

```
try  
{
```

```
// Convert input value to numeric and assign
```

1

Introduction to Programming and Visual C# 2008

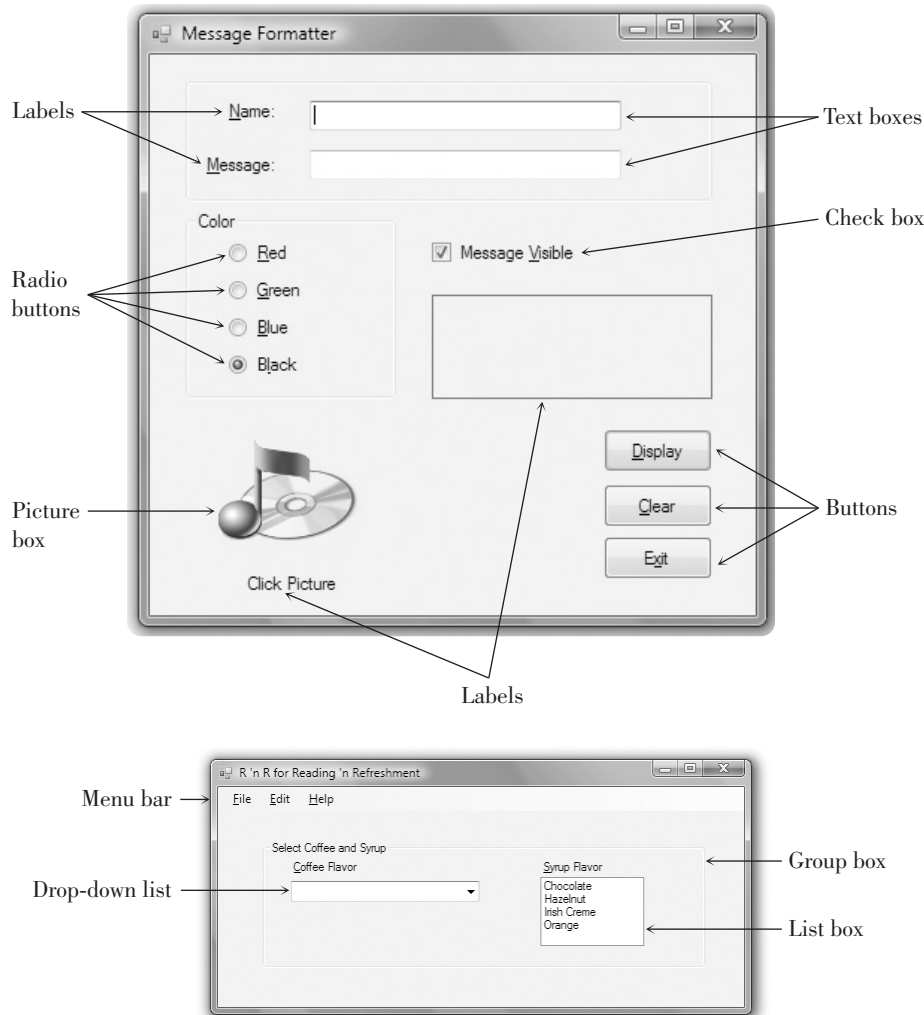
at the completion of this chapter, you will be able to . . .

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

Writing Windows Applications with Visual C#

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 C# (C sharp) 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 C# and Windows Forms.



Beginning in Chapter 9 you will create programs using Web Forms and Visual Web Developer. You can run Web applications in a browser such as Internet Explorer or Mozilla FireFox, on the Internet, or on a company intranet. Figure 1.2 shows a Web Forms application.

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

Figure 1.2

A Web Forms application running in a browser.



The Windows Graphical User Interface

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

Programming Languages—Procedural, Event Driven, and Object Oriented

There are literally hundreds of programming languages. Each was developed to solve a particular type of problem. Most traditional languages, such as BASIC, C, COBOL, FORTRAN, PL/1, 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#, Java, and Visual Basic (VB) use a different approach: *object-oriented programming (OOP)*.

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 method (a set of programming statements) 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 method you have written to do the calculation.

The Object Model

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

Objects

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

Properties

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

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

Methods

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

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

Events

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

Classes

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

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



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

we say *automobile*, we are not referring to a particular auto, but we know that an automobile has a make and model, a color, an engine, and a number of doors. These elements are the *properties* of the Automobile class.

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

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

The events of an Automobile class could be `Arrive` or `Crash`. In a C# program, you write event-handling methods that specify the actions you want to take when a particular event occurs for an object. For example, you might write a method to handle the `yourAuto.Crash` event.

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

Microsoft's Visual Studio

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

The .NET Framework

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

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

C#

Microsoft C# is a part of Visual Studio. You also can purchase C# by itself (without the other languages but *with* the .NET Framework). C# is available in an **Express Edition**, a **Standard Edition**, a **Professional Edition**, and four specialized versions of **Team System Editions** for large enterprise application development. You can find a matrix showing the features of each edition in Help. Anyone planning to do professional application development that includes the advanced features of database management should use the Professional Edition or the Team System Database version. The full Professional Edition is available to educational institutions through the Microsoft Academic Alliance program and is the best possible deal. When a campus department purchases the Academic Alliance, the school can install Visual Studio on all classroom and lab computers and provide the software to all students and faculty at no additional charge. For more information, have your instructor visit: <http://msdn.microsoft.com/en-us/academic/default.aspx>

Microsoft provides an Express Edition of each of the programming languages, which you can download for free (www.microsoft.com/express/download/). You can use Visual C# Express for Windows development and Visual Web Developer Express for the Web applications in Chapters 9 and 10.

This text is based on the Professional Edition of Visual Studio 2008, the current version. You cannot run the projects in this text in any earlier version of C#.

Writing C# Programs

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

The Three-Step Process

Planning

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

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

2. *Plan the properties.* For each object, write down the properties that you plan to set or change during the design of the form.
3. *Plan the C# code.* In this step you plan the classes and methods 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 C# **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 *End the project* or *Quit*.

Programming

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

1. *Define the user interface.* When you define the user interface, you create the forms and controls that you designed in the planning stage.

Think of this step as defining the objects you will use in your application.

2. *Set the properties.* When you set the properties of the objects, you give each object a name and define such attributes as the contents of a label, the size of the text, and the words that appear on top of a button and in the form's title bar.






You might think of this step as describing each object.




3. *Write the code.* You will use C# programming statements (called *C# 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.

C# Application Files

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

File Name	File Icon	Description
HelloWorld.sln		The solution file . A text file that holds information about the solution and the projects it contains. This is the primary file for the solution—the one that you open to work on or run your project. Note the “9” on the icon, which refers to Visual Studio version 9.
HelloWorld.suo		Solution user options file. Stores information about the state of the integrated development environment (IDE) so that all customizations can be restored each time you open the solution.
HelloForm.cs		A .cs (C#) file that holds the code methods that you write. 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.
HelloForm.Designer.cs		A .cs (C#) file created by the Form Designer that holds the definition of the form and its controls. You should not modify this file directly, but instead make changes in the Designer and allow it to update the file.
HelloForm.resx		A resource file for the form. This text file defines all resources used by the form, including strings of text, numbers, and any graphics.

File Name	File Icon	Description
HelloWorld.csproj		The project file that describes the project and lists the files that are included in the project.
HelloWorld.csproj.user		The project user options file. This text file holds IDE option settings so that the next time you open the project, all customizations will be restored.
Program.cs		A .cs (C#) file that contains automatically generated code that runs first when you execute your application.

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

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

The Visual Studio Environment

The **Visual Studio environment** is where you create and test your projects. A development environment such as Visual Studio is called an **integrated development environment (IDE)**. The IDE consists of various tools, including a form designer, which allows you to visually create a form; an editor, for entering and modifying program code; a compiler, for translating the C# 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.

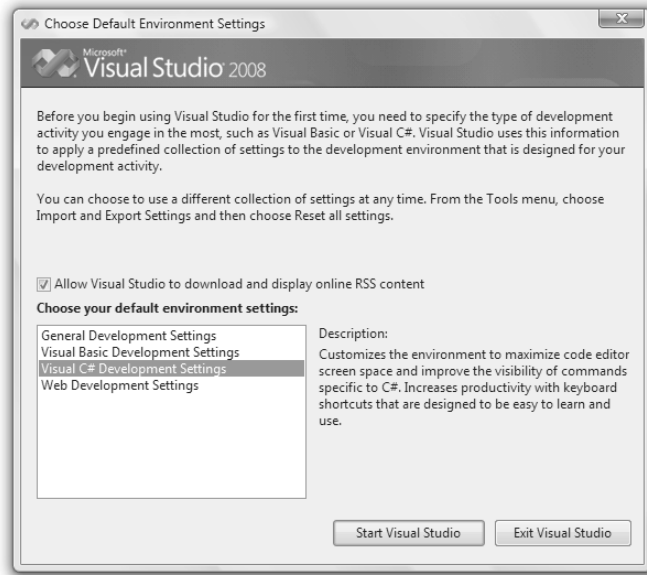
In versions of Visual Studio prior to .NET, each language had its own IDE. For example, to create a Visual Basic project you would use the Visual Basic IDE, and to create a C++ project you would use the C++ IDE. But in Visual Studio, you use the one IDE to create projects in any of the supported languages.

Default Environment Settings

The full version of Visual Studio 2008 provides an option that allows the programmer to select the default profile for the IDE. The first time you open Visual Studio, you are presented with the *Choose Default Environment Settings* dialog box (Figure 1.3), where you can choose *Visual C# Development Settings*. This text uses the Visual C# settings.

Figure 1.3

The first time you open the Visual Studio IDE, you must select the default environment settings for Visual C# development.



Note: If you plan to develop in more than one language, such as VB and C#, you can save each group of settings and switch back and forth between the two. Select *Tools / Import and Export Settings* and choose to *Reset all settings*.

The IDE Initial Screen

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

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

You can open an existing project or begin a new project using the Start Page or the *File* menu. The examples in this text use the menus.

The New Project Dialog

You will create your first C# projects by selecting *File / New Project*, which opens the *New Project* dialog (Figure 1.5). In the *New Project* dialog, you may need to expand the node for *Other Languages*, depending on your installation. Under *Visual C#*, select *Windows*, and in the *Templates* pane, select *Windows Forms Application*. You also give the project a name in this dialog. Deselect the check box for *Create directory for solution*, which creates an extra level of folders for our single-project solutions.

Figure 1.4

The Visual Studio IDE with the Start Page open, as it first appears in Windows Vista, without an open project. You can close the Start Page by clicking on its Close button.

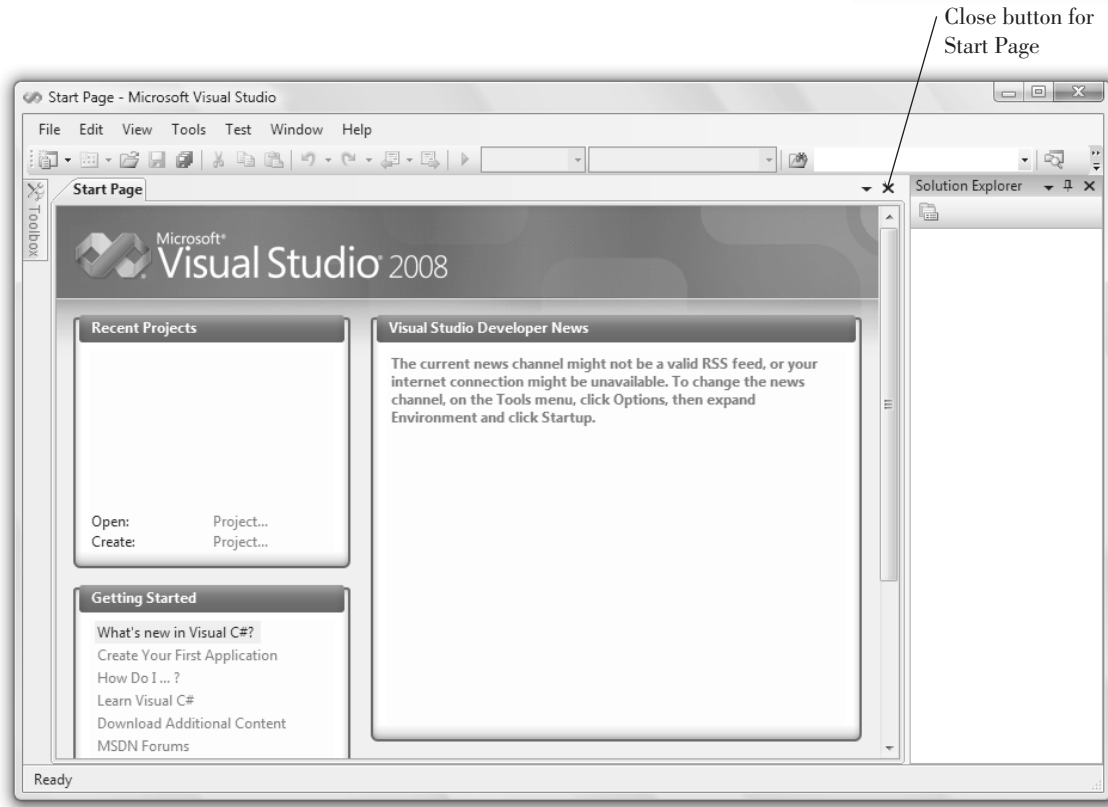
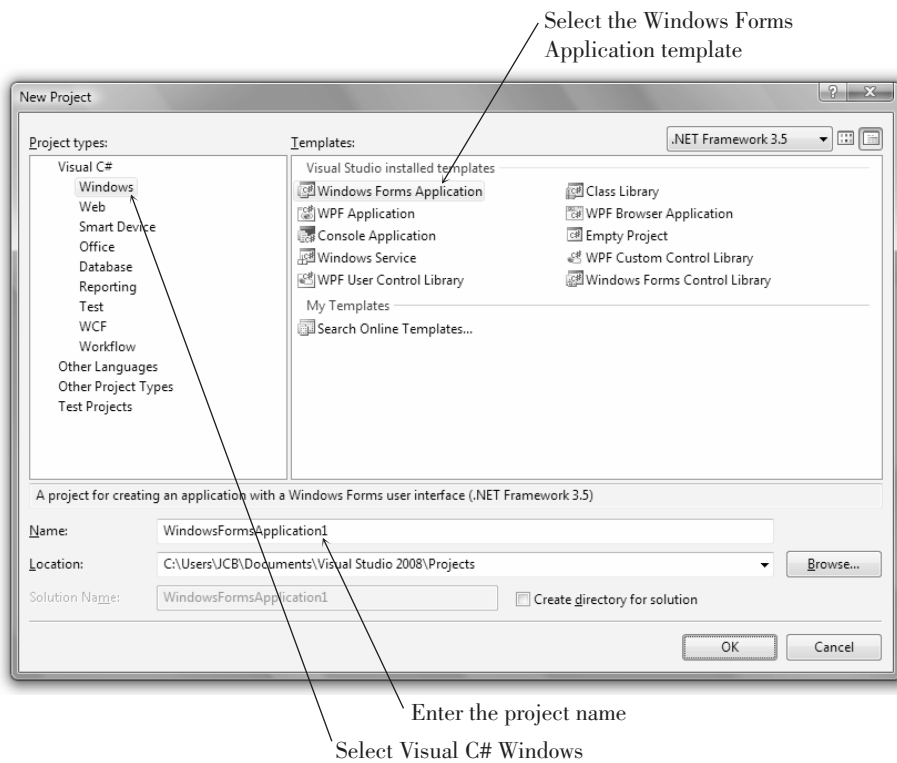


Figure 1.5

Begin a new C# Windows project using the Windows Forms Application template.



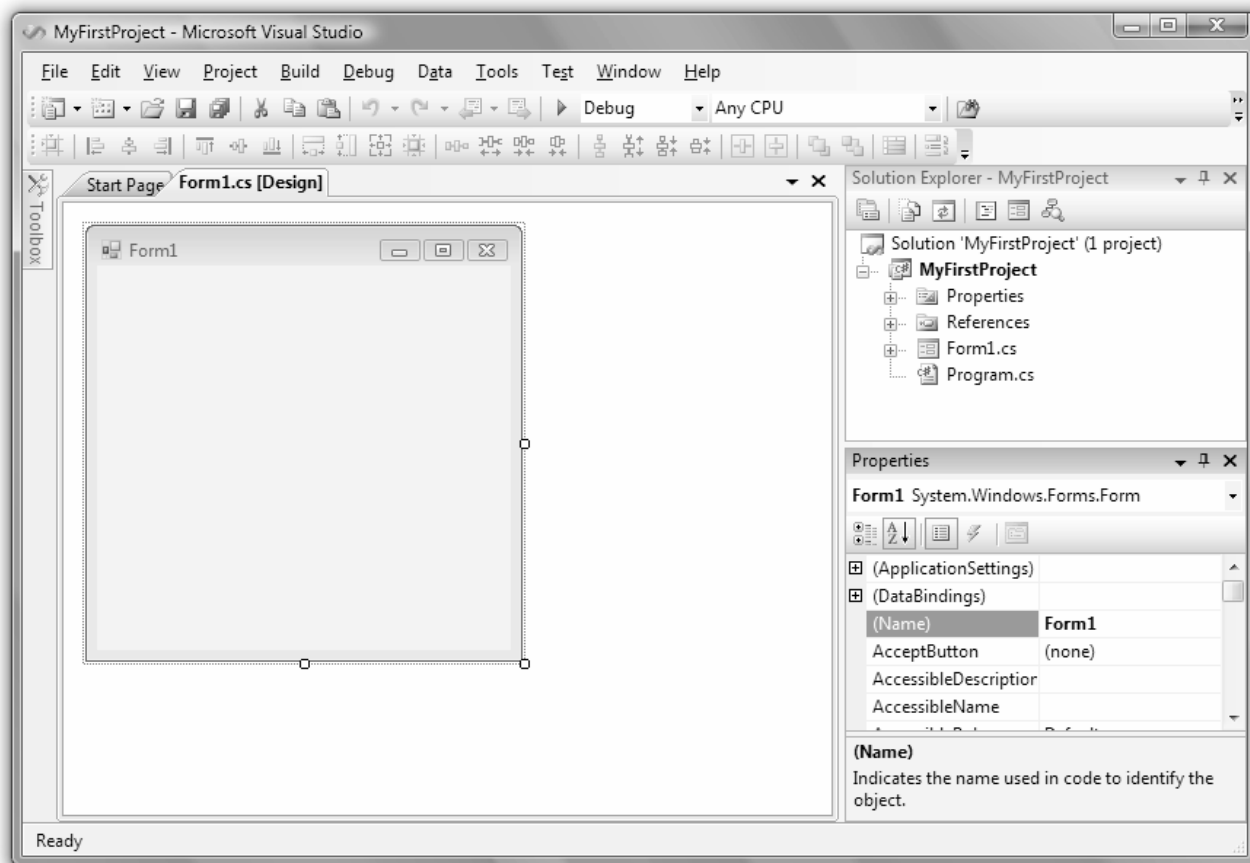
The IDE Main Window

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

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

Figure 1.6

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

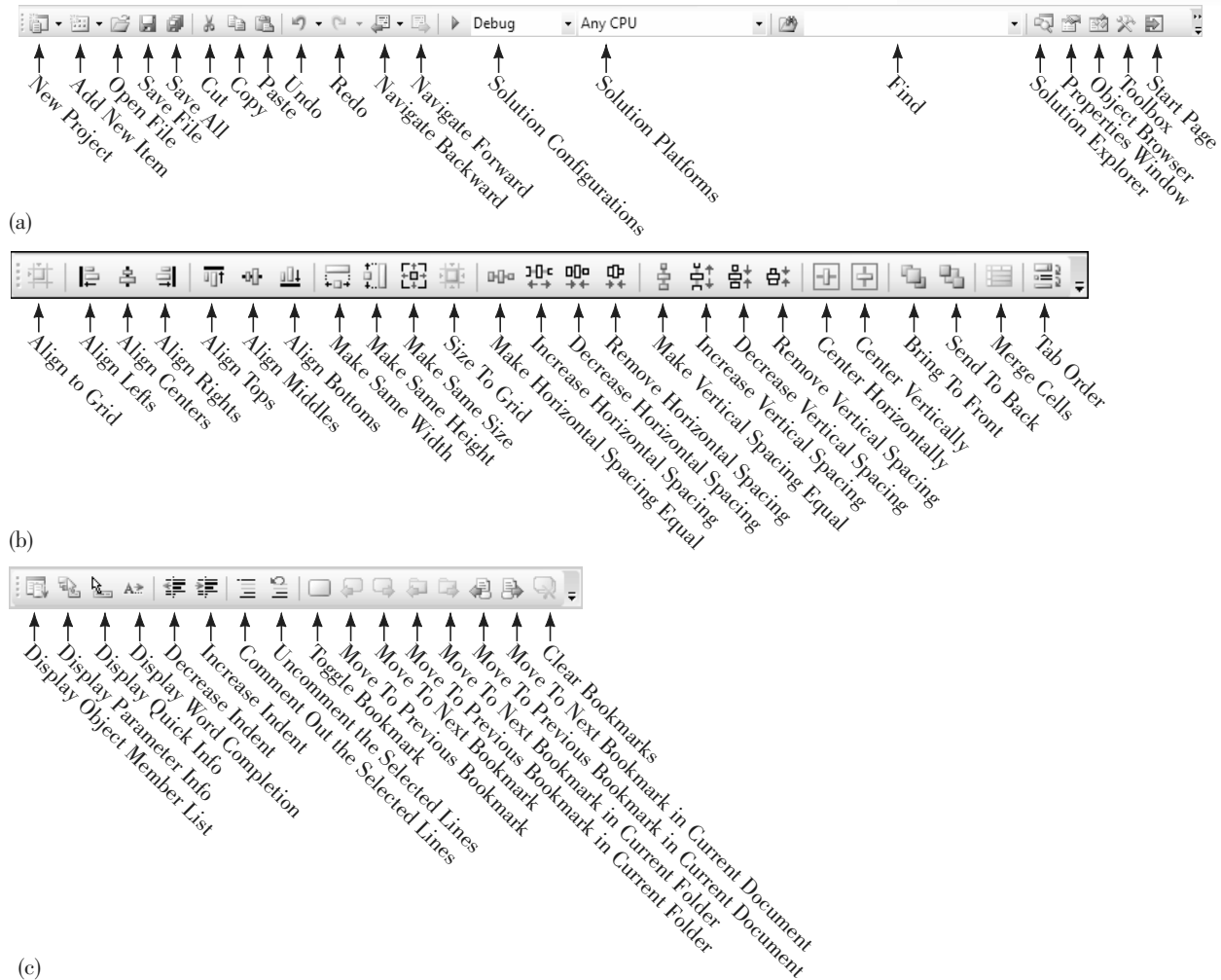


The Toolbars

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

Figure 1.7

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



The Document Window

The largest window in the center of the screen is the **Document window**. Notice the tabs across the top of the window, which allow you to switch between open documents. The items that display in the Document window include the Form Designer, the Code Editor, the Project Designer, the Database Designer, and the Object Browser.

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

The Form Designer

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



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

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

The Solution Explorer Window

The **Solution Explorer window** holds the filenames for the files included in your project and a list of the classes it references. The Solution Explorer window and the environment's title bar hold the name of your solution (.sln) file, which is WindowsFormsApplication1 by default unless you give it a new value in the *New Project* dialog box. In Figure 1.6, the name of the solution is MyFirstProject.

The Properties Window

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

TIP

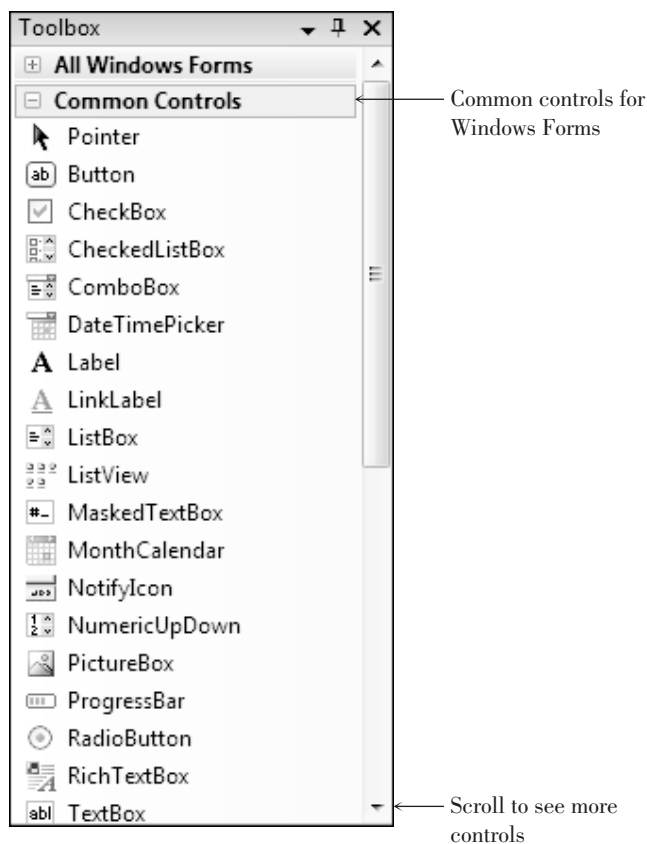
You can sort the properties in the window either alphabetically or by categories. Use the buttons on the Properties window. ■

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 C# you are using (Express, Standard, Professional, or Team System). Figure 1.8 shows the toolbox.

Figure 1.8

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



TIP

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

Help

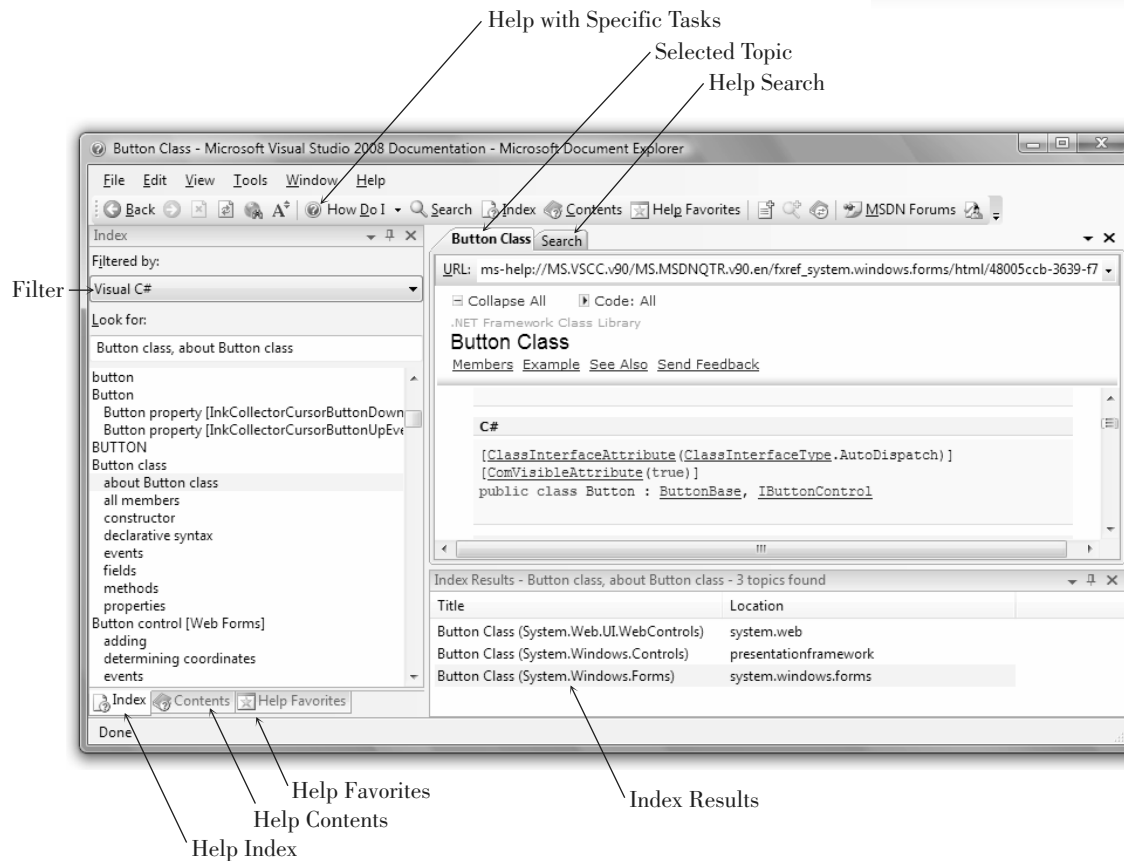
Visual Studio has an extensive **Help** feature, which includes the Microsoft Developer Network library (MSDN). You can find reference materials for C#, C++, VB, and Visual Studio; several books; technical articles; and the Microsoft Knowledge Base, a database of frequently asked questions and their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

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

When you make a selection from the *Help* menu, the requested item appears in a new window that floats on top of the IDE window (Figure 1.9), so you can keep both open at the same time. It’s a good idea to set the *Filtered By* entry to *Visual C#*.

Figure 1.9

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



Design Time, Run Time, and Debug Time

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

Writing Your First C# Project

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

Figure 1.10

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



Set Up Your Workspace

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

Run Visual Studio

These instructions assume that Visual Studio 2008 is installed in the default location. If you are running in a classroom or lab, the program may be installed in an alternate location, such as directly on the desktop.

- STEP 1:** Click the Windows *Start* button and move the mouse pointer to *All Programs*.
- STEP 2:** Locate *Microsoft Visual Studio 2008*.
- STEP 3:** If a submenu appears, select *Microsoft Visual Studio 2008* or *Microsoft Visual C# 2008 Express*.

Visual Studio will start and display the Start Page (refer to Figure 1.4). If you are using Visual Studio Professional and this is the first time that VS has been opened for this user, you will need to select *Visual C# Development Settings* from the *Choose Default Environment Settings* dialog box (refer to Figure 1.3).

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

Start a New Project

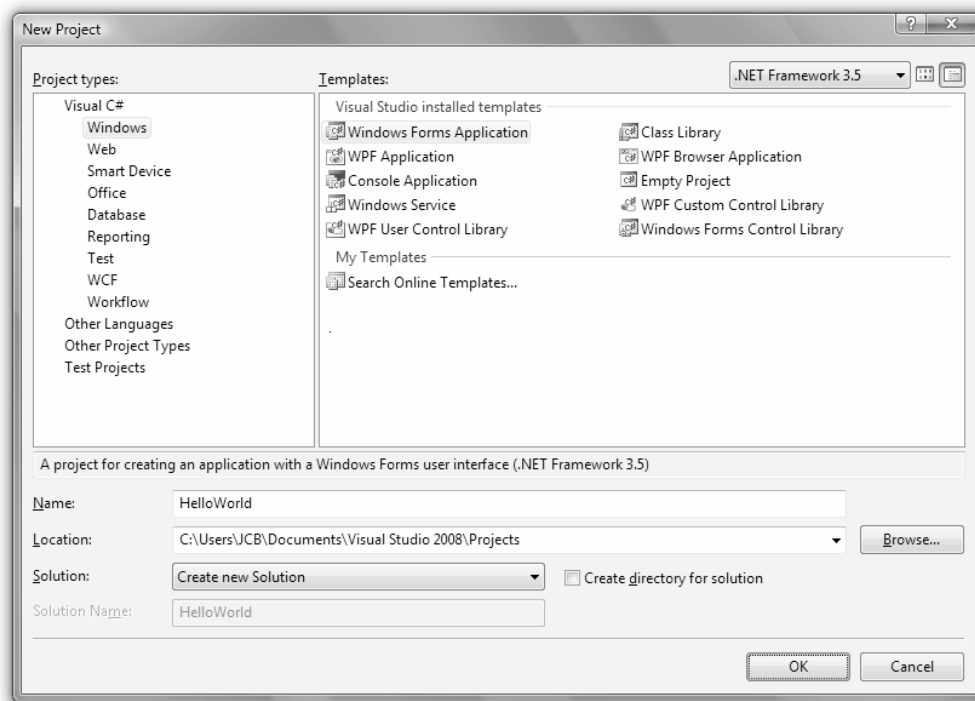
- STEP 1:** Select *File / New / Project*; the *New Project* dialog box opens (refer to Figure 1.5). Make sure that *Visual C#* and *Windows* are selected for

Project types and *Windows Forms Application* is selected for the template. If you are using Visual C# Express, the dialog box differs slightly and you don't have to choose the language, but you can still choose a Windows Forms Application.

STEP 2: Enter "HelloWorld" (without the quotes) for the name of the new project (Figure 1.11) and click the *OK* button. The new project opens (Figure 1.12). At this point, your project is stored in a temporary directory. You can specify a new location for the project later when you save it.

Figure 1.11

Enter the name for the 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 *Window / Reset Window Layout* and responding *Yes*. The IDE should now match Figure 1.12.
- STEP 2:** Point to the icon for the toolbox at the left of the IDE window. The Toolbox window pops open. Notice the pushpin icon at the top of the window (Figure 1.13); clicking this icon pins the window open rather than allowing it to Auto Hide.
- STEP 3:** Click the Auto Hide pushpin icon for the Toolbox window; the toolbox will remain open.

Figure 1.12

The Visual Studio IDE with the new HelloWorld C# project. Your screen may look significantly different from the figure since the environment can be customized.

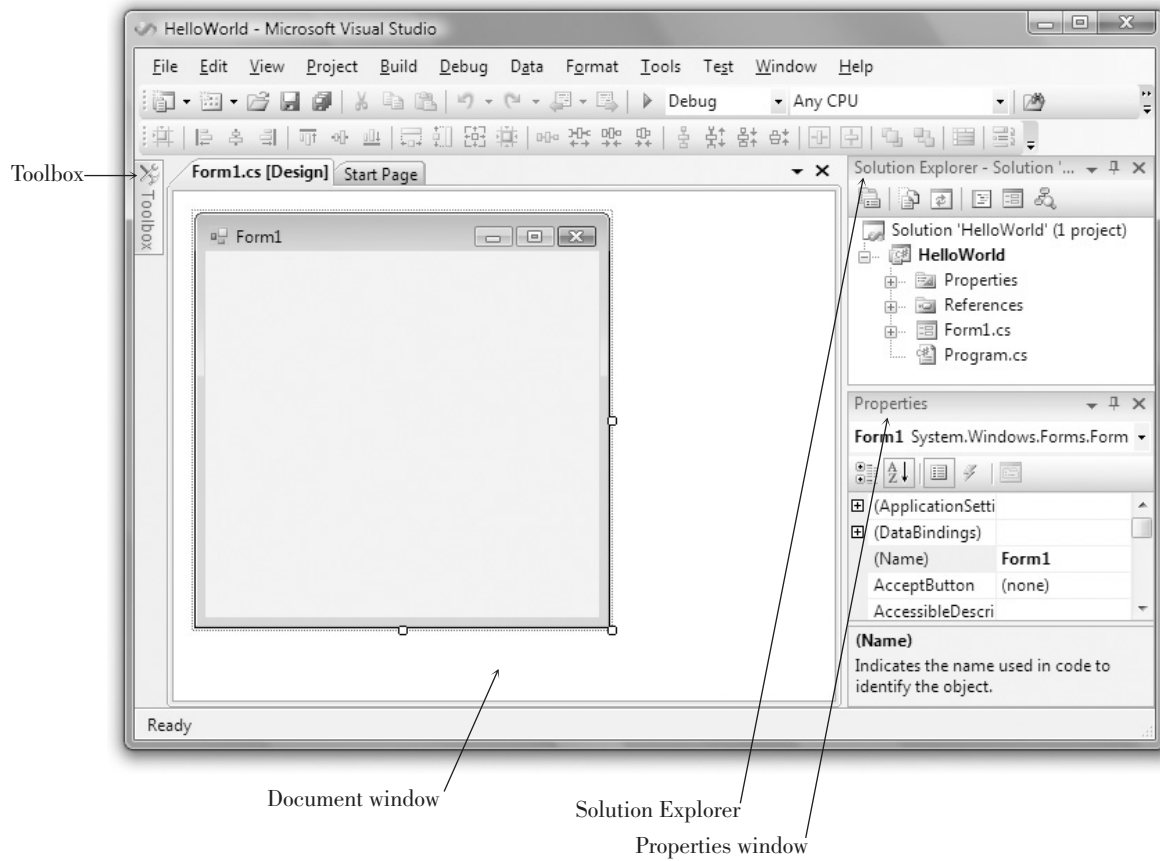
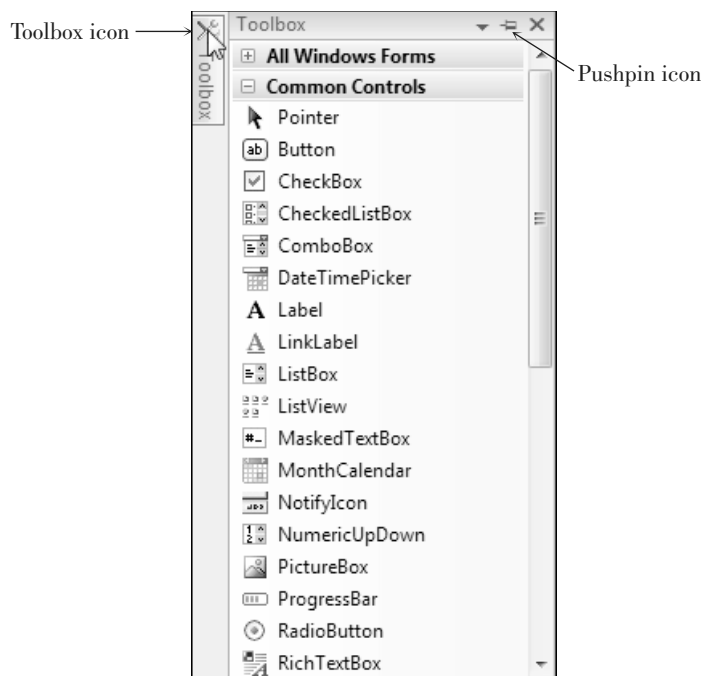


Figure 1.13

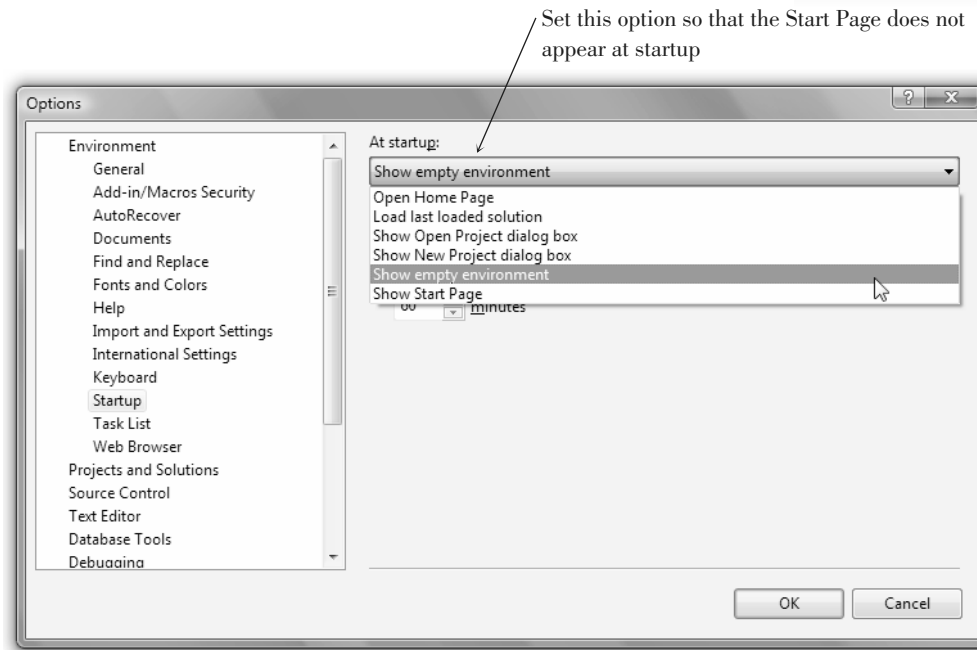
The Toolbox window.



STEP 4: Optional: Select *Tools / Options*. In the *Options* dialog box, select *Startup* under *Environment*, drop down the *At startup* list and select *Show empty environment* (Figure 1.14), and click *OK*. This selection causes the Start Page to not appear and will make your environment match the illustrations in this text. Note that you can show the Start Page at any time by selecting *View / Other Windows / Start Page*.

Figure 1.14

Select *Show empty environment* for the environment's *Startup* option in the *Options* dialog box.

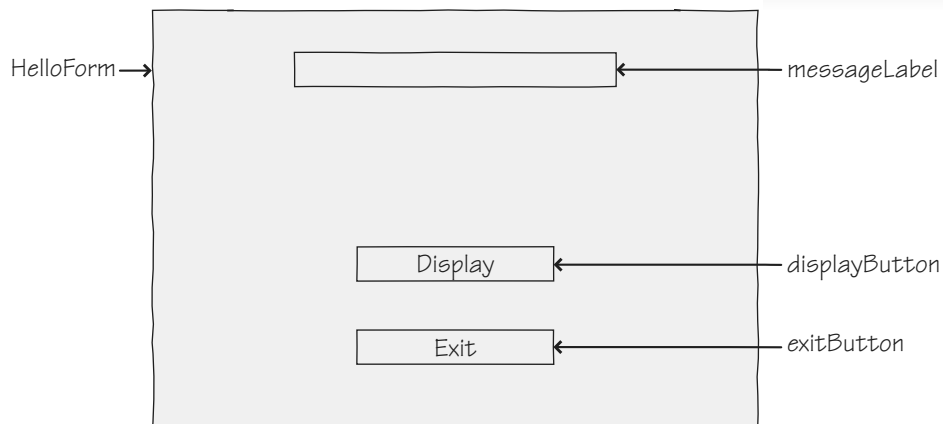


Plan the Project

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.

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.

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

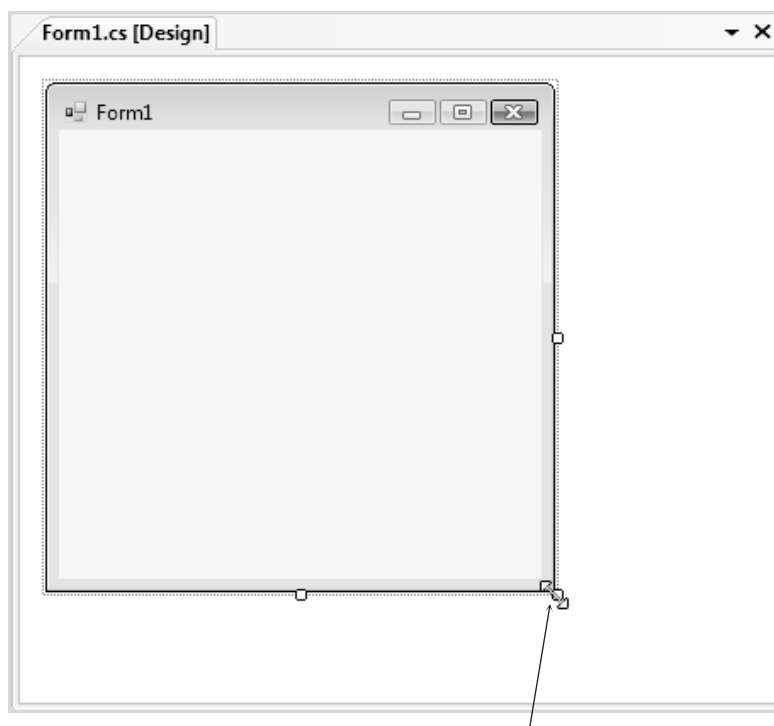


Figure 1.16

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

Place Controls on the Form

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

STEP 1: Point to the Label tool in the toolbox and double-click; a Label control appears on the form. Drag the label to the desired location (Figure 1.17). Later you will adjust the label's size.

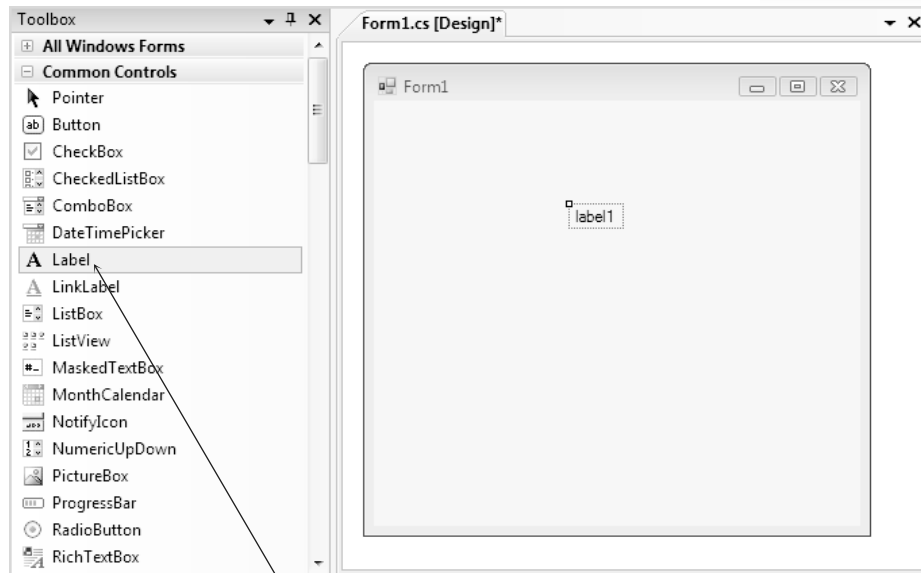
As long as the label is selected, you can press the Delete key to delete it, or drag it to a new location.

You can tell that a label is selected; it has a dotted border, as shown in Figure 1.17, when the `AutoSize` property is `true` (the default) or sizing handles if you set the `AutoSize` property to `false`.

STEP 2: 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 1.18). When you release the mouse button, the new button should appear selected and have

Figure 1.17

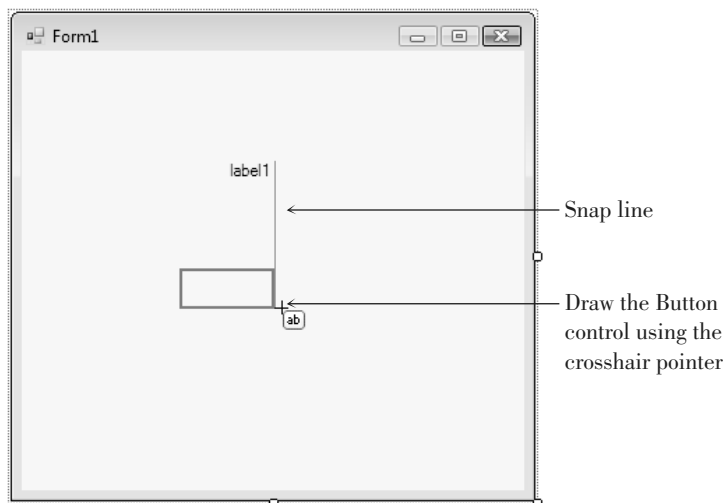
The newly created label appears outlined, indicating that it is selected. Notice that the contents of the label are set to the control's name (*label1*) by default.



Double-click the Label tool

Figure 1.18

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



resizing handles. The blue lines that appear are called **snap lines**, which can help you align your controls.

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

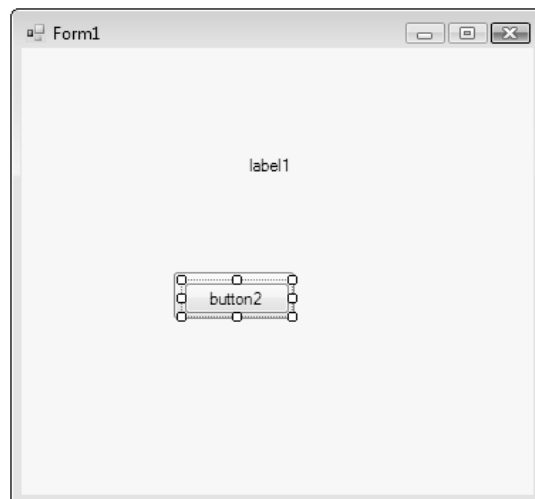
Selecting, Deleting, Moving, and Resizing Controls on a Form.**Table 1.1**

Select a control	Click on the control.
Delete a control	Select the control and then press the 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 and has resizing handles; then either point to one of the handles, press the mouse button, and drag the handle; or drag the form's bottom border to change the height or the side border to change the width. Note that the default format for labels does not allow resizing.

STEP 3: While the first button is still selected, point to the Button tool in the toolbox and double-click. A new button of the default size will appear on top of the last-drawn control (Figure 1.19).

Figure 1.19

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



STEP 4: Keep the new button selected, point anywhere inside the button (not on a handle), and drag the button below your first button (Figure 1.20).

STEP 5: Select each control and move and resize the controls as necessary. Make the two buttons the same size and line them up. Use the snap lines to help with the size and alignment. Note that you can move but not resize the label.

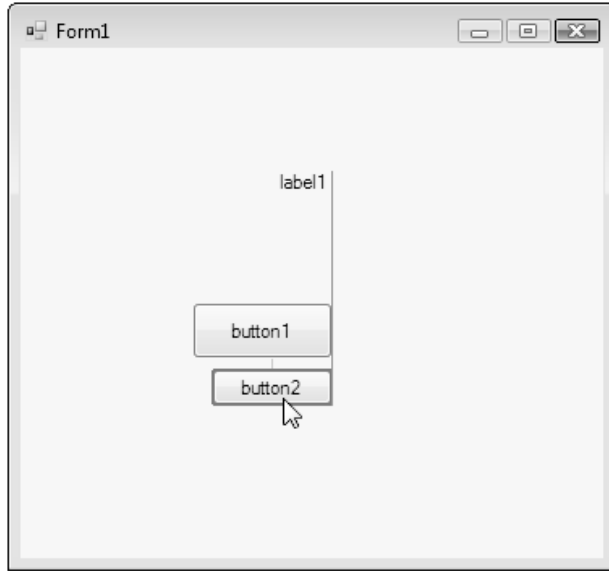
At this point you have designed the user interface and are ready to set the properties.

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

STEP 1: Click on the label you placed on the form; a dotted outline appears around the control. If the Properties window is not displaying, select *View / Properties Window* or press the F4 key. Click on the title bar of the Properties window to make it the active window (Figure 1.21).

Figure 1.20

Drag the new button (button2) below button1.

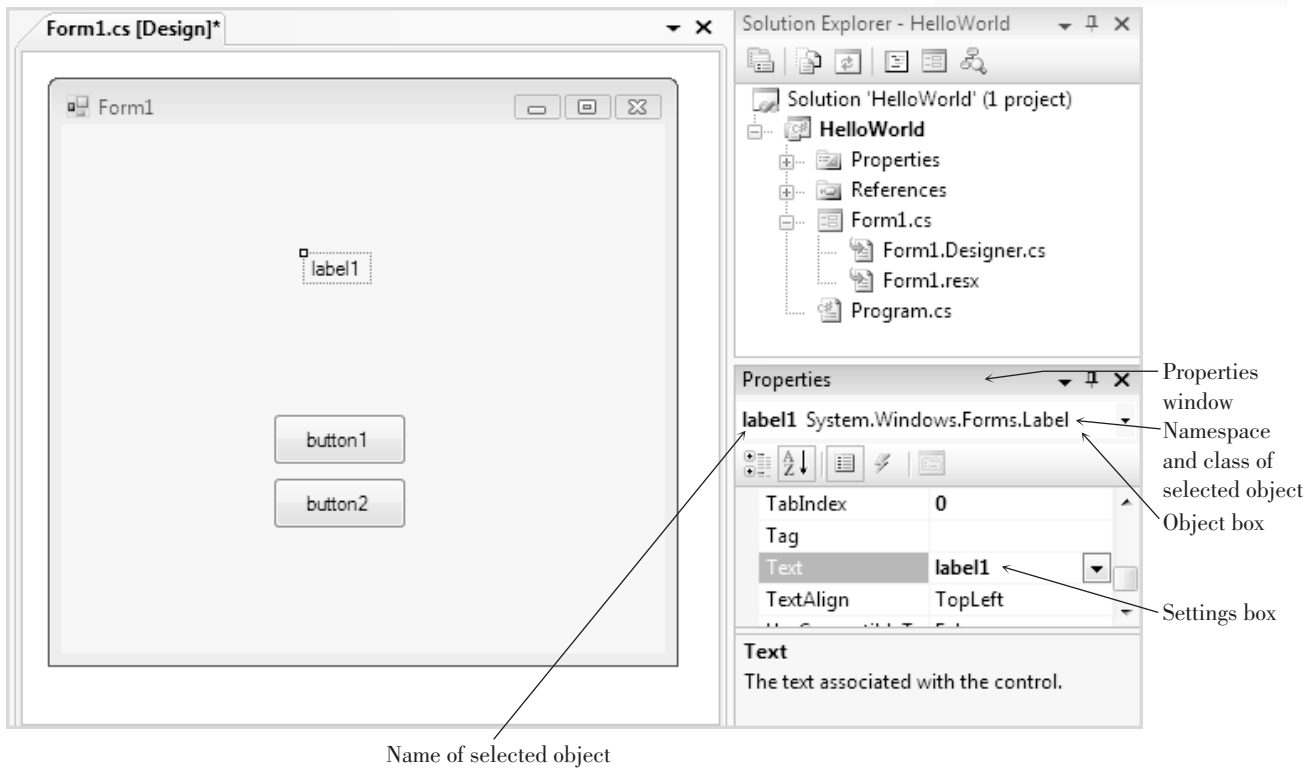


TIP

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

Figure 1.21

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.

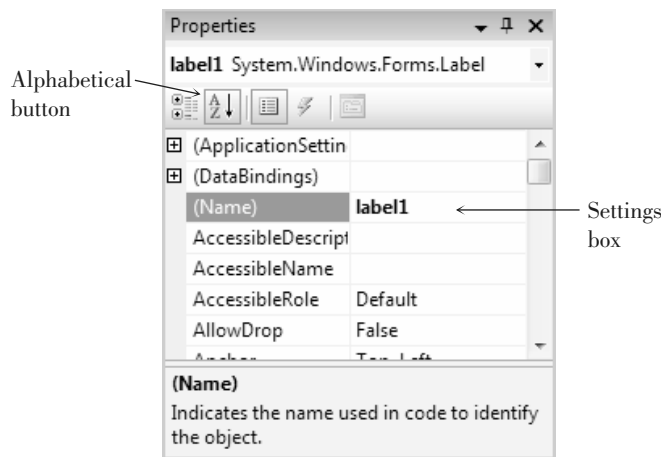
STEP 2: In the Properties window, click on the Alphabetical button to make sure the properties are sorted in alphabetic order. Then select the Name property, which appears near the top of the list. Click on *(Name)* and notice that the Settings box shows *label1*, the default name of the label (Figure 1.22).

TIP

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

Figure 1.22

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

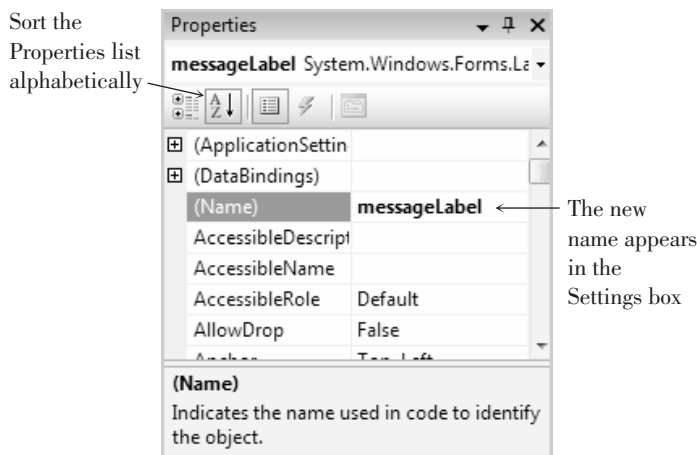


STEP 3: Type “messageLabel” (without the quotation marks). See Figure 1.23. As a shortcut, you may wish to delete the “1” from the end of “label1”, press the Home key to get to the beginning of the word, and then type “message”. Change the “l” for label to uppercase.

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

Figure 1.23

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



- STEP 4:** Select the `AutoSize` property and change the value to `False`. You can easily change a property from `True` to `False` in several ways: (1) Click in the word “`True`” and type only the letter “`f`”, and the value changes automatically; (2) Double-click on either the property name (`AutoSize`) or the property value (`True`), and the value toggles each time you double-click; or (3) Click on either the property name or the property value, and a drop-down arrow appears at the right end of the Settings box. Drop down the list and make your selection from the possible values (`True` or `False`, in this case).
- STEP 5:** Click on the `Text` property to select it. (Scroll the Properties list if necessary.)

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

- STEP 6:** Double-click on `label1` in the Settings box; the entry should appear selected (highlighted). See Figure 1.24.

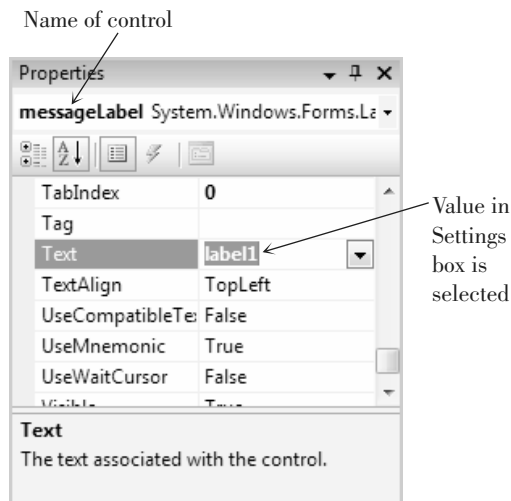


Figure 1.24

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

- STEP 7:** Press the Delete key to delete the value of the `Text` property. Then press Enter and notice that the label on the form appears empty. 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.

All you see is a very small selection border (Figure 1.25), and if you click anywhere else on the form, which deselects the label, you cannot see it at all.

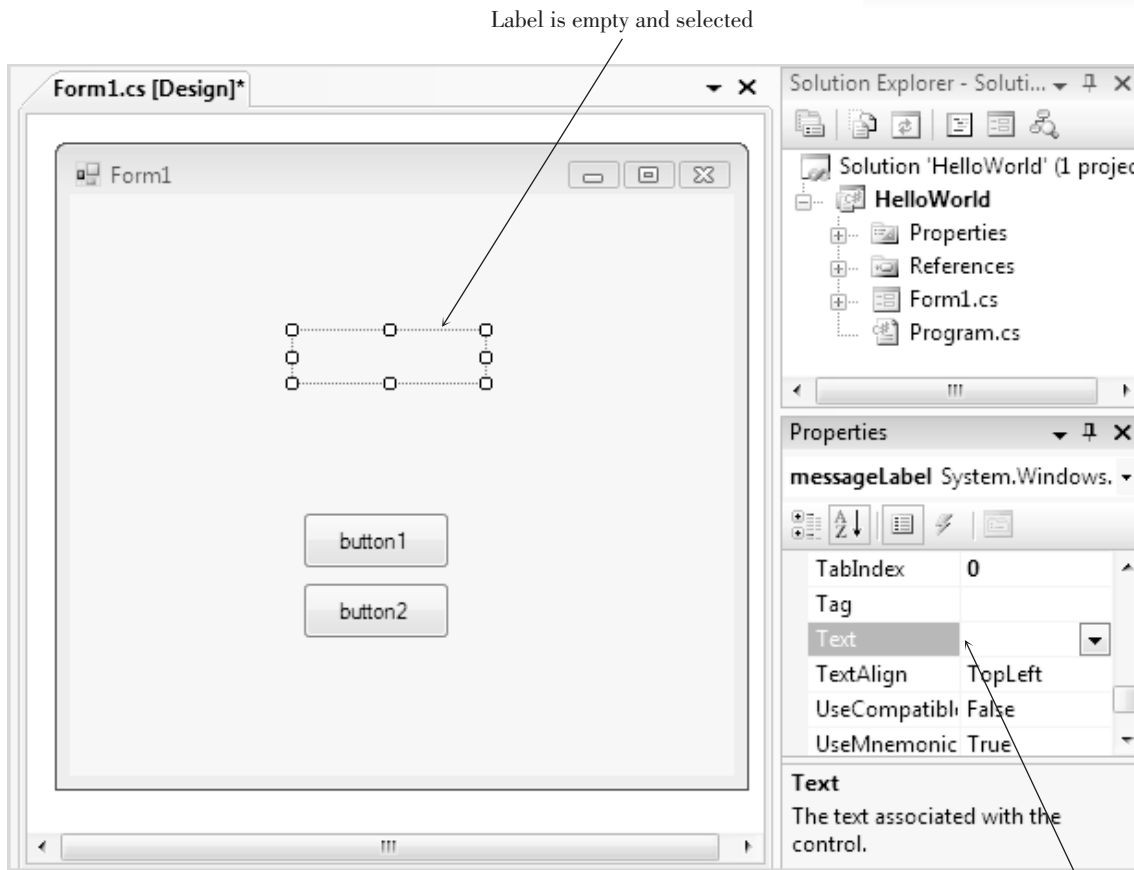
If you need to select the label after deselecting it, you can click in the approximate spot on the form or use the Properties window: Drop down the Object list at the top of the window; you can see a list of all controls on the form and can make a selection (Figure 1.26).

✓ TIP

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

Figure 1.25

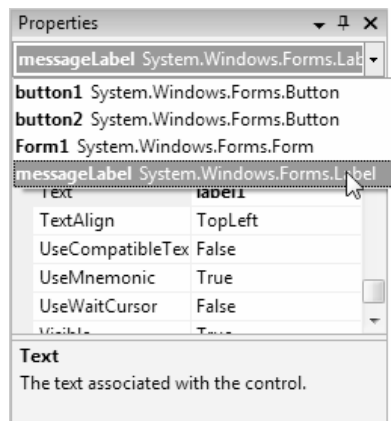
Delete the value for the Text property from the Settings box; the label on the form also appears empty.



Text deleted from the Settings box

Figure 1.26

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



Lock the Controls

STEP 1: Point anywhere on the form and click the right mouse button to display a **context menu**. On the context menu, select *Lock Controls* (Figure 1.27). Locking prevents you from accidentally moving the

controls. When your controls are locked, a selected control has a small lock icon in the upper-left corner instead of resizing handles (Figure 1.28).

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.

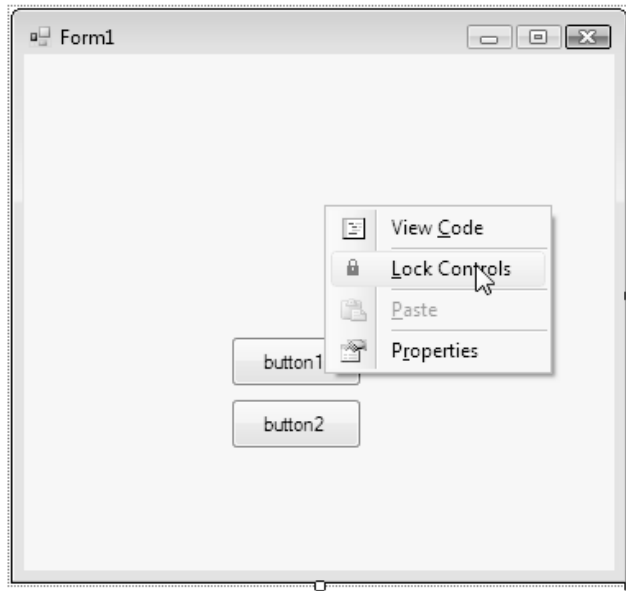


Figure 1.27

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

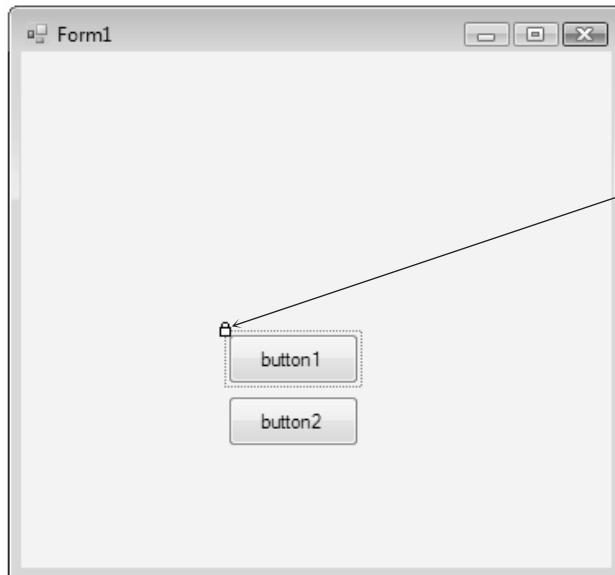


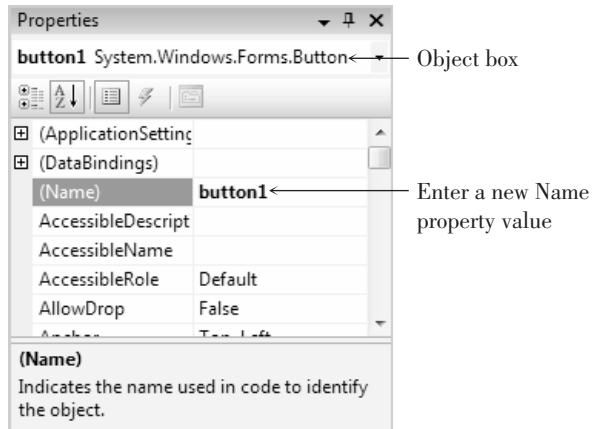
Figure 1.28

After you lock the controls on a form, a selected control has a lock icon instead of resizing handles.

The Button control is selected and locked

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 (Figure 1.29).

**Figure 1.29**

Change the properties of the first button.

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

STEP 2: Change the Name property of the button to “displayButton” (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 “Display” (without the quotation marks). This step changes the words that appear on top of the button.

Set the Name and Text Properties for the Second Button

STEP 1: Select `button2` and change its Name property to “exitButton.”

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

Change Properties of the Form

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

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

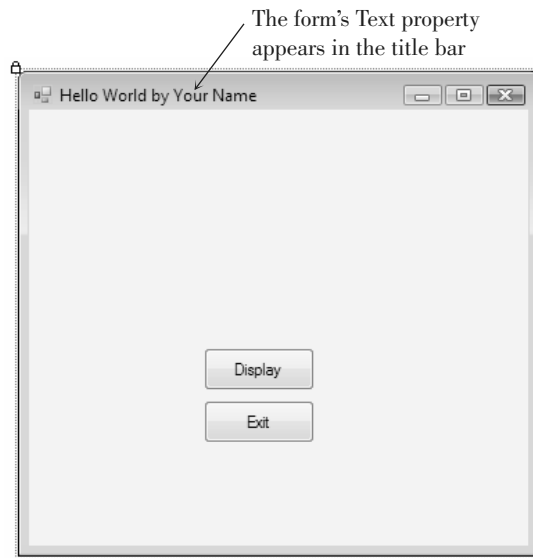
The Text property of a form determines the text that appears in the title bar. Your screen should now look like Figure 1.30.

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

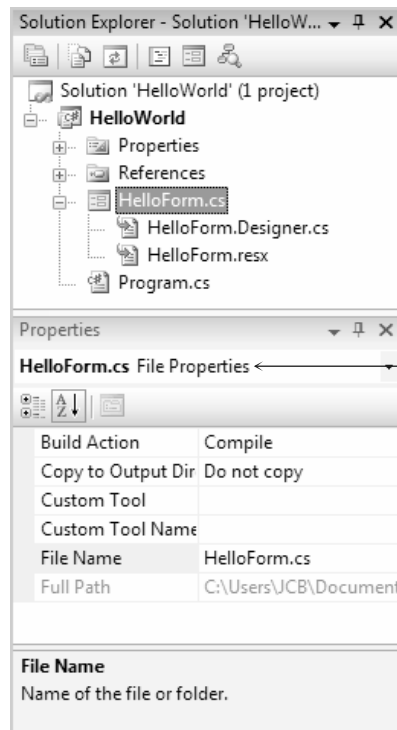
STEP 4: In the Solution Explorer, right-click on *Form1.cs* and choose *Rename* from the context menu. Change the file name to “HelloForm.cs”, making sure to retain the .cs extension. Press Enter when finished and click *Yes* on the confirmation dialog box. This changes the name of the file that saves to disk (Figure 1.31) as well as the name of the class.

TIP

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

**Figure 1.30**

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

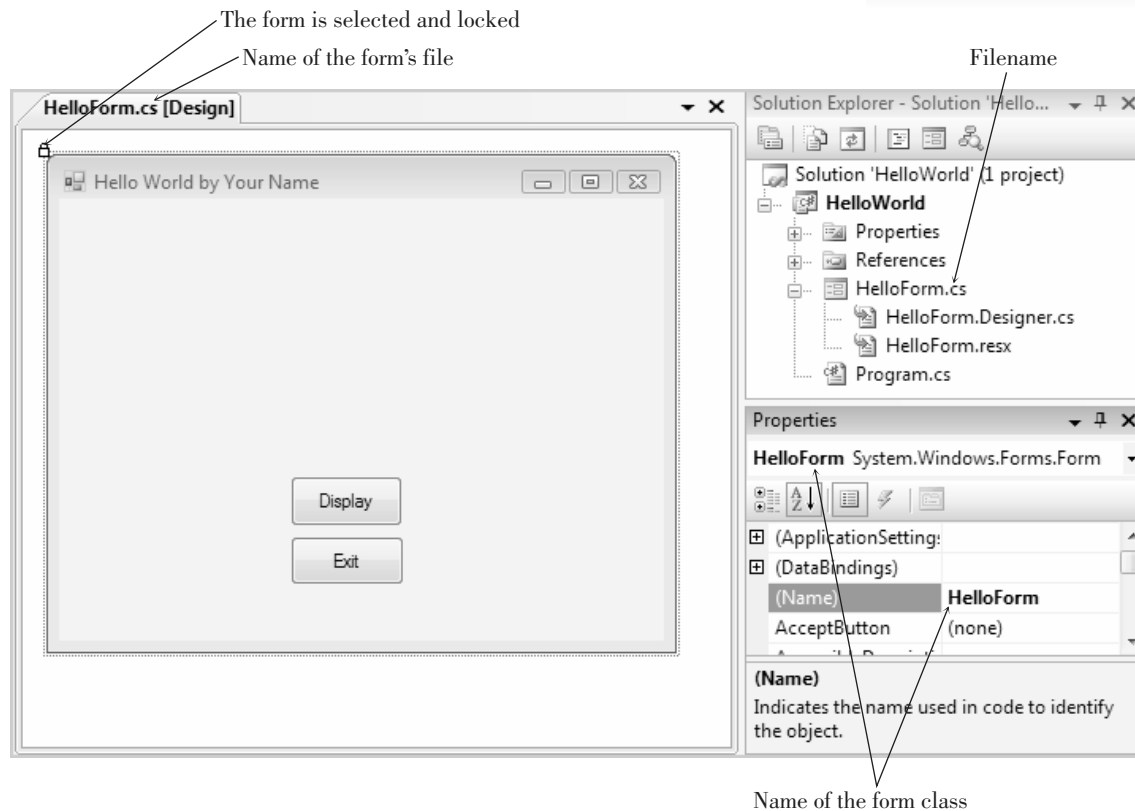
**Figure 1.31**

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

STEP 5: Click on the form in the Document window, anywhere except on a control. The name of the file appears on the tab at the top of the Designer window and the Properties window shows properties for the form's class, not the file. The C# designer changed the name of the form's class to match the name of the file (Figure 1.32).

Figure 1.32

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



Write Code

C# Events

While your project is running, the user can do many things, such as move the mouse around; click 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 C# 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 code for a particular event, then C# will respond to the event and automatically execute your method. *C# ignores events for which no methods are written.*

C# Event Handlers

You write code in C# in methods. For now, each method will begin with the words `private void` and the code will be enclosed in opening and closing braces `{ }`.

C# automatically names your **event-handling methods** (also called **event handlers**). The name consists of the object name, an underscore (`_`), and the name of the event. For example, the Click event for your button called `displayButton` will be `displayButton_Click`. For the sample project you are writing, you will have a `displayButton_Click` method and an `exitButton_Click` method.

✓ TIP

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

C# Code Statements

This first project requires two C# statements: the **comment** and the **assignment statement**. You also will execute a method of an object.

The Comment Statement

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

Good programming practices dictate that programmers include comments to clarify their projects. Every method should begin with a comment that describes its purpose. Every project should have comments 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 comments 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 comments.

C# comments begin with slashes. Most of the time, your comments will be on a separate line. You also can add slashes and a comment to the right end of a line of code.

The Comment Statement—Examples

Examples

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

Multiline Comments You also can create multiline comments by placing `/*` at the beginning and `*/` at the end. The enclosing symbols can be on lines by themselves or on existing lines. As you type additional lines between the beginning and ending symbols, the editor adds an asterisk at the start of each line, indicating that it is a comment line. However, you do not need the `*` at the beginning of each line. When you want to turn multiple lines of code into comments, just add the opening `/*` and ending `*/`.

```
/*  
 * Project:      Ch01HandsOn  
 * Programmer:   Bradley/Millspaugh  
 * Date:        June 2009  
 * Description:  This project displays a Hello World message  
 *              using labels and buttons.  
 * */  
  
/*Project:      Ch01HandsOn  
Programmer:    Bradley/Millspaugh  
Date:         June 2009  
Description:   This project displays a Hello World message  
              using labels and buttons. */
```

Ending a Statement

Most C# statements must be terminated by a semicolon (;). Comments and a few other statements (which you will learn about later) do not end with a semicolon. A C# statement may extend over multiple lines; the semicolon indicates that the statement is complete.

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 that appears on the right side of the equal sign is assigned to the property named on the left of the equal sign. It is often helpful to read the equal sign as “is replaced by.” For example, the following assignment statement would read “messageLabel.Text is replaced by Hello World.”

```
messageLabel.Text = "Hello World";
```

The Assignment Statement—General Form

General
Form

```
Object.Property = value;
```

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

The Assignment Statement—Examples

Examples

```
titleLabel.Text = "A Snazzy Program";  
addressLabel.Text = "1234 South North Street";  
messageLabel.AutoSize = true;  
numberInteger = 12;
```

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

Ending a Program by Executing a Method

To execute a method of an object, you write:

```
Object.Method();
```

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

Examples

```
helloButton.Hide();  
messageLabel.Show();
```

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

```
this.Close();
```

In most cases, you will include `this.Close()` in the event-handling method for an *Exit* button or an *Exit* menu choice.

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

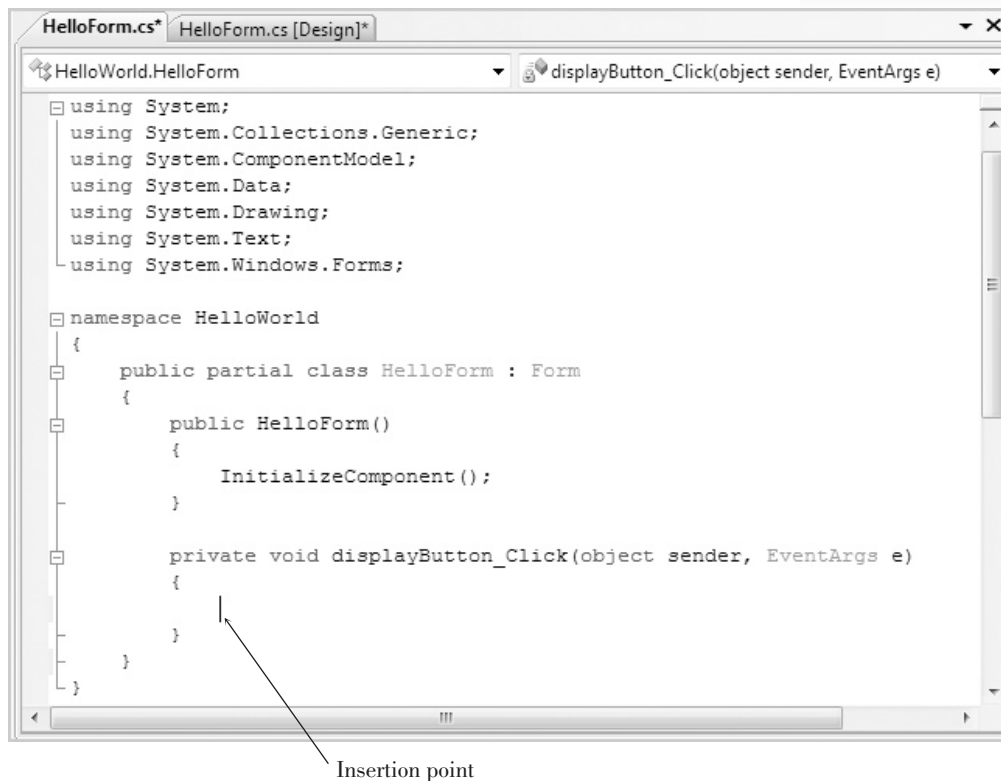
Code the Event-Handling Methods for Hello World

Code the Click Event Handler for the Display Button

STEP 1: Double-click the *Display* button. The Visual Studio editor opens with the header line of your method already in place, with the insertion point indented inside the opening and closing braces (Figure 1.33).

Figure 1.33

The Editor window, showing the first line of the `displayButton_Click` event handler with the insertion point between the opening and closing braces.



STEP 2: Type this comment statement:

```
// Display the Hello World message.
```

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

Follow good coding conventions and indent all lines between the opening and closing braces. The smart editor attempts to help you follow this convention. Also, always leave a blank line after the comments at the top of a method.

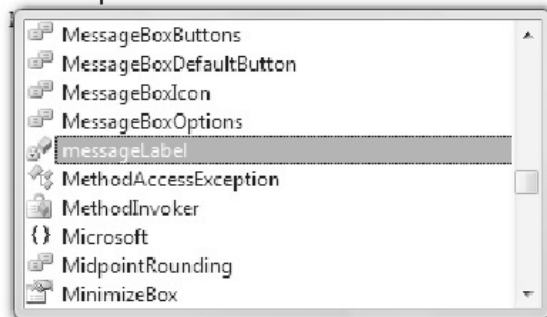
STEP 3: Press Enter twice and then type this assignment statement:

```
messageLabel.Text = "Hello World";
```

Note: When you type the names of objects and properties, allow IntelliSense to help you. When you type the first character of a name, such as the “m” of “messageLabel”, IntelliSense pops up a list of possible object names from your program (Figure 1.34). When several items match the first letter, you can type additional characters until you get a match, or use your keyboard down arrow or the mouse to highlight the correct item. To accept the correct item when it is highlighted, press the punctuation character that should follow the item, such as the period, spacebar, equal sign, semicolon, Tab key, or Enter key, or double-click the item with your mouse. For example, accept “messageLabel” by pressing the period and accept “Text” by pressing the spacebar, since those are the characters that follow the selected items.

```
private void displayButton_Click(object sender, EventArgs e)
{
    // Display the Hello World message.

    mes|
```



The assignment statement

```
messageLabel.Text = "Hello World";
```

assigns the literal “Hello World” to the Text property of the control called messageLabel. Compare your screen to Figure 1.35.

Figure 1.34

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

TIP

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

STEP 4: Return to the Form Designer (refer to Figure 1.32) by clicking on the *HelloForm.cs [Design]* tab on the Document window (refer to Figure 1.35).

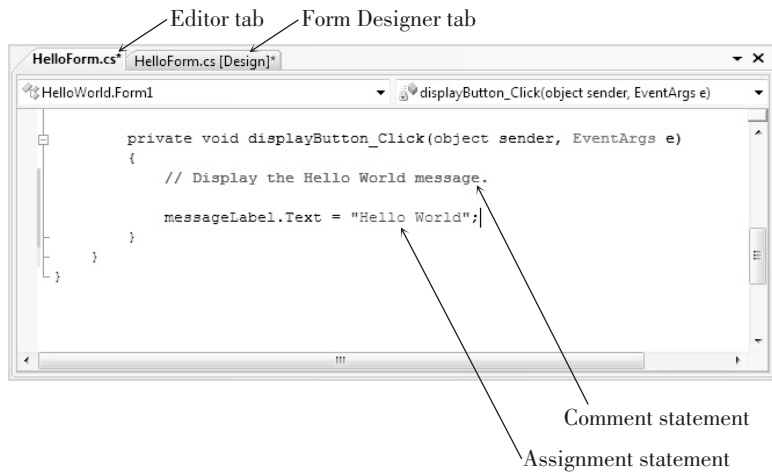


Figure 1.35

Type the comment and assignment statement for the *displayButton_Click* event handler.

Code the Click Event Handler for the Exit Button

STEP 1: Double-click the *Exit* button to open the editor for the *exitButton_Click* event handler.

STEP 2: Type this comment:

```
// Exit the project.
```

STEP 3: Press Enter twice and type this C# statement:

```
this.Close();
```

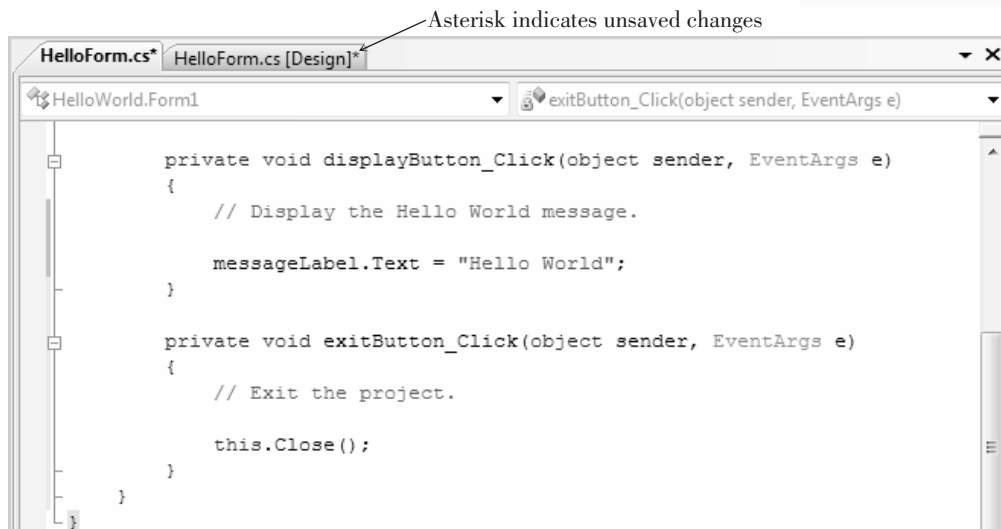
STEP 4: Make sure your code looks like the code shown in Figure 1.36.



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

Figure 1.36

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



Run the Project

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

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

Start the Project Running

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

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

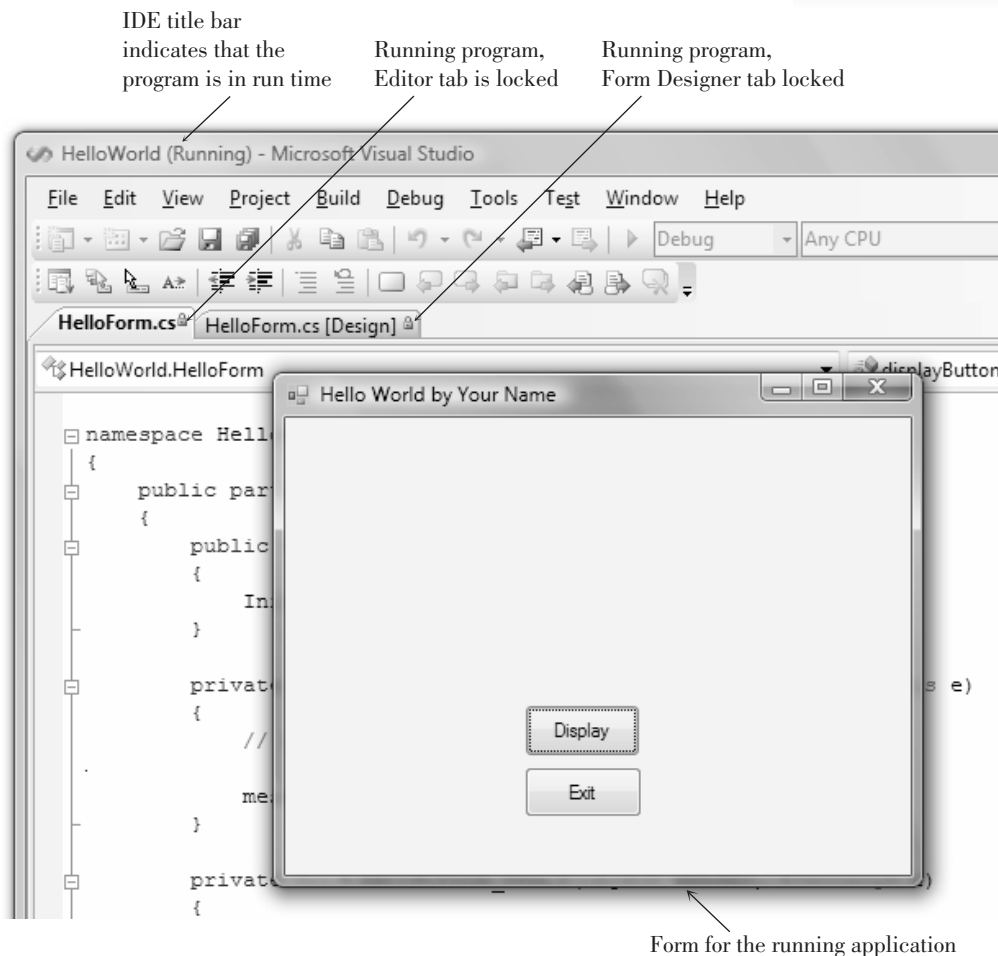
If all went well, the form appears and the Visual Studio title bar now indicates that you are in run time (Figure 1.37).

TIP

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

Figure 1.37

The form of the running application.



Click the Display Button

STEP 1: Click the *Display* button. Your “Hello World” message appears in the label (Figure 1.38).



Figure 1.38

Click the *Display* 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 such as this one, you will usually save your work as you go along. Unless you (or someone else) have changed the setting in the IDE’s *Options* dialog box, your files are automatically saved in a temporary location each time you build (compile) or execute (run) your project. After you have performed a save to a different location, files are automatically resaved each time you compile or run. You also can save the files as you work.



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

Save the Files

STEP 1: Open the Visual Studio *File* menu and choose *Save All*. This option will save the current form, project, and solution files.

Note: When saving a project, do not attempt to save a modified version by giving the project a new name. If you want to move or rename the project, it must be closed. See Appendix C for help.

Close the Project

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

Open the Project

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

- Select *Open Project* from the Visual Studio *File* menu and browse to find your *.sln* file, which has a small “9” as part of the file’s icon.

- Choose the project from the *File / Recent Projects* menu item.
- Choose the project from Recent Projects (if available) on the Start Page (*View / Other Windows / Start Page*).

Open the Project File

STEP 1: Open your project by choosing one of the previously listed techniques. Remember that the file to open is the .sln file.

If you do not see your form on the screen, check the Solution Explorer window—it should say *HelloWorld* for the project. Select the icon for your form: *HelloForm.cs*. You can double-click the icon or single-click and click on the *View Designer* button at the top of the Solution Explorer (Figure 1.39); your form will appear in the Designer window. Notice that you also can click on the *View Code* button to display your form’s code in the Editor window.

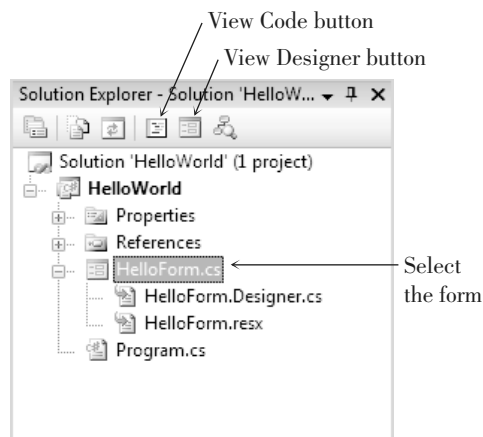


Figure 1.39

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.

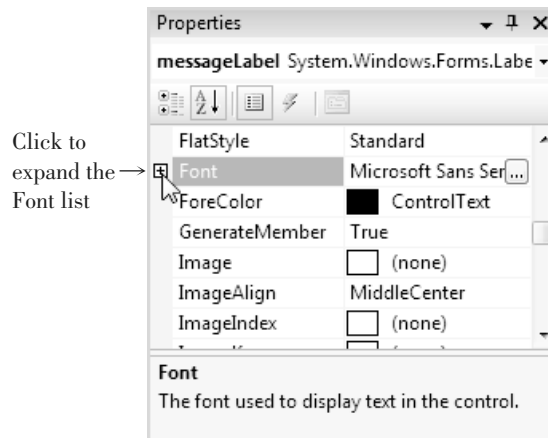
Modify the Project

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.

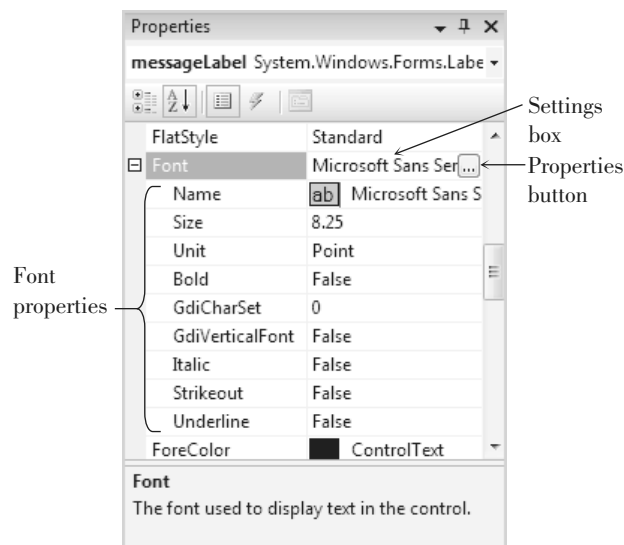
Change the Size and Alignment of the Message

- STEP 1:** Right-click the form to display the context menu. If your controls are currently locked, select *Lock Controls* to unlock the controls so that you can make changes.
- STEP 2:** Drop down the Object list at the top of the Properties window and select *messageLabel*, which will make the label appear selected.
- STEP 3:** Scroll to the Font property in the Properties window. The Font property is actually a Font object that has a number of properties. To see the Font properties, click on the small plus sign on the left (Figure 1.40); the Font properties will appear showing the current values (Figure 1.41).

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

Figure 1.40

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

Figure 1.41

You can change the individual properties of the Font object.

STEP 4: Click the Properties button for the font (the button with the ellipsis on top) to display the *Font* dialog box (Figure 1.42). 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.

STEP 5: Select the *TextAlign* property. The Properties button that appears with the down-pointing arrow indicates a drop-down list of choices. Drop down the list (Figure 1.43) and choose the center box; the alignment property changes to *MiddleCenter*.

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 1.44). (You can resize the form if necessary.)

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

TIP

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

Figure 1.42

Choose 12 point on the Font dialog box.

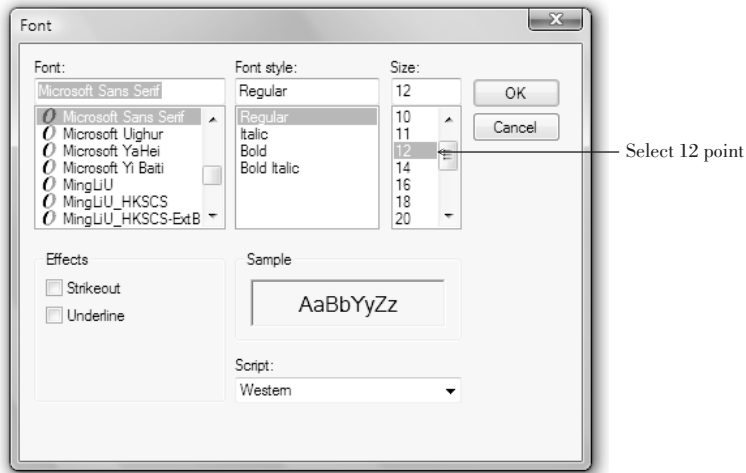


Figure 1.43

Select the center box for the TextAlign property.

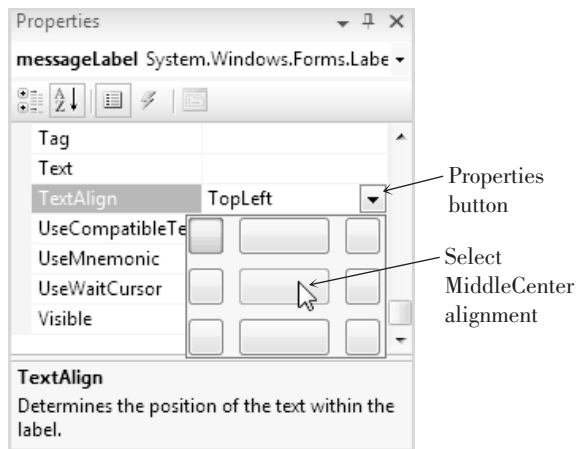


Figure 1.44

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



TIP

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

The Label's AutoSize Property Earlier you changed the `AutoSize` property of `messageLabel` to `False`, a step that allows you to set the size of the label yourself. When `AutoSize` is set to `True` (the default), the label resizes automatically to accommodate the `Text` property, which can be an advantage when the text or font size may change. However, if you plan to delete the `Text` property, as you did for `messageLabel`, the label resizes to such a tiny size that it is difficult to see.

Any time that you want to set the size of a label yourself, change the `AutoSize` property to `False`. This setting also allows you to create taller labels that allow a long `Text` property to wrap to multiple lines. If you set the `Text` property to a very long value when `AutoSize` is set to `True`, the label will resize only to the edge of the form and cut off any excess text, but if `AutoSize` is set to `False` and the label has been resized to a taller height, the long `Text` property will wrap.

Change the Text of the Display Button

Because we plan to display the message in one of two languages, we'll change the text on the *Display* button to "English" and move the buttons to allow for another button.

STEP 1: Select the *Display* button and change its `Text` property to "English."

STEP 2: Move the *English* button and the *Exit* button to the right and leave room for a *Spanish* button (Figure 1.45).



Figure 1.45

Move the English and Exit buttons and add a Spanish button.

Add a Spanish Button

STEP 1: Add a new button. Move and resize the buttons as necessary, referring to Figure 1.45.

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

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

TIP

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

Add an Event Handler for the Spanish Button

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

STEP 2: Add a comment:

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

STEP 3: Press Enter twice and type the following line of C# code.

```
messageLabel.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 Comments

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

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

STEP 2: Press Enter to create a blank line.

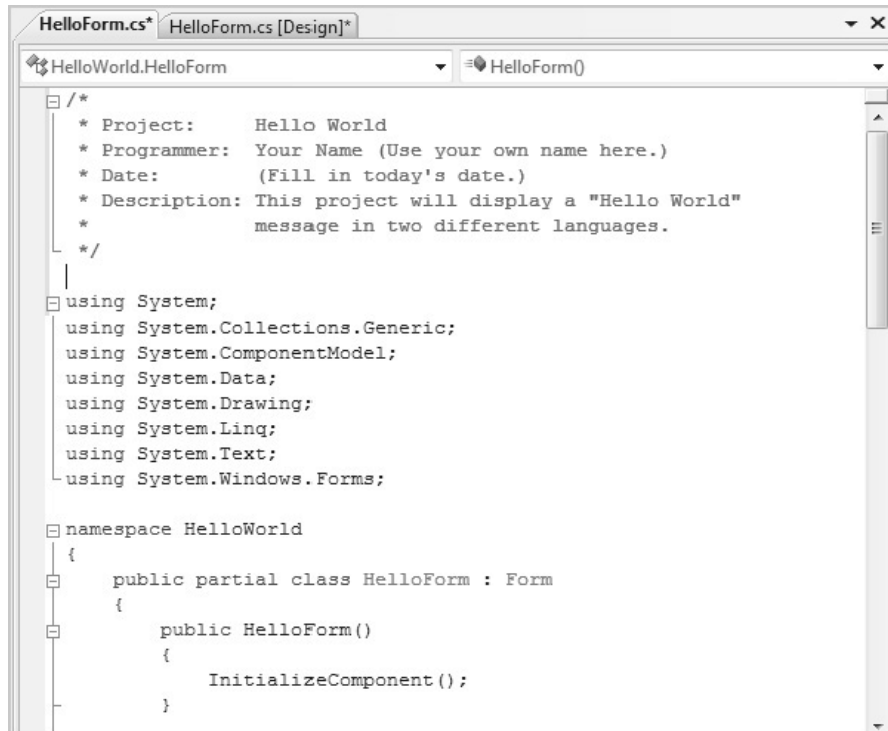
Warning: If you accidentally deleted the first line, click *Undo* (or press Ctrl + Z) and try again.

STEP 3: Move the insertion point up to the blank line and type the following comments, one per line (Figure 1.46):

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



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



```

HelloForm.cs* HelloForm.cs [Design]*
HelloWorld.HelloForm
HelloForm()
/*
 * Project:    Hello World
 * Programmer: Your Name (Use your own name here.)
 * Date:      (Fill in today's date.)
 * Description: This project will display a "Hello World"
 *             message in two different languages.
 */
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace HelloWorld
{
    public partial class HelloForm : Form
    {
        public HelloForm()
        {
            InitializeComponent();
        }
    }
}

```

Figure 1.46

Enter the comments at the top of the form file.

Finish Up

STEP 1: Run the project again. Test each language button multiple times; then click the *Exit* button.

Print the Code

Select the Printing Options

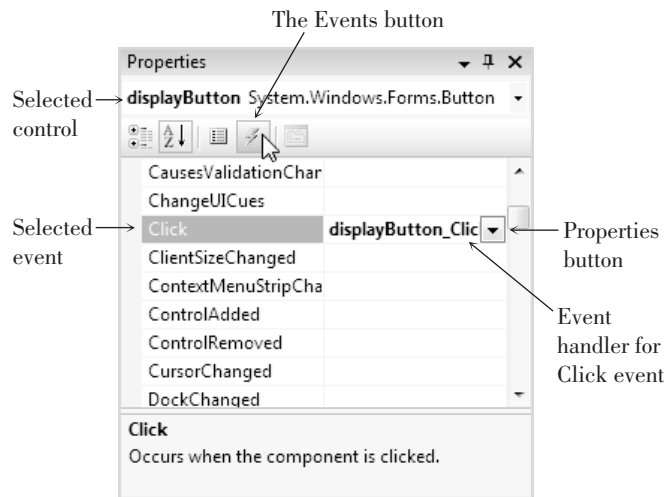
STEP 1: Make sure that the Editor window is open and showing your form's code. The *File/Print* command is disabled unless the code is displaying and its window selected.

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

View Event Handlers

You also can get to the event-handling methods for a control using the Properties window in design mode. With a button control selected, click on the *Events* button (lightning bolt) in the Properties window; all of the events for that control display (Figure 1.47). If you've already written code for the Click event, the method name appears bold in the Properties window. When you double-click on the event, the editor takes you to the method in the code window.

To write an event-handling method for any of the available events of a control, double-click the event name. You will be transferred to the Code Editor window with the insertion point inside the template for the new event handler. You also can click in any event name in the Properties window and then drop down a list of all previously written methods and select a method to assign as the event handler.

**Figure 1.47**

Click on the **Events** button to see the available events for a selected control. Any event handlers that are already written appear in bold. Double-click an event to jump to the Editor window inside the event handler for that method, or drop down the list to select a method to assign as the handler for the event.

A Sample Printout

This output is produced when you print the form's code. An explanation of some of the features of the code follows the listing.

```
C:\Users\...\. . \Ch01HelloWorld\HelloForm.cs      1
/*
 * Project:      Hello World
 * Programmer:   Your Name (Use your own name here.)
 * Date:         (Fill in today's date.)
 * Description:  This project will display a "Hello World"
 *              message in two different languages.
 */

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace HelloWorld
{
    public partial class HelloForm : Form
    {
        public HelloForm()
        {
            InitializeComponent();
        }

        private void displayButton_Click(object sender, EventArgs e)
        {
            // Display the Hello World message.
        }
    }
}
```

```
        messageLabel.Text = "Hello World";
    }

    private void exitButton_Click(object sender, EventArgs e)
    {
        // Exit the project.

        this.Close();
    }

    private void spanishButton_Click(object sender, EventArgs e)
    {
        // Display the Hello World message in Spanish.

        messageLabel.Text = "Hola Mundo";
    }
}
}
```

Automatically Generated Code

In the preceding code listing, you see many statements that you wrote, plus some more that appeared “automatically.” Although a programmer *could* begin a C# program by using a simple text editor and write all of the necessary statements to make the program run, using the development tools of the Visual Studio IDE is much quicker and more efficient. The IDE adds a group of statements by default and sets up the files for the project to accommodate the majority of applications. Later, when your programs include database tables, you will have to write additional `using` statements.

The Using Statements

The `using` statements appear at the top of the file after the comments that you wrote. `Using` statements provide references to standard groups of classes from the language library. For example, the statement `using System.Windows.Forms;` allows your program to refer to all of the `Windows` controls that appear in the toolbox. Without the `using` statement, each time that you wanted to refer to a `Label` control, for example, you would have to specify the complete reference: `System.Windows.Forms.Label.messageLabel`. Instead, in the program with the `using` statement, you can just refer to `messageLabel`.

The Namespace Statement

As mentioned earlier, a namespace provides a way to refer to programming components by location or organization. In the `Label` example in the preceding section, “`Label`” is the class and “`System.Windows.Forms`” is the namespace, or library grouping where “`Label`” is found. You can think of a namespace as similar to a telephone area code: In any one area code, a single phone number can appear only once, but that same phone number can appear in any number of other area codes.

Using the .NET Framework, every program component is required to have a namespace. The VS IDE automatically adds a `namespace` statement to your program. The default namespace is the name of your solution, but you can use a different name if you wish. Many companies use the namespace to

organize applications such as the company name and functional organization, `LookSharpFitnessCenter.Payroll`, for example.

In Visual Studio, one solution can contain multiple projects. All of the solutions in this text contain only one project, so you can think of a solution and a project as being equal.

The Class Statement

In object-oriented programming, code is organized into classes. A new class can be based on (inherit from) another class, which gives the new class all of the properties and methods of the original class (the base class).

When you create a new form, you declare a new class (HelloForm in the earlier example). The new class inherits from the Form base class, which makes your new form behave like a standard form, with a title bar, maximize and minimize buttons, and resizable borders, among other behaviors.

A class may be split into multiple files. VS uses this feature, to place most of the code automatically generated by the Form Designer in a separate file that is part of the form's class.

The automatically generated statement

```
public partial class HelloForm : Form
```

means that this is a new class called HelloForm that inherits from the Form class. The new class is a partial class, so another file can exist that also contains statements that are part of the HelloForm class. You will learn more about classes and files in later chapters.

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.

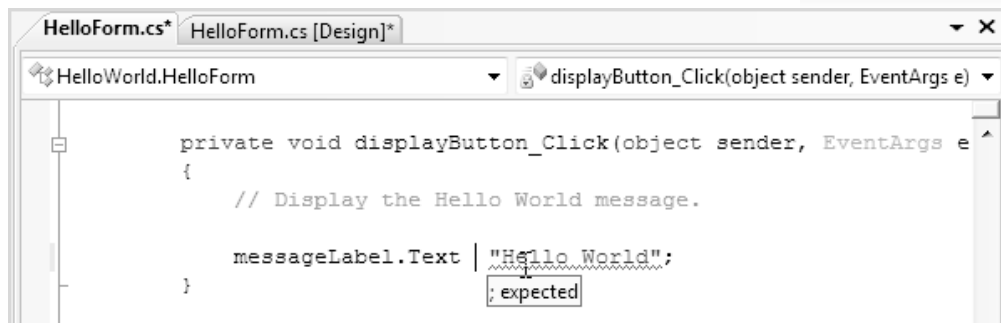
Syntax Errors

When you break C#'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*.

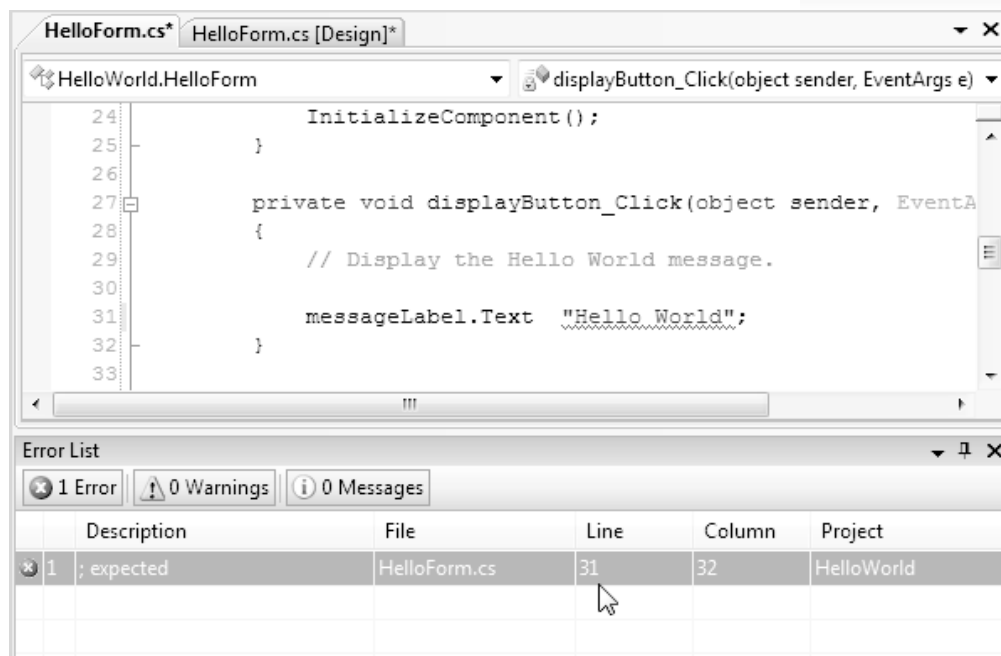
The editor identifies syntax errors as you move off the offending line. A red squiggly line appears under the part of the line that the editor cannot interpret. You can view the error message by pausing the mouse pointer over the error, which pops up a box that describes the error (Figure 1.48). You also can display an Error List window, which appears at the bottom of the Editor window and shows all error messages along with the line number of the statement that caused the error. You can display line numbers on the source code (Figure 1.49) with *Tools / Options / Text Editor / C# / General / Display / Line Numbers*.

Figure 1.48

The editor identifies a syntax error with a squiggly red line; you can point to an error to pop up the error message.

**Figure 1.49**

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



The quickest way to jump to an error line is to point to a message in the Error List window and double-click. The line in error will display in the Editor window with the error highlighted (Figure 1.50).

If a syntax error is found by the compiler, you will see the dialog box shown in Figure 1.51. Click *No* and return to the editor, correct your errors, and run the program again.

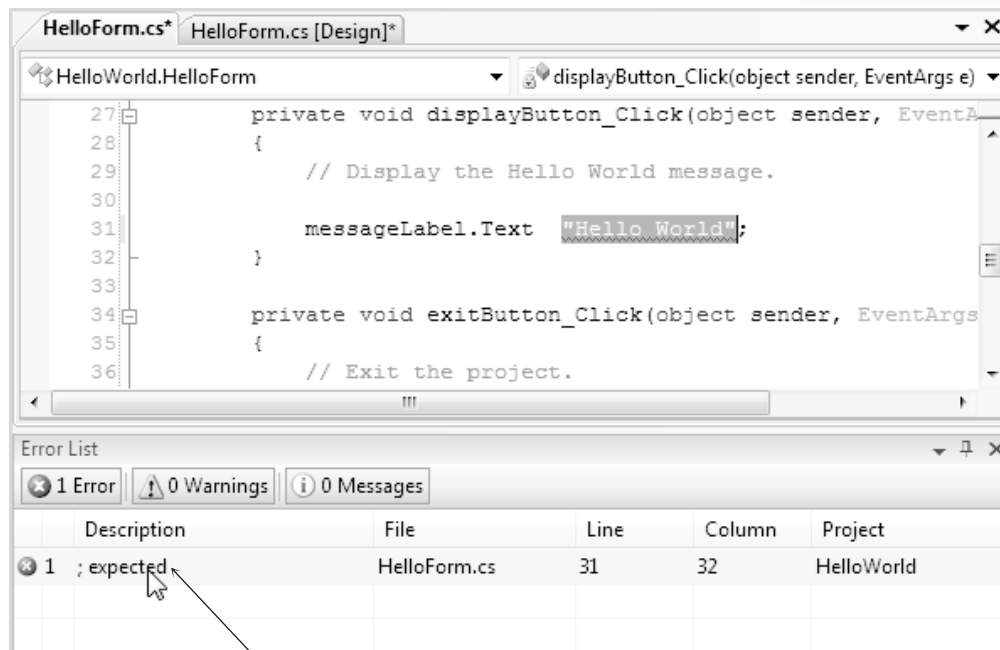
Run-Time Errors

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

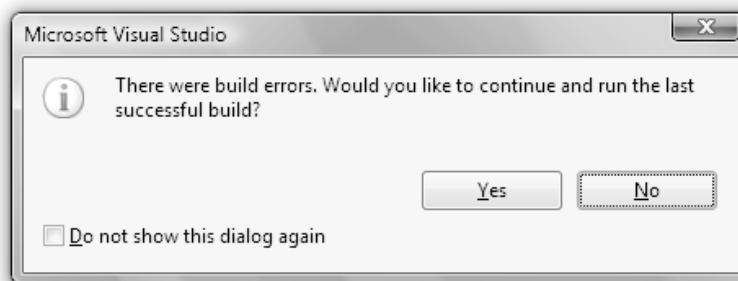
Statements that cannot execute correctly cause run-time errors. The statements are correctly formed C# statements that pass the syntax checking; however, the statements fail to execute due to some serious issue. You can cause

Figure 1.50

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

**Figure 1.51**

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



run-time errors by attempting to do impossible arithmetic operations, such as calculate with nonnumeric data, divide by zero, or find the square root of a negative number.

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

Logic Errors

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

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

For example, the Hello World project in this chapter has event-handling methods for displaying “Hello World” in English and in Spanish. If the contents of the two methods 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 void spanishButton_Click(object sender, EventArgs e)
{
    // Display the Hello World message in Spanish.

    messageLabel.Text = "Hello World";
}
```

Project Debugging

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

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

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

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

A Clean Compile

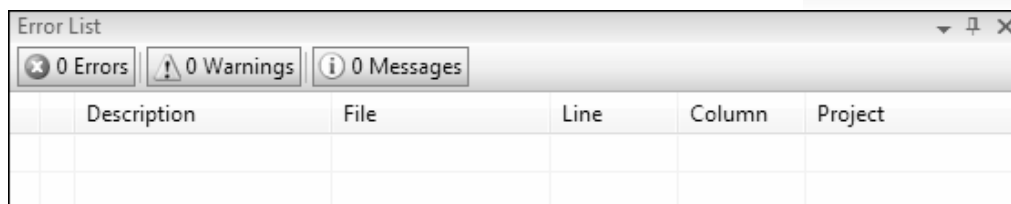
When you start executing your program, the first step is called *compiling*, which means that the C# statements are converted to Microsoft Intermediate Language (MSIL). Your goal is to have no errors during the compile process: a **clean compile**. Figure 1.52 shows the Error List window for a clean compile: 0 Errors; 0 Warnings; 0 Messages.

TIP

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

Figure 1.52

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



Modifying an Event Handler

When you double-click a Button control to begin writing an event-handling method for the Click event, several things happen. As an example, say that you

have a button on your form called *button1*. If you double-click *button1*, the Editor window opens with a template for the new method:

```
private void button1_Click(object sender, EventArgs e)
{
}

```

The insertion point appears between the opening and closing braces, where you can begin typing your new method. But behind the scenes, VS also adds a line to the (hidden) *FormName.Designer.cs* file that assigns this new method to the Click event of the button.

As long as you keep the name of the button unchanged and don't delete the method, all is well. But if you want to rename the button, or perhaps delete the method (maybe you accidentally double-clicked a label or the form and have a method that you really don't want or need), then you will need to take additional steps.

Deleting an Event Handler

Assume that you have double-clicked the form called *Form1* and now have an extra event handler that you do not want. If you simply delete the event handler, your program generates an error message due to the extra code that appears in the Form's designer.cs file. When you double-click on the form, the extra Form Load event handler looks like this:

```
private void Form1_Load(object sender, EventArgs e)
{
}

```

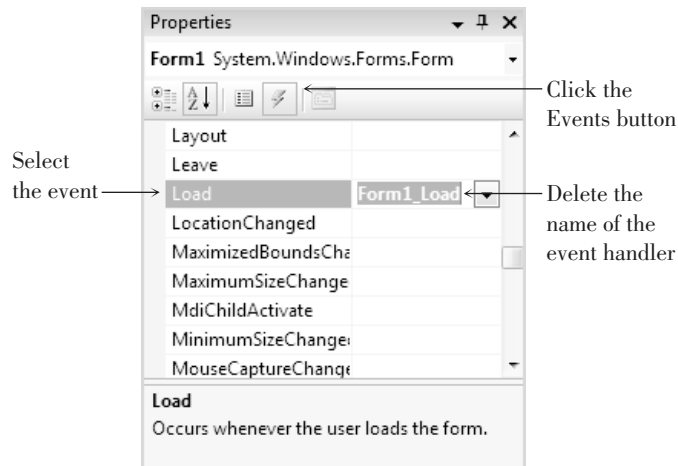
If you delete these lines of code and try to run the program, you receive an error message that “WindowsApplication1.Form1' does not contain a definition for 'Form1_Load'.” If you double-click on the error message, it takes you to a line in the *Form1.Designer.cs* file. You can delete the line of code that it takes you to, which, in this example, is

```
this.Load += new System.EventHandler(this.Form1_Load);
```

The preferable way to remove the statement that assigns the event handler is to use the Properties window in the designer. First, make sure to select the form or control that has the unwanted event handler assigned; then click on the *Events* button in the Properties window (Figure 1.53). You will see the event-handling method's name for the name of the event. You can select and delete the name of the method, which removes the assignment statement from the Designer.cs file, and you will not generate an error message when you delete the code lines.

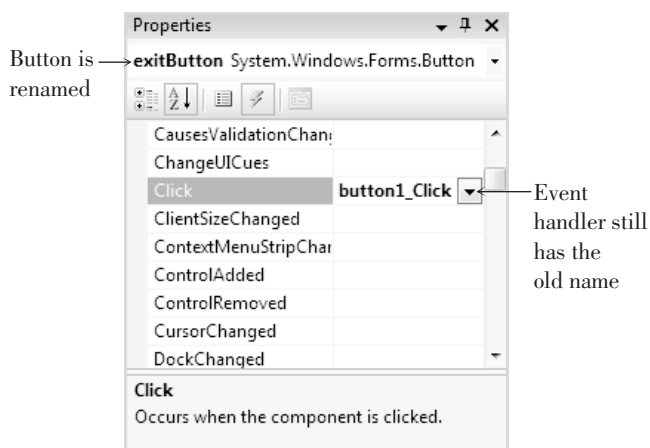
Renaming a Control

You can receive an error if you rename a control after you write the code for its event. For this example, assume that you add a button that is originally called *button1*. You write the code for the *button1_Click* event handler and then decide to change the button's name to *exitButton*. (This scenario occurs quite often, especially with beginning programmers.)

**Figure 1.53**

*To remove the form's event handler, select the form and click on the **Events** button in the Properties window. Then delete the entry for the event handler—`Form1_Load` in this example.*

If you simply change the Name property of `button1` to `exitButton` in the Form Designer, your program will still run without an error message. But you may be surprised to see that the event handler is still named `button1_Click`. If you check the events in the Properties window, you will see why (Figure 1.54): Although the control was renamed, the event handler was not. And if you type a new name into the Properties window (`exitButton_Click`, for example), a new (empty) method template will appear in your code. The code that you wrote in the `button1_Click` method is still there and the new `exitButton_Click` method is empty. One solution is to just cut-and-paste the code from the old method to the new one. You can safely delete the empty `button1_Click` method since it no longer is assigned as the event handler.

**Figure 1.54**

Even though you rename a control, the event handler is not renamed automatically.

Another way to change the name of an event handler is to use refactoring, which allows you to make changes to an existing object. After you change the name of the control using the designer, switch to the Editor window and right-click on the name of the event-handling method (`button1_Click` in this example). From the context menu, select *Refactor / Rename*. The *Rename* dialog box

shows the current name of the method (Figure 1.55). Enter the new name, making sure to include the “_Click.” When you click *OK*, you see a *Preview Changes-Rename* dialog box with the proposed changes highlighted (Figure 1.56). Click *Apply* and all references to the old name are changed to the new one, which corrects the line in the Designer.cs file that assigns the event handler.

Figure 1.55

Change the name of the event-handling method using Refactor / Rename, which changes the name of the method and the assignment of the event handler in the form’s Designer.cs file.

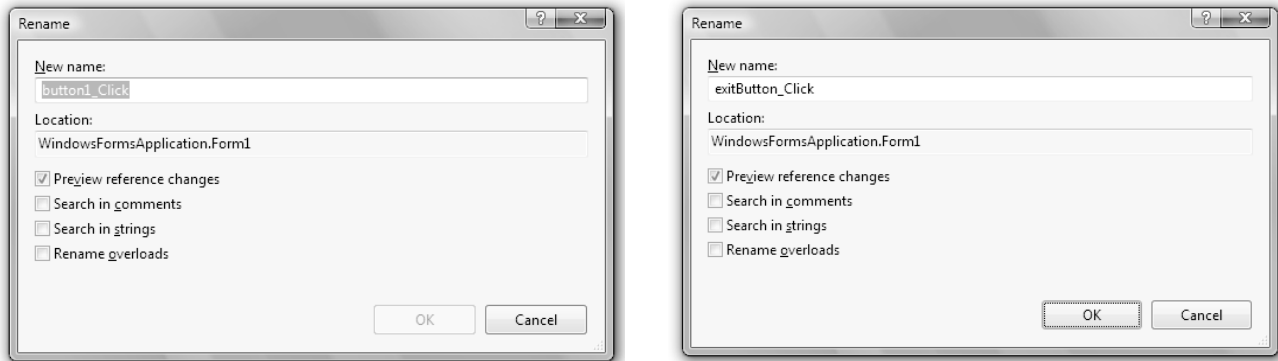
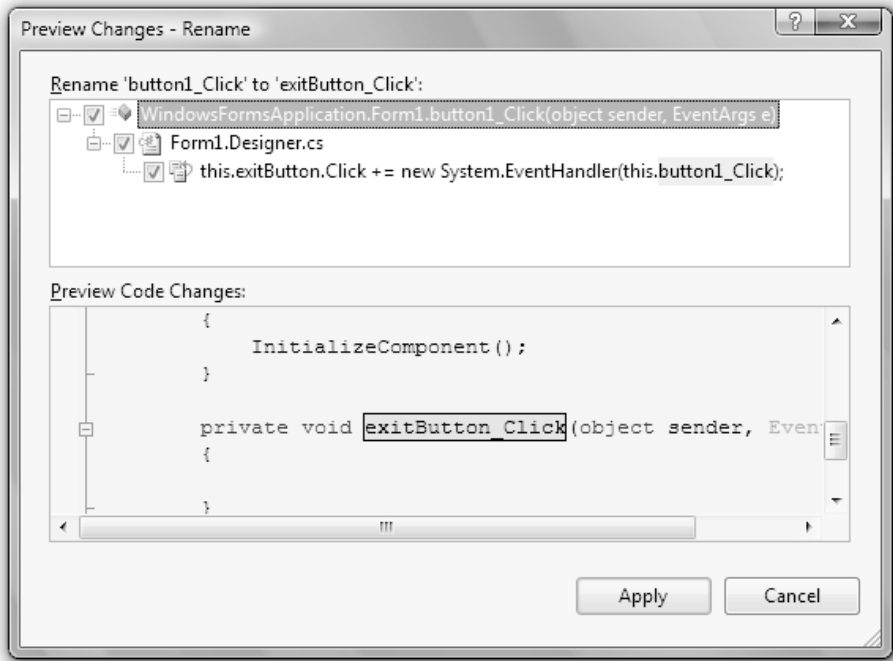


Figure 1.56

The Preview Changes-Rename dialog box shows the changes that you are about to make. Click Apply to complete the Rename operation.



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 C# rules for naming objects, methods, 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, C# requires the name to begin with a letter or an underscore. The name can contain letters, digits, and underscores. An object name cannot include a space or punctuation mark and cannot be a reserved word, such as `button` or `Close`, but can contain one. For example, `exitButton` and `closeButton` are legal. C# is case sensitive, so `exitbutton`, `ExitButton`, and `exitButton` refer to three different objects.

The Naming Conventions

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

Examples

```
messageLabel  
exitButton  
discountRateLabel
```

Do not keep the default names assigned by C#, 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 program 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.

For forms and other classes, capitalize the first letter of the name and all other words within the name. You will find this style of capitalization referred to as **pascal casing** in the MSDN Help files. Always append the word *Form* to the end of a form name.

Examples

```
HelloForm  
MainForm  
AboutForm
```

Refer to Table 1.2 for sample object names.

Recommended Naming Conventions for C# Objects.

Table 1.2

Object Class	Example
Form	DataEntryForm
Button	exitButton
Label	totalLabel
TextBox	paymentAmountTextBox
RadioButton	boldRadioButton
CheckBox	printSummaryCheckBox
PictureBox	landscapePictureBox
ComboBox	bookListComboBox
ListBox	ingredientsListBox
SoundPlayer	introPageSoundPlayer

Visual Studio Help

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

The VS Help facility includes 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, C#, Visual Basic, and C++. You will want to filter the information to display only the Visual C# and related information.

Installing and Running MSDN

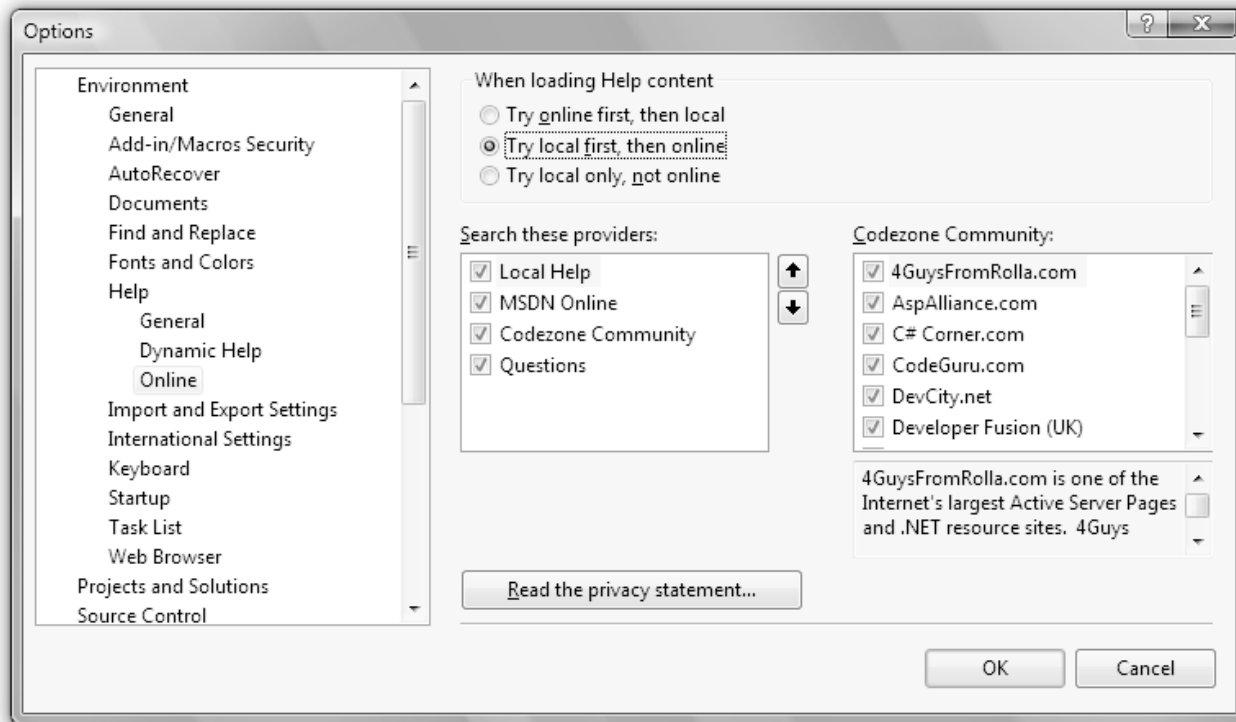
You can run MSDN from a hard drive, or online. Of course, if you plan to access MSDN online, you must have a live Internet connection as you work.

Depending on how you install C#, you are given the option to refer first to online, first to local, or only to local. You can change this setting later in the *Options* dialog box (Figure 1.57). Select *Tools / Options* and expand the *Environment* node and the *Help* node. Click on *Online*. You can choose the options to *Try online first, then local*; *Try local first, then online*; or *Try local only, not online*. Notice also that you can select sites to include in Help topics.

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

Figure 1.57

In the Options dialog box, you can specify the preferred source for Help content and choose the Help providers.



Viewing Help Topics

The Help system in Visual Studio 2008 allows you to view the Help topics in a separate window from the VS IDE, so you can have both windows open at the same time. When you choose *How Do I*, *Search*, *Contents*, *Index*, or *Help Favorites* from the *Help* menu, a new window opens on top of the IDE window (Figure 1.58). You can switch from one window to the other, or resize the windows to view both on the screen if your screen is large enough.

You can choose to filter the Help topics so that you don't have to view topics for all of the languages when you search for a particular topic. In the *Index* or *Contents* window, drop down the *Filtered by* list and choose *Visual C# Express Edition* for the Express Edition or *Visual C#* for the Professional Edition (Figure 1.59).

In the *Search* window, you can choose additional filter options, such as the technology and topic type. Drop down a list and select any desired options (Figure 1.60).

In the *Help Index* window, you see main topics and subtopics (indented beneath the main topics). All main topics and some subtopics have multiple entries available. When you choose a topic that has more than one possible entry, the *Index Results* pane opens up below the main Document window (refer to Figure 1.58). Click on the entry for which you are searching and the corresponding page appears in the Document window. For most controls, such as the Label control that appears in Figure 1.58, you will find references for mobile controls, Web controls, and Windows Forms. For now, always choose Windows Forms. Chapters 1 to 8 deal with Windows Forms exclusively; Web Forms are introduced in Chapter 9.

Figure 1.58

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

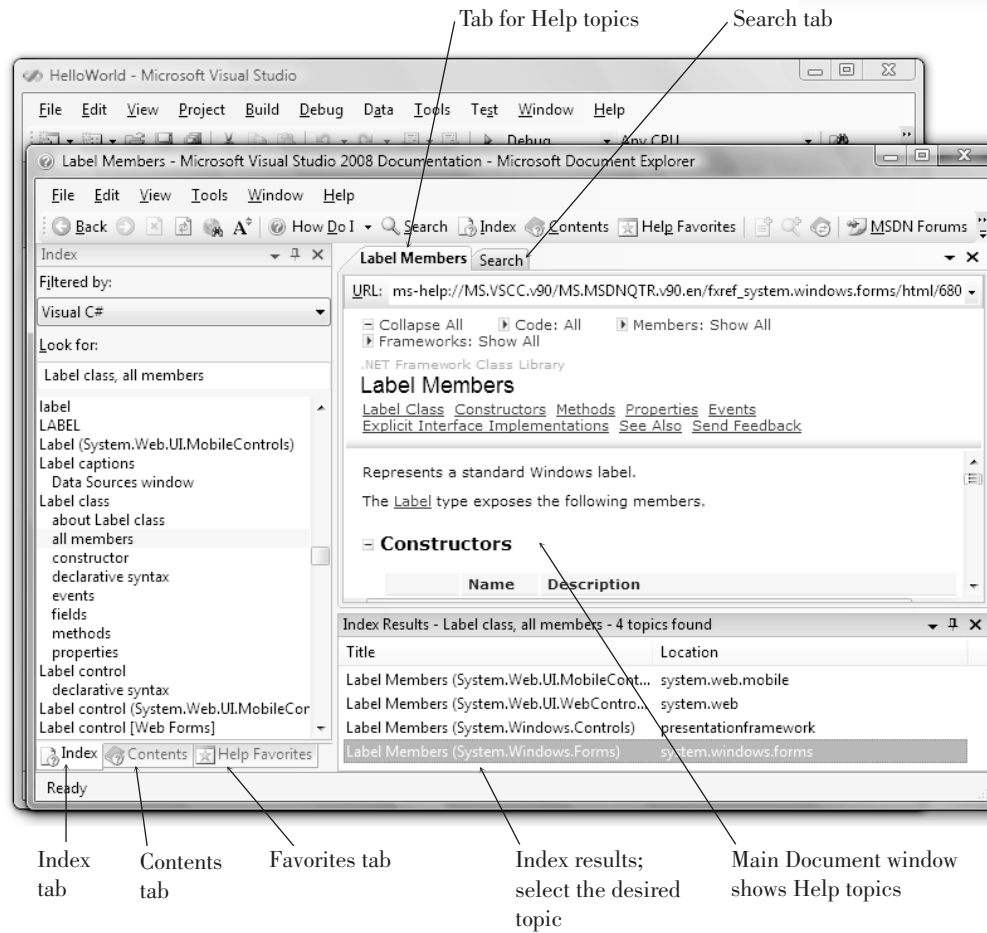


Figure 1.59

Filter the Help topics so that only the C# topics appear.

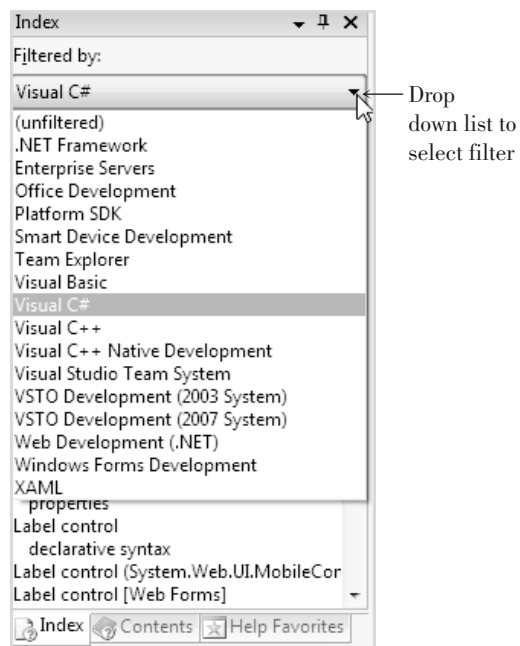
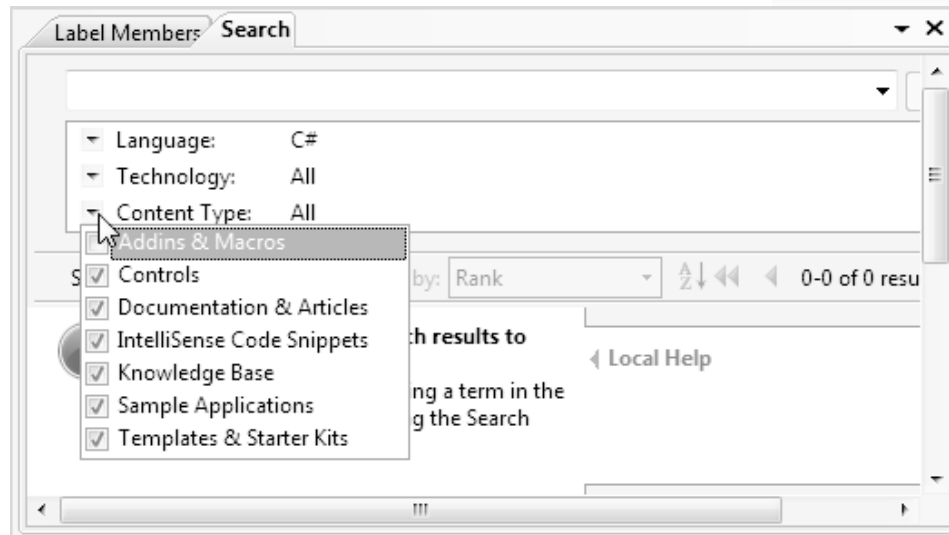


Figure 1.60

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



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

Context-Sensitive Help

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

Managing Windows

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

- To close a window that is a part of a tabbed window, click the window's *Close* button. Only the top window will close.
- To switch to another window that is part of a tabbed window, click on its tab.

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

Feedback 1.1

Note: Answers for Feedback questions appear in Appendix A.

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

Your Hands-On Programming Example

Write a program for the Look Sharp Fitness Center to display the current promotions. Include a label for the current special and buttons for each of the following departments: Clothing, Equipment and Accessories, Juice Bar, Membership, and Personal Training.

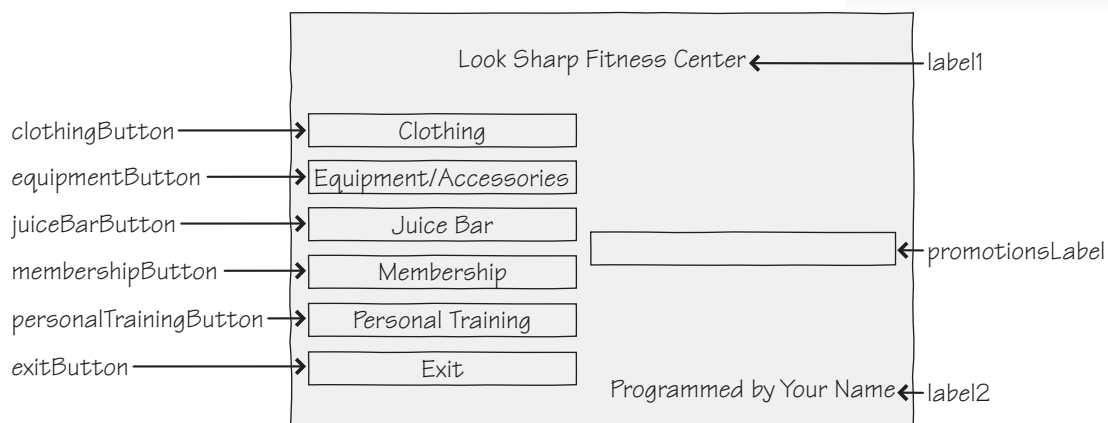
The user interface should also have an *Exit* button and a label with the programmer’s name. Use appropriate names for all controls. Make sure to change the Text property of the form.

Planning the Project

Sketch a form (Figure 1.61), which your users sign off as meeting their needs.

Figure 1.61

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



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

Plan the Objects and Properties

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

Object	Property	Setting	
PromotionForm	Name	PromotionForm	
	Text	Current Promotions	
	StartPosition	CenterScreen	
label1	Text	Look Sharp Fitness Center	Hint: Do not change the name of this label.
	Font	18 pt.	
label2	Text	Programmed by Your Name	
promotionsLabel	Name	promotionsLabel	
	AutoSize	True	
	Text	(blank)	
	TextAlign	MiddleLeft	
clothingButton	Name	clothingButton	
	Text	Clothing	
equipmentButton	Name	equipmentButton	
	Text	Equipment/Accessories	
juiceBarButton	Name	juiceBarButton	
	Text	Juice Bar	
membershipButton	Name	membershipButton	
	Text	Membership	
personalTrainingButton	Name	personalTrainingButton	
	Text	Personal Training	
exitButton	Name	exitButton	
	Text	Exit	

Plan the Event Methods You will need event-handling methods for each button.

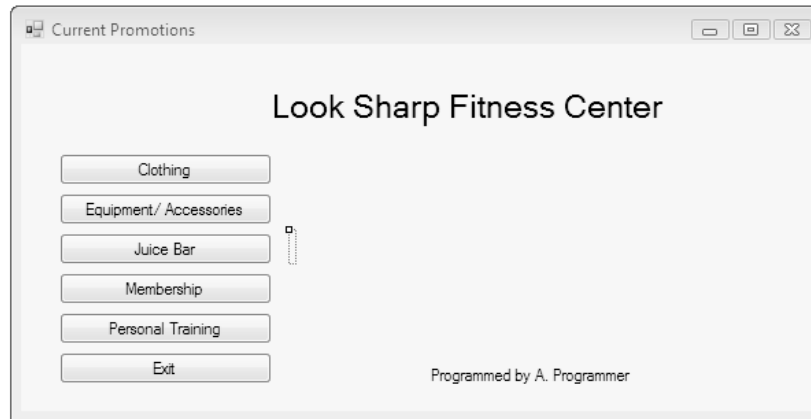
Method	Actions—Pseudocode
clothingButton_Click	Display “Take an extra 30% off the clearance items.” in the label.
equipmentButton_Click	Display “Yoga mats—25% off.”
juiceBarButton_Click	Display “Try a free serving of our new WheatBerry Shake.”
membershipButton_Click	Display “First month personal training included.”
personalTrainingButton_Click	Display “3 free sessions with membership renewal.”
exitButton_Click	End the project.

Write the Project Follow the sketch in Figure 1.61 to create the form. Figure 1.62 shows the completed form.

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

Figure 1.62

The form for the hands-on programming example.



The Project Coding Solution

```

/*
 * Project:      Ch01HandsOn
 * Programmer:   Bradley/Millspaugh
 * Date:        June 2009
 * Description:  This project displays current sales for
 *              each department.
 */

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Ch01HandsOn
{
    public partial class PromotionsForm : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}

```

```
private void exitButton_Click(object sender, EventArgs e)
{
    // End the project.

    this.Close();
}

private void clothingButton_Click(object sender, EventArgs e)
{
    // Display current promotion.

    promotionsLabel.Text = "Take an extra 30% off the clearance items.";
}

private void equipmentLabel_Click(object sender, EventArgs e)
{
    // Display current promotion.

    promotionsLabel.Text = "Yoga mats--25% off.";
}

private void juiceBarButton_Click(object sender, EventArgs e)
{
    // Display current promotion.

    promotionsLabel.Text = "Try a free serving of our new WheatBerry Shake.";
}

private void membershipButton_Click(object sender, EventArgs e)
{
    // Display current promotion.

    promotionsLabel.Text = "First month personal training included.";
}

private void personalTrainingButton_Click(object sender, EventArgs e)
{
    // Display current promotion.

    promotionsLabel.Text = "3 free sessions with membership renewal.";
}
}
```

Summary

1. Visual C# is an object-oriented language primarily used to write application programs that run in Windows or on the Internet using a graphical user interface (GUI).
2. In the OOP object model, classes are used to create objects that have properties, methods, and events.
3. The current release of C# is called Visual C# 2008 and is one part of Visual Studio. C# is available individually in an Express Edition or in Visual Studio Professional Edition and Team System versions.

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-handling methods.
6. The three steps to creating a C# project are (1) define the user interface, (2) set the properties, and (3) write the code.
7. A C# application is called a *solution*. Each solution may 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 .csproj, and form files and additional C# files have an extension of .cs. In addition, the Visual Studio environment and the C# 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. Visual Studio has three modes: design time, run time, and debug time.
10. You can customize the Visual Studio IDE and reset all customizations back to their default state.
11. You create the user interface for an application by adding controls from the toolbox to a form. You can move, resize, and delete the controls.
12. The Name property of a control is used to refer to the control in code. The Text property holds the words that the user sees on the screen.
13. C# code is written in methods. Method bodies begin and end with braces { }.
14. Project comments are used for documentation. Good programming practice requires comments in every method and at the top of a file.
15. Most C# statements must be terminated by a semicolon. A statement may appear on multiple lines; the semicolon determines the end of the statement. Comments and some other statements do not end with semicolons.
16. 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.
17. The `this.Close()` method terminates program execution.
18. Each event to which you want to respond requires an event-handling method, also called an *event handler*.
19. You can print out the C# code for documentation.
20. Three types of errors can occur in a C# project: syntax errors, which violate the syntax rules of the C# language; run-time errors, which contain a statement that cannot execute properly; and logic errors, which produce erroneous results.
21. Finding and fixing program errors is called *debugging*.
22. You must have a clean compile before you run the program.
23. Following good naming conventions can help make a project easier to debug.
24. C# Help has very complete descriptions of all project elements and their uses. You can use the *How Do I*, *Contents*, *Index*, *Search*, *Help Favorites*, or context-sensitive Help.

Key Terms

- assignment statement 30
- button 19
- camel casing 52
- class 4
- clean compile 48
- code 6
- comment 30
- context menu 25
- context-sensitive Help 56
- control 3
- debug time 14
- debugging 48
- design time 14
- Document window 12
- event 4
- event handler 29
- event-handling method 29
- exception 46
- Express Edition 5
- form 3
- Form Designer 12
- graphical user interface (GUI) 3
- Help 14
- integrated development environment (IDE) 8
- label 19
- logic error 47
- method 4
- namespace 23
- object 4
- object-oriented programming (OOP) 3
- pascal casing 52
- Professional Edition 5
- project file 8
- Properties window 13
- property 4
- pseudocode 6
- resizing handle 20
- run time 14
- run-time error 46
- snap lines 20
- solution 6
- Solution Explorer window 13
- solution file 7
- Standard Edition 5
- syntax error 45
- Team System 5
- Text property 24
- this 32
- toolbar 11
- toolbox 3
- user interface 6
- Visual Studio environment 8

Review Questions

1. What are objects and properties? How are they related to each other?
2. What are the three steps for planning and creating C# projects? Describe what happens in each step.
3. What is the purpose of these C# file types: .sln, .suo, and .cs?
4. When is C# in design time? run time? debug time?
5. What is the purpose of the Name property of a control?
6. Which property determines what appears on the form for a Label control?
7. What is the purpose of the Text property of a button? the Text property of a form?
8. What does `displayButton_Click` mean? To what does `displayButton` refer? To what does `Click` refer?
9. What is a C# 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. Describe the two types of comments in a C# program and tell where each is generally used.

12. What is meant by the term *debugging*?
13. What is a syntax error, when does it occur, and what might cause it?
14. What is a run-time error, when does it occur, and what might cause it?
15. What is a logic error, when does it occur, and what might cause it?
16. Tell the class of control and the likely purpose of each of these object names:
 - addressLabel
 - exitButton
 - nameTextBox
17. What does context-sensitive Help mean? How can you use it to see the Help page for a button?

Programming Exercises

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

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

“Hello World” in French: Bonjour tout le monde
“Hello World” in Italian: Ciao Mondo
- 1.2 Create a project that displays the hours for each department on campus. Include buttons for Student Learning, Financial Aid, Counseling, and the Bookstore. Each button should display the hours for that department in a label. The interface should have one label for the hours, one label for the programmer name, buttons for each department, and an *Exit* button.

Make sure to use meaningful names for your new buttons, following the naming conventions in Table 1.2. Include comments at the top of every method and at the top of the file.
- 1.3 Write a project that displays four sayings, such as “The early bird gets the worm” or “A penny saved is a penny earned.” (You will want to keep the sayings short, as each must be entered on one line. However, when the saying displays on your form, you can set the label’s properties to allow long lines to wrap within the label.)

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.

If your sayings are too long to display on one line, set the label’s Auto-Size property to False and resize the height of the label to hold multiple lines. You may change the Font properties of the label to the font and size of your choice.

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

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

- 1.4 Write a project to display company contact information. Include buttons and labels for the contact person, department, and phone. When the user clicks on one of the buttons, display the contact information in the corresponding label. Include a button to 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 comments at the top of every method and at the top of the file.

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

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

Also include an *Exit* button.

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

Case Studies

Custom Supplies 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 CS 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

Christopher's Car Center

Christopher's Car 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.

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

Auto Sales

Service Center

Detail Shop

Employment Opportunities

Label Text

Family wagon, immaculate condition \$12,995

Lube, oil, filter \$25.99

Complete detail \$79.95 for most cars

Sales position, contact Mr. Mann
551-2134 x475

Xtreme Cinema

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 Xtreme Cinema.

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 comments at the top of every method and at the top of the file.

Test Data

Button

Comedy

Drama

Action

Sci-Fi

Horror

New Releases

Location

Aisle 1

Aisle 2

Aisle 3

Aisle 4

Aisle 5

Back Wall

Cool Boards

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

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

Include a label that holds your name at the bottom of the form and change the title bar of the form to Cool 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 comments at the top of every method and at the top of the file.

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

