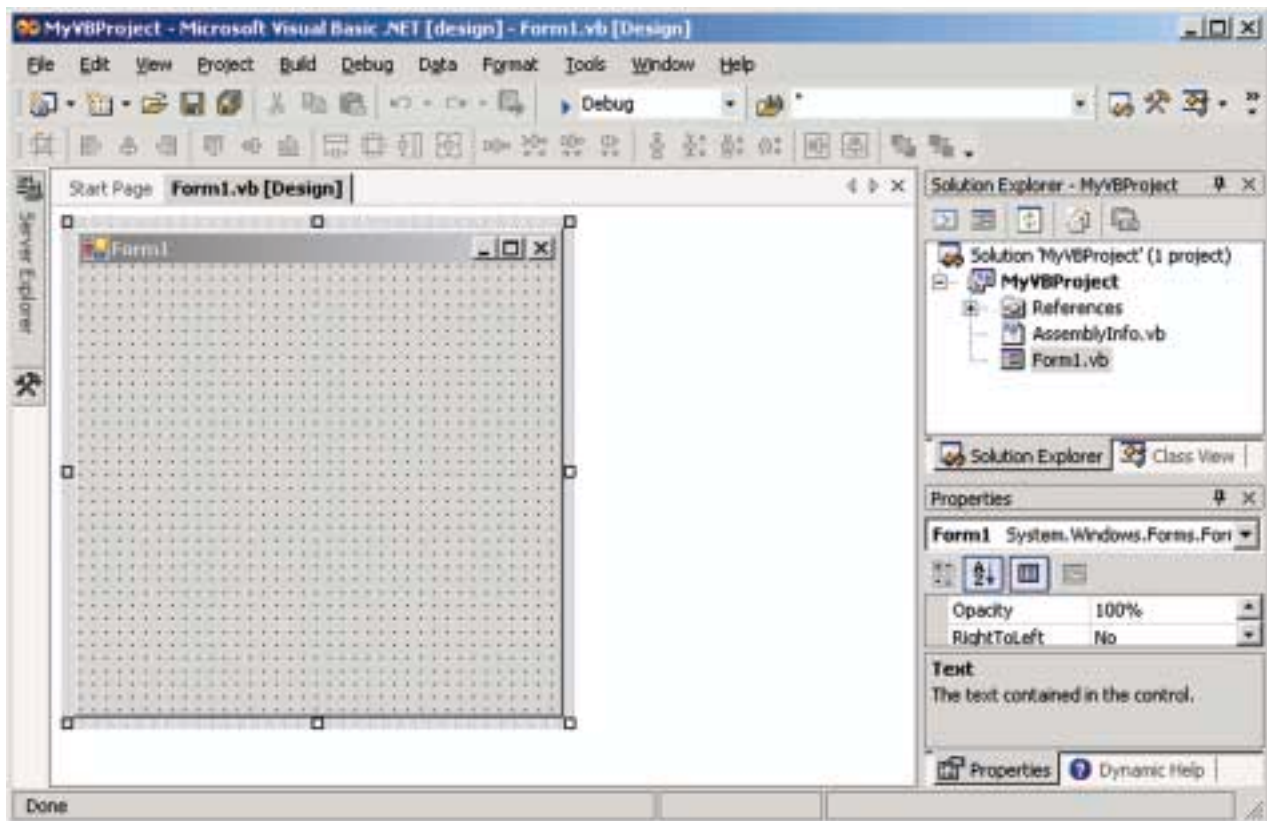
## The IDE Main Window

Figure 1.5 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 1.5; in all likelihood you will want to customize the placement of the various windows.

The IDE main window holds the Visual Studio menu bar and the toolbars.

**Figure 1.5**

*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 can also be selected from a menu. Figure 1.6a shows the toolbar buttons on the Standard toolbar, which displays in the main window of the IDE; Figure 1.6b shows the Layout toolbar, which displays in the Form Designer; and Figure 1.6c shows the Text Editor toolbar, which appears when the Editor window is displayed.

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.

**Figure 1.6**



---

## 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 Object Browser, and the pages of Help that you request.

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



**Use** Ctrl + Tab to cycle through the open documents in the Documents window. ■

---

## The Form Designer

The **Form Designer** is where you design a form that makes up your user interface. In Figure 1.5, the Form Designer for Form1 is currently displaying. You can drag the form's borders to change the size of the form.

When you begin a new Visual Basic Windows project, a new form is added to the project with the default name Form1. When you save the file, you should give it a new 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 window's title bar holds the name of your solution (.sln) file, which is WindowsApplication1 by default until you save it with a new name.

---

## The Properties Window

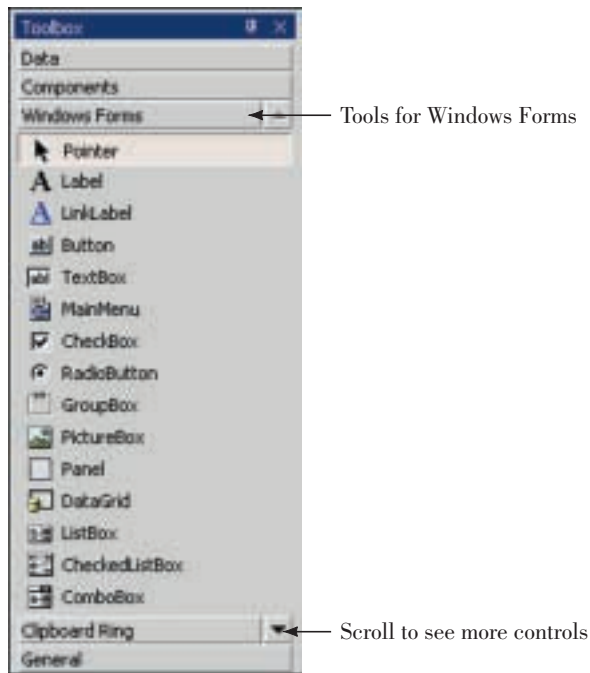
You use the **Properties window** to set the properties for the objects in your project. See “Set Properties” later in this chapter for instructions on changing properties.

---

## 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 (Professional or Enterprise). Figure 1.7 shows the toolbox.

- The toolbox autohides by default. Click the push-pin icon to pin it open.

**Figure 1.7**

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

## Help

Visual Studio has an extensive **Help** feature that is greatly expanded for .NET. Help includes the Microsoft Developer Network library (MSDN), which contains 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.

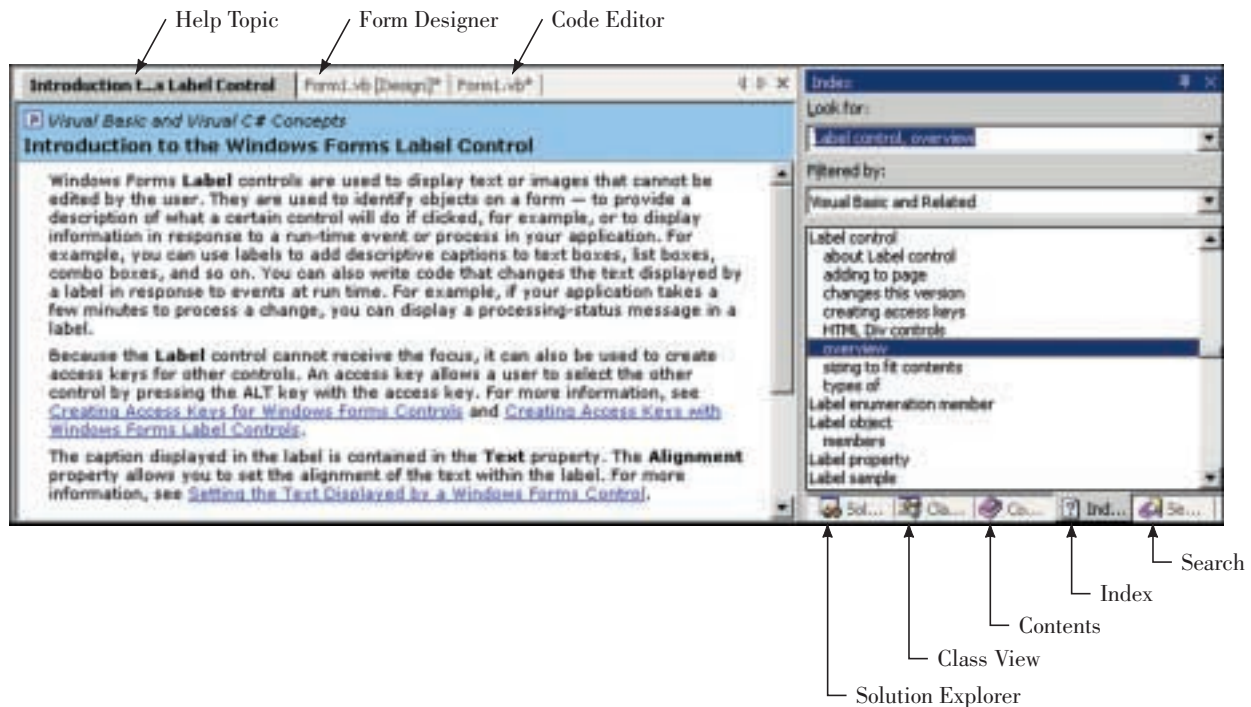
Help includes the entire reference manual, as well as many coding examples. See the topic “Visual Studio Help” later in this chapter for help on Help.

When you select *Contents*, *Index*, or *Search* from the *Help* menu, the requested item appears as another tabbed window on top of the Solution Explorer window. It’s a good idea to set the *Filtered By* entry to *Visual Basic and Related*. Once you select a topic, the corresponding Help page appears in the main Document window.

In Figure 1.8, notice the tabs across the bottom of the Solution Explorer window and on the top of the Document window. The window for the Solution Explorer now shows the Help Index and the tabs allow you to switch between the Help Index, Help Contents, the Class View, and the Solution Explorer. Use the tabs on the Document window to switch back to the Form Designer (*Form1.vb [Design]\**), the Code Editor (*Form1.vb\**), or the Help topic (*Introduction t...s Label Control*).

**Figure 1.8**

The Help Index displays on a tab in the Solution Explorer window and the Help text appears on a tab in the Document window.

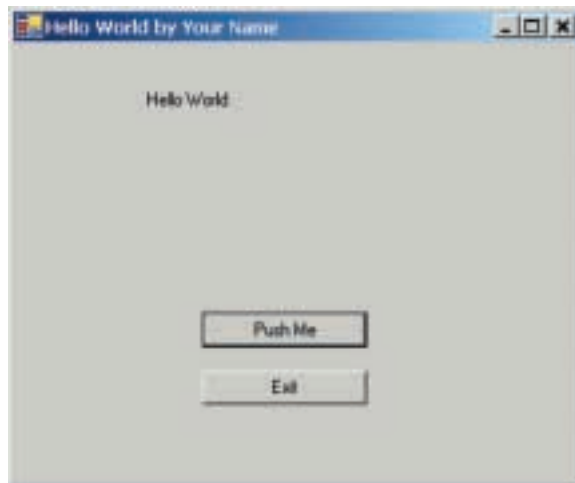


## Design Time, Run Time, and Break Time

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 project execution, you are in **break time**. The window title bar in Figure 1.5 indicates that the project is currently in design time.

## Writing Your First Visual Basic Project

For your first VB project, you will create a form with three controls (see Figure 1.9). 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.

**Figure 1.9**

The *Hello World* form. The “Hello World” message will appear in the label when the user clicks on the *Push Me* 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.

### Run Visual Studio

These instructions assume that Visual Studio .NET 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 on the Windows *Start* button and move the mouse pointer to *Programs*.

**STEP 2:** Locate *Microsoft Visual Studio .NET*.

**STEP 3:** In the submenu that pops up, select *Microsoft Visual Studio .NET*.

Visual Studio will start and display the Start Page (refer to Figure 1.3).

*Note:* The VS IDE can be customized to not show the Start Page.

### Start a New Project

**STEP 1:** Click on the New Project button. The *New Project* dialog box opens. (Refer to Figure 1.4.) Make sure that *Visual Basic Projects* is selected for *Project Types* and *Windows Application* is selected for *Templates*.

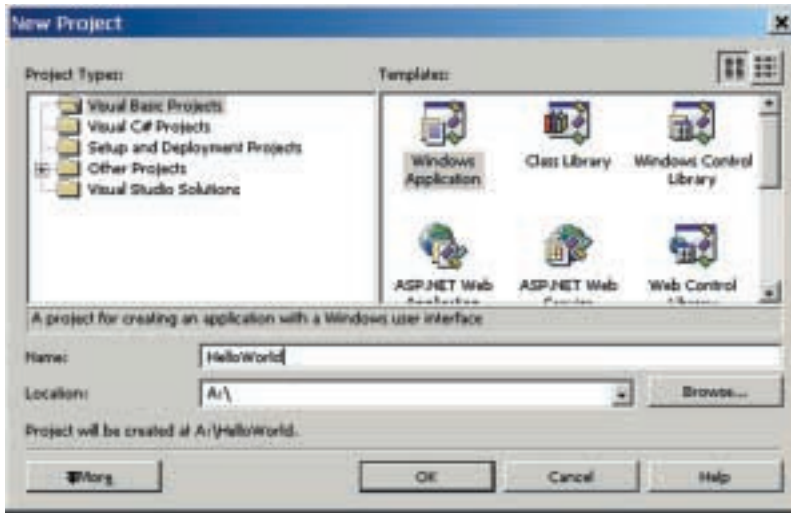
**STEP 2:** For *Location*, browse to select the path for your new project. *Note:* In this exercise we will use a diskette in the A: drive. You may use a path on the hard drive or network, if you prefer.

Do not create a new folder for your project; the VS IDE automatically creates a new folder for each new solution. If you create a folder yourself, you will have a folder within a folder.

**STEP 3:** Enter “HelloWorld” (without the quotes) for the name of the new project (Figure 1.10) and click on the OK button. The new project opens (Figure 1.11). *Note:* Your screen may look significantly different from the figure since the environment can be customized.

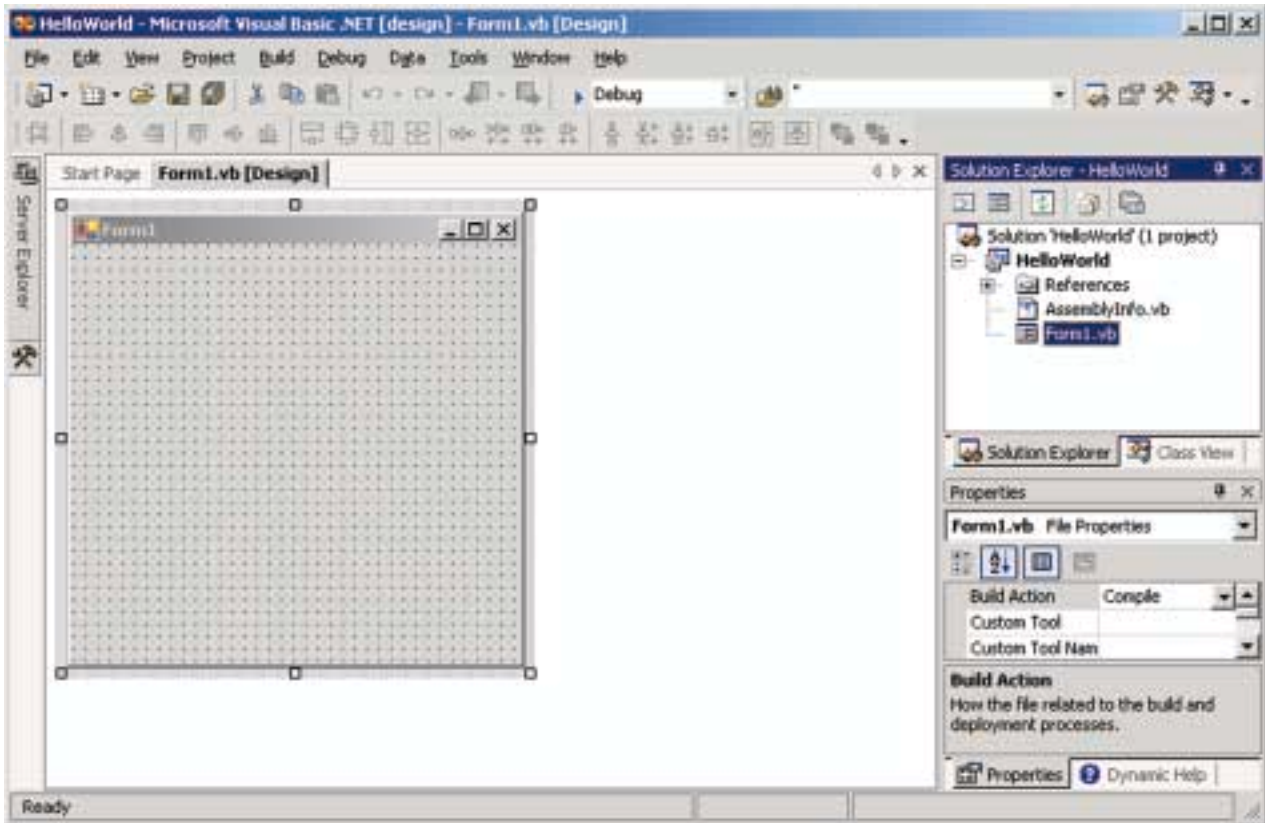
**Figure 1.10**

Select the path and enter the name for the new project.



**Figure 1.11**

Begin a new project.





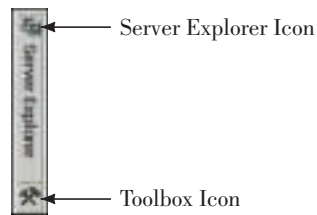
## Set Up Your Environment

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 *Tools / Options / Environment / General / Reset Window Layout*; click OK on both dialogs.

The Server Explorer and toolbox are both set to AutoHide in the same location. You don't need the Server Explorer, but you do need the toolbox.

**STEP 2:** Point to the icon for the Server Explorer at the top of the hidden window's title bar (Figure 1.12). The Server Explorer will open.



**Figure 1.12**

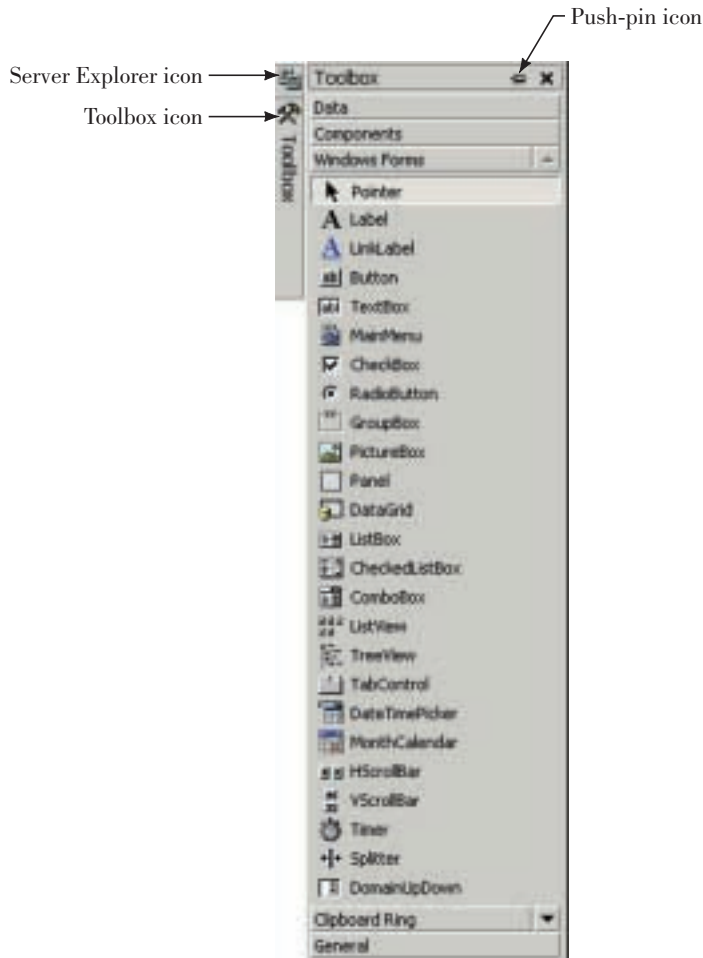
*The title bar of the hidden window for the Server Explorer and the toolbox. Point to the correct icon to display the desired window.*

**STEP 3:** Point to the icon for the toolbox at the bottom of the window's title bar. The Toolbox window opens. Notice the push-pin icon at the top of the window (Figure 1.13); clicking this icon makes the window remain on the screen rather than AutoHide.

- The AutoHide behavior of the toolbox can be confusing. Use the push-pin icon at the top of the toolbox to fix it on the screen.

**Figure 1.13**

*The Toolbox window.*

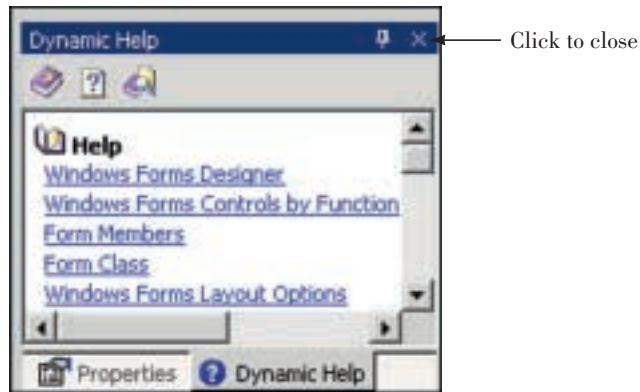


**STEP 4:** Click the AutoHide push-pin icon for the Toolbox window. The toolbox will remain open and tabs appear at the bottom of the window for the toolbox and the Server Explorer.

**STEP 5:** Click on the tab for the Server Explorer to make its window appear. Then click on the window's Close button to permanently close the window.

*Note:* You can reopen the Server Explorer from the *View* menu, if you wish.

**STEP 6:** In the lower-right corner of the screen, click on the tab for *Dynamic Help* to bring its tabbed window to the top (Figure 1.14). Then click the window's Close button to close the Dynamic Help window. Later you can experiment with Dynamic Help turned on, but the feature slows the environment significantly.

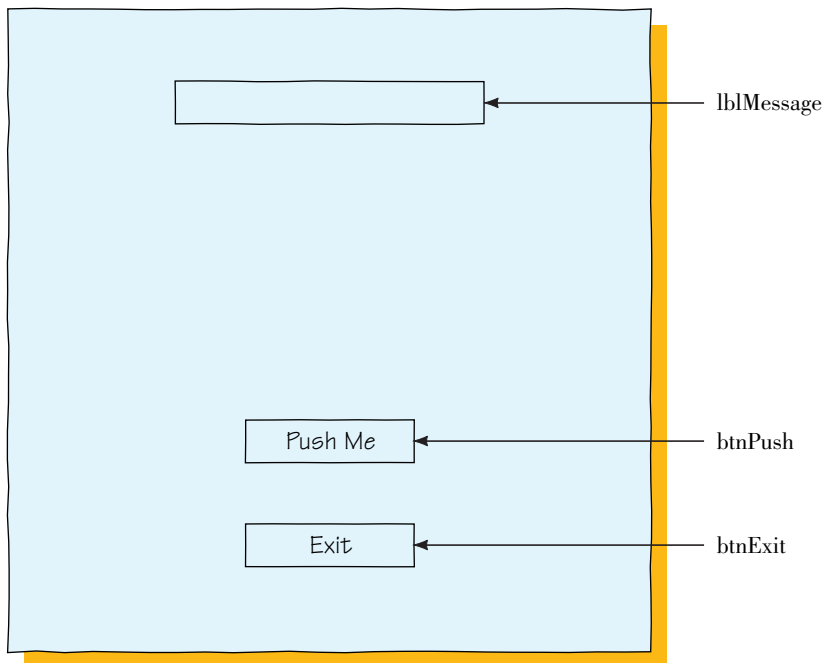
**Figure 1.14**

Click on the Close button to close the Dynamic Help window.

## Plan the Project

The first step in planning is to design the user interface. Figure 1.15 shows a sketch of the form that includes a label and two buttons. You will refer to the sketch as you create the project.

- The `PrintForm` method is no longer supported.

**Figure 1.15**

A sketch of the Hello World 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

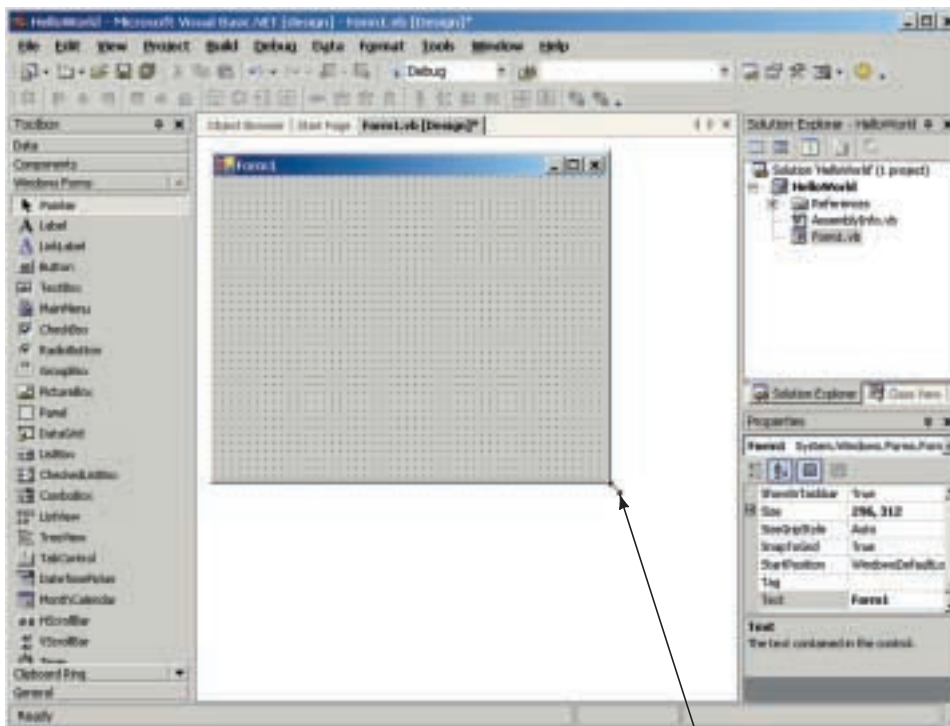
### Set Up the Form

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. The grid of dots on the form is there to help you align the controls; the grid does not appear when you run the program.

**STEP 1:** Resize the form in the Document window: Drag the handle in the lower-right corner down and to the right (see Figure 1.16).

**Figure 1.16**

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



— Drag handle to enlarge form

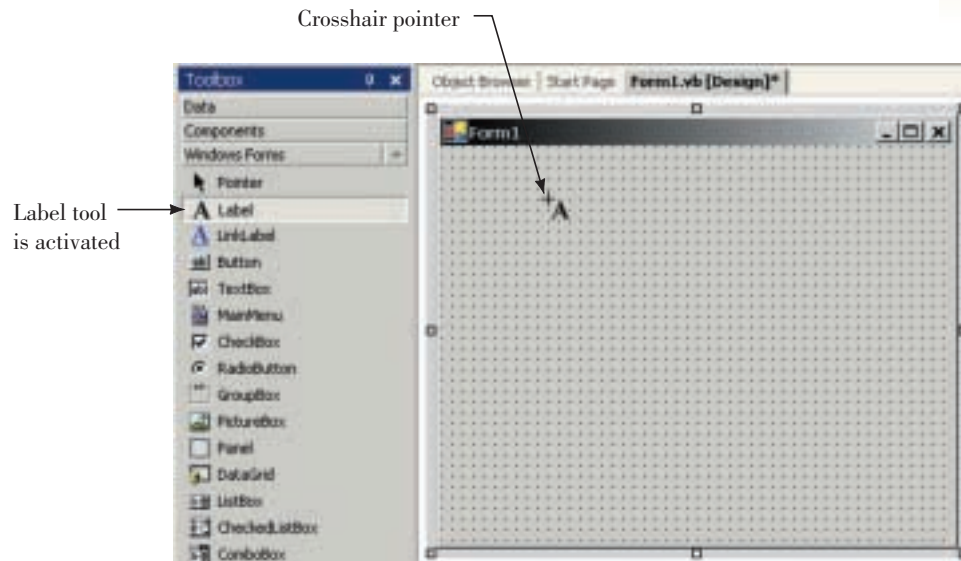
## Place Controls on the Form

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

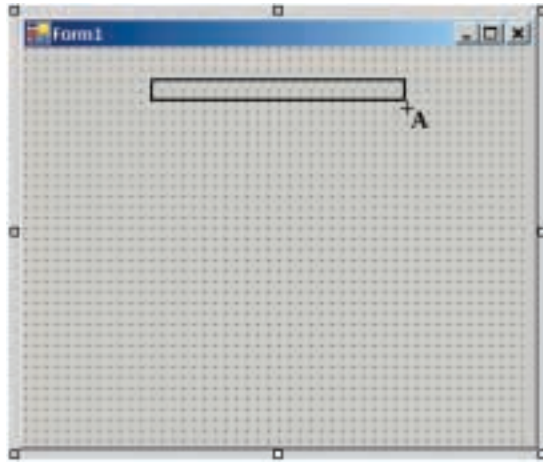
**STEP 1:** Point to the Label tool in the toolbox and click. Then move the pointer over the form. Notice that the pointer becomes a crosshair with a big A, and the Label tool looks as if it has been pressed, indicating it is the active tool (Figure 1.17).

**Figure 1.17**

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

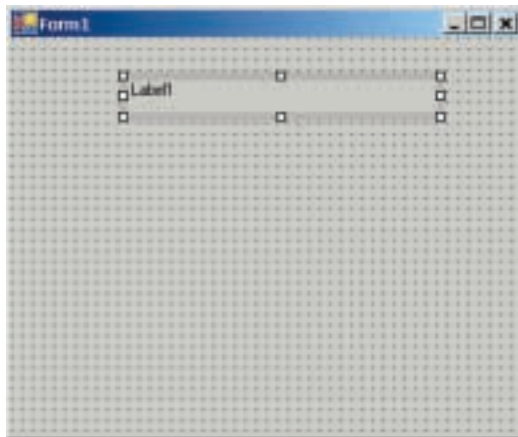


**STEP 2:** Point to a spot where you want one corner of the label, press the mouse button, and drag the pointer to the opposite corner (Figure 1.18). When you release the mouse button, the label and its default contents (Label1) will appear (Figure 1.19).



**Figure 1.18**

*Drag the mouse pointer diagonally to draw the label on the form.*



**Figure 1.19**

*The newly created label has eight small handles, indicating that it is selected. Notice that the contents of the label are set to Label1 by default.*

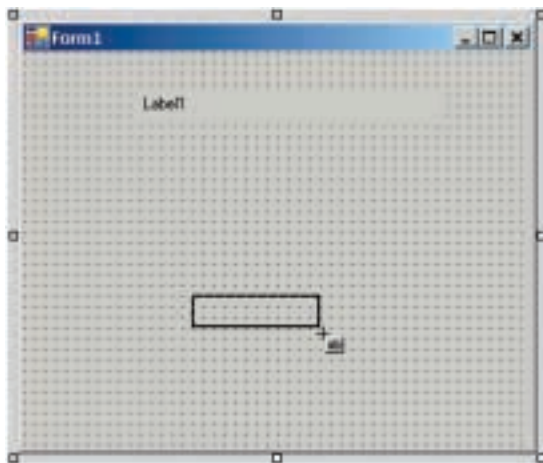
The label has eight small square **handles**, indicating that the control is currently selected. While a control is selected, you can delete it, resize it, or move it. Refer to Table 1.1 for instructions for selecting, deleting, resizing, and moving controls. Click outside of a control to deselect it.

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

Table 1.1

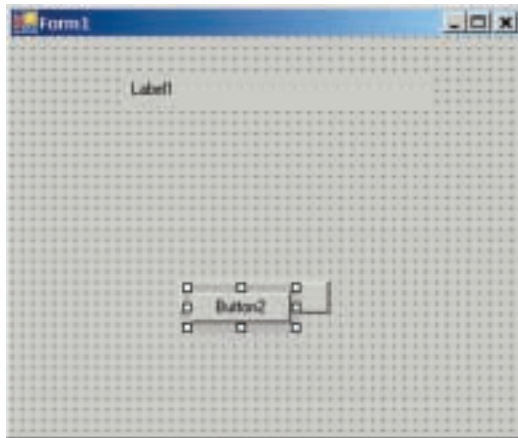
Select a control	Click on the control.
Delete a control	Select the control and then press the Delete key on the keyboard.
Move a control	Select the control, point inside the control (not on a handle), press the mouse button, and drag it to a new location.
Resize a control	Make sure the control is selected; then point to one of the handles, press the mouse button, and drag the handle. Drag a side handle to change the width, a bottom or top handle to change the height, or a corner handle to resize in two directions.

**STEP 3:** Draw a button on the form: 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 1.20). The new button should have selection handles.

**Figure 1.20**

Select the Button tool and drag diagonally to create a new Button control.

**STEP 4:** Create another button using this alternative method: 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 1.21).



**Figure 1.21**

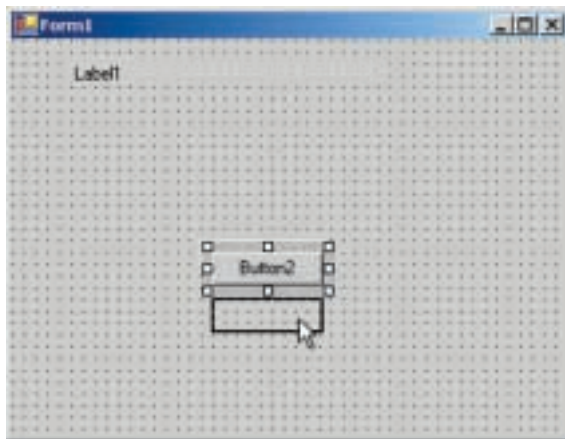
*Place a new button on the form by double-clicking the Button tool in the toolbox. The new button appears on top of the previously selected control.*

- Double-clicking a toolbox tool places the new default-sized control on top of the last-drawn control rather than in the center of the form. If no control is selected, the new control appears at the top-left of the form.

**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 5:** Keep the new button selected, point anywhere inside the button (not on a handle), and drag the button below your first button (Figure 1.22). As you drag the control, you see only its outline; when you release the mouse button, the control is actually moved to its new location.



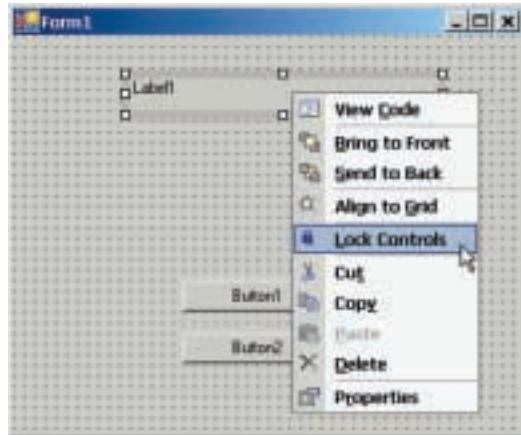
**Figure 1.22**

*Drag the new button (Button2) below Button1. An outline of the control shows the new location for the control.*

**STEP 6:** Select each control and move and resize the controls as necessary. Make the two buttons the same size and line them up.



**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 1.23). Locking prevents you from accidentally moving the controls. When your controls are locked, a selected control has no handles.



**Figure 1.23**

*After the controls are placed into the desired location, lock them in place by selecting *Lock Controls* from the context menu.*

*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.

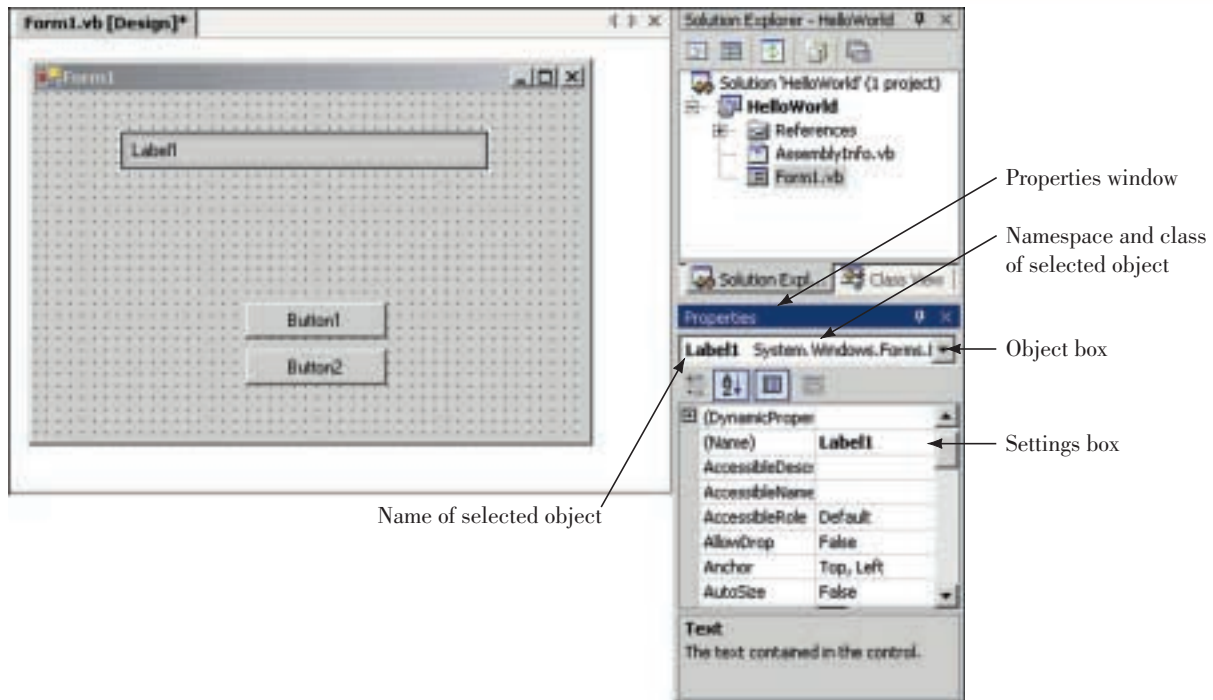
## Set Properties

### Set the Name and Text Properties for the Label

**STEP 1:** Click on the label you placed on the form; a shaded outline appears around the control. Next click on the title bar of the Properties window to make it the active window (Figure 1.24).

**Figure 1.24**

The currently selected control is shown in the 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.

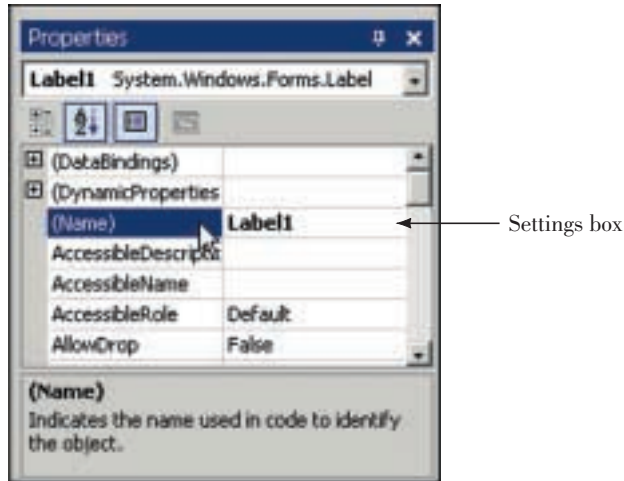
**STEP 2:** Select the Name property. You may have to scroll up; Name is located near the top of the list. Click on *(Name)* and notice that the Settings box shows *Label1*, the default name of the label (Figure 1.25).



If the Properties window is not visible, you can press the F4 key to show it. ■

**Figure 1.25**

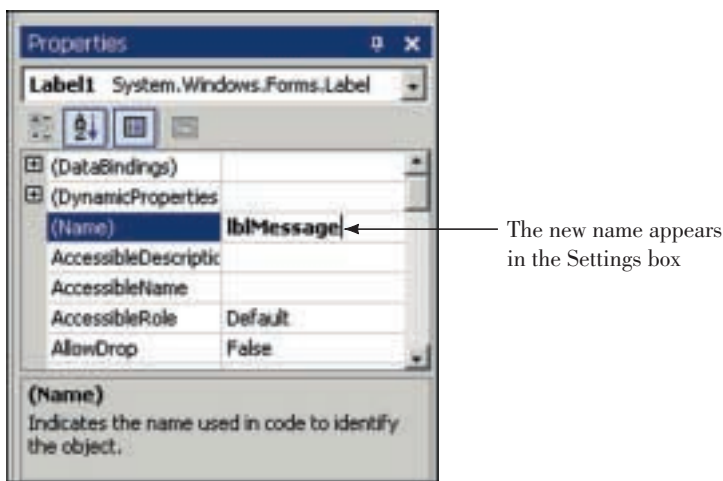
*The Properties window. Click on the Name property to change the value in the Settings box.*



**STEP 3:** Type “lblMessage” (without the quotation marks). See Figure 1.26. After you change the name of the control, you can see the new name in the Object box’s drop-down list.

**Figure 1.26**

*Type “lblMessage” into the Settings box for the Name property.*



**STEP 4:** Click on the Text property to select it. Scroll the list if necessary. Following the Name property, all properties are in alphabetic order.

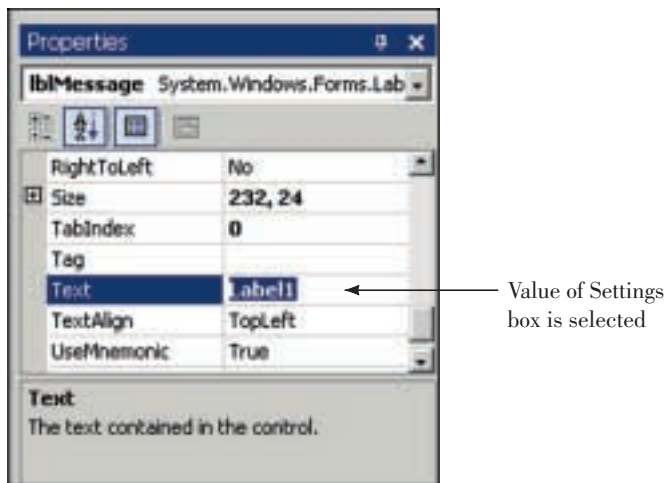
The **Text property** of a control determines what will be displayed on the form. Because nothing should display when the program begins, you must delete the value of the Text property (as described in the next two steps).

**STEP 5:** Double-click on *Label1* in the Settings box; the entry should appear selected (highlighted). See Figure 1.27.

- The Caption property of a form, label, and button changes to the Text property.

**Figure 1.27**

*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 now appears empty (Figure 1.28). Changes do not appear until you press Enter or move to another property or control.

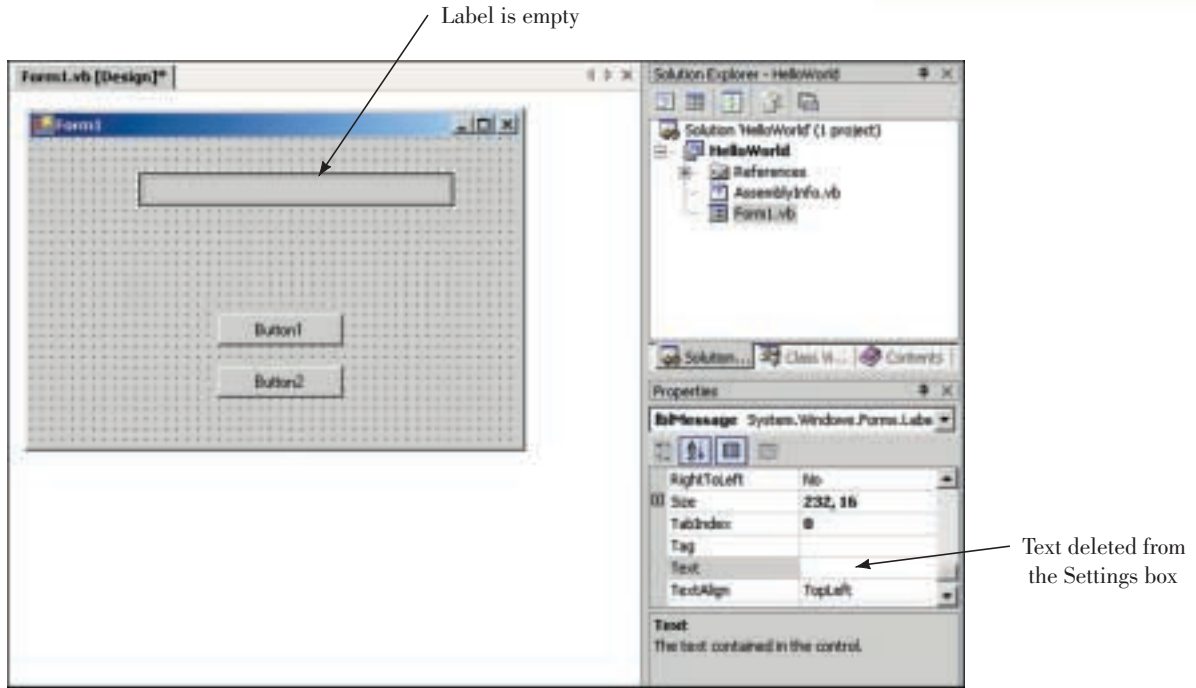
As an alternate technique, you can double-click on the property name, which automatically selects the entry in the Settings box. Then you can press the Delete key or just begin typing to change the entry.

### TIP

Don't confuse the Name property with the Text property. You use the Name property to refer to the control in your Basic code. The Text property tells 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. ■

**Figure 1.28**

Delete the value for the Text property from the Settings box; the label on the form also appears empty.

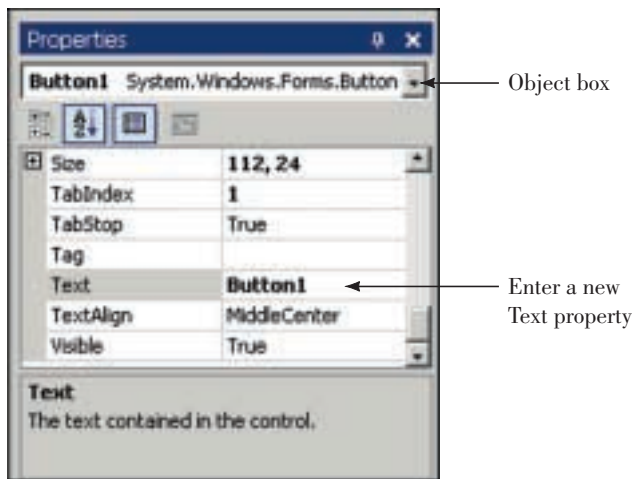


**Set the Name and Text Properties for the First Button**

**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 1.29.

**Figure 1.29**

Change the Text property for the first button.



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.

**STEP 2:** Change the Name property of the button to “btnPush” (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 “btnExit”.

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



**Always** set the Name property of controls before writing code. ■

### Change Properties of the Form

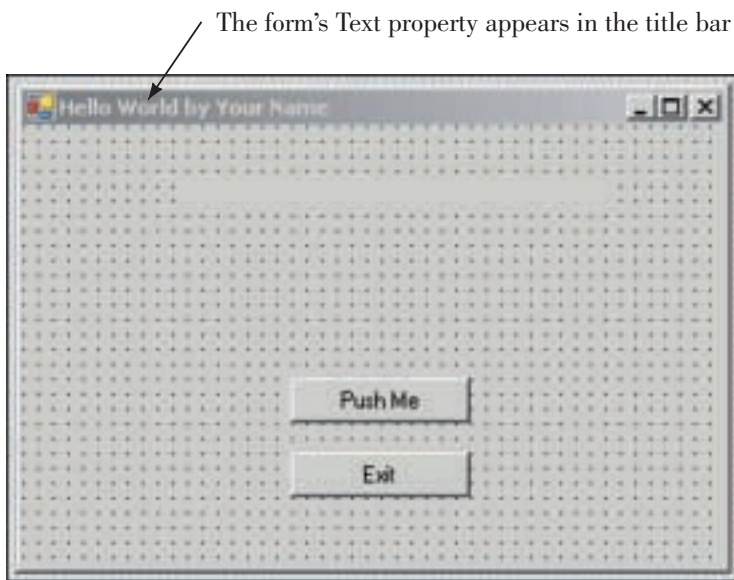
**STEP 1:** Click anywhere on the form, except on a control. The Properties window Object box should now show the form as the selected object (*Form1* as the object’s name and *System.Windows.Forms.Form* as its class).

**STEP 2:** Change the Text property to “Hello World by Your Name” (again, no quotation marks).

The Text property of a form determines the text to appear in the title bar. Your screen should now look like Figure 1.30.

**Figure 1.30**

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

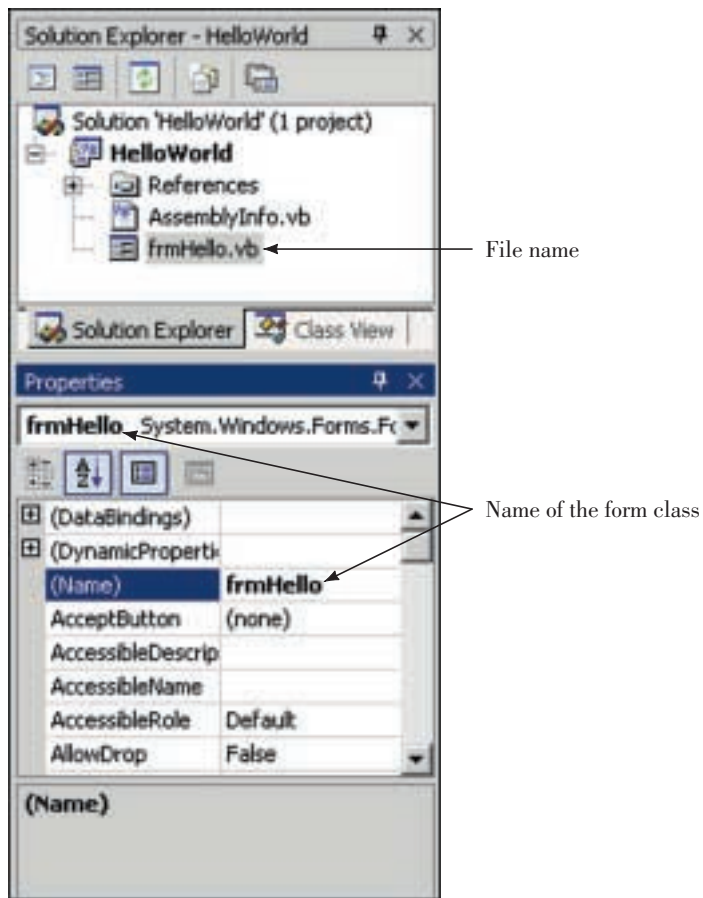


- STEP 3:** Click on the `StartPosition` property and notice the arrow on the property setting, indicating a drop-down list. Drop down the list and select `CenterScreen`. This will make your form appear in the center of the screen when the program runs.
- STEP 4:** Change the form's `Name` property to "frmHello". This step changes the name of the form's class, but not the name of the form's file, which is still `Form1`.
- STEP 5:** In the Solution Explorer, right-click on `Form1.vb` and choose *Rename* from the shortcut menu. Change the file name to "frmHello.vb", making sure to retain the `.vb` extension. Press Enter when finished. Now the form's class and its file should both be renamed (Figure 1.31).

- The IDE does not have a *Form Layout* window. Use the `StartPosition` property of the form.
- Changing the name of the form does not change the name of the form's file. Rename the file using the Solution Explorer to keep the project pointing to the correct filename.

**Figure 1.31**

*Change the name of the form class and the name of the form's file.*

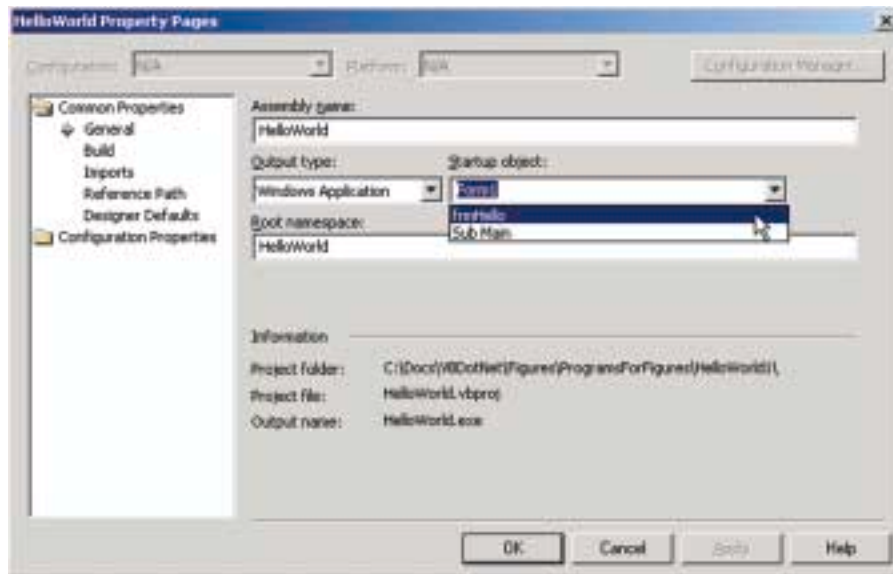


### Set the Project's Startup Object

Whenever you change the name of the form, you must take one more step so that VB knows which form to run when the project begins. Each project has a *Startup Object*—the object with which to begin execution. By default, the *Startup Object* is `Form1`. If you change the name of the form, you must set the project's *Startup Object* property to the new name of the form.

- When you change the name of the startup form, VB does not automatically change the project's *Startup Object*. You must open the *Project Properties* dialog box and change it manually.

**STEP 1:** In the Solution Explorer, click on HelloWorld to select the project. Then you can either select *Project / Properties* or right-click on the project name in the Solution Explorer and select *Properties* from the shortcut menu. In the *Project Properties* dialog box, drop down the list for *Startup object* and select *frmHello* (Figure 1.32).



**Figure 1.32**

In the project's *Property Pages* dialog box, drop down the list for *Startup object* and select the name of your form.

If you ever receive an error message when you attempt to run a project telling you that it can't find Form1, you know that you have forgotten this step.

## 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



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.

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 btnPush will be btnPush\_Click. For the sample project you are writing, you will have a btnPush\_Click procedure and a btnExit\_Click procedure.

## Visual Basic Code Statements

This first project requires two Visual Basic statements: the remark and the assignment statement. You will also 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 sub procedure should begin with a remark that describes the purpose of the sub. 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  
lblMessage.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 assignment statement in the example would read “lblMessage.Text is replaced by 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

```
lblTitle.Text = "A Snazzy Program"
lblAddress.Text = "1234 South North Street"
lblMessage.AutoSize = True
intNumber = 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.

- All method names require parentheses.

### Examples

```
btnHello.Hide()
lblMessage.Show()
```

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

```
Me.Close()
```

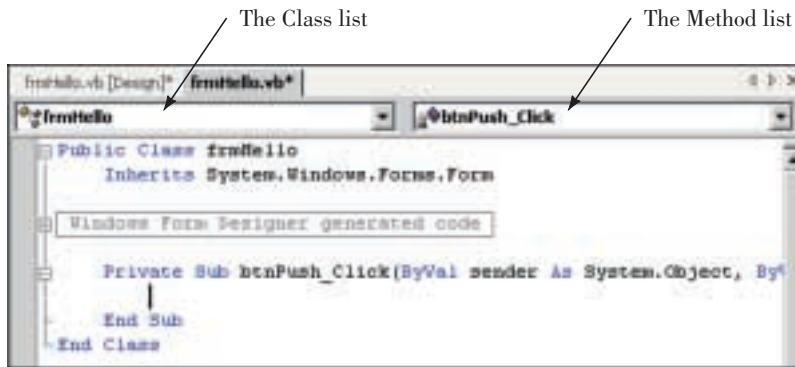
In most cases, you will include `Me.Close()` in the sub procedure for an Exit button or an *Exit* menu choice.

## Code the Event Procedures for Hello World

### Code the Click Event for the Push Me Button

**STEP 1:** Double-click on 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 1.33). For now, you can ignore the extra lines of code that appear above your sub procedure.

- All references to the *Code window* are changed to the *Editor window* or just *editor*.

**Figure 1.33**

The Editor window, showing the first and last lines of the `btnPush_Click` sub 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 the 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:

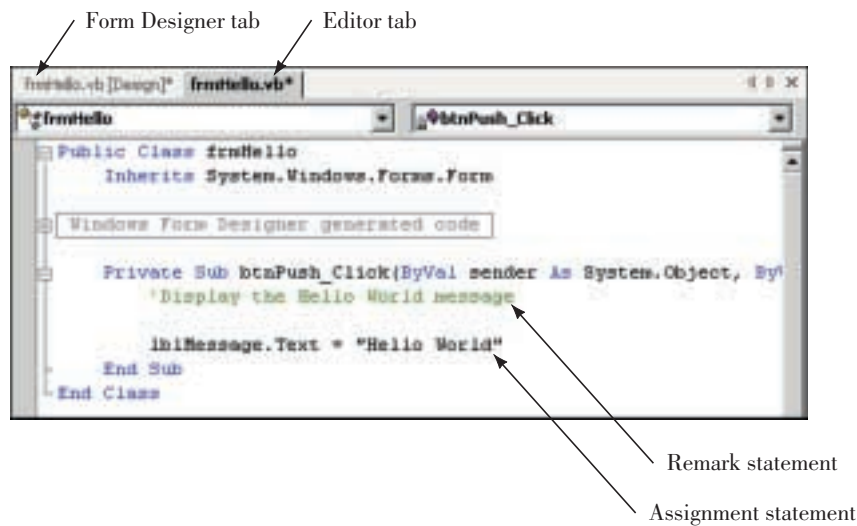
```
lblMessage.Text = "Hello World"
```

*Note:* When you type the period after `lblMessage`, an IntelliSense list pops up showing the properties and methods available for a Label control. Although you can type the entire word `Text`, you can allow IntelliSense to help you. As soon as you type the `T`, the list automatically scrolls to the first word that begins with `T`. Type the next letter, `e`, and the property `Text` appears highlighted. You can press the spacebar to select the word and continue typing the rest of the statement.



Allow the editor and IntelliSense to help you. If the IntelliSense list does not pop up, you likely misspelled the name of the control. 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. ■

This assignment statement assigns the literal “Hello World” to the Text property of the control called lblMessage. Compare your screen to Figure 1.34.



**Figure 1.34**

Type the remark and assignment statement for the `btnPush_Click` event sub procedure.

**STEP 4:** Return to the form (Figure 1.30) by clicking on the `frmHello.vb [Design]` tab on the Document window (Figure 1.34).

### Code the Click Event for the Exit Button

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

**STEP 2:** Type this remark:

```
'Exit the project
```

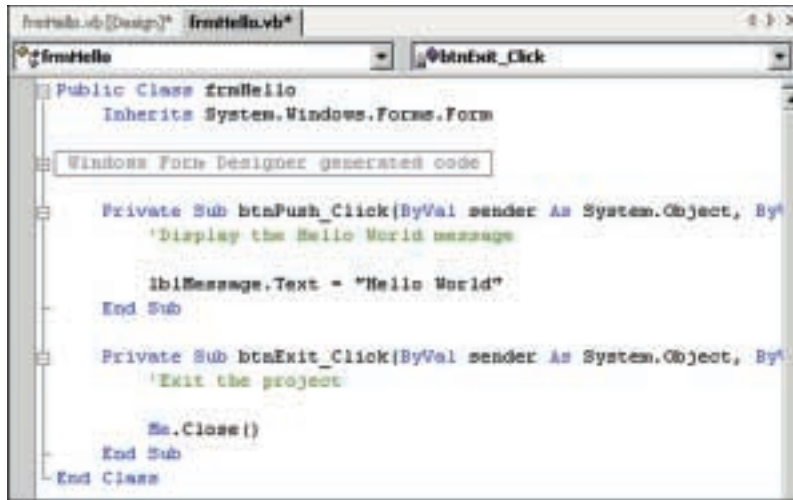
**STEP 3:** Press Enter twice and type this Basic statement:

```
Me.Close()
```



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

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



```
frmHello.vb [Design] frmHello.vb
Public Class frmHello
    Inherits System.Windows.Forms.Form

    Windows Form Designer generated code

    Private Sub btnPush_Click(ByVal sender As System.Object, ByVal
        'Display the Hello World message

        lblMessage.Text = "Hello World"
    End Sub

    Private Sub btnExit_Click(ByVal sender As System.Object, ByVal
        'Exit the project

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

**Figure 1.35**

Type the code for the `btnExit_Click` event procedure.

## Run the Project

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

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

## Start the Project Running

**STEP 1:** Choose one of the three methods previously listed to start your project running.

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



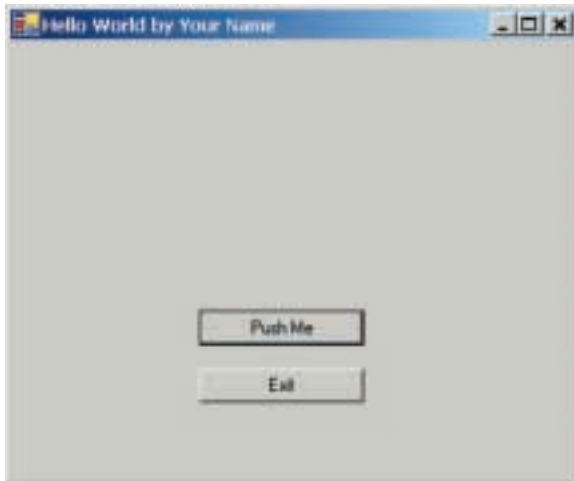
**TIP**

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

If all went well, the Visual Studio title bar now indicates that you are in run time, and the grid dots have disappeared from your form (Figure 1.36). (The grid dots help you align the controls; you may turn them off if you prefer.)

**Figure 1.36**

*When you run the project, the form's grid dots disappear.*

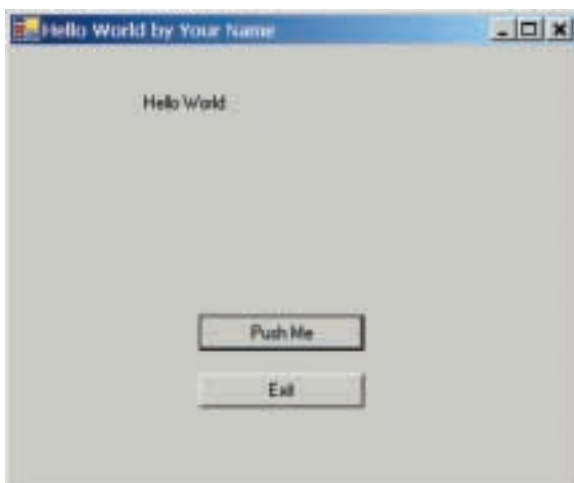


### Click the Push Me Button

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

**Figure 1.37**

*Click on the Push Me button and “Hello World” appears in the label.*



### Click the Exit Button

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

## 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.

### 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 path for the files when you first created the project.

### Close the Project

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

After your project closes, you should again see the Visual Studio Start Page. This time you may see your project on the list.

*Note:* If the Start Page does not appear, display it with *Help/Show Start Page*.

## Open the Project

---

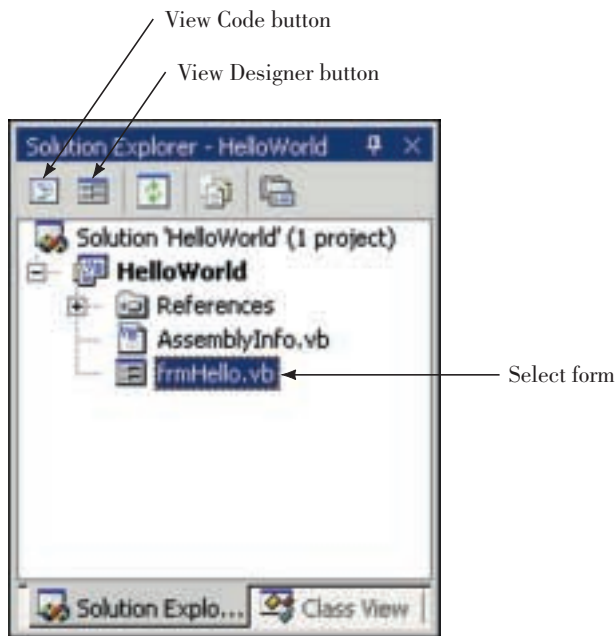
Now is the time to test your save operation by opening the project from disk. You can choose one of three ways to open a saved project:

- If your project appears on the Start Page, you can open it by clicking on its name.
- Click on the Open Project button on the Start Page and browse to find your .sln file.
- Select *Open Solution* from the Visual Studio *File* menu and browse to find your .sln file.

### 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 .sln file.

**STEP 2:** If you do not see your form on the screen, check the Solution Explorer window—it should say *HelloWorld* for both the solution and the project. Select the icon for your form: *frmHello.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 Form Designer. Notice that you can also 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, 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:** Click on the label on your form, which will make selection handles appear. (If you see a dark border instead of selection handles, you must unlock the controls, as described in Step 1.)
- STEP 3:** Widen the label on both ends by dragging the handles wider. (Drag the right end farther right and the left end farther left.)

- Locked controls now have a dark border instead of selection handles. If no control is selected, the form has the dark border that indicates the controls are locked.

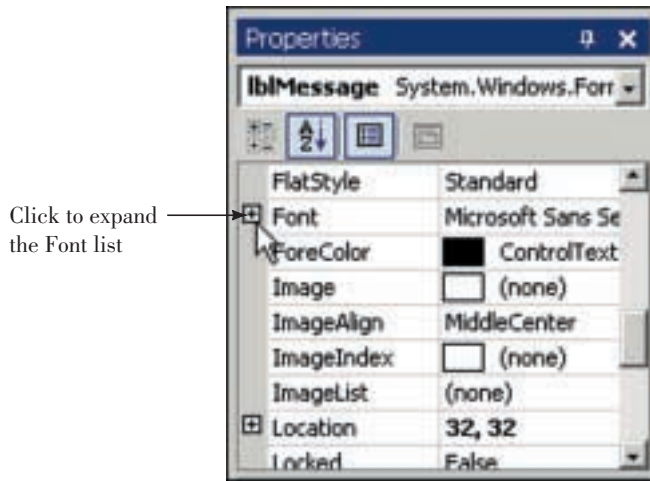


**STEP 4:** With the label still selected, scroll to the Font property. The Font property is actually a Font object that has a number of properties. To see the Font properties, click on the small plus sign on the left (Figure 1.39); the Font properties will appear showing the current values (Figure 1.40).

- The Font property changes to show the individual properties of the Font object.

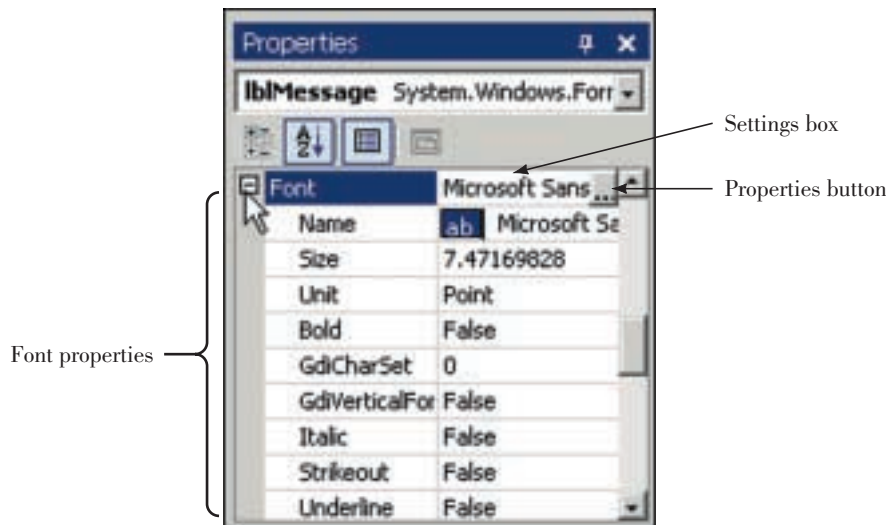
**Figure 1.39**

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.



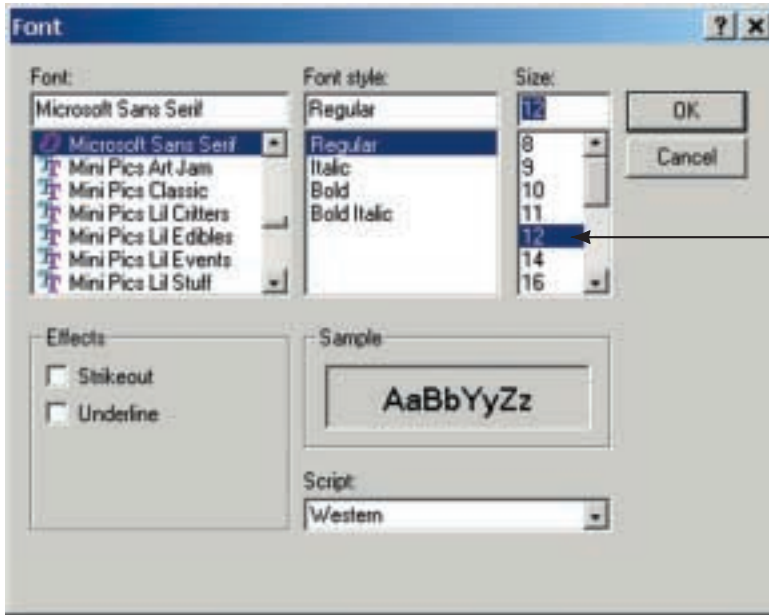
You can change any of the Font properties in the Properties window, such as setting the Font's Size, Bold, or Italic properties. You can also display the *Font* dialog box and make changes there.

To display the *Font* dialog box, click on the button with an ellipsis on top, which appears in the Settings box. The button is called the *Properties button* (sometimes the *Builder button*); the ellipsis indicates that clicking on the button will display a dialog box with choices.

**STEP 5:** Click on the Properties button 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 from the *Font* dialog box.



Select 12 point

**STEP 6:** 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*.

- The *Alignment* property becomes *TextAlign*.

**Figure 1.42**

Select the center box for the *TextAlign* property.

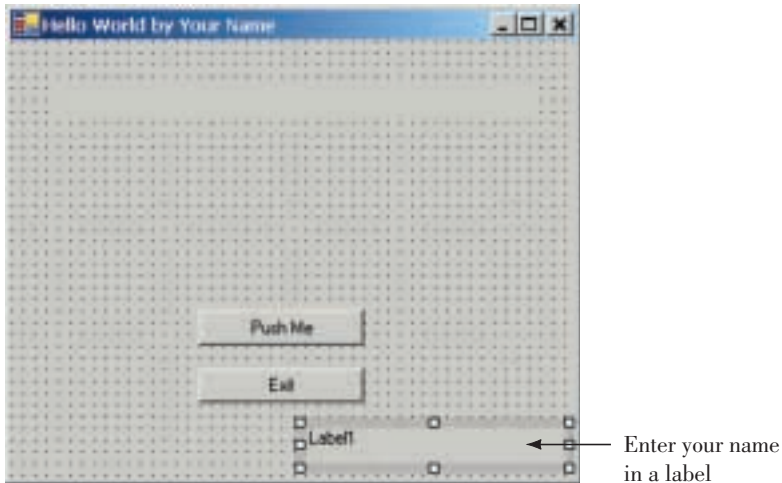


### Add a New Label for Your Name

**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.)

**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 name this label because it will never be referred to in the code.

### Change the Location and Text of the Push Me Button

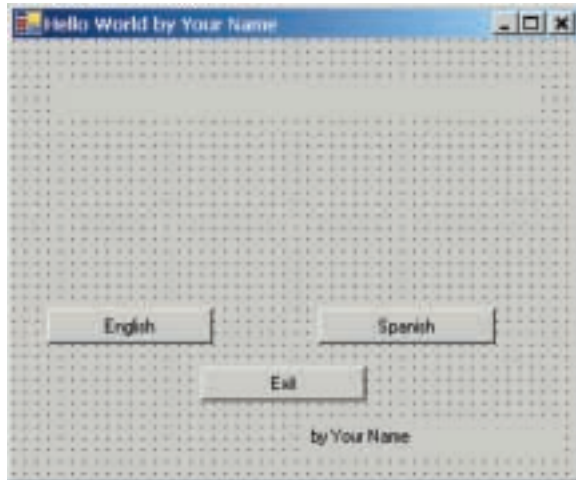
Because we plan to display the message in one of two languages, we'll change the text on the Push Me button to "English" and move the button to allow for a second button.

**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).

**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 it as necessary, referring to Figure 1.44.

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

**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 btnSpanish\_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:

```
lblMessage.Text = "Hola Mundo"
```

**STEP 4:** 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 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 frmHello). Make sure that you have an insertion point; if the entire first line is selected, press the left arrow to set the insertion point.

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 2:** Move the insertion point up to the blank line and type the following remarks, one per line (Figure 1.45):

- References to *General Declarations section* are changed to *Declarations section*.

**Figure 1.45**

Enter remarks in the Declarations section of the form file.



```
'Project:      Hello World
'Programmer:   Your Name (Use your own name here.)
'Date:         Today's Date
'Description:  This project will display a "Hello World"
message in two different languages.
```

### Explore the Editor Window

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

- The two list boxes at the top of the Editor window are called the *Class list* and the *Method list*. They work a little differently from the old lists. You must select from both lists to jump to a procedure. And to find the *Form\_Load* procedures, select (*Base Class Events*) from the *Class list* and *Load* from the *Method list*.

**STEP 2:** Click on the left down-pointing arrow to view the Class list. Notice that every object in your form is listed there (Figure 1.46). At the top of the list, you see the name of your form and project: *frmHello*.



**Figure 1.46**

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

**STEP 3:** Click on *frmHello* to select it. Then notice the Method list on the right, which says (*Declarations*). Clicking on (*Declarations*) is the quick way to jump to the Declarations section of a module.

**STEP 4:** Drop down the Class list (the left list) and select *btnSpanish*.

**STEP 5:** Drop down the Method 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.

**STEP 6:** Select the Click event from the Method list; the insertion point jumps to the event procedure for *btnSpanish*. You are currently viewing the *btnSpanish\_Click* event procedure.

To write code for more than one event for an object, use the Method drop-down list. You can jump to another procedure by selecting its name from the list. Selecting a new event from the Method list causes the Editor to generate the `Sub` and `End Sub` lines for that procedure.

**STEP 7:** Select *frmHello* in the Class list and drop down the Method list. Notice that your event procedures are all listed. Try selecting *btnPush\_Click* and *btnExit\_Click* to jump to each of those procedures.

## 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.

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

- The IDE will print only code. It's no longer possible to print a form image or form text.

If you want to have students turn in a printout of a form, have them press `Alt + Print Screen` while the program is running. This saves the current window to the Clipboard. Then have them paste the Clipboard contents into a Word document. You can instruct them to add their name and class info before printing.

You can change an option to print or not print the heading line on the code printout.

## 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.

If you are using a color printer, the colors on the screen will also appear on the printed output.

```
A:\HelloWorld\frmHello.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.

Public Class frmHello
    Inherits System.Windows.Forms.Form

Windows Form Designer generated code

    Private Sub btnPush_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) ↵
        Handles btnPush.Click
            'Display the Hello World Message

            lblMessage.Text = "Hello World"
        End Sub

    Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) ↵
        Handles btnExit.Click
            'Exit the project

            Me.Close()
        End Sub

    Private Sub btnSpanish_Click(ByVal sender As System.Object, ByVal e As System. ↵
        EventArgs) Handles btnSpanish.Click
            'Display the Hello World message in Spanish

            lblMessage.Text = "Hola Mundo"
        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

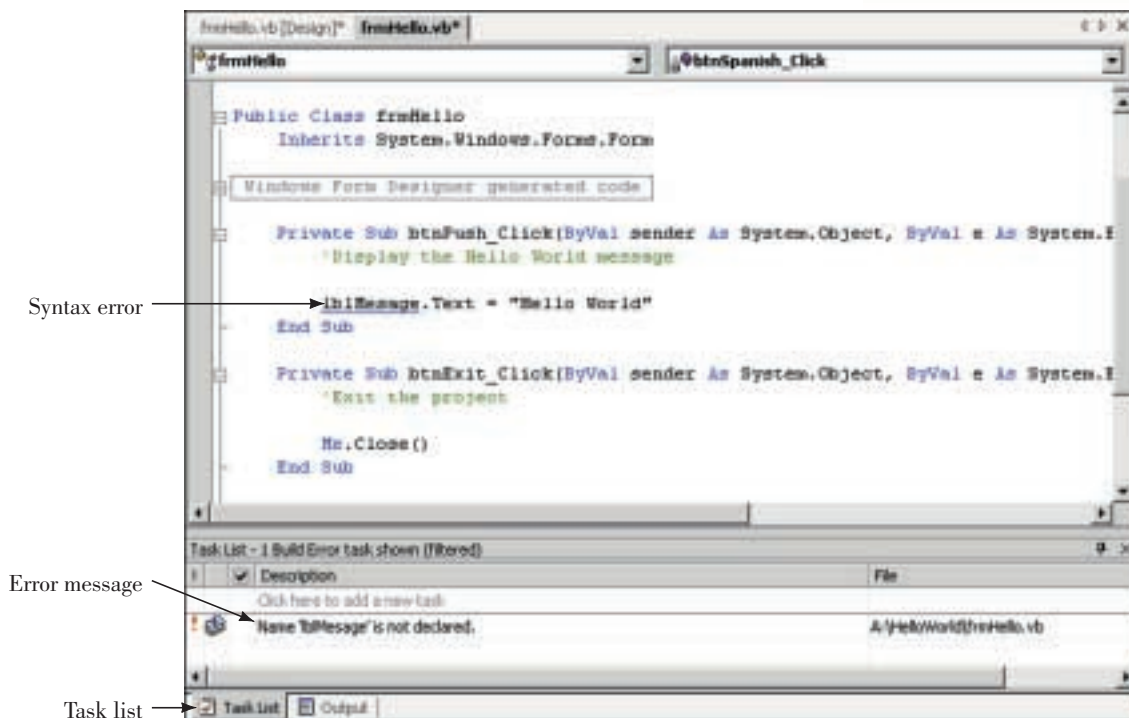
are found and reported by the compiler as it attempts to convert the code into intermediate machine language. A compiler-reported syntax error may be referred to as a *compile error*.

The editor can correct some syntax errors by making assumptions and not even report the error to you. For example, if you type the opening quote of “Hello World” but 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?

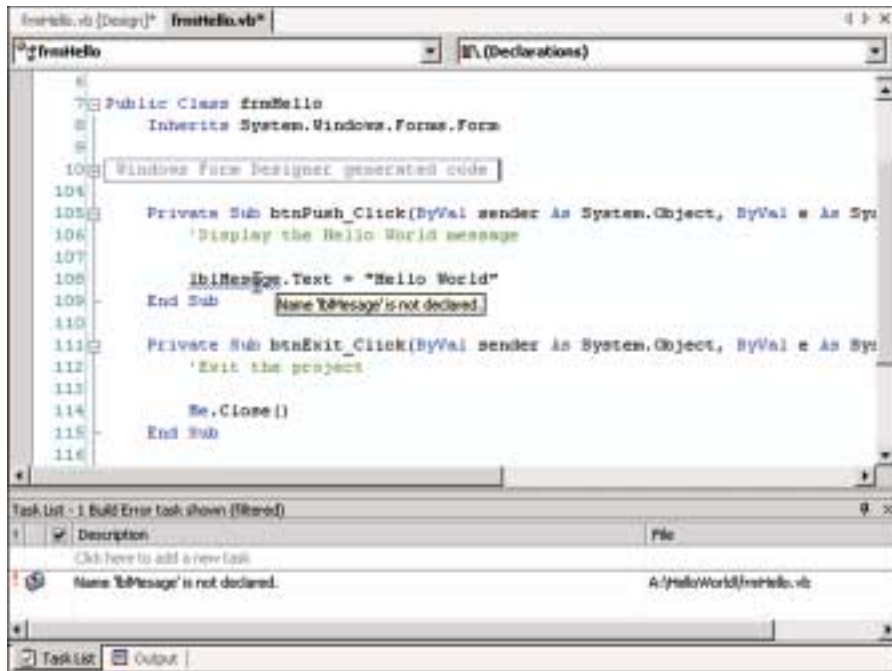
The editor identifies syntax errors as you move off the offending line: A blue squiggly line appears under the part of the line that the editor cannot interpret, and a message appears in the Task list at the bottom of the screen (Figure 1.47). Notice also that the Task list shows the line number of the statement that caused the error. You can display line numbers on the source code (Figure 1.48) with *Tools/Options/Text Editor/Basic/Display/Line Numbers*. You can also pause the mouse pointer over the error line to pop up an error message (refer to Figure 1.48).

**Figure 1.47**

The editor identifies a syntax error with a squiggly blue line and places a message in the Task list.



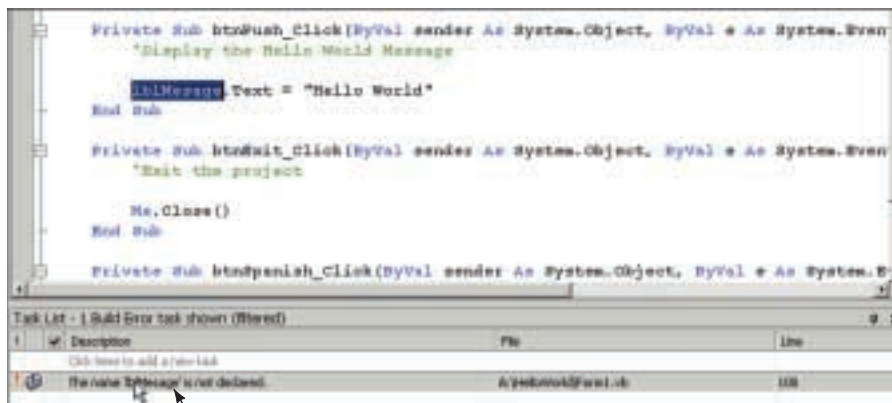


**Figure 1.48**

You can display line numbers in the source code to help identify the lines, and you can point to an error to pop up the error message.

*Note:* If the Task list does not appear, show it with *View/Other Windows/Task List*.

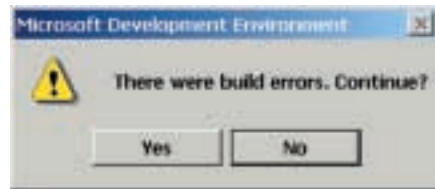
The quickest way to jump to an error line is to point to a message in the Task list and double-click. The line in error will display in the Editor window with the error highlighted (Figure 1.49).

**Figure 1.49**

Quickly jump to the line in error by double-clicking on the error message in the Task list.

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

If a syntax error is found by the compiler, you will see the dialog box shown in Figure 1.50. Click No and return to the editor, correct your errors, and run the program again.



**Figure 1.50**

*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. Run-time errors can be caused by attempting to do impossible arithmetic operations, such as calculate with non-numeric 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.

- No edit-and-continue as in VB 6. If you modify code during break time, you must restart the program. Attempting to continue from break time causes it to execute the “old” unchanged code. This behavior is due to the fact that VB must completely compile a program before running it.

## 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 OK 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 printing “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 cmdSpanish_Click
    'Display the Hello World Message in Spanish

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

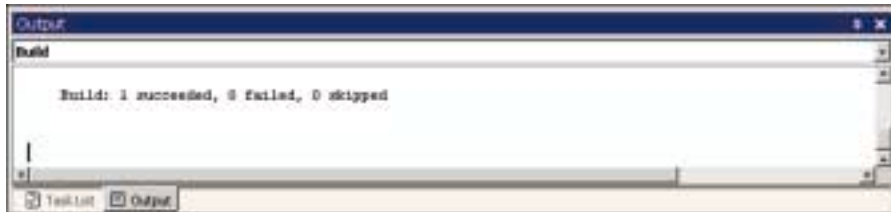
## Project Debugging

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.

### A Clean Compile

After you locate the problem and fix it, you must recompile the program and run it again. Each time you compile the program, you must have a **clean compile**, which means zero errors (Figure 1.51). You are looking for this line in the Output window:



**Figure 1.51**

*Zero build errors means that you have a clean compile.*

```
Build: 1 succeeded, 0 failed, 0 skipped
```

You can confuse yourself if you try to run a program without first getting a clean compile. For example, say you *do* get a clean compile and run the program once; then you make some modifications to the program and tell it to run again. If you ignore the error message (refer to Figure 1.50) and attempt to run it anyway, you will actually be running the last cleanly compiled version, without the changes that you just made.

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

## 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. The name can contain letters, digits, and underscores. An object name cannot include a space or punctuation mark.

## The Naming Conventions

This text follows standard naming conventions, which help make projects more understandable. Always begin a name with a lowercase three-letter prefix, which identifies the object type (such as label, button, or form) and capitalize the first character after the prefix (the “real” name of the object). For names with multiple words, capitalize each word in the name. All names must be meaningful and indicate the purpose of the object.

## Examples

```
lblMessage
btnExit
frmDataEntry
lblDiscountRate
```

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 that never change during project execution. These 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 the list of object prefixes.

- The maximum length for identifiers changes from 40 characters to 16,383—probably not a problem for most programmers. In the text we just omitted the statement concerning a maximum length.

## Recommended Naming Conventions for Visual Basic Objects

**Table 1.2**

Object Class	Prefix	Example
Form	frm	frmDataEntry
Button	btn	btnExit
TextBox	txt	txtPaymentAmount
Label	lbl	lblTotal
Radio button	rad	radBold
CheckBox	chk	chkPrintSummary
Horizontal scroll bar	hsb	hsbRate
Vertical scroll bar	vsb	vsbTemperature
PictureBox	pic	picLandscape
ComboBox	cbo	cboBookList
ListBox	lst	lstIngredients

## Visual Studio Help

Visual Studio has an extensive Help facility, which contains lots more information than you will ever use. You can look up any Basic statement, object, 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 is greatly changed and expanded in the .NET version. Help includes all of 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 and related information.

### Installing and Running MSDN

You can run MSDN from a hard drive, from a network drive, from a CD, or from the Web. If you run from a CD, you must keep the CD in the drive while you develop programs, and, of course, if you plan to access MSDN from the Web, you must have a live Internet connection as you work.

When you install Visual Studio, by default MSDN is installed on the hard drive. If you don't want to install it there, you must specifically choose this option. You can access MSDN on the web at <http://msdn.microsoft.com>.

Or, if you want to go directly to VB documentation, add this link to your favorites:

<http://msdn.microsoft.com/library/en-us/vbl7/html/vboriVBLangRefTopNode.asp>

The expanded 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.

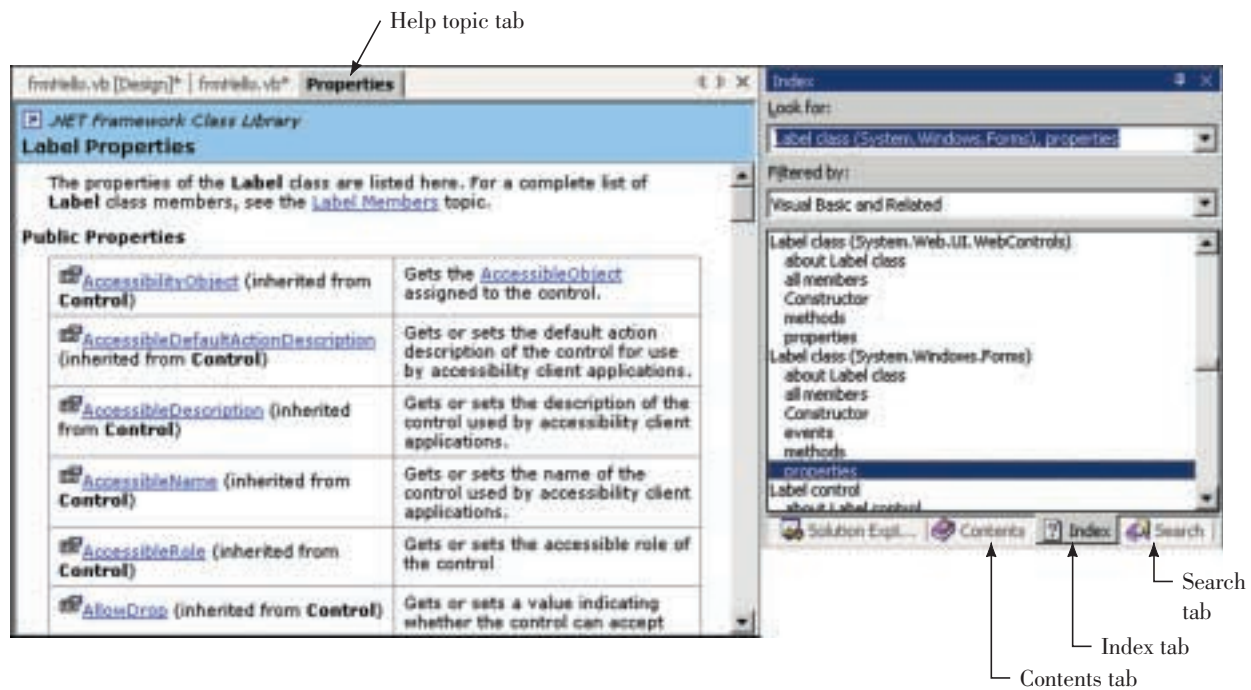
- Help is totally different. The Contents/Index/Search pops up in a new tabbed window in the Solution Explorer's space. A selected topic appears as a tabbed window in the Document window. And if several topics match the selected item in Contents/Index/Search, a new tabbed window appears at the bottom of the screen, docked with the Task List and Output window.

## Viewing Help Topics

You view the Help topics in various windows in the VS IDE. When you choose *Contents*, *Index*, or *Search* from the *Help* menu, a new tabbed window opens in the same location as the Solution Explorer (Figure 1.52). Select a topic and the correct page appears in the Document window. Notice the new tab at the top of the Document window in Figure 1.52.

**Figure 1.52**

The *Help Index*, *Contents*, and *Search* windows appear as tabs in the Solution Explorer window. Note that this window was widened to show the text in the tabs.



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. Drop down the *Filtered By* list and choose *Visual Basic and Related* (Figure 1.53).

**Figure 1.53**

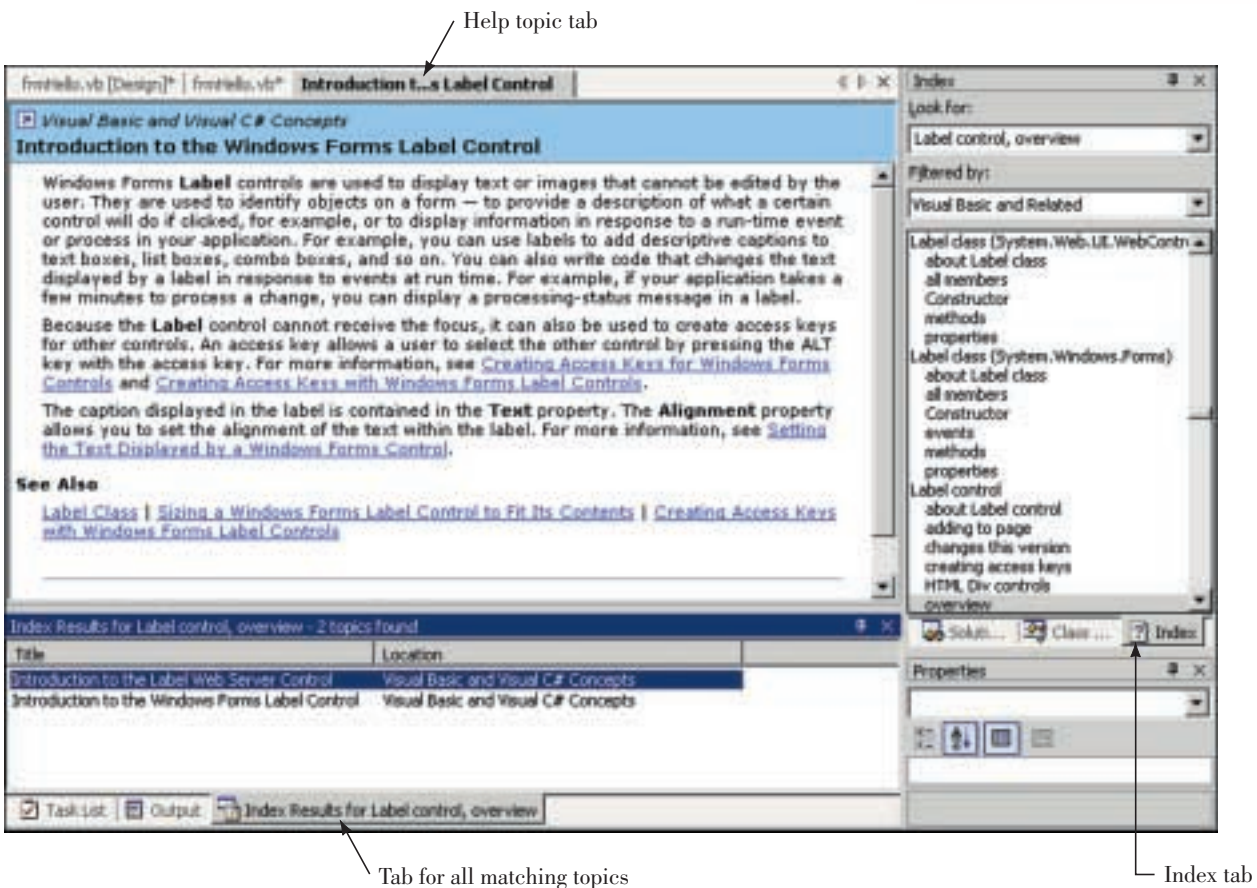
*Filter the Help topics so that only Visual Basic topics appear.*



Sometimes you may select a topic that has several pages from which to choose. When that happens, a new tabbed window opens in the Task List window (Figure 1.54). Double-click the topic you wish to view and the selected page appears in the Document window.

**Figure 1.54**

*Multiple matching topics appear in a tabbed window at the bottom of the screen. Select a topic from the list and the corresponding page appears in the Document window.*



Many Help topics have entries for both Windows Forms and Web 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. Select *Help/Contents* and choose *Visual Studio .NET/Getting Assistance/Using Help in Visual Studio .NET*.

## 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 corresponding Help topic will appear in the Document window, 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 hide a window that is a part of a tabbed window, right-click the tab and select *Hide* from the context menu.
- 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. Use the *Help* menu's *Index*, filter by *Visual Basic and Related*, and type “button control”. In the Index list, notice that one heading covers Web Forms and another covers Windows Forms. Under *Button control (Windows Forms)* select *overview*. At the bottom of the screen, you should see a new tabbed window showing the two topics that Help found. Double-click on the “Button Control (Windows Forms)” entry; the corresponding page should appear 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. Display the *Help* menu and view all of the options. Try the *Contents*, *Index*, *Search*, *Index Results*, and *Search Results* options. Notice the *Next Topic* and *Previous Topic* items and the *Show Start Page* item; you can use this command to show the Start Page if its tab does not appear in the Document window.



## Summary

1. Visual Basic is an object-oriented language 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 (VB) is called .NET, which corresponds to Version 7. Visual Basic .NET is part of Visual Studio .NET. VS .NET has an Academic Edition, a Professional Edition, an Enterprise Developer Edition, and an Enterprise Architect 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, 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 break time.
10. You can customize the Visual Studio IDE and reset all customizations 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.
19. Three types of errors can occur in a Visual Basic project: syntax errors (violating the syntax rules of Basic statements), run-time errors (containing a statement that cannot execute properly), and logic errors (producing erroneous results).

20. Finding and fixing programming errors is called *debugging*.
21. You must have a clean compile each time you modify a program before you can 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 Contents, Index, Search, or context-sensitive Help.

## Key Terms

Academic Edition	5	logic error	50
assignment statement	33	method	4
break time	14	namespace	26
Button	21	object	4
class	4	object-oriented programming (OOP)	3
clean compile	51	procedure	32
code	6	Professional Edition	5
context menu	25	project file	7
context-sensitive Help	56	Properties window	12
control	4	property	4
debugging	51	pseudocode	6
design time	14	remark statement	33
Declarations section	45	run time	14
Document window	12	run-time error	50
Enterprise Architect Edition	5	solution	7
Enterprise Developer Edition	5	Solution Explorer window	12
event	4	solution file	7
event procedure	33	sub procedure	32
form	4	syntax error	47
Form Designer	12	Text property	28
graphical user interface (GUI)	3	toolbar	10
handle	22	toolbox	12
Help	13	user interface	6
integrated development environment (IDE)	8	Visual Studio environment	8
Label	21		

## 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? break 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 `btnPush_Click` mean? To what does `btnPush` 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 a syntax error, when does it occur, and what might cause it?
13. What is a run-time error, when does it occur, and what might cause it?
14. What is a logic error, when does it occur, and what might cause it?
15. Tell the class of control and the likely purpose of each of these object names:
  - `lblAddress`
  - `btnExit`
  - `txtName`
16. 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. (Begin the name with lowercase “btn”.) Include remarks at the top of every procedure and in the Declarations section of the code.

“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 two buttons to display the greeting, 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 in the Declarations section of the code.

Select a different font name and font size for the greeting label. If you wish, you can also 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 code statement. However, when the saying displays on your form, long lines will wrap within the label if the label is large enough.)

Make a button for each saying with a descriptive Text property for each, as well as 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 label is large enough to display your longest saying and that 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 in the Declarations section of the code.

- 1.4 Write a project to display company contact information. Include buttons and labels for 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 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 in the Declarations section of the code.

- 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,” and “Daily Fish.”

Also include 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 Exit.

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

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 button for Exit.

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 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 in the Declarations section of the code.

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 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.

Follow good programming conventions for object names; include remarks at the top of every procedure and in the Declarations section of the code.

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