PART 1

Getting Started with Microsoft Visual Basic .NET

CHAPTER

1

Opening and Running a Visual Basic .NET Program

In this chapter you will learn how to:

- ✔ *Start Microsoft Visual Studio .NET.*
- ✔ *Use the Visual Studio development environment.*
- ✔ *Open and run a Visual Basic program.*
- ✔ *Change a property setting.*
- ✔ *Move, resize, dock, and auto hide tool windows.*
- ✔ *Use online Help and exit Visual Studio.*

Microsoft Visual Basic .NET is an important upgrade and enhancement of the popular Visual Basic development system, a product that enjoys an installed base of more than 3 million programmers worldwide. This chapter introduces you to what's new in Visual Basic .NET and gives you the skills you will need to get up and running with the Visual Studio .NET development environment quickly and efficiently. You should read this chapter whether you are new to Visual Basic programming or you have used previous versions of the Visual Basic compiler. The most important advantage of Visual Basic .NET is that it has been designed to make you even more productive in your daily development work—especially if you need to use information in databases or create solutions for the Internet—but an important additional benefit is that once you become comfortable with the development environment in Visual Studio, you can use the same tools to write programs for Microsoft Visual C++ .NET, Microsoft Visual C# .NET, and other third-party tools and compilers.

In this chapter, you'll learn how to start Visual Basic and how to use the integrated development environment to open and run a simple program. You'll learn the

essential Visual Studio menu commands and programming procedures; you'll open and run a simple Visual Basic program named MusicTrivia; you'll change a programming setting called a property; and you'll practice moving, sizing, docking, and auto hiding tool windows. You'll also learn how to get more information by using online Help and how to exit the development environment safely.

Upgrade Notes: What's New in Visual Basic .NET?

Upgrading from Visual Basic 6 to Visual Basic .NET involves a unique set of challenges—I'm very happy with the new features, but I also notice that more than a few familiar tools and controls have really changed. For this reason, I begin each chapter in this book with a sidebar that highlights the changes. Remember that you don't need *any* programming experience to learn Visual Basic .NET using this book. But if you have some Visual Basic 6 knowledge already, I want to give you a short executive summary. So to begin with, here is my list of Visual Basic .NET upgrade notes for this chapter:

- Visual Basic is now fully part of Visual Studio—it shares the Visual Studio development environment with Microsoft Visual C++ .NET, Microsoft Visual C# .NET, and several other programming tools. Although Visual Basic .NET and Visual C++ .NET are still different programming languages, they share the same development environment.
- As part of its new development environment, Visual Studio offers a new Get Started pane, which shows recently used projects and lets you open new or existing source files. Additional links in the Get Started pane provide you with access to Visual Studio Web sites, profile information, and contacts in the Visual Studio development community.
- The Visual Studio development environment contains several new and modified programming tools. The Project window is now called Solution Explorer, and there is a new context-sensitive help window called Dynamic Help. You'll find that the Toolbox has changed quite a bit—it's now subdivided into several functional categories, from Windows Forms to Web Forms to Data.
- Most of the programming tool windows have an auto hide feature to hide the tool as a tab when it isn't needed.

- Projects are now saved in a different way. You give your project a name *before* you create it. The project itself now is spread over several files and folders—even more than in Visual Basic 6. In Visual Basic 6, programs that were made up of multiple projects were called *project groups*; now they are called *solutions*.

The Visual Studio .NET Development Environment

Although the programming language you'll be learning in this book is Visual Basic, the development environment you'll be using to write programs is called the Microsoft Visual Studio .NET development environment. Visual Studio is a powerful and customizable programming workshop that contains all the tools you need to build robust programs for Microsoft Windows quickly and efficiently. Most of the features in Visual Studio apply equally to Visual Basic .NET, Visual C++ .NET, and Visual C# .NET. Use the following procedures to start Visual Studio now.

important

If you haven't yet installed this book's practice files, work through "Finding Your Best Starting Point" and "About the CD-ROM and Practice Files" at the beginning of the book. Then return to this chapter.

Start Visual Studio .NET

Start button

1 On the Windows taskbar, click the Start button, point to Programs, and then point to the Microsoft Visual Studio .NET folder. The icons in the Microsoft Visual Studio .NET folder appear in a list.

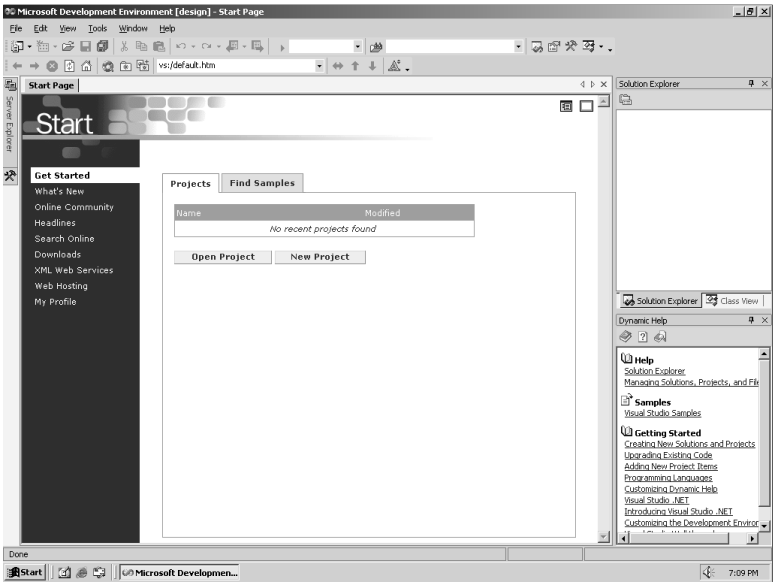
important

To perform the steps in this book, you must have one of the following Visual Studio .NET editions installed: Visual Basic .NET Standard, Visual Studio .NET Professional, Visual Studio .NET Enterprise Developer, or Visual Studio .NET Enterprise Architect. Also—don't try to use this book if you have an earlier version of the Visual Basic software; if that's your situation, you'll be better served by locating an earlier edition of my book, such as *Microsoft Visual Basic Professional 6.0 Step by Step*.

2 Click the Microsoft Visual Studio .NET program icon.

Visual Studio starts, and you see the development environment on the screen with its many menus, tools, and component windows. (These windows are sometimes called *tool windows*.) If this is a new installation of Visual Studio, you should see a Start Page with a set of links. (If you don't see the Start Page, click Show Start Page on the Help menu.)

The first thing Visual Studio .NET displays when you launch it is the Start Page.



3 On the left side of the Start Page window, click the My Profile link.

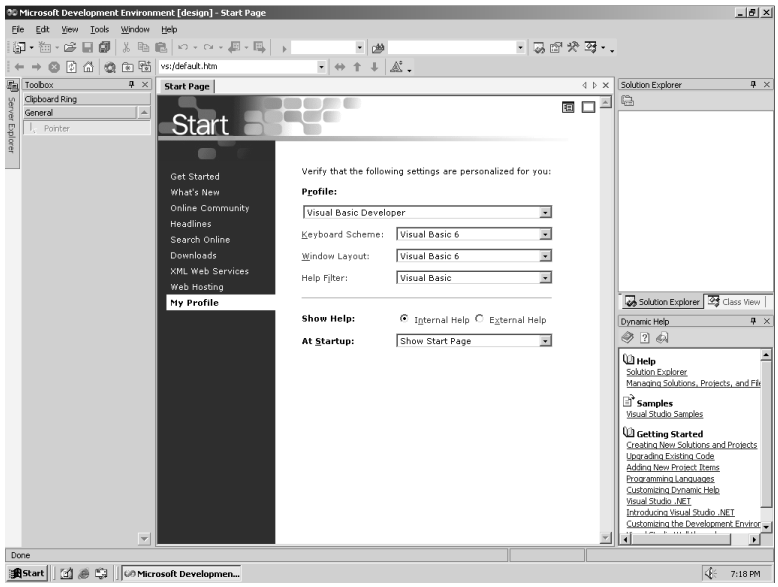
Visual Studio displays the My Profile settings. The My Profile pane allows you to configure Visual Studio with your personal preferences. This is an important screen because Visual Studio can be used to build different types of programs (Visual Basic, Visual C++, and so on), so you want to identify yourself as a Visual Basic developer right away.

4 In the Profile drop-down list box, select Visual Basic Developer from the list of choices.

Visual Studio immediately configures the development environment for Visual Basic programming, displays the Toolbox, and makes adjustments to window characteristics and code formatting styles so that your programs will look like standard Visual Basic solutions.

5 In the At Startup drop-down list box, be sure that Show Start Page is selected.

Setting this option will ensure that the Start Page will appear each time you open Visual Studio.



6 Investigate a few of the remaining settings—and notice that Visual Studio changed a few items on the list when you identified yourself as a Visual Basic programmer.

You can come back to My Profile at any time in the future to fine-tune your profile selections. They will mean more to you after you've written a few programs and have looked at some of the Visual Basic code your friends and coworkers have produced.

7 When you're finished with the profile options, click the Get Started link on the left side of the Start Page window.

Now you see the Get Started page in the development environment, the typical "home" page for Visual Studio users and the place you'll begin when you want to open an existing project or create a new one. The following illustration shows some of the common features of the Get Started pane, including a list of recent Visual Studio projects that have been open (although your list might be empty), a button that you can use to open an existing project, and a button that you can use to create a new project.

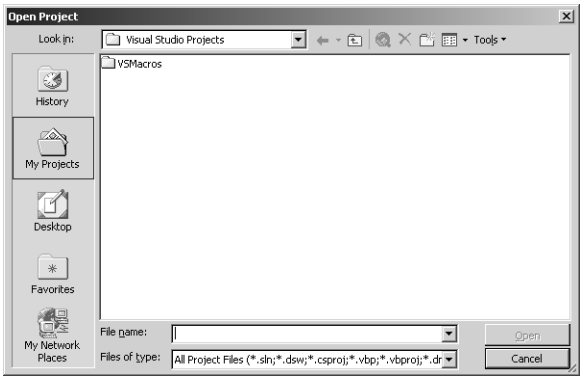
The regular Start Page (or home page) has buttons and links that allow you to quickly open an existing programming solution or create a new one.

The first thing most developers do when they start Visual Studio is open an existing project—either a completed solution they want to work with again or an ongoing development project. Try opening an existing project now that I created for you—the MusicTrivia program.

Open a Visual Basic project

1 Click the Open Project button on the Get Started pane of the Start Page.

The Open Project dialog box appears on the screen with several options. Even if you haven't used Visual Basic before, the Open Project dialog box will seem straightforward, like the familiar Open box in Microsoft Word or Microsoft Excel.



tip

In the Open Project dialog box, you'll see a number of shortcut icons along the left side of the window. The My Projects icon is particularly useful—it opens the Visual Studio Projects folder inside the My Documents folder on your system. By default, Visual Studio saves your projects in the Visual Studio Projects folder and gives each project its own folder.

2 Browse to the folder c:\vbnetsbs on your hard disk.

The folder c:\vbnetsbs is the default location for this book's extensive sample file collection, and you'll find the files there if you followed the setup instructions in "About the CD-ROM and Practice Files" at the beginning of this book. If you didn't install the sample files, do so now using the CD included with this book.

3 Open the chap01\musictrivia folder, and then double-click the MusicTriva project file (MusicTrivia.vbproj).

Visual Studio loads the MusicTrivia form, properties, and program code for the MusicTrivia project. The Start Page probably will still be visible, but Solution Explorer in the upper right corner of the screen will list some of the files in the project.

tip

If you don't see file name extensions in the Open Project dialog box, the hide file extensions option may be turned on. You can change this option in Windows Explorer. On the Tools menu in Windows Explorer, click Folder Options. On the View tab of the Folder Options dialog box, you can uncheck the Hide File Extensions For Known File Types check box.

Projects and Solutions

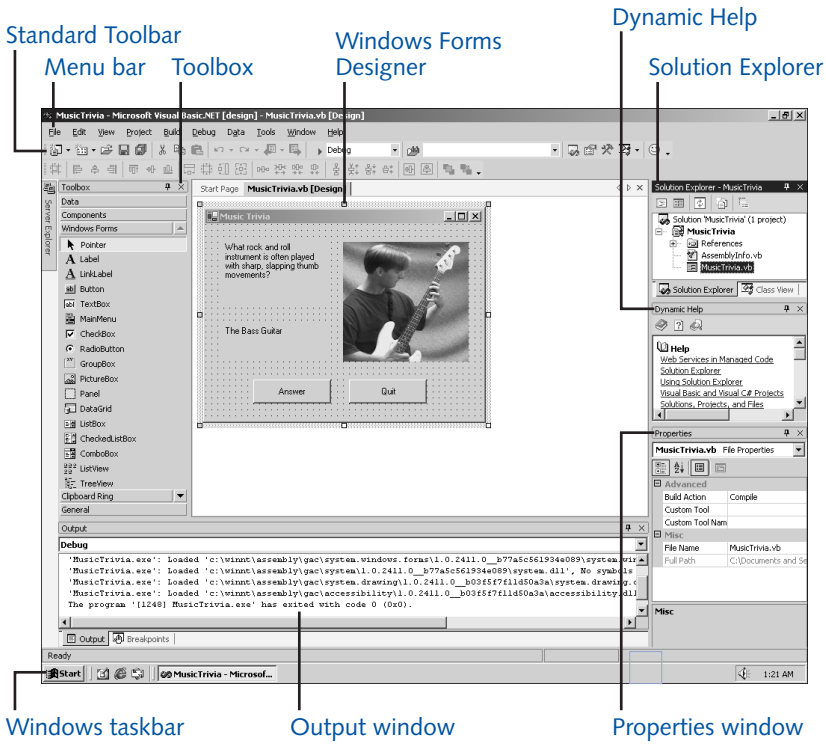
In Visual Studio, programs in development are typically called *projects* or *solutions* because they contain many individual components, not just one file. Visual Basic .NET programs include a project file (.vbproj) and a solution file (.sln). A project file contains information specific to a single programming task. A solution file contains information about one or more projects. Solution files are useful to manage multiple related projects and are similar to project group files (.vbg) in Visual Basic 6. The samples included with this book typically have a single project for each solution, so opening the project file (.vbproj) will have the same effect as opening the solution file (.sln). But for a multiproject solution, you will want to open the solution file.

The Visual Studio .NET Tools

At this point, you should take a few moments to study the development environment and identify some of the programming tools and windows that you'll be using as you complete this course. If you've written Visual Basic programs before, you'll recognize many (but not all) of the development tools. Collectively, these features are the components that you use to construct, organize, and test your Visual Basic programs. A few of the programming tools also help you learn more about the resources on your system, including the larger world of databases and Web site connections available to you. There are also several powerful online Help tools.

The *menu bar* provides access to most of the commands that control the development environment. Menus and commands work as they do in all Windows-based programs, and you can access them by using the keyboard or the mouse. Located below the menu bar is the *Standard toolbar,* a collection of buttons that serve as shortcuts for executing commands and controlling the Visual Studio development environment. If you've used Excel or Word, the toolbar should be a familiar concept. To activate a button on the toolbar, click the button using the mouse. Along the bottom of the screen is the Windows *taskbar*. You can use the taskbar to switch between various Visual Studio .NET components and to activate other Windows-based programs. You may also see taskbar icons for Microsoft Internet Explorer and other programs.

The following illustration shows some of the tools, windows, and other elements in the Visual Studio development environment. Don't worry that this illustration looks different from your current development environment view. You'll learn more about these elements as you step through the chapter.



The main tools that you will see in the Visual Studio development environment are Solution Explorer (formerly called the Project Explorer), the Properties window, Dynamic Help, the Windows Forms Designer, the Toolbox, and the Output window. You may also see more specialized tools such as Server Explorer and Class View, but these are often tabs along the margins of the development environment or at the bottom of tool windows or they aren't visible at all. If a tool isn't visible and you want to see it, click the View menu and select the tool you want to use. The View menu is now pretty full, so Microsoft moved some of the lesser-used tools onto a submenu called (cleverly) Other Windows. Check there if you don't see what you need.

The exact size and shape of the tools and windows depends on how your development environment has been configured. Visual Studio allows you to align and attach, or *dock*, windows to make just the elements that you want visible. You can also hide tools as tabs along the edge of the development environment to move them out of the way until you need them again. Trying to sort out which tools are important to you now and which you can learn about later is a difficult early challenge when you're first learning the busy Visual Studio interface. Your development environment will probably look best if you set your monitor and Windows desktop settings so that they maximize your screen space, but even then things can get a little crowded. (For example, I'm using a desktop property setting of 1024 x 768 for some of the screen shots in this book—an attribute you can change by right-clicking the Windows desktop and clicking Properties.)

What Microsoft is doing with all of this tool complexity is adding many new and useful features to the development environment, while simultaneously providing clever mechanisms for dealing with the clutter. (These mechanisms include features such as docking, auto hiding, and a few other things that I describe later). If you're just starting out with Visual Basic, the best way to resolve this feature tension is to hide the tools that you don't plan to use often and make room for the important ones. The crucial tools for Visual Basic programming—the ones you'll start using right away in this book—are Solution Explorer, Properties window, Windows Forms Designer, Toolbox, Dynamic Help, and the Output window. You won't use the Server Explorer, Class View window, Resource View window, Object Browser, or Debug windows until later in this book.

In the following exercises, you'll start experimenting with the crucial tools in the Visual Studio development environment. You'll also learn to hide the tools you won't use for a while.

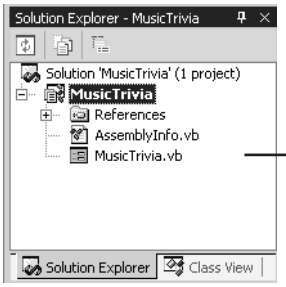
The Windows Forms Designer

If you completed the last exercise, the MusicTrivia project will be loaded in the Visual Studio development environment. However, the user interface, or *form*, for the project may not yet be visible in Visual Studio. (More sophisticated projects might contain several forms, but this simple trivia program needs only one.) To make the form of the MusicTrivia project visible in the development environment, you display it by using Solution Explorer.

Display the Windows Forms Designer

1 Locate the Solution Explorer pane near the upper right corner of the Visual Studio development environment. If you don't see Solution Explorer (if it is hidden as a tab in a location that you cannot see or isn't currently visible), click Solution Explorer on the View menu to display it.

When the MusicTrivia project is loaded, Solution Explorer looks like this:



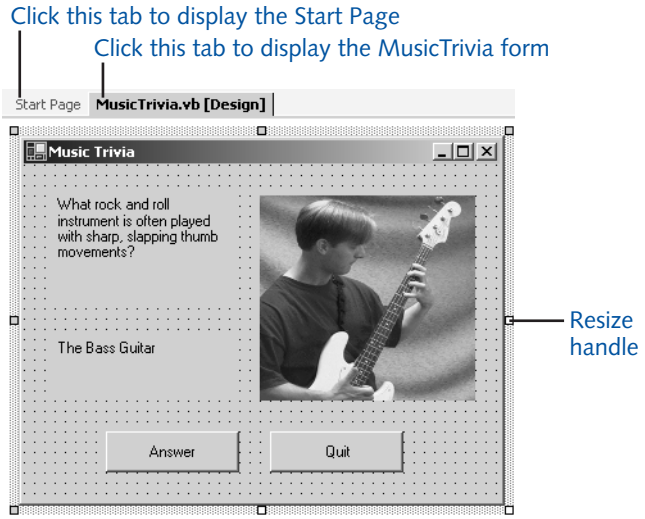
2 Click the MusicTrivia.vb form in the Solution Explorer window.

All form files, including this one, have a tiny form icon next to them so that you can easily identify them. When you click the form file, Visual Studio highlights it in Solution Explorer, and some information about the file appears in the Properties window (if you currently have it visible).

View Designer button

3 Click the View Designer button in Solution Explorer to display the program's user interface.

The MusicTrivia form is displayed in the Windows Forms Designer, as shown here:



Notice that a tab for the Start Page is still visible at the top of the Windows Forms Designer. You can click this tab to display the Start Page, change your profile settings, or open additional project files. To return to Windows Forms Designer view, click the tab labeled "MusicTrivia.vb [Design]" at the top of the MusicTrivia form.

Now try running a Visual Basic program with Visual Studio.

tip

If you don't see the Start Page and MusicTrivia.vb [Design] tabs, your development environment might be in MDI view instead of Tabbed Documents view. To change this option, click Options on the Tools menu. On the left side of the Options dialog box, click the Environment folder and then click General. On the right under Settings, click the Tabbed Documents radio button and then click OK. The next time you start Visual Studio, the various windows that you open will have tabs, and you can switch between them with a simple button click.

Running a Visual Basic Program

MusicTrivia is a simple Visual Basic program designed to get you familiar with the programming tools in Visual Studio. The form you see now has been customized with five objects (two labels, a picture, and two buttons), and I've added three lines of program code to make the trivia program ask a simple question and display the appropriate answer. (The program "gives away" the answer now because it is currently in design mode, but the answer will be hidden when you run the program.) You'll learn more about creating objects and adding program code in the next chapter. For now, try running the program in the Visual Studio development environment.

Run the MusicTrivia program

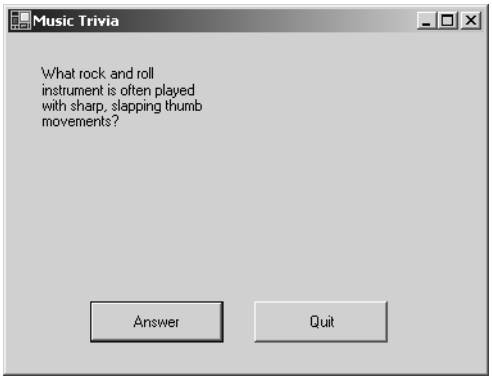
Start button

1 Click the Start button on the Standard toolbar to run the MusicTrivia program in Visual Studio.

tip

You can also press F5 or click the Start command on the Debug menu to run a program in the Visual Studio development environment. Note that the placement of the Start command is different than it is in the Visual Basic 6 compiler.

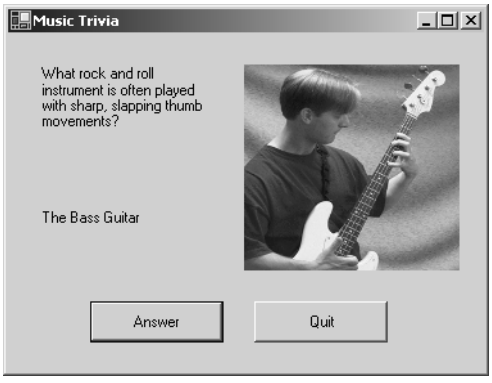
Visual Studio loads and compiles the project into an assembly (a structured collection of modules, data, and manifest information for a program) and then runs the program in the development environment. An icon for the program also appears on the Windows taskbar. During compilation, the Output window documents several of the loading and compiling steps and records any errors that occurred so that you can fix them. After a moment, you'll see the MusicTrivia form again, this time with the photograph and answer label hidden from view:



MusicTrivia now asks its important question: What rock and roll instrument is played with sharp, slapping thumb movements?

2 Click the Answer button to reveal the solution to the question.

When you do, the program displays the answer (Bass Guitar) below the question and then displays a photograph of an obscure Seattle bass player demonstrating the technique. The test program works.



3 Click Quit to close the program.

The form closes, and the Visual Studio development environment becomes active again. Notice that the form looks a little different now in the development environment—a grid of alignment dots is visible on the surface of the form, the two labels are surrounded by some gray space, and resize handles surround the form. These features are visible only when a form is in design mode, and they will help you design and align your user interface. The grid is also a quick giveaway if you're wondering whether a program is running in Visual Studio. (You'll practice using these features in the next chapter.)

The Properties Window

Use the Properties window to change the characteristics, or *property settings*, of the user interface elements on a form. A property setting is a quality of one of the objects in your user interface. For example, the trivia question the MusicTrivia program displays can be modified to appear in a different font or font size or with a different alignment. (With Visual Studio .NET, you can display text in any font installed on your system, just as you would in Excel or Word.) You can change property settings by using the Properties window while you are creating your user interface, or you can add program code via the Code Editor to change one or more property settings while your program is running.

The Properties window contains an Object drop-down list box that itemizes all the user interface elements (objects) on the form; the window also lists the property settings that can be changed for each object. (You can click one of two convenient buttons to view properties alphabetically or by category.) You'll practice changing the Font property of the first label in the MusicTrivia program now.

Change a property

1 Click the Label1 object on the form. (Label1 contains the text "What rock and roll instrument is played with short, slapping thumb movements?")

To work with an object on a form, you must first select the object. When you select an object, resize handles surround it, and the property settings for the object are displayed in the Properties window.

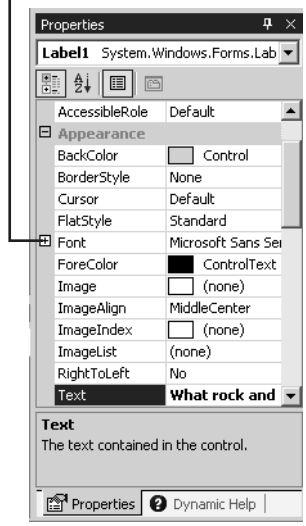
Properties Window button

2 Click the Properties Window button on the Standard toolbar.

The Properties window is another tool that might or might not be visible in Visual Studio, depending on how it has been configured and used on your system. It usually appears below Solution Explorer on the right side of the development environment.

You'll see a window similar to the following:

Font category



The Properties window lists all the property settings for the first label object (Label1) on the form. (In all, 45 properties are available to labels.) Property names are listed in the left column of the window, and the current setting for each property is listed in the right column. Because there are so many properties (and some that are rarely modified), Visual Studio organizes them into categories and displays them in outline view. If a category has a plus sign (+) next to it, you can click the collection title to display all the properties in that category. If a category has a minus sign (–) next to it, the properties are all visible, but you can hide the list under the category name by clicking on the minus sign.

Alphabetic button

Categorized button

tip

The Properties window also has two useful buttons that you can use to further organize properties. The Alphabetic button lists all the properties in alphabetical order and puts them in just a few categories. (Click this button if you know the name of a property and want to locate it quickly.) The Categorized button breaks down the property list into many logical categories. (Click this button if you don't know much about the object you are customizing and would benefit from a more conceptual organization.) If you're new to Visual Studio and Visual Basic, I recommend that you use the Categorized configuration until you get familiar with the most common objects and properties.

3 Scroll in the Properties window list box until the Font property is visible.

The Properties window scrolls like a regular list box.

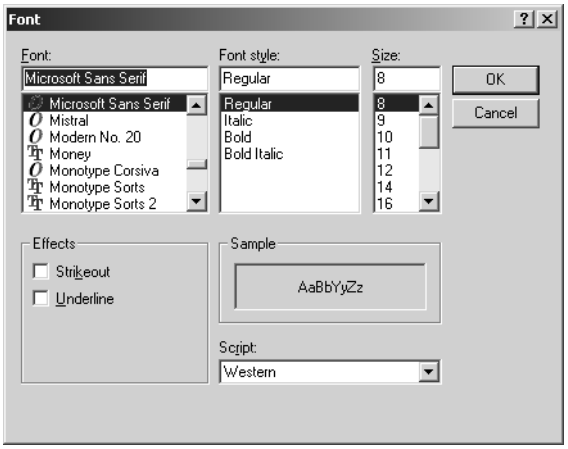
4 Click the Font property name (in the left column).

The current font (Microsoft Sans Serif) is partially displayed in the right column, and a button with three dots on it appears by the font name. This button is called an ellipsis button and means that a dialog box is available to customize the property setting.

Ellipsis button

5 Click the Font ellipsis button in the Properties window.

Visual Studio displays the Font dialog box, which allows you to specify new formatting characteristics for the text in the selected label on your form. The Font dialog box contains more than one formatting option; for each option you select, a different property setting will be modified.



6 Change the size of the font from 8 point to 10 point, and then change the font style from Regular to Italic. Click OK to confirm your selections.

Visual Studio records your selections and adjusts the property settings accordingly. You can examine the changes that Visual Studio made by viewing your form in the Windows Forms Designer and by expanding the Font category in the Properties window.

Now change a property setting for the Label2 object (the label that contains the text "The Bass Guitar").

7 In the Windows Forms Designer, click the second label object (Label2).

When you select the object, resize handles surround it.

8 Click the Font property again in the Properties window.

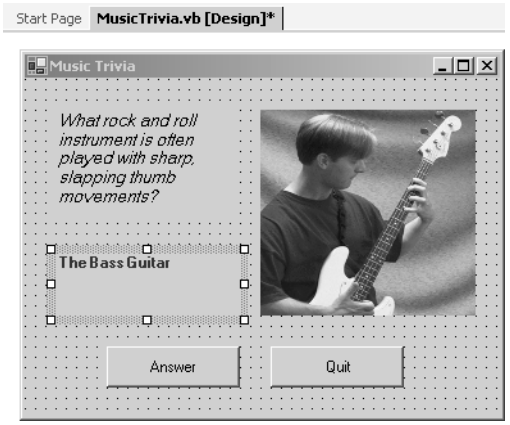
The Label2 object has its own unique set of property settings—although the property names are the same as the Label1 object, the values in the property settings are distinct and allow the Label2 object to act independently on the playing field of the form.

9 Click the Font ellipsis button, set the font style to Bold, and then click OK.

10 Scroll to the ForeColor property in the Properties window, and then click it in the left column.

11 Click the ForeColor drop-down arrow in the right column, and then click a dark purple color on the Custom tab.

The text in the Label2 object should now appear bold and in the color purple on the form.



Congratulations! You've just learned how to set properties in a Visual Basic program using the Visual Studio Properties window—one of the important skills in becoming a Visual Basic programmer.

Thinking About Properties

In Visual Basic, each user interface element in a program (including the form itself) has a set of definable properties. You can set properties at design time by using the Properties window. Properties can also be referenced in code to do meaningful work as the program runs. (User interface elements that receive input often use properties to convey information to the program.) At first, you may find properties a difficult concept to grasp. Viewing them in terms of something from everyday life can help.

(continued)

continued

Consider this bicycle analogy: a bicycle is an object you use to ride from one place to another. Because a bicycle is a physical object, it has several inherent characteristics. It has a brand name, a color, gears, brakes, and wheels, and it's built in a particular style. (It may be a touring bike, a mountain bike, or a bicycle built for two.) In Visual Basic terminology, these characteristics are *properties* of the bicycle object. Most of the bicycle's properties would be defined while the bicycle was being built. But others (tires, travel speed, age, and options such as reflectors and mirrors) could be properties that change as the bicycle is used. As you work with Visual Basic, you'll find object properties of both types.

Moving and Resizing the Programming Tools

With numerous programming tools to contend with on the screen, the Visual Studio development environment can become a pretty busy place. To give you complete control over the shape and size of the elements in the development environment, Visual Studio lets you move, resize, dock, and auto hide most of the interface elements that you use to build programs.

To move one of the tool windows in Visual Studio, simply click the title bar and drag the object to a new location. If you align one window along the edge of another window, it will attach itself, or *dock*, to that window. Dockable windows are advantageous because they always remain visible. (They won't become hidden behind other windows.) If you want to see more of a docked window, simply drag one of its borders to view more content.

If you ever want to completely close a window, click the close button in the upper right corner of the window. You can always open the window again later by clicking the appropriate command on the View menu.

Auto Hide is enabled

If you want an option somewhere between docking and closing a window, you might try "auto hiding" a tool window to the side of the Visual Studio development environment by clicking the tiny Auto Hide pushpin button on the right side of the tool's title bar. This action removes the window from the docked position and places the title of the tool at the edge of the development environment in a tab that is quite unobtrusive. When you perform the auto hide action, you'll notice that the tool window will still be visible as long as you keep the mouse in the area of the window. When you move the mouse to another part of the development environment, the window will slide out of view.

To restore a window that has auto hide enabled, click the tool tab at the edge of the development environment or hold your mouse over the tab. (You can recognize a window that has auto hide enabled because the pushpin in its title bar is pointing sideways.) Holding the mouse over the title allows you to use the tools in what I call "peek-a-boo" mode—in other words, you can quickly display an auto hidden window by clicking its tab, check or set the information you need, and then move the mouse to make the window disappear. If you ever need the tool displayed permanently, click the Auto Hide pushpin button again so that the point of the pushpin faces down, and the window will remain visible.

Auto Hide is disabled

Docking and auto hiding techniques definitely take some practice to master. Use the following exercises to hone your windows management skills and experiment with the features of the Visual Studio development environment along the way. After you complete the exercises here, feel free to configure the Visual Studio tools in a way that seems comfortable for you.

Moving and Resizing Tool Windows in Visual Studio

To move and resize one of the programming tool windows in Visual Studio, follow these steps. This exercise demonstrates how you manipulate the Properties window, but you can move around a different tool window if you want.

Move and resize the Properties window

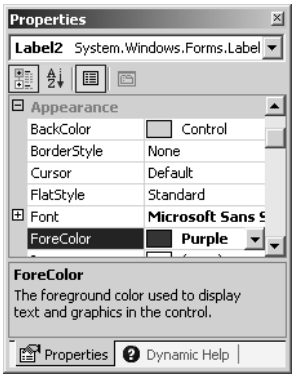
Properties Window button

1. Click the Properties Window button on the Standard toolbar if the Properties window isn't visible in the development environment.

   The Properties window is activated in the development environment, and its title bar is highlighted.

2. Double-click the Properties window title bar to display the window as a floating (nondocked) window.

   You'll see a Properties window that looks like the following:



3 Using the Properties window title bar, drag the window to a new location in the development environment but don't allow it to be docked.

Moving windows around the Visual Studio development environment gives you some flexibility with the tools and your programming environment. Now resize the Properties window to see more of an object's property settings at once.

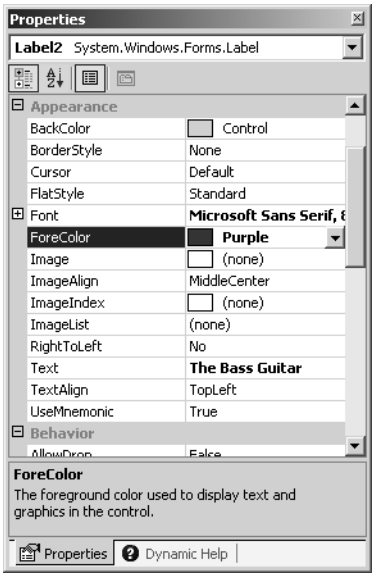
4 Move the mouse to the lower right corner of the Properties window until it becomes a resizing pointer.

Resizing pointer

You resize windows in Visual Studio just as you resize other application windows in the Microsoft Windows operating system.

5 Drag the lower right border of the window down and to the right to enlarge the window.

Your Properties window will now look bigger:



A bigger window lets you work more quickly and with more clarity of purpose. Feel free to move or resize a window when you need to see more of it.

Docking a Tool in Visual Studio

If a tool is floating over the development environment, you can return it to its original docked position by double-clicking the window's title bar. (Notice that this is the same technique that you used in the last exercise to expand a docked window—double-clicking a title bar works like a *toggle*, a state that switches

back and forth between two standard positions.) You can also attach or dock a floating tool in a new place when it's in its floating, expanded position. You might want to do this if you need to make more room in Visual Studio for a particular programming task, such as creating a user interface with the Windows Forms Designer. Try docking the Properties window in a new location now.

Dock the Properties window

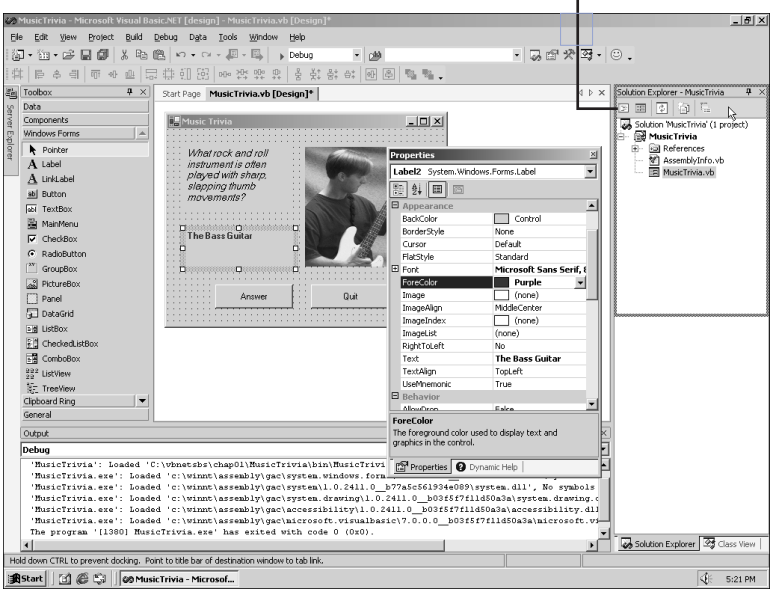
1 Verify that the Properties window (or another tool that you want to move) is floating over the Visual Studio development environment in an undocked position.

   If you completed the last exercise, the Properties window will be in an undocked position now.

2 Drag the title bar of the Properties window to the top, bottom, right, or left edge of the development environment (your choice!) until the border of the window snaps to the window edge you selected.

   This snapping behavior signifies that the window will be docked when you release the mouse button. Note that there are several valid docking locations for tool windows in Visual Studio, so you may want to try two or three different spots until you find one that looks right to you. (A window should be located in a place that is handy but not in the way of other needed tools.)

Docking the Properties window



3 Release the mouse button to dock the Properties window.

The window snaps into place in its new home.

tip

To prevent docking as you drag a window, hold down the Ctrl key as you drag. If you want the window you are dragging to be linked to another window by tabs, drag the window directly onto the title bar of the other window. When windows are linked together in this manner, a tab for each window will appear at the bottom of a shared window, and you can switch back and forth between windows by clicking the tabs. Tabbing windows provides an efficient way to use the space of one window for two or more purposes. (Solution Explorer and the Class View window are often tabbed together, for example.)

4 Try docking the Properties window several more times in different places to get the hang of how docking and tabbing works.

I guarantee that a few of these window procedures will seem confusing at first, but after a while they will become routine for you. In general, you want to create window spaces that have enough room for the information you need to see and use while you work on your more important tasks in the Windows Forms Designer and Code Editor.

Hiding a Tool in Visual Studio

Visual Studio .NET includes a mechanism for hiding and displaying tools quickly, called auto hide. The auto hide feature is available for most tool windows. To hide a tool window, click the Auto Hide pushpin button on the right side of the title bar to conceal the window beneath a tool tab on the edge of the development environment, and click it again to restore the window to its docked position. You can also use the Auto Hide command on the Window menu to enable auto hide for a tool window. Note that the auto hide feature and pushpin button are available only for docked windows—you won't see the Auto Hide command or the pushpin for an active window floating on the top of the development environment.

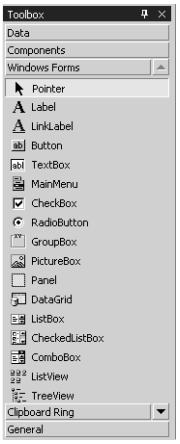
Use the auto hide feature

1 Locate the Toolbox in the development environment (a window that is usually open on the left side of the Windows Forms Designer).

The Toolbox contains many of the controls that you will use to build Visual Basic applications. For example, I used the Label, Button, and PictureBox

controls to create the objects you've seen in the MusicTrivia program. There are several different control collections in the Toolbox, and you can access them by clicking on the tabs that you see within the Toolbox.

If you don't see the Toolbox now, click Toolbox on the View menu. The following illustration shows you what it looks like.



2 Locate the Auto Hide pushpin button on the title bar of the Toolbox.

The pushpin is currently in the "down," or "pushed in," position, meaning that the Toolbox is "pinned" open and auto hide is disabled.

Auto Hide is enabled

3 Click the pushpin button in the Toolbox title bar, and keep the mouse pointer within the Toolbox.

The pushpin button changes direction (it now points to the left), indicating that the Toolbox is no longer pinned open and auto hide is enabled, and a tab appears on the left side of the development environment with the word Toolbox on it. You may also notice that the Windows Forms Designer shifted left. However, if your mouse pointer is still resting on the top of the Toolbox, nothing will have changed in the Toolbox itself—the designers of Visual Studio decided that it would be best if a window with auto hide enabled didn't disappear until you move the mouse to another part of the Visual Studio development environment.

4 Move the mouse away from the Toolbox.

As soon as you move the mouse away, the Toolbox slides off the screen and is hidden beneath the small Toolbox tab. (You may also see a Server Explorer tab above the Toolbox tab—an indication that another tool has auto hide

enabled. Indeed, depending on how Visual Studio is currently configured, you may now notice that there are other windows in the development environment with auto hide enabled.)

The benefit of enabling auto hide for windows is that they free up considerable work area in Visual Studio but are also quickly accessible.

5 Hold the mouse pointer over the Toolbox tab. (You can also click the Toolbox tab if you wish.)

The Toolbox immediately slides back into view, and you can begin using Toolbox controls to build your user interface. (We'll do this in Chapter 2.)

6 Move the mouse away from the Toolbox, and the tool disappears again.

Auto Hide is disabled

7 Finally, display the Toolbox again and then click the pushpin button on the Toolbox title bar.

The Toolbox returns to its familiar docked position, and you can use it without worrying about it sliding away.

Spend some time moving, resizing, docking, and auto hiding tool windows in Visual Studio now, to create your version of the perfect work environment. As you work through this book, you'll want to adjust your window settings periodically to adapt your work area to the new tools you're using. When the need arises, come back to this section and practice your skills again.

Getting Help

Visual Studio .NET includes an online reference that you can use to learn more about the Visual Studio development environment, the Visual Basic programming language, the resources in the .NET Framework, and the remaining tools in the Visual Studio suite. Take a moment to explore your Help resources before moving to the next chapter, where you will build your first program.

tip

Visual Studio .NET online Help is provided by your Visual Studio .NET CDs. If you have plenty of disk space, you can install all the Visual Studio .NET documentation on your system from these CDs during Setup.

You can access Help information in several ways.

To get Help information	Do this
About the task you're currently working on	Click the Dynamic Help tab in the development environment to see a list of Help topics related to the features you're using, or on the Visual Studio Help menu, click Dynamic Help.
By topic or activity	On the Visual Studio Help menu, click Contents.
While working in the Code Editor	Click the keyword or program statement you're interested in and then press F1.
While working in a dialog box	Click the Help button in the dialog box.
By searching for a specific keyword	On the Help menu, click Search and type the term you're looking for.
In a window separate from Visual Studio	On the Windows taskbar, click the Start button, point to Programs, point to Microsoft Visual Studio .NET, and then click Microsoft Visual Studio .NET Documentation. Use the Contents, Index, and Search tabs to find information.
From Visual Studio Web sites and newsgroups	On the Visual Studio Start Page, click Online Community and then the Web site or newsgroup you're interested in.
About contacting Microsoft for product support	On the Help menu, click Technical Support.

In this section, you'll learn how to use the new Dynamic Help feature in Visual Studio .NET. The goal of this tool is to anticipate the questions you'll ask based on the current context of your work in Visual Studio. I believe you'll find this searching mechanism vastly superior to the general search tools provided with Visual Basic 6 because Dynamic Help uses contextual logic to limit the material you see to the particular compiler or tool that you are using. (In other words, Dynamic Help doesn't bring up Visual C++ or Visual C# topics unless you are working with those tools.)

In this section, you'll also learn how to perform a full-text search of the Visual Studio Help system. Full-text search can be helpful when you want to search for specific keywords.

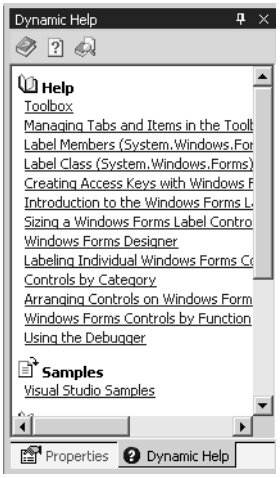
Because you just completed an exercise on auto hide with the Toolbox in Visual Studio, it is likely that the Dynamic Help system has been gathering information for you about the Toolbox and other recent commands you've issued in Visual Studio. Let's open Dynamic Help now and see.

Get help using Dynamic Help

1 Click the Dynamic Help tab in the development environment, or click Dynamic Help on the Visual Studio Help menu.

The Dynamic Help window appears, as shown here:

The Help menu is your door to the Visual Studio .NET Help system.



The Dynamic Help window is an integrated part of Visual Studio. The window can be moved, resized, docked, or auto hidden to suit your needs, and you can leave it open all the time or open it only when you need it.

2 Click a topic in the Dynamic Help window. It could be about the Toolbox or the Properties window.

The MusicTrivia form is hidden under the MusicTrivia.vb [Design] tab, and the Help topic appears in the main window of the development environment.

tip

If you didn't install the documentation on your system during Setup, the Dynamic Help window won't display any topics. To install the documentation, run the Visual Studio .NET Setup again and select the Documentation item.

3 Browse a few of the other Help topics in the Dynamic Help listing, and see how close these picks are to the actual activities you have been working on.

tip

You can adjust how Dynamic Help comes up with Help information by clicking the Options command on the Tools menu, opening the Environment folder, clicking the Dynamic Help topic, and identifying which topics you want Help to list and how many links there should be. These settings will help you further manage the amount of information Help offers you.

Do a full-text search in Help

1. On the Visual Studio Help menu, click Search.

   The Search window appears in the development environment.

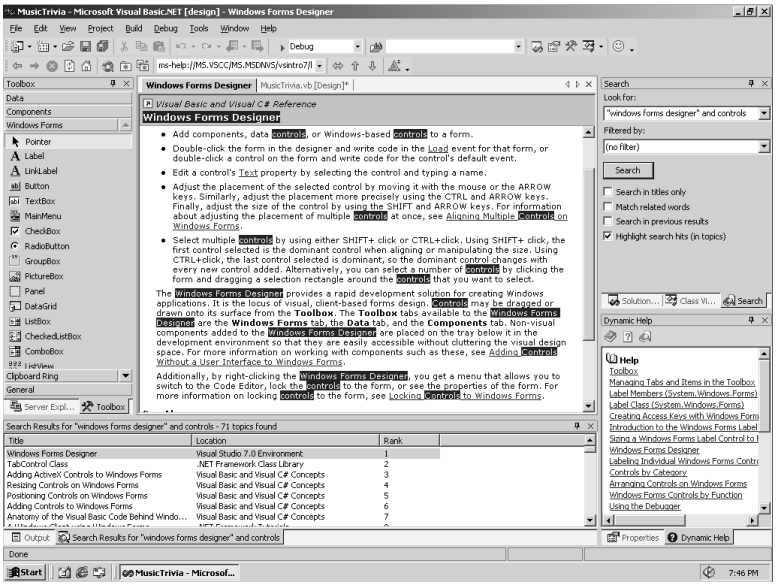
2. In the Look For drop-down list box, type **"windows forms designer" and controls** (include the quotes because the name contains spaces).

3. Check the Highlight Search Hits (In Topics) check box, and then click the Search button.

   The Search Results window appears with many topics that match your search criteria.

4. In the Search Results window, double-click a topic to see its contents.

   The topic appears in the main window, and the text that you searched for is highlighted as shown here:



tip

The Help Contents and Index windows are also integrated into the Visual Studio development environment. You can open these help windows by clicking the Contents or Index command on the Help menu. Keep in mind that you don't have to use the Help features one at a time. (See the Help Information table earlier in this section for other Help suggestions.)

One Step Further: Exiting Visual Studio .NET

Each chapter in this book concludes with a section entitled "One Step Further" that enables you to practice an additional skill related to the topic at hand. After the "One Step Further" tutorial, I've compiled a Quick Reference table that reprises the important concepts discussed in each chapter.

When you're finished using Visual Studio for the day, save any projects that are open and close the development environment. Give it a try.

Exit Visual Studio

Save All button

1. Save any changes you have made to your program by clicking the Save All button on the Standard toolbar.

   In contrast with Visual Basic 6, in Visual Studio .NET you give your program a name when you begin the project, not when you're ready to save it, so you won't need to provide any filename information now. As you'll learn, there are also many more folders for projects in Visual Studio .NET than there were in Visual Basic 6. Each project fits in a folder of its own, and several subfolders are created below the main project folder to hold files as the program is built and compiled.

2. On the File menu, click the Exit command.

   The Visual Studio .NET program exits. Time to move on to your first program in Chapter 2!

Chapter 1 Quick Reference

To	Do this
Start Visual Studio .NET	Click the Start button on the taskbar, point to Programs, point to the Microsoft Visual Studio .NET folder, and then click the Microsoft Visual Studio .NET program icon.
Open an existing project	Start Visual Studio .NET. Click the File menu, point to Open, and then click Project. *or* On the Start Page, click Open Project.
Run a program	Click the Start button on the Standard toolbar. *or* Press F5.
Set properties	Click the object on the form containing the properties you want to set, and then click the Properties Window button on the Standard toolbar to display the Properties window (if it isn't open).
Display and resize a tool window	Double-click the window's title bar to display it as a floating window. Resize the window by dragging the edges of the window.
Move a tool window	Double-click the window's title bar to display it as a floating window, and then drag the title bar.
Dock a tool window	Double-click the title bar. *or* Drag the tool to the edge of another tool until it snaps into place.
Enable auto hide for a tool window	Click the Auto Hide pushpin button on the right side of the tool's title bar. The tool window hides behind a small tab at the edge of the development environment until you hold the mouse over it.
Disable auto hide for a tool window	Click the tool tab, and then click the Auto Hide pushpin button again.
Quit Visual Studio .NET	On the File menu, click Exit.

Auto Hide enabled

Auto Hide disabled

CHAPTER

2

Writing Your First Program

In this chapter you will learn how to:

- ✔ *Create the user interface for a new program.*
- ✔ *Set the properties for each object in your user interface.*
- ✔ *Write program code.*
- ✔ *Save and run the program.*
- ✔ *Build an executable file.*

As you learned in Chapter 1, the Microsoft Visual Studio .NET environment contains several powerful tools to help you run and manage your programs. Visual Studio also contains everything you need to build your own applications for Windows from the ground up. In this chapter, you'll learn how to create a simple but attractive user interface with the controls in the Visual Studio Toolbox. Next you'll learn how to customize the operation of these controls with special characteristics called property settings. Then you'll see how to identify just what your program should do by writing program code. Finally, you'll learn how to save and run your new program (a Las Vegas–style slot machine) and how to compile it as an executable file.

Upgrade Notes: What's New in Visual Basic .NET?

If you're experienced with Visual Basic 6, you'll notice some new features in Visual Basic .NET, including the following:

- The Visual Studio .NET development environment provides a few different menus and toolbars with which you can build your programs. For example, Visual Basic 6 included Format, Run, and Add-Ins menus, which aren't included in Visual Studio. Most of the commands have been relocated—you'll find many of the Run menu commands on the Debug menu.
- The CommandButton control is named the Button control in Visual Studio .NET, and many of its properties and methods have changed. For example, the Caption property is now named the Text property.
- Some of the properties and methods for the Label control are new or have changed. For example, the Caption property is now named the Text property, and the TextAlign property has more alignment options than the previous Alignment property.
- The Image control has been removed from Visual Studio. To display pictures, use the PictureBox control.
- Visual Basic .NET code contains more compiler-generated statements than you saw in Visual Basic 6. In particular, Visual Basic .NET adds to the top of each form a block of code labeled "Windows Forms Designer generated code," which defines important form characteristics and shouldn't be modified. (Add your own program code below this code block.)
- Visual Studio can create two types of executable files for your project, a *debug build* and a *release build*. Debug builds contain debugging information and are used when testing and debugging your program. Release builds are optimized and smaller and are used when you complete your program.

Lucky Seven: Your First Visual Basic Program

The Windows-based application you're going to construct is Lucky Seven, a game program that simulates a lucky number slot machine. Lucky Seven has a simple user interface and can be created and compiled in just a few minutes using Visual Basic. (If you'd like to run a completed version of Lucky before you start, you can find it in the c:\vbnetsbs\chap02\lucky7 folder on your hard disk.) Here's what your program will look like when it's finished:

Programming Steps

The Lucky Seven user interface contains two buttons, three lucky number boxes, a digital photo depicting your winnings, and the label "Lucky Seven." I produced these elements by creating seven objects on the Lucky Seven form and then changing several properties for each object. After the interface was designed, I added program code for the Spin and End buttons to process the user's button clicks and produce the random numbers. To re-create Lucky Seven, you'll follow three essential programming steps in Visual Basic: create the user interface, set the properties, and write the program code. The following table summarizes the process for Lucky Seven.

Programming step	Number of items
1. Create the user interface.	7 objects
2. Set the properties.	12 properties
3. Write the program code.	2 objects

Creating the User Interface

In this exercise, you'll start building Lucky Seven by creating a new project and then using controls in the Toolbox to construct the user interface.

Create a new project

1 Start Visual Studio.

2 On the Visual Studio Start Page, click the Get Started link and then click the New Project button.

> **tip**
> You can also start a new programming project by clicking the File menu, pointing to New, and then clicking Project.

The New Project dialog box appears.

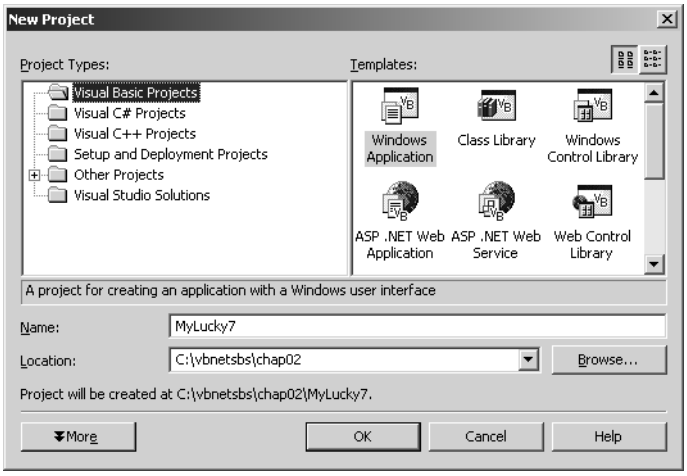
Visual Studio can create operating system components and Web applications, in addition to the standard Windows application that you'll create in this chapter. The New Project dialog box lets you specify the type of program or component that you want to create, the language you'll use to create the program or component, and the name and location you'll use for the files.

3 In the list of Project Types, verify that the Visual Basic Projects folder is selected, and then, in the list of Templates, click the Windows Application icon.

Once these project settings are applied, Visual Studio will set up the development environment for Visual Basic .NET Windows application programming.

4 In the Name text box, type **MyLucky7** and then specify the c:\vbnetsbs\chap02 folder in the Location text box. (If you'd like, click the Browse button to specify the project location.)

Visual Studio assigns the name MyLucky7 to your program and prepares to create a new folder on your hard disk for the project named MyLucky7.



tip

If your copy of Visual Basic .NET didn't come with some of the other tools in the Visual Studio .NET development suite (such as Visual C++ .NET), you'll see a few differences when you work through the procedures in this book. For example, you might not have all the project types and templates shown in the previous illustration. I wrote this book with an installation of the Professional Edition of the Visual Studio .NET suite, which includes Visual C++ .NET, Visual C# .NET, and other development tools. It isn't necessary that you have these extra tools—I won't be using Visual C++ .NET or Visual C# .NET in this book—but I wanted you to see an installation that would be typical in the workplace of a professional software developer.

5 Click OK to create the new project in Visual Studio.

Visual Studio cleans the slate for a new programming project and displays in the center of the screen a blank Windows form (typically called simply "form") you can use to build your user interface.

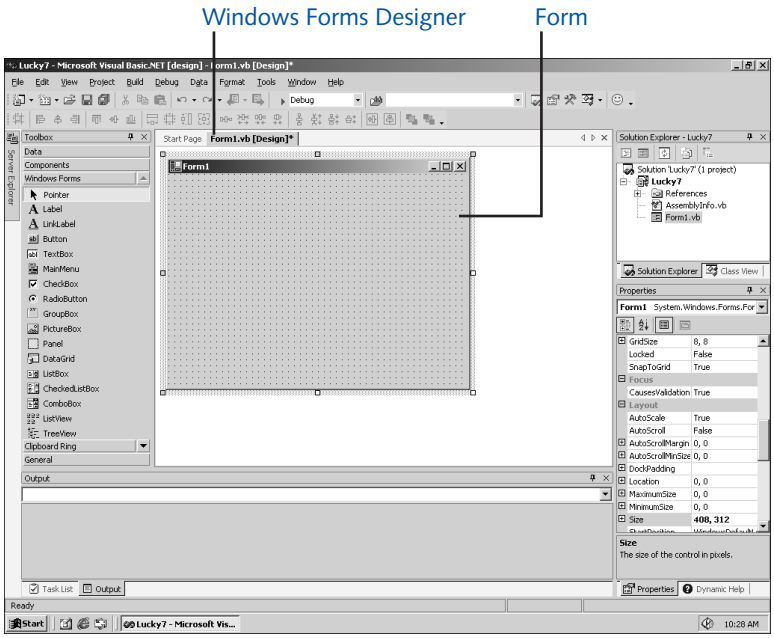
Now you'll enlarge the form, and then you'll create the two buttons in the interface.

Create the user interface

1 Position the mouse pointer over the lower right corner of the form until the mouse changes into a resizing pointer, and then drag to increase the size of the form to make room for the objects in your program.

As you resize the form, scroll bars may appear in the Windows Forms Designer to give you access to the entire form you're creating. Depending on your screen resolution and the Visual Studio tools you have open, there may not be enough room for you to see the entire form at once. Don't worry about this—your form can be small, or it can fill the entire screen, and the scroll bars will give you access to the entire form.

Size your form so that it is about the size of the form shown here:



To see the entire form without obstruction, you can resize or close the other programming tools as you learned in Chapter 1. (Return to Chapter 1 if you have questions about resizing windows or tools.)

Now you'll practice adding a button object on the form.

2 Click the Button control in the Toolbox.

Button control

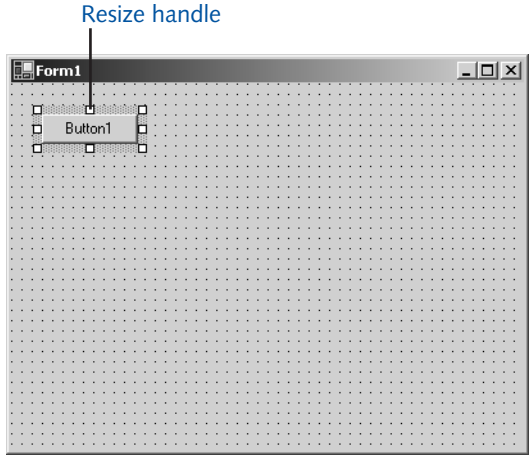
3 Place the mouse pointer over the form.

The mouse pointer changes to crosshairs and a button icon. The crosshairs are designed to help you draw the rectangular shape of a button. When you hold down the left mouse button and drag, the button object takes shape and snaps to the grid formed by the intersection of dots on the form.

Try creating your first button now.

4 Move the mouse pointer close to the upper left corner of the form, hold down the left mouse button, and then drag down and to the right. Stop dragging, and release the mouse button when you've drawn a button similar to the one shown here:

The name of the button is Button1.



A button with resize handles appears on the form. The button is named Button1, the first button in the program. (You might make a mental note of this button name—you'll see it again later when you write your program code.)

You can move buttons by dragging them with the mouse and you can resize them by using the resize handles whenever Visual Basic is in *design mode* (whenever the Visual Studio programming environment is active). When a program is running, however, the user won't be able to move interface elements unless you have changed a special property in the program to allow this. You'll practice moving and resizing the button now.

Move and resize a button

1 Position the mouse pointer over the button so that it changes to a four-headed arrow, and then drag the button down and to the right.

The button snaps to the grid when you release the mouse button. The form grid is designed to help you edit and align different user interface elements. You can change the size of the grid by clicking the Options command on the Tools menu, clicking the Windows Forms Designer folder, and then modifying the GridSize property.

The grid helps you design your user interface.

2 Position the mouse pointer on the lower right corner of the button.

When the mouse pointer rests on a resize handle of a selected object, it changes into a resizing pointer. You can use the resizing pointer to change the size of an object.

3 Enlarge the button by holding down the left mouse button and dragging the pointer down and to the right.

When you release the mouse button, the button changes size and snaps to the grid.

4 Use the resizing pointer to return the button to its original size, and then move the button back to its original location on the form.

Now you'll add a second button to the form, below the first button.

Add a second button

Button control

1 Click the Button control in the Toolbox.

2 Draw a button below the first button on the form. (For consistency, create a button of the same size.)

> **tip**
> To quickly add a control to your form, double-click the control in the Toolbox and a default-size control will be added to your form.

You can delete an object by selecting the object on the form and then pressing Delete.

3 Move or resize the button as necessary after you place it. If you make a mistake, feel free to delete the button and start over.

Now you'll add the labels used to display the numbers in the program. A *label* is a special user interface element designed to display text, numbers, or symbols when a program runs. When the user clicks the Lucky Seven program's Spin button, three random numbers appear in the label boxes. If one of the numbers is a 7, the user hits the jackpot.

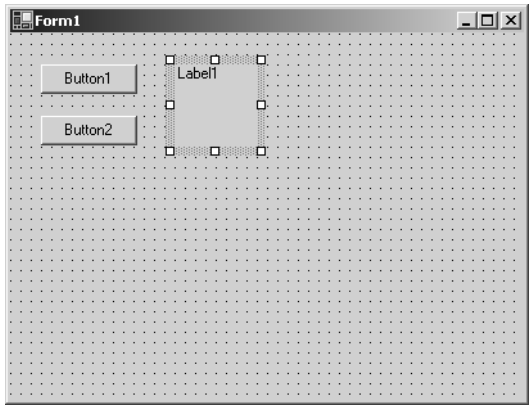
Add the number labels

Label control

1. Click the Label control in the Toolbox.
2. Place the mouse pointer over the form.

   The mouse pointer changes to crosshairs and a letter *A* icon.
3. Draw a small rectangular box like the one shown in the following illustration.

   The label object you have created is named Label1, the first label in the program. Now you'll create two more labels, named Label2 and Label3, on the form.



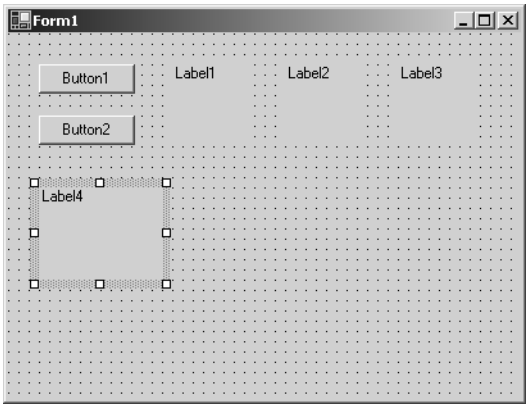
4. Click the Label control in the Toolbox, and then draw a label box to the right of the first label.

   Make this label the same size as the first. The text "Label2" will appear in the label.
5. Click the Label control again, and add a third label to the form, to the right of the second label.

   The text "Label3" appears in the label.

   Now you'll use the Label control to add a descriptive label to your form. This will be the fourth and final label in the program.
6. Click the Label control in the Toolbox.
7. Draw a larger rectangle directly below the two buttons.

When you've finished, your four labels should look like those in the illustration on the following page. (You can resize the label objects if they don't look quite right.)



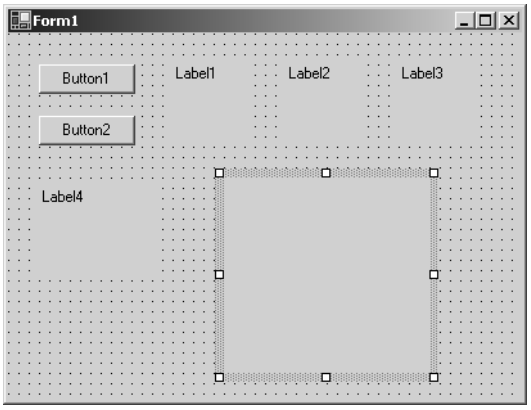
Now you'll add a *picture box* to the form to graphically display the payout you'll receive when you draw a 7 and hit the jackpot. A picture box is designed to display bitmaps, icons, digital photos, and other artwork in a program. One of the best uses for a picture box is to display a JPEG image file.

Add a picture

PictureBox control

1. Click the PictureBox control in the Toolbox.
2. Using the PictureBox control, draw a large rectangular box directly beneath the three number labels.

   When you've finished, your picture box object should look like this:



   This object will be named PictureBox1 in your program; you'll use this name later in the program code.

Now you're ready to customize your interface by setting a few properties.

Setting the Properties

As you discovered in Chapter 1, you can change properties by selecting objects on the form and changing their settings in the Properties window. You'll start by changing the property settings for the two buttons.

Set the button properties

1 Click the first button (Button1) on the form.

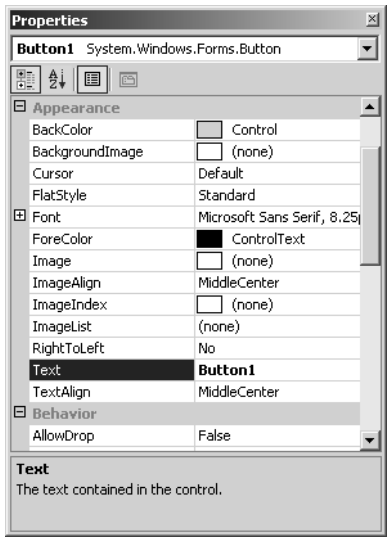
The button is selected and is surrounded by resize handles.

2 Double-click the Properties window title bar.

> **tip**
> If the Properties window isn't visible, click the Properties Window command on the View menu or press F4.

3 Resize the Properties window so that there is plenty of room to see the property names and their current settings.

Once you get used to setting properties, it's OK to use the Properties window without enlarging it, but making it bigger helps when you first try it out. The Properties window in the following illustration is a good size for setting properties:



The Properties window lists the settings for the first button. These include settings for the background color, text, font height, and width of the button. Because there are so many properties, Visual Studio organizes them into categories and displays them in outline view. If you want to see the properties in a category, click the plus sign (+) next to the category title.

4 Scroll in the Properties window until you see the Text property, located in the Appearance category.

5 Double-click the Text property in the left column of the Properties window.

The current Text setting ("Button1") is highlighted in the Properties window.

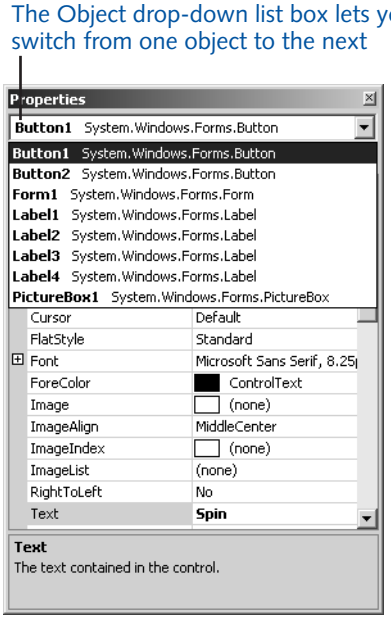
6 Type **Spin,** and press Enter.

The Text property changes to "Spin" in the Properties window and on the button on the form. Now you'll change the Text property of the second button to "End". (You'll select the second button in a new way this time.)

7 Open the Object drop-down list box at the top of the Properties window.

A list of the interface objects in your program appears:

The Object drop-down list box lets you switch from one object to the next.



8 Click Button2 System.Windows.Forms.Button (the second button) in the list box.

The property settings for the second button appear in the Properties window, and Visual Studio highlights Button2 on the form.

9 Double-click the current Text property ("Button2"), type **End**, and then press Enter.

The text of the second button changes to "End".

> **tip**
> Using the Object drop-down list is a handy way to switch between objects in your program. You can also switch between objects on the form by clicking each object.

Now you'll set the properties for the labels in the program. The first three labels will hold the random numbers generated by the program and will have identical property settings. (You'll set most of them as a group.) The descriptive label settings will be slightly different.

Set the number label properties

To select more than one object on a form, hold down the Shift key while clicking the objects.

1 Click the first number label (Label1), and then, holding down the Shift key, click the second and third number labels. (If the Properties window is in the way, move it to a new place.)

A selection rectangle and resize handles appear around each label you click. When you've selected all three labels, release the Shift key. You'll change the TextAlign, BorderStyle, and Font properties now so that the numbers that appear in the labels will be centered, boxed, and identical in font and point size. (Each is located in the Appearance category of the Properties window.)

> **tip**
> When more than one object is selected, only those properties that can be changed as a group are displayed in the Properties window.

2 Click the TextAlign property in the Properties window, and then click the drop-down arrow that appears to the right.

A graphical assortment of alignment options appears in the list box; these settings let you align text anywhere within the borders of the label object.

3 Click the center option (MiddleCenter).

The TextAlign property for each of the selected labels changes to MiddleCenter.

Now you'll change the BorderStyle property.

4 Click the BorderStyle property, and then click the drop-down arrow that appears to the right.

A list of the valid property settings (None, FixedSingle, and Fixed3D) appears in the list box.

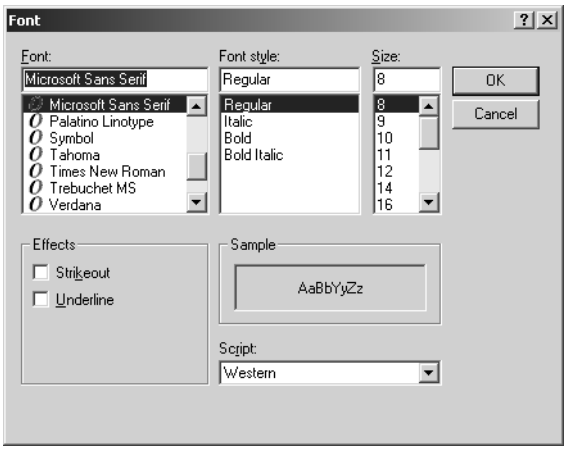
5 Click FixedSingle in the list box to add a thin border around each label.

Now you'll change the font for the labels by changing settings for the Font property.

Ellipsis button

6 Click the Font property in the Properties window, and then click the ellipsis button (the button with three dots that is located next to the current font setting).

The Font dialog box appears, as shown here:



7 Change the font to Times New Roman, the font style to Bold, and the point size to 24, and then click OK.

The label text appears in the font, style, and size you specified.

Now you'll delete the text for the three labels so that the boxes will be empty when the program starts. (Your font selections will remain with the labels because they're stored as separate properties.) To complete this operation, you'll first need to select each of the labels individually.

8 Click a blank area on the form to remove the selection from the three labels, and then click the first label.

9. Double-click the Text property in the Properties window, press the Delete key, and then press Enter.

   The text of the Label1 object is deleted. You'll use program code to put a random "slot machine" number in this property later in this chapter.

10. Delete the text in the second and third labels on the form.

You've finished with the first three labels. Now you'll change the Text, Font, and ForeColor properties of the fourth label.

Set the descriptive label properties

1. Click the fourth label object (Label4) on the form.
2. Change the Text property in the Properties window to **Lucky Seven**.
3. Click the Font property, and then click the ellipsis button.
4. Use the Font dialog box to change the font to Arial, the font style to Bold, and the point size to 18. Click OK.

   The font in the label box is updated. Notice that the text in the box wrapped to two lines. This is an important concept: the contents of an object must fit inside the object. If they don't, the contents will wrap or be truncated.

   Now you'll change the foreground color of the text.
5. Click the ForeColor property in the Properties window, and then click the drop-down arrow in the second column.

   Visual Studio displays a list box with Custom, Web, and System tabs for setting foreground colors (the color of your text) in the label object. The Custom tab contains many of the colors available in your system. The Web tab sets colors for Web pages and lets you pick colors using their common names. The System tab displays the current colors used for user interface elements in your system. (The list reflects the current settings on the Appearance tab of the Display Properties dialog box.)
6. Click the dark purple color on the Custom tab.

   The text in the label box changes to dark purple.

Now you're ready to set the properties for the last object.

The Picture Box Properties

The picture box object will contain a picture of a person paying you money when you hit the jackpot (that is, when at least one seven appears in the number labels on the form). This picture is a digitized image from an unpublished fourteenth century German manuscript stored in JPEG format. You need to set the SizeMode property to accurately size the picture and set the Image property to

specify the name of the JPEG file that you will load into the picture box. You also need to set the Visible property, which specifies the picture state at the beginning of the program.

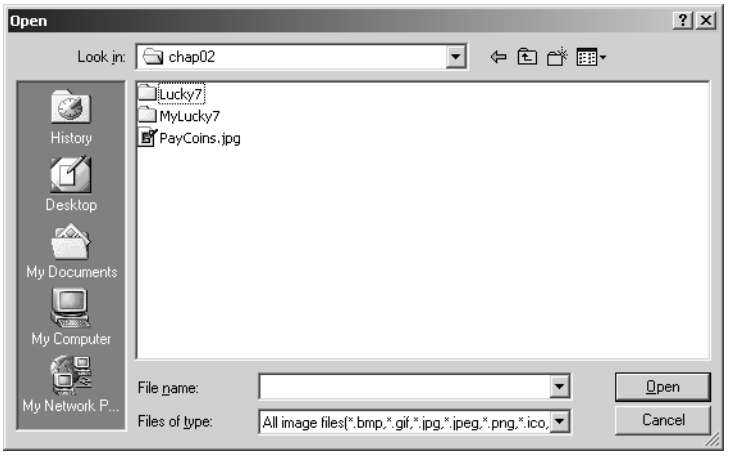
Set the picture box properties

1. Click the picture box object on the form.
2. Click the SizeMode property in the Properties window (listed in the Behavior category), click the drop-down arrow, and then click StretchImage.

   Setting SizeMode to StretchImage before you open a graphic causes Visual Studio to resize the graphic to the exact dimensions of the picture box. (Typically, you set this property before you set the Image property.)
3. Double-click the Image property in the Properties window, and then click the ellipsis button in the second column.

   The Open dialog box appears.
4. In the dialog box, navigate to the c:\vbnetsbs\chap02 folder.

   The digital photo PayCoins.jpg appears in the list.



5. Select the file PayCoins.jpg in the dialog box, and then click Open.

   The PayCoins photo is loaded into the picture box on the form. Because the photo is relatively small (24KB), it opens quickly on the form.

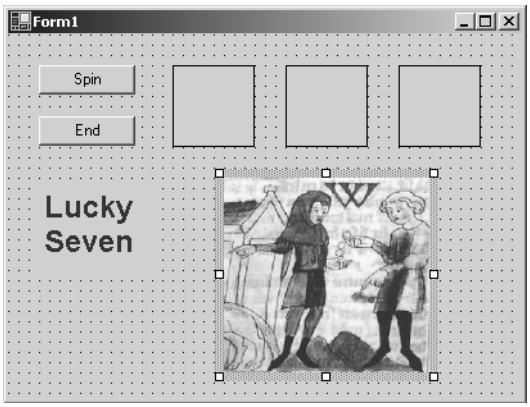
   Now you'll change the Visible property to False so that the image will be invisible when the program starts. (You'll use program code to make it visible later.)

6 Click the Visible property in the Behavior category of the Properties window. Click the Visible drop-down arrow.

The valid settings for the Visible property appear in a list box.

7 Click False to make the picture invisible when the program starts.

8 The Visible property is set to False. This affects the picture box when the program runs, but not now while you are designing it. Your completed form should look similar to this:



9 Double-click the title bar of the Properties window to return it to the docked position.

Writing the Code

Now you're ready to write the code for the Lucky Seven program. Because most of the objects you've created already "know" how to work when the program runs, they're ready to receive input from the user and process it automatically. The inherent functionality of objects is one of the great strengths of Visual Basic—once objects are placed on a form and their properties are set, they're ready to run without any additional programming. However, the "meat" of the Lucky Seven game—the code that actually calculates random numbers, displays them in boxes, and detects a jackpot—is still missing from the program. This computing logic can be built into the application only by using program statements—code that clearly spells out what the program should do each step of the way. Because the program is driven by the Spin and End buttons, you'll associate the code for the game with those buttons. You enter and edit Visual Basic program statements in the *Code Editor*.

Program code is entered in the Code Editor.

Reading Properties in Tables

In this chapter, you've set the properties for the Lucky Seven program step by step. In future chapters, the instructions to set properties will be presented in table format unless a setting is especially tricky. Here are the properties you've set so far in the Lucky Seven program in table format, as they'd look later in the book.

Object	Property	Setting
Button1	Text	"Spin"
Button2	Text	"End"
Label1, Label2, Label3	BorderStyle	Fixed Single
	Font	Times New Roman, Bold, 24-point
	Text	(empty)
	TextAlign	MiddleCenter
Label4	Text	"Lucky Seven"
	Font	Arial, Bold, 18-point
	ForeColor	Purple
PictureBox1	Image	"c:\vbnetsbs\chap02\paycoins.jpg"
	SizeMode	StretchImage
	Visible	False

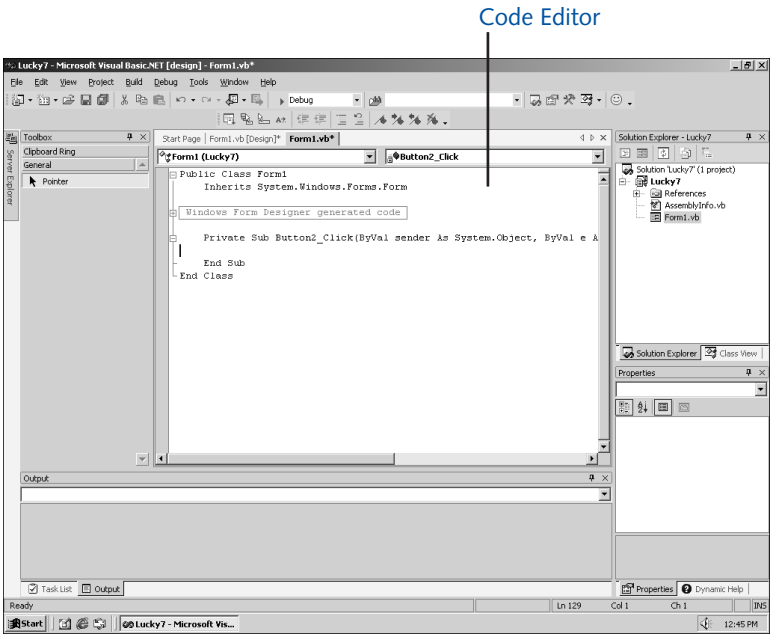
In the following steps, you'll enter the program code for Lucky Seven in the Code Editor.

Use the Code Editor

1 Double-click the End button on the form.

After a few moments, the Code Editor appears in the center of the Visual Studio development environment, as shown on the following page.

Inside the Code Editor are program statements associated with the current form. Program statements that are used together to perform some action are typically grouped in a programming construct called a *procedure*. A common type of procedure is a Sub procedure, sometimes called a subroutine. Sub procedures include a Sub keyword in the first line and end with End Sub. Procedures typically get executed when certain events occur, such as when a button is clicked. When a procedure is associated with a particular object and an event, it is called an *event handler* or an *event procedure*.



When you double-clicked the End button (Button2), Visual Studio automatically added the first and last lines of the End button event procedure, as the following code shows. (The first line was wrapped to stay within the book margins.) You will notice other lines of code in the Code Editor, which Visual Studio has added to define important characteristics of the form. (You can recognize these statements by the words, "Windows Form Designer generated code.") Ignore these statements for now and don't modify them. You'll learn about this boilerplate code in later chapters.

```
Private Sub Button2_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles Button2.Click


End Sub
```

The body of a procedure fits between these lines and is executed whenever a user activates the interface element associated with the procedure. In this case, the event is a mouse click, but as you'll see later in the book, it could also be a different type of event.

2 Type **End**, and then press the Down arrow key.

After you type the statement, the letters turn blue, indicating that Visual Basic recognizes it as a valid statement, or *keyword*, in the program.

The End statement stops the execution of a program.

You use the program statement End to stop your program and remove it from the screen. The Visual Basic programming language contains several hundred unique keywords such as this, complete with their associated operators and symbols. The spelling of and spacing between these items are critical to writing program code that will be accurately recognized by the Visual Basic compiler. As you enter these keywords and perform other edits, the Code Editor will handle many of the formatting details for you, including indents, spacing, and adding parentheses that you need.

tip

Another name for the exact spelling, order, and spacing of keywords in a program is *statement syntax*.

When you pressed the Down arrow key, the End statement was indented to set it apart from the Private Sub and End Sub statements. This indenting scheme is one of the programming conventions you'll see throughout this book to keep your programs clear and readable. The group of conventions regarding how program code is organized in a program is often referred to as *program style*.

Now that you've written the code associated with the End button, you'll write code for the Spin button. These programming statements will be a little more extensive and will give you a chance to learn more about program syntax and style. You'll study each of the program statements later in this book, so you don't need to know everything about them now. Just focus on the general structure of the program code and on typing the program statements exactly as they are printed. (Visual Basic is fussy about spelling and the order in which keywords and operators appear.)

Write code for the Spin button

View Designer button

1 Click the View Designer button in Solution Explorer window to display your form again.

When the Code Editor is visible, you won't be able to see the form you're working on. The View Designer button is one mechanism you can use to display it again. (If more than one form is loaded in Solution Explorer, click the form you want to display first.)

tip

To display the form again, you can also click the tab labeled "Form1.vb [Design]" at the top edge of the Code Editor. If you don't see tabs at the top of the Code Editor, enable Tabbed Documents view in the Options dialog box, as discussed in a Tip in Chapter 1.

2 Double-click the Spin button.

After a few moments, the Code Editor appears, and an event procedure associated with the Button1 button appears near the Button2 event procedure.

Although you changed the text of this button to "Spin", its name in the program is still Button1. (The name and the text of an interface element can be different to suit the needs of the programmer.) Each object can have several procedures associated with it, one for each event it recognizes. The click event is the one we're interested in now because users will click the Spin and End buttons when they operate the program.

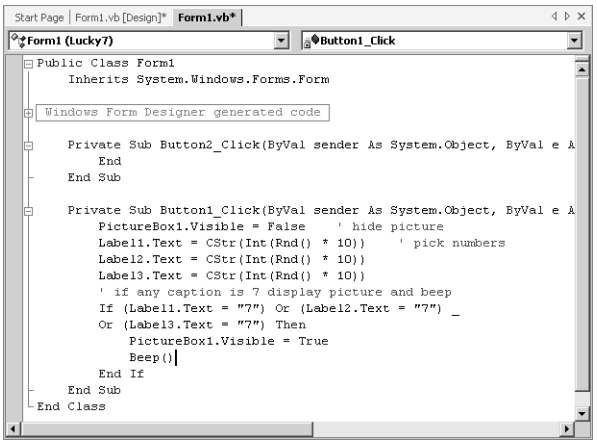
3 Type the following program lines between the Private Sub and End Sub statements, pressing Enter after each line, indenting with Tab, and taking care to type the program statements exactly as they appear here. (The Code Editor will scroll to the left as you enter the longer lines.) If you make a mistake (usually identified by jagged underline), delete the incorrect statements and try again.

tip

As you enter the program code, Visual Basic formats the text and displays different parts of the program in color to help you identify the various elements. When you begin to type a property, Visual Basic also displays the available properties for the object you're using in a list box, so you can double-click the property or keep typing to enter it yourself. If Visual Basic displays an error message, you may have misspelled a program statement. Check the line against the text in this book, make the necessary correction, and continue typing. (You can also delete a line and type it from scratch.) In addition, Visual Basic might also add code automatically when it is necessary. For example, when you type the following code, Visual Basic will automatically add the End If line. Readers of previous editions of this book have found this first typing exercise to be the toughest part of this chapter—"But Mr. Halvorson, I *know* I typed it just as written!"—so please give this program code your closest attention. I promise you, it works!

```
PictureBox1.Visible = False    ' hide picture
Label1.Text = CStr(Int(Rnd() * 10))    ' pick numbers
Label2.Text = CStr(Int(Rnd() * 10))
Label3.Text = CStr(Int(Rnd() * 10))
' if any caption is 7 display picture and beep
If (Label1.Text = "7") Or (Label2.Text = "7") _
Or (Label3.Text = "7") Then
    PictureBox1.Visible = True
    Beep()
End If
```

When you've finished, the Code Editor should look like this:



4 Click the Save All command on the File menu to save your additions to the program.

The Save All command saves everything in your project—the project file, the form file, any code modules, and other related components in your application. Note that if you want to save just the item you are currently working on (the form, the code module, or something else), you can use the Save command on the File menu. If you want to save the current item with a different name, you can use the Save As command.

A Look at the Button1_Click Procedure

The Button1_Click procedure is executed when the user clicks the Spin button on the form. The procedure uses some pretty complicated statements, and, because I haven't formally introduced them yet, it may look a little confusing. However, if you take a closer look you'll probably see a few things that look

familiar. Taking a peek at the contents of the procedures will give you a feel for the type of program code you'll be creating later in this book. (If you'd rather not have a look, feel free to skip to the next section, "Running Visual Basic .NET Applications.")

The Button1_Click procedure performs three tasks: it hides the digital photo, creates three random numbers for the number labels, and displays the photo when the number 7 appears. Let's look at each of these steps individually.

Hiding the photo is accomplished with the following line:

```
PictureBox1.Visible = False    ' hide picture
```

This line is made up of two parts: a program statement and a comment. The program statement (PictureBox1.Visible = False) sets the Visible property of the picture box object (PictureBox1) to False (one of two possible settings). You might remember that you set this property to False once before by using the Properties window. You're doing it again now in the program code because the first task is a spin and you need to clear away a photo that might have been displayed in a previous game. Because the property will be changed at runtime and not at design time, the property must be set by using program code. This is a handy feature of Visual Basic, and we'll talk about it more in Chapter 3.

The second part of the first line (the part displayed in green type on your screen) is called a *comment*. Comments are explanatory notes included in program code following a single quotation mark ('). Programmers use comments to describe how important statements work in a program. These notes aren't processed by Visual Basic when the program runs; they exist only to document what the program does. You'll want to use comments often when you write Visual Basic programs to leave an easy-to-understand record of what you're doing.

The next three lines handle the random number computations. Does this concept sound strange? You can actually make Visual Basic generate unpredictable numbers within specific guidelines—in other words, you can create random numbers for lottery contests, dice games, or other statistical patterns. The Rnd function in each line creates a random number between 0 and 1 (a number with a decimal point and several decimal places), and the Int function rounds the product of the random number and 10 to the nearest decimal place. This computation creates random numbers between 0 and 9 in the program—just what we need for this particular slot machine application.

```
Label1.Text = CStr(Int(Rnd() * 10))    ' pick numbers
```

We then need to jump through a little hoop in our code—we need to copy these random numbers into the three label boxes on the form, but before we do so the

numbers need to be converted to text with the CStr (convert to string) function. Notice how CStr, Int, and Rnd are all connected together in the program statement—they work collectively to produce a result like a mathematical formula. After the computation and conversion, the values are then assigned to the Text properties of the first three labels on the form, and the assignment causes the numbers to be displayed in boldface, 24-point, Times New Roman type in the three number labels. The following illustration shows how Visual Basic evaluates one line of code step by step to generate the random number 7 and copy it to a label object. Visual Basic evaluates the expression just like a mathematician solving a mathematical formula.

Example:

```
Label1.Text = CStr(Int(Rnd() * 10))
```

Code	Result
`Rnd()`	`0.7055475`
`Rnd() * 10`	`7.055475`
`Int(Rnd() * 10)`	`7`
`CStr(Int(Rnd() * 10))`	`"7"`
`Label1.Text = CStr(Int(Rnd() * 10))`	

The last group of statements in the program checks whether any of the random numbers is seven. If one or more of them is, the program displays the medieval manuscript depiction of a payout, and a beep announces the winnings.

```
' if any caption is 7 display picture and beep
If (Label1.Text = "7") Or (Label2.Text = "7") _
Or (Label3.Text = "7") Then
    PictureBox1.Visible = True
    Beep()
End If
```

The complete Lucky 7 program is located in the c:\vbnetsbs\chap02\lucky7 folder.

Each time the user clicks the Spin button, the Button1_Click procedure is called and the program statements in the procedure are executed again.

Running Visual Basic .NET Applications

Congratulations! You're ready to run your first real program. To run a Visual Basic program from the programming environment, you can click Start on the Debug menu, click the Start button on the Standard toolbar, or press F5. Try running your Lucky Seven program now. If Visual Basic displays an error message, you may still have a typing mistake or two in your program code. Try to fix it by comparing the printed version in this book with the one you typed, or load Lucky7 from your hard disk and run it.

Run the Lucky Seven program

Start button

1 Click the Start button on the Standard toolbar.

The Lucky Seven program compiles and runs in the programming environment. After a few seconds, the user interface appears, just as you designed it.

2 Click the Spin button.

The program picks three random numbers and displays them in the labels on the form, as follows:

Because a seven appears in the first label box, the digital photo appears depicting your payoff and the computer beeps. You win! (The sound you hear depends on your Sounds And Multimedia setting in Windows Control Panel—to make this game sound really cool, change the Default Beep sound to something more dynamic.)

3 Click the Spin button 15 or 16 more times, watching the results of the spins in the number boxes.

About half the time you spin, you hit the jackpot—pretty easy odds. (The actual odds are about 2.8 times out of 10; you're just lucky at first.) Later on you might want to make the game tougher by displaying the photo only when two or three sevens appear, or by creating a running total of winnings.

4 When you've finished experimenting with your new creation, click the End button.

The program stops, and the programming environment reappears on your screen.

tip

If you run this program again, you may notice that Lucky Seven displays exactly the same sequence of random numbers. There is nothing wrong here—the Visual Basic Rnd function was designed to display a repeating sequence of numbers at first so that you can properly test your code using output that can be reproduced again and again. To create truly "random" numbers, use the Randomize function in your code, as shown in the exercise at the end of this chapter.

Building an Executable File

Your last task in this chapter is to complete the development process and create an application for Windows, or an *executable file*. Windows applications created with Visual Basic have the filename extension .exe and can be run on any system that contains Microsoft Windows and the necessary support files. (Visual Basic installs these support files—including the dynamic link libraries and the .NET Framework files—automatically.) If you plan to distribute your applications, see Chapter 14 in this book. In that chapter, you'll learn more about optimizing your Visual Basic .NET applications and how to use *assemblies* to distribute solutions.

At this point, you need to know that Visual Studio has the ability to create two types of executable files for your project, a *debug build* and a *release build*. Debug builds are the default type in Visual Studio, and you'll use them often as you test and debug your program. Debug builds contain debugging information that makes them run slower, but the Visual Basic compiler is able to produce the builds quite quickly. When your project is complete, however, you want to compile your application using a release build, which includes numerous optimizations and doesn't contain unneeded debugging information.

tip

Although there are many similarities, Visual Basic .NET approaches compiling and distributing your applications in a different way than Visual Basic 6. For example, the Make Project.exe command on the Visual Basic 6 File menu has been replaced with the Build Solution command on the Build menu, and assemblies and manifests are now a feature of compilation and deployment. For additional details about this conceptual change, see Chapter 14.

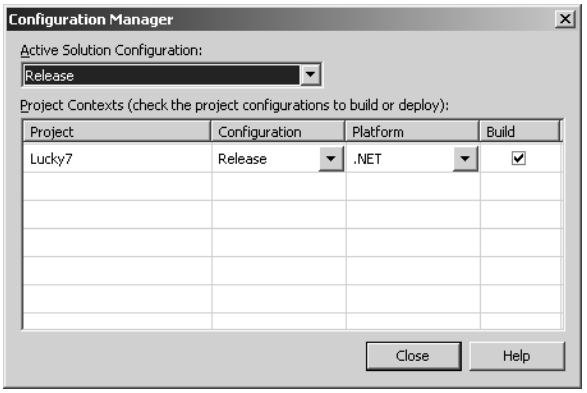
Try creating a release build named MyLucky7.exe now.

Create an executable file

1. On the Build menu, click the Configuration Manager command.

   The Configuration Manager dialog box appears. This dialog box lets you switch between debug builds and release builds, and it contains additional program settings such as the operating platform for which you are creating the application.

2. Click Release in the Active Solution Configuration list box, and then click Close.



Your project will now be compiled and optimized as a release build.

tip

You can also specify the debug or release build type using the Solution Configurations drop-down list box on the Standard toolbar.

3. On the Build menu, click the Build Solution command.

   The Build Solution command creates a bin folder where your project is located (if the folder doesn't already exist) and compiles the source code in your project. The result is an executable file named MyLucky7.exe. To save you time, Visual Studio often creates these files as you develop your application; however, it is always a good idea to recompile your application manually with the Build Solution command when you reach an important milestone. In particular, you'll need to recompile if you switch from debug build to release build.

   Try running this program now by using the Run command on the Start menu.

Start button

4. On the Windows taskbar, click the Start button and then click Run.

   The Run dialog box appears.

5. Click the Browse button, and then navigate to the c:\vbnetsbs\chap02\mylucky7\bin folder.
6. Click the MyLucky7.exe application icon, click Open, and then click OK.
7. The Lucky Seven program loads and runs in Windows—you've run the program outside the Visual Studio development environment.
8. Click Spin a few times to verify the operation of the game, and then click End.

tip

You can also run Windows applications, including compiled Visual Basic programs, by opening Windows Explorer and double-clicking the executable file. To create a shortcut icon for MyLucky7.exe on the Windows desktop, right-click the Windows desktop, point to New, and then click Shortcut. When you are prompted for the location of your application file, click Browse and select the MyLucky7.exe executable file. Click the Open, Next, and Finish buttons, and Windows will place an icon on the desktop that you can double-click to run your program.

9. On the File menu, click Exit to close Visual Studio and the MyLucky7 project.

   The Visual Studio development environment closes. If you decide later that you want to make extensive changes to this program (more than the changes listed on the following pages), use the Configuration Manager command on the Build menu to change the build type from release build to debug build.

One Step Further: Adding to a Program

You can restart Visual Studio at any time and work on a programming project you have stored on disk. You'll restart Visual Studio now and add a special statement named Randomize to the Lucky Seven program.

Reload Lucky Seven

1 On the Windows taskbar, click the Start button, point to Programs, point to Microsoft Visual Studio .NET, and then click the Microsoft Visual Studio .NET program icon.

A list of the most recent projects that you have worked on appears on the Visual Studio Start Page. Because you just finished working with Lucky Seven, MyLucky7 should be the first project on the list.

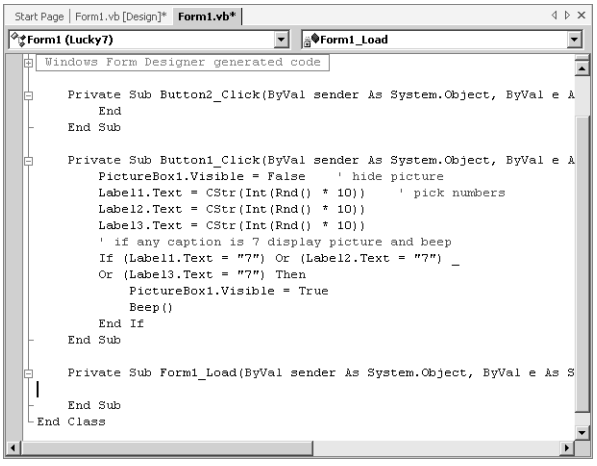
2 Click the MyLucky7 link to open the Lucky Seven project.

The Lucky Seven program opens, and the MyLucky7 form appears. (If you don't see the form, click Form1.vb in Solution Explorer and then click the View Designer button.)

Now you'll add the Randomize statement to the Form_Load procedure, a special procedure that is associated with the form and that is executed each time the program is started.

3 Double-click the form (not one of the objects) to display the Form_Load procedure.

The Form_Load procedure appears in the Code Editor, as shown here:



4 Type **Randomize**, and then press the Down arrow key.

The Randomize statement is added to the program and will be executed each time the program starts. Randomize uses the system clock to create a truly random starting point, or "seed," for the Rnd statement used in the Button1_Click procedure. As I mentioned earlier, without the Randomize statement the Lucky Seven program produces the same string of random spins every time you restart the program. With Randomize in place, the program will spin randomly every time it runs. The numbers won't follow a recognizable pattern.

5 Run the new version of Lucky Seven, and then save the project. If you plan to use the new version a lot, you may want to create a new .exe file too.

6 When finished, click Close Solution on the File menu.

The files associated with the Lucky Seven program are closed.

Chapter 2 Quick Reference

To	Do this
Create a user interface	Use Toolbox controls to place objects on your form, and then set the necessary properties. Resize the form and the objects as appropriate.
Move an object	Position the mouse over the object until you get the four-headed arrow, and then drag the object.
Resize an object	Click the object to select it, and then drag the resize handle attached to the part of the object you want to resize.
Delete an object	Click the object, and then press the Delete key.
Open the Code Editor	Double-click an object on the form (or the form itself). *or* Select a form or a module in the Solution Explorer and then click the View Code button.
Write program code	Type Visual Basic program statements associated with the object you want to program in the Code Editor.
Save a program	On the File menu, click the Save All command. *or* Click the Save All button on the Standard toolbar.

To	Do this
Save a form file	Make sure the form is open, and then, on the File menu, click the Save command. *or* Click the Save button on the Standard toolbar.
Change the build type from debug build to release build	On the Build menu, click the Configuration Manager command. Click the build type you want (Debug or Release) in the Active Solution Configuration drop-down list box. *or* Select the build type from the Solution Configurations drop-down list box on the Standard toolbar.
Create an .exe file	On the Build menu, click the Build command.
Reload a project	On the File menu, point to Open, and then click the Projects command. *or* On the File menu, point to Recent Projects, and then click the desired project. *or* Click the project in the recent projects list on the Visual Studio Start Page.

CHAPTER

3

Working with Toolbox Controls

In this chapter you will learn how to:

- ✔ *Use TextBox and Button controls to create a "Hello World" program.*
- ✔ *Use the DateTimePicker control to display your birth date.*
- ✔ *Use CheckBox, RadioButton, ListBox, and ComboBox controls to process user input.*
- ✔ *Use a LinkLabel control to display a Web page on the Internet.*
- ✔ *Install ActiveX controls.*

As you learned in Chapters 1 and 2, Microsoft Visual Studio .NET controls are the graphical tools you use to build the user interface of a Visual Basic program. Controls are located in the Toolbox in the development environment, and you use them to create objects on a form with a simple series of mouse clicks and dragging motions. Windows Forms controls are specifically designed for building Windows applications, and you'll find them organized on the Windows Forms tab of the Toolbox. (You used a few of these controls in the previous chapter.) You'll learn about other controls, including the tools you use to build Web Forms and database applications, later in the book.

In this chapter, you'll learn how to display information in a text box, work with date and time information on your system, process user input, and display a Web page within a Visual Basic .NET program. The exercises in this chapter will help you design your own Visual Basic applications and will teach you more about objects, properties, and program code. You'll also learn how to add older ActiveX controls to the Toolbox so that you can extend the functionality of Visual Basic.

Upgrade Notes: What's New in Visual Basic .NET?

If you're experienced with Visual Basic 6, you'll notice some new features in Visual Basic .NET, including the following:

- A new control named DateTimePicker helps you prompt the user for date and time information. The new LinkLabel control is designed to display and manage Web links on a form.
- The OptionButton control has been replaced with a new RadioButton control.
- The Frame control has been replaced with a new GroupBox control.
- The ListIndex property in the ListBox control has been replaced by a property called SelectedIndex. The same change was made to the ComboBox control.
- There is no longer an Image control. You use the PictureBox control instead.
- Images are added to picture box objects using the System.Drawing.Image.FromFile method (not the LoadPicture function).
- Web browsers and other applications are now started using the System.Diagnostics.Process.Start method.
- ActiveX controls are added to the Toolbox in a new way and are "wrapped" by Visual Studio so that they can be used in Visual Basic .NET applications.

The Basic Use of Controls: The "Hello World" Program

A great tradition in introductory programming books is the "Hello World" program. Hello World is the name given to a short program that demonstrates how the simplest utility can be built and run in a given programming language. In the days of character-based programming, Hello World was usually a two-line or three-line program typed in a program editor and assembled with a stand-alone compiler. With the advent of graphical programming tools, however, the typical Hello World has grown into a complex program containing dozens of

lines and requiring several programming tools for its construction. Fortunately, creating a Hello World program is still quite simple with Visual Basic .NET. You can construct a complete user interface by creating two objects, setting two properties, and entering one line of code. Give it a try.

Create a Hello World program

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

   Visual Studio displays the New Project dialog box, which prompts you for the name and location of your project and the template that you want to use to open it. (Unlike previous versions of Visual Basic, you begin a project by specifying a name for your program.)

tip

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

3. Make sure the Visual Basic Projects folder is selected, and then click the Windows Application template.

   These selections indicate that you'll be building a stand-alone Visual Basic Windows application.
4. Type **MyHello** in the Name text box, and then click the Browse button.

   The Project Location dialog box opens. You use this dialog box to specify the location of your project and to create new folders for your projects if necessary. Although you can save your projects in any location that you want (the folder \My Documents\Visual Studio Projects is a common location), in this book I will instruct you to save your projects in the c:\vbnetsbs folder, the default location for your Step by Step practice files. If you ever want to remove all of the files associated with this programming course, you will know just where the files are, and you will be able to remove them easily by deleting the entire folder.

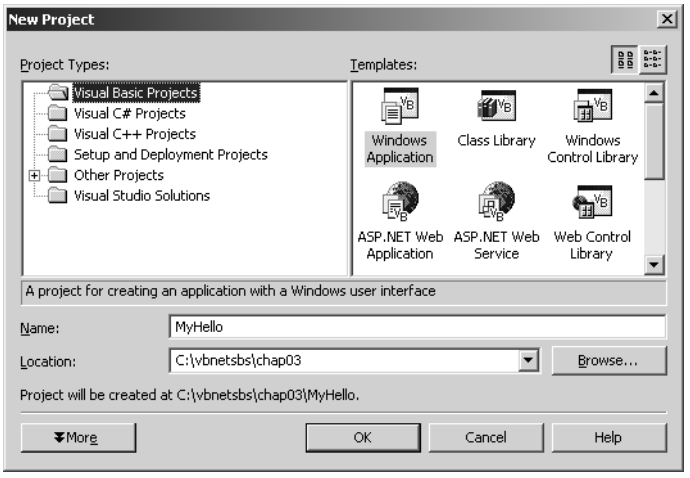
tip

Throughout this book, I ask you to create sample projects with the My prefix, to distinguish your own work from the sample files I include on the companion CD.

5 Click the Desktop icon in the Project Location dialog box, double-click the My Computer icon, and then browse to the folder c:\vbnetsbs\chap03.

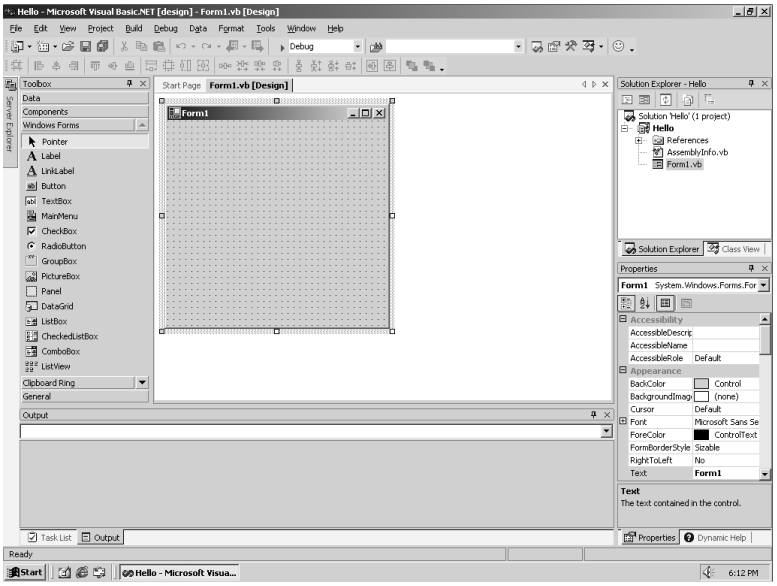
6 Click the Open button to indicate that the MyHello project and its supporting files will be saved in the c:\vbnetsbs\chap03 folder in a subfolder named MyHello.

The New Project dialog box now looks like this:



7 Click OK to create your new project.

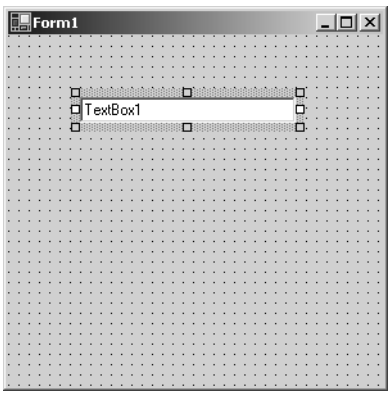
The new project is created and a blank form appears in the Windows Forms Designer, as shown in the following illustration. The two controls you'll use in this exercise, Button and TextBox, are labeled in the Toolbox. If your programming tools are configured differently now, take a few moments to organize them as shown in the illustration. (Chapter 1 describes how to configure the Visual Studio development environment if you need a refresher course.)



8 Click the TextBox control on the Windows Forms tab of the Toolbox.

TextBox control

9 Draw a text box similar to this:



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

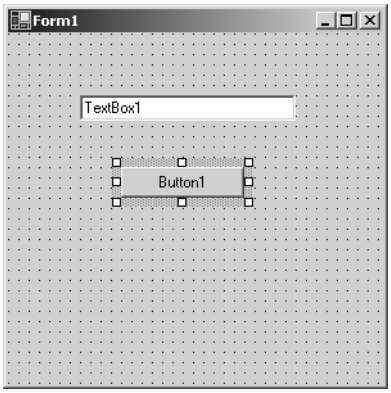
You'll add that button now.

Button control

10 Click the Button control in the Toolbox.

11 Draw a button below the text box on the form.

Your form should look like this:



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

For more information about setting properties, see the section "The Properties Window" in Chapter 1.

12 Set the following properties for the text box and button objects, using the Properties window. The setting "(empty)" for TextBox1 means that you should delete the current setting and leave the property blank. Settings you need to type in are shown in quotation marks. You shouldn't type the quotation marks.

Object	Property	Setting
TextBox1	Text	(empty)
Button1	Text	"OK"

The complete Hello World program is located in the c:\vbnetsbs\chap03\hello folder.

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

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

tip

After you type the TextBox1 object name and a period, Visual Studio displays a list box containing all the valid properties for text box objects, to jog your memory if you've forgotten the complete list. You can select a property from the list by double-clicking it, or you can continue typing and enter it yourself. (I usually just keep on typing, unless I'm exploring new features.)

The statement you've entered changes the Text property of the text box to "Hello, world!" when the user clicks the button at runtime. (The equal sign assigns everything between the quotation marks to the Text property of the TextBox1 object.) This example changes a property at runtime—one of the most common uses of program code in a Visual Basic program. Your statement is in an *event procedure*—an instruction that is executed when the Button1 object is clicked. It changes the property setting (and therefore the text box contents) immediately after the user clicks the button.

Now you're ready to run the Hello World program.

Run the Hello World program

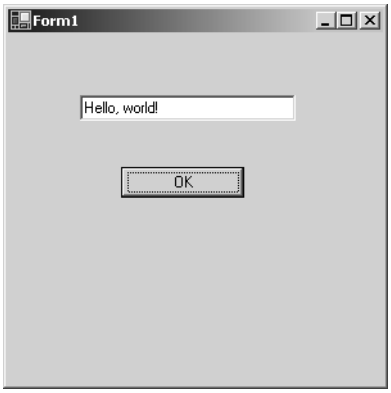
Start button

1. Click the Start button on the Standard toolbar.

   The Hello World program compiles and after a few seconds runs in the Visual Studio development environment.

2. Click the OK button.

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



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

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

tip

To stop a program running in Visual Studio, you can also click the Stop Debugging button on the Visual Studio Debug toolbar to close the program. (In Visual Basic 6, this button was named End.)

Save All button

4 Click the Save All button on the Visual Studio Standard toolbar to save your changes.

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

Using the DateTimePicker Control

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

The Birthday Program

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

Build the Birthday program

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

The files associated with the Hello World program are closed.

2 On the File menu, point to New, and then click Project.

The New Project dialog box appears.

3 Create a new Visual Basic Windows Application project named **MyBirthday** in the c:\vbnetsbs\chap03 folder.

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

DateTimePicker control

4 Click the DateTimePicker control in the Toolbox.

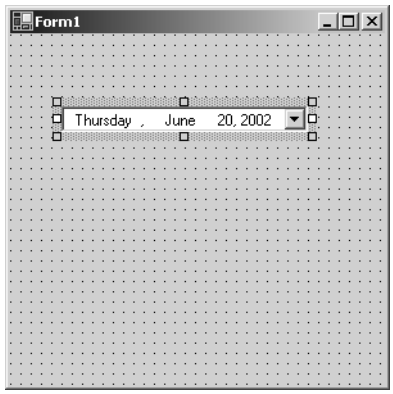
Down scroll arrow

Up scroll arrow

tip

If you don't see the DateTimePicker in the Toolbox, it may be lower in the Toolbox list hidden from your view. To scroll down in the Toolbox list, click the down scroll arrow next to the Clipboard Ring tab. To scroll back up, click the up scroll arrow next to the Windows Forms tab.

5 Draw a date time picker object in the middle of the form, as shown in the following:



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

Button control

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

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

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

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

Double-click an object to display its default event procedure.

8 Double-click the button object on the form, and then type the following program statement between the Private Sub and End Sub statements in the Button1_Click event procedure:

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

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

The second and third lines collectively form one program statement and have been broken by the line continuation character (_) because the statement was a bit too long to print in our book. (See Tip box below for an explanation of this useful convention for breaking longer lines.) The statement DateTimePicker1.Value.DayOfYear.ToString() uses the date time picker object to calculate the day of the year in which you were born, counting from January 1. This is accomplished by the DayOfYear property and the ToString method, which converts the numeric result of the date calculation to a textual value that is more easily displayed by the MsgBox function.

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

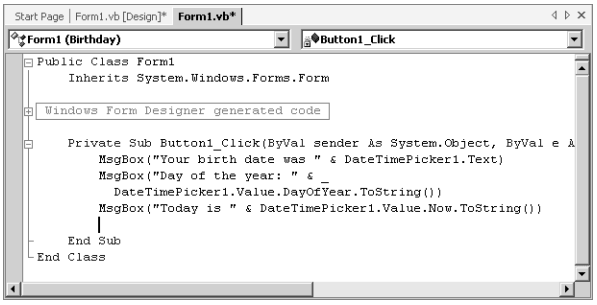
The fourth line in the program code uses the Now property to check your computer's system clock for the current date and time and displays that information in a message box after converting it to a string, or textual, value.

tip

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

After you enter the code for the Button1_Click event procedure, the Code Editor should look similar to the following illustration.

The complete Birthday program is located in the c:\vbnetsbs\chap03\birthday folder.



Save All button

9 Click the Save All button to save your changes to disk.

Now you're ready to run the Birthday program.

Run the Birthday program

Start button

1 Click the Start button on the Standard toolbar.

The Birthday program starts to run in the development environment. The current date is displayed in the date time picker.

2 Click the drop-down arrow in the date time picker to display the object in calendar view.

Your form will look like the illustration on the following page. (You'll see a different date.)

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

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

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

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

When you select the date, the date time picker will close.

5 Type your birth year in place of the year that is currently selected, and then click the drop-down arrow again.

The calendar reappears in the year of your birth.

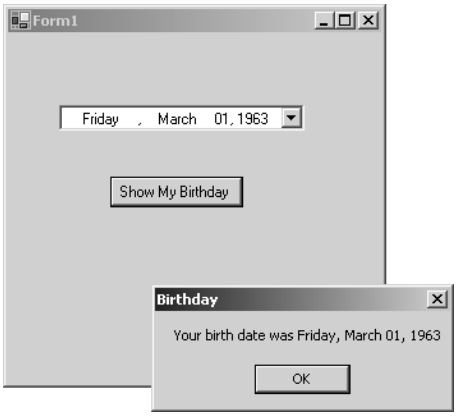
6 Click the scroll arrows again to locate the month in which you were born, and then click the exact day on which you were born.

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

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

7 Click the Show My Birthday button.

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



8 Click OK in the message box.

A second message box appears indicating which day of the year you were born on.

9 Click OK to display the final message box.

The current date and time are displayed—the program works!

You'll find the date time picker object to be quite capable—not only does it remember the new date or time information that you enter, but it keeps track of the current date and time as well, and it can display this date and time information in a variety of useful formats.

tip

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

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

You're finished using the DateTimePicker control for now.

A Word About Terminology

So far in this book I've used several different terms to describe items in a Visual Basic program. Although I haven't defined each of them formally, it's worth listing several of them now to clear up any confusion. Can you tell the difference yet?

Program statement A program statement is a keyword in the code that does the work of the program. Visual Basic program statements create storage space for data, open files, perform calculations, and do several other important tasks. Most keywords are shown in blue type in the Code Editor.

Variable A variable is a special container used to hold data temporarily in a program. The programmer creates variables using the Dim statement to store the results of a calculation, create filenames, process input, and so on. Numbers, names, and property values can be stored in variables.

Control A control is a tool you use to create objects on a Visual Basic form. You select controls from the Toolbox and use them to draw objects on a form with the mouse. You use most controls to create user interface elements, such as buttons, picture boxes, and list boxes.

Object An object is the name of a user interface element you create on a Visual Basic form with a Toolbox control. You can move, resize, and customize objects by using property settings. Objects have what is known as *inherent functionality*—they know how to operate and can respond to certain situations on their own. (A list box "knows" how to scroll, for example.) And objects are the members of *classes*, which serve as the blueprints for defining what an object does. You can program Visual Basic objects by using customized event procedures for different situations in a program. In Visual Basic, the form itself is also an object.

Property A property is a value, or characteristic, held by an object. For example, a button object has a Text property to specify the text that appears on the button and an Image property to specify the path to an image file that should appear on the button face. In Visual Basic, properties can be

set at design time by using the Properties window or at runtime by using statements in the program code. In code, the format for setting a property is

```
Object.Property = Value
```

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

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

could be used in the program code to set the Text property of the Button1 object to "Hello".

Event procedure An event procedure is a block of code that is executed when an object is manipulated in a program. For example, when the Button1 object is clicked, the Button1_Click event procedure is executed. Event procedures typically evaluate and set properties and use other program statements to perform the work of the program.

Method A method is a special statement that performs an action or a service for a particular object in a program. In program code, the notation for using a method is

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

where *Object* is the name of the object you want to work with, *Method* is the action you want to perform, and *Value* is an optional argument to be used by the method. For example, the statement

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

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

Controls for Gathering Input

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

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

Experiment with the CheckBox control

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

   The New Project dialog box appears.
3. Create a new Visual Basic Windows Application project named **MyCheckBox** in the c:\vbnetsbs\chap03 folder.

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

CheckBox control

4. Click the CheckBox control in the Toolbox.
5. Draw two check box objects on the form, one above the other.

   Check boxes appear like objects on your form just as other objects do.

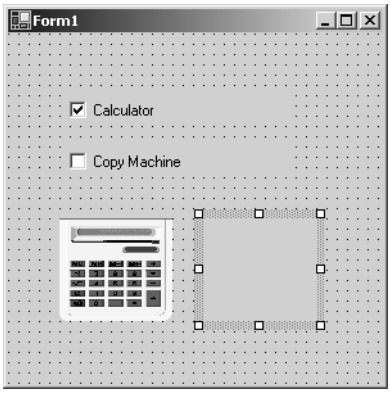
PictureBox control

6. Click the PictureBox control and draw two square picture box objects beneath the two check boxes.
7. Set the following properties for the check box and label objects:

Object	Property	Setting
CheckBox1	Checked	True
	Text	"Calculator"
CheckBox2	Text	"Copy machine"
PictureBox1	Image	c:\vbnetsbs\chap03\calcultr.bmp
	SizeMode	StretchImage
PictureBox2	SizeMode	StretchImage

In this demonstration program, you'll use the check boxes to display and hide images of a calculator and a copy machine. The Text property of the check box object determines the contents of the check box label in the user interface. The Checked property lets you set a default value for the check box. Setting Checked to True will place a check mark in the box, and setting Checked to False (the default setting) will remove the check mark. I use the SizeMode properties in the picture boxes to size the images so that they stretch to fit in the picture box.

Your form should look like this:



8 Double-click the first check box object to open the Code Editor for the CheckBox1_CheckedChanged event procedure, and then enter the following program code:

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

The CheckBox1_CheckChanged event procedure runs only if the user clicks in the first check box object. The event procedure uses an If…Then decision structure (described in Chapter 6) to check the current status, or "state," of the first check box, and it displays a calculator picture from the c:\vbnetsbs\chap03 folder if a check mark is in the box. The CheckState property holds a value of 1 if there is a check mark present and 0 if there is no check mark present. I use the Visible property to display the picture if a check mark is present or hide the picture if a check mark isn't present.

Notice that I wrapped the long line that loads the image into the picture box object by using the line continuation (_) character.

View Designer button

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

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

The complete CheckBox program is located in the c:\vbnetsbs\chap03\checkbox folder.

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

Save All button

10 Click the Save All button on the Standard toolbar to save your changes.

Run the CheckBox program

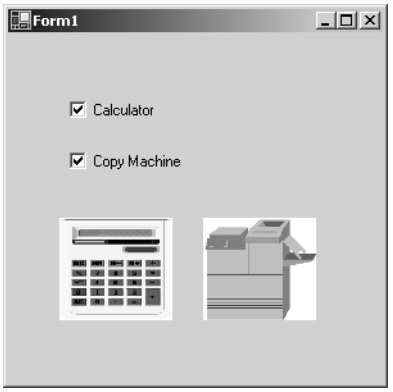
Start button

1 Click the Start button on the Standard toolbar.

Visual Basic runs the program in the development environment. Because you placed a check mark in the first check box, the calculator image appears on the form.

2 Click the Copy machine check box.

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



3 Experiment with different combinations of check boxes, clicking the boxes several times to test the program. The program logic you added with a few

short lines of Visual Basic code manages the boxes perfectly. (You'll learn much more about program code in upcoming chapters.)

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

The Input Controls Demo

Now that you've had a little experience with check boxes, run and examine the Input Controls demonstration program that I created to simulate an electronic ordering environment that makes more extensive use of check boxes, radio buttons, a list box, and a combo box. If you work in a business that does a lot of order entry, you might want to expand this into a full-featured graphical order entry program. After you experiment with Input Controls, spend some time learning how the four input controls work in the program. They were created in a few short steps by using Visual Basic and the techniques you just learned.

Run the Input Controls program

1 On the File menu, point to Open, and then click Project.

The Open Project dialog box appears.

2 Open the c:\vbnetsbs\chap03\input controls folder, and then double-click the Input Controls project file (Input Controls.vbproj).

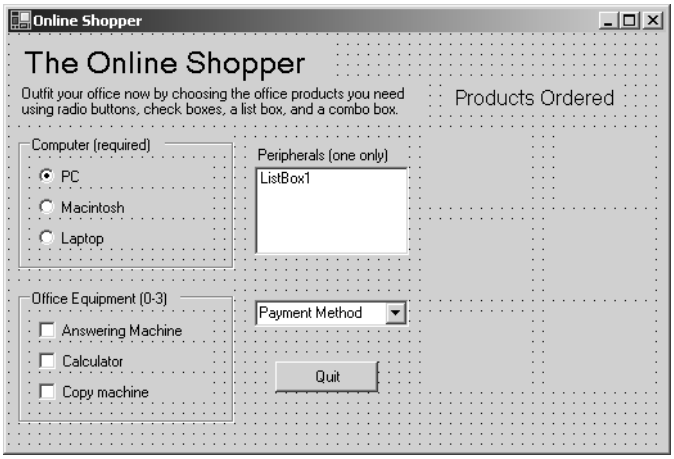
The Input Controls project opens in the development environment.

View Designer button

3 If the project's form isn't visible, click the Form1.vb form in Solution Explorer and then click the View Designer button.

4 Move or close the windows that block your view of the form so that you can see how the objects are laid out.

You'll see a form similar to this:



The Input Controls form contains radio button, check box, list box, combo box, picture box, button, and label objects. These objects work together to create a simple order entry program that demonstrates how the Visual Basic input objects work. When the Input Controls program is run, it loads images from the \vbnetsbs\chap03\input controls folder on drive C and displays them in the six picture boxes on the form.

tip

If you installed the practice files in a location other than the default c:\vbnetsbs folder, the statements in the program that load the artwork from the disk will contain an incorrect path. (Each statement begins with c:\vbnetsbs\chap03\input controls, as you'll see soon.) If this is the case, you can make the program work by renaming your practice files folder \vbnetsbs or by changing the paths in the Code Editor using the editing keys or the Find And Replace submenu on the Edit menu.

Start button

5 Click the Start button on the Standard toolbar.

The program runs in the development environment.

6 Click the Laptop radio button in the Computer box.

Radio buttons allow the user to select only one item from a list.

The image of a laptop computer appears in the Products Ordered area on the right side of the form. The user can click various options and the current choice is depicted in the order area on the right. In the Computer box, a group of *radio buttons* is used to gather input from the user. (Note: Radio buttons were called option buttons in Visual Basic 6.) Radio buttons force the user to choose one (and only one) item from a list of possibilities.

When radio buttons are placed inside a group box object on a form, the radio buttons are considered to be part of a group and only one option can be chosen. To create a group box, click the GroupBox control on the Windows Forms tab of the Toolbox, and then draw the control on your form. (The GroupBox control replaces the Frame control in Visual Basic 6.) You can give the group of radio buttons a title (as I have) by setting the Text property of the group box object. When you move a group box object on the form, the controls within it also move.

7 Click the Answering Machine, Calculator, and Copy Machine check boxes in the Office Equipment box.

Check boxes let the user select any number of items.

Check boxes are used in a program so that the user can select more than one option at a time from a list. Click the Calculator check box again, and notice that the picture of the calculator disappears from the order area. Because

each user interface element is live and responds to click events as they occur, order choices are reflected immediately. The code that completes these tasks is nearly identical to the code you entered earlier in the CheckBox program.

8 Click Satellite Dish in the Peripherals list box.

List boxes let the user select one item from a variable-length list of choices.

A picture of a satellite dish is added to the order area. A *list box* is used to get a user's single response from a list of choices. List boxes are created with the ListBox control, and may contain many items to choose from (scroll bars appear if the list of items is longer than the list box). Unlike radio buttons, a default selection isn't required in a list box. In a Visual Basic program, items can be added to, removed from, or sorted in a list box while the program is running. If you would like to see check marks next to the items in your list box, use the CheckedListBox control in the Toolbox instead of the ListBox control.

9 Now choose U.S. Dollars (sorry, no credit) from the payment list in the Payment Method combo box.

Combo boxes take up less space than list boxes.

A *combo box,* or drop-down list box, is similar to a regular list box, but it takes up less space. (The "combo" in a combo box basically comes from a "combination" of an editable text box and a drop-down list.) Visual Basic automatically handles the opening, closing, and scrolling of the list box. All you do as a programmer is create the combo box using the ComboBox control in the Toolbox, set the Text property to provide directions or a default value, and then write code to add items to the combo box and to process the user's combo box selection. You'll see examples of each task in the program code for the Input Controls demonstration.

After you make your order selections, your screen should look something like this:

10 Practice making a few more changes to the order list in the program (try different computers, peripherals, and payment methods), and then click the Quit button in the program to exit.

The program closes when you click Quit, and the programming environment appears.

Looking at the Input Controls Program Code

Chapters 5, 6, and 7 discuss program code in detail.

Although you haven't had much formal experience with program code yet, it's worth taking a quick look at a few event procedures in Input Controls to see how the program processes input from the user interface elements. In these procedures, you'll see the If…Then and Select Case statements at work. You'll learn about these and other decision structures in Chapter 6. For now, concentrate on the CheckState property, which changes when a check box is selected, and the SelectedIndex property, which changes when a list box is selected.

Examine the check box code and the list box code

1 Be sure the program has stopped running, and then double-click the Answering Machine check box in the Office Equipment box to display the CheckBox1_CheckedChanged event procedure in the Code Editor.

You'll see the following program code:

```
'If the CheckState property for a check box is 1, it has a mark in it
If CheckBox1.CheckState = 1 Then
    PictureBox2.Image = System.Drawing.Image.FromFile _
      ("c:\vbnetsbs\chap03\input controls\answmach.bmp")
    PictureBox2.Visible = True
Else
    'If there is no mark, hide the image
    PictureBox2.Visible = False
End If
```

The first line of this event procedure is called a comment. Comments are displayed in green type and are simply notes written by the programmer to describe what is important or interesting about this particular piece of program code. (Comments are also occasionally generated by automated programming tools that compile programs or insert code snippets.) I wrote this comment to remind myself that the CheckState property contains a crucial value in this routine—a value of 1 if the first check box was checked.

The rest of the event procedure is nearly identical to the one you just wrote in the CheckBox program. If you scroll down in the Code Editor, you'll see a similar event procedure for the CheckBox2 and CheckBox3 objects.

2 At the top edge of the Code Editor, click the tab labeled "Form1.vb [Design]" to display the form again, and then double-click the Peripherals list box on the form.

The ListBox1_SelectedIndexChanged event procedure appears in the Code Editor. You'll see the following program statements:

```
'The item you picked (0-2) is held in the SelectedIndex property
Select Case ListBox1.SelectedIndex
    Case 0
        PictureBox3.Image = System.Drawing.Image.FromFile _
          ("c:\vbnetsbs\chap03\input controls\harddisk.bmp")
    Case 1
        PictureBox3.Image = System.Drawing.Image.FromFile _
          ("c:\vbnetsbs\chap03\input controls\printer.bmp")
    Case 2
        PictureBox3.Image = System.Drawing.Image.FromFile _
          ("c:\vbnetsbs\chap03\input controls\satedish.bmp")
End Select
```

Here you see code that executes when the user clicks an item in the Peripherals list box in the program. In this case, the important keyword is ListBox1.SelectedIndex, which is read "the SelectedIndex property of the first list box object." After the user clicks an item in the list box, the SelectedIndex property returns a number that corresponds to the location of the item in the list box. (The first item is numbered 0, the second item is numbered 1, and so on.)

In the previous code, SelectedIndex is evaluated by the Select Case decision structure, and a different image is loaded depending on the value of the SelectedIndex property. If the value is 0, a picture of a hard disk is loaded; if the value is 1, a picture of a printer is loaded; if the value is 2, a picture of a satellite dish is loaded. You'll learn more about how the Select Case decision structure works in Chapter 6.

3 At the top edge of the Code Editor, click the Form1.vb [Design] tab to display the form again, and then double-click the form (not any of the objects) to display the code associated with the form itself.

Statements in the Form1_Load event procedure run when the program starts.

The Form1_Load event procedure appears in the Code Editor. This is the procedure that is executed each time the Input Controls program starts, or *loads*. Programmers put program statements in this special procedure when they want them executed every time a form loads. (Your program can display more than one form, or none at all, but the default behavior is that Visual Basic loads and runs the Form1_Load event procedure each time the

user runs the program.) Often, as in the Input Controls program, these statements define an aspect of the user interface that couldn't be created by using Toolbox controls or the Properties window.

Here is what the Form1_Load event procedure looks like for this program:

```
'These program statements run when the form loads
PictureBox1.Image = System.Drawing.Image.FromFile _
  ("c:\vbnetsbs\chap03\input controls\pcomputr.bmp")
'Add items to a list box like this:
ListBox1.Items.Add("Extra hard disk")
ListBox1.Items.Add("Printer")
ListBox1.Items.Add("Satellite dish")
'Combo boxes are also filled with the Add method:
ComboBox1.Items.Add("U.S. Dollars")
ComboBox1.Items.Add("Check")
ComboBox1.Items.Add("English Pounds")
```

Three lines in this event procedure are comments displayed in green type. The second line in the event procedure loads the personal computer image into the first picture box. (This line is broken in two using a space and the line continuation character, but the compiler still thinks of it as one line.) Loading an image establishes the default setting reflected in the Computer radio button group box. The next three lines add items to the Peripherals list box (ListBox1) in the program. The words in quotes will appear in the list box when it appears on the form. Below the list box program statements, the items in the Payment Method combo box (ComboBox1) are specified. The important keyword in both these groups is Add, which is a special function, or method, to add items to list box and combo box objects.

You're finished using the Input Controls program. Take a few minutes to examine any other parts of the program you're interested in, and then move on to the next exercise.

Using the LinkLabel Control

Providing access to the Web is now a standard feature of many Windows applications, and Visual Studio .NET makes adding this functionality easier than ever before. You can create sophisticated Web-aware applications using Web Forms and other technologies in Visual Studio, or you can open a simple Web page using your computer's Web browser using just a few lines of program code.

In this exercise, you'll learn to use the LinkLabel control to display text on a Visual Basic form that looks and acts just like an Internet link. The LinkLabel control is a new addition to Visual Basic, and when you combine it with the

Process.Start method, you can quickly open links on your form using Internet Explorer, Netscape Navigator, or another browser. Use the LinkLabel control now to connect to the Microsoft Press home page on the Internet.

> **tip**
> To learn more about writing Web-aware Visual Basic .NET applications, read Chapters 21-22.

Create the WebLink program

1. On the File menu, click Close Solution to close the Input Controls project.
2. On the File menu, point to New, and then click Project.

   The New Project dialog box appears.
3. Create a new Visual Basic Windows Application project named **MyWebLink** in the c:\vbnetsbs\chap03 folder.

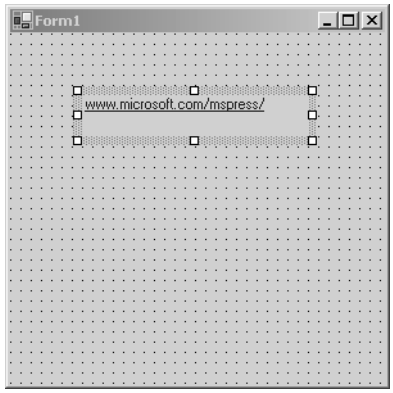
   The new project is created and a blank form appears in the Windows Forms Designer.

LinkLabel control

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

   Link label objects look like label objects, except that all label text is displayed in blue underlined type on the form.
5. Set the Text property of the link label object to **www.microsoft.com/mspress/** (the Web site for Microsoft Press).

   Your form will look like this:



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

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

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

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

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

```
' Change the color of the link by setting LinkVisited to True.
LinkLabel1.LinkVisited = True
' Use the Process.Start method to open the default browser
' using the Microsoft Press URL:
System.Diagnostics.Process.Start _
  ("http://www.microsoft.com/mspress/")
```

I've included comments in the program code to give you some practice entering them. As soon as you enter the single quote character ('), Visual Studio changes the color of the line to green, identifying the line as a comment. Comments are for documentation purposes only—they aren't evaluated or executed by the compiler.

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

The second program statement (that was broken into two lines) runs the default Web browser (such as Internet Explorer) if the browser isn't already running. (If the browser is running, the URL just loads immediately.) The Start method in the Process class performs the important work, by starting a *process* or executable program session in memory for the browser. The Process class, which manages many other aspects of program execution, is a member of the System.Diagnostics object library, which Visual Basic .NET

programmers called the System.Diagnostics namespace. By including an Internet address or *URL* along with the Start method, I'm letting Visual Basic know that I want to view a Web site, and Visual Basic is clever enough to know that the default system browser is the tool that would best display that URL, even though we didn't identify the browser by name.

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

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

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

Or if I had wished to run a different application with the Start method, such as the application Microsoft Word (and open the document c:\myletter.doc), I could have used the following syntax:

The complete WebLink program is located in the c:\vbnetsbs\chap03\weblink folder.

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

As you can see, the Start method in the Process class is very useful, and we'll make use of it again in this book. Now that you've entered your code, save your project and run it. (If you experimented with the Start syntax as I showed you, restore the original code shown at the beginning of step 8.)

Save All button

9 Click the Save All button on the Standard toolbar to save your changes.

Run the WebLink program

Start button

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

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

2 Click the link to open the Web site shown (*www.microsoft.com/mspress/*).

Recall that it is only a happy coincidence that the link label Text property contains the same URL as the site you named in the program code. You may enter any text you like in the link label. You can also use the Image property for a link label to specify a picture to display in the background of the link label. Look on the following page to see what the Microsoft Press Web page looks like (in English) when the WebLink program displays it using Internet Explorer.

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

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

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

You're finished writing code in this chapter.

One Step Further: Installing ActiveX Controls

Visual Studio .NET is a new product, so many developers will find that the current list of Windows Forms controls isn't as extensive as the ActiveX controls that were available to Visual Basic 6 programmers. If you want to expand the collection of controls that you have access to in Visual Studio .NET, you can load and use older ActiveX (COM) components and controls. These customizable programming tools were provided by Visual Basic 6, Visual C++ 6, Microsoft Office, and other third-party products. When you add older ActiveX controls to your Toolbox, Visual Studio displays them with the .NET controls, and you can use the ActiveX controls on your forms with few limitations. The only technical qualification is that Visual Studio must create a *wrapper* for the control, which makes the objects in the control usable in a .NET program. In most cases, Visual Studio .NET handles the creation of a control wrapper automatically when the ActiveX control is loaded.

tip

ActiveX controls on your computer typically have an .ocx or .dll filename extension. These library files are routinely added to your system when you install a new application program, and you can reuse them in your own programs if you can figure out how to use the objects, methods, and properties that they provide, or *expose*. Visual Studio .NET "learns" about the presence of new ActiveX controls when they are cataloged in the Windows system registry.

The Microsoft Chart Control

You probably have numerous ActiveX controls on your system now, especially if you have a previous version of Visual Basic or Microsoft Office on your system. Try installing one of the ActiveX controls now, even if you're not sure how to use it yet. (You'll start putting ActiveX controls to work later in the book, so you should learn the installation technique now.) The control you'll open in this exercise is the Microsoft Chart control, which allows you to build charts on your forms. The Microsoft Chart control is included in several versions of Microsoft Office. If you don't have Office or this control, pick another one.

Install the Chart control

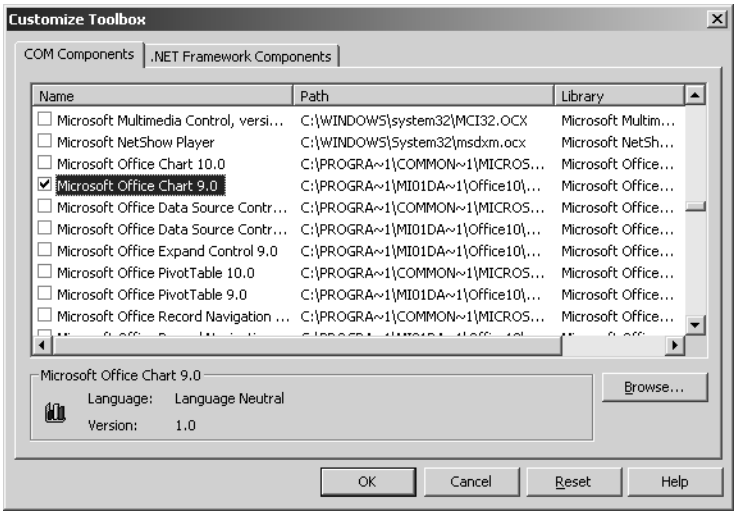
1. If you still have the Code Editor open for the WebLink project, display the form.
2. Click the Toolbox, and open the tab to which you would like to add an ActiveX control. For example, click the Windows Forms tab.

   ActiveX controls are added to individual tabs in the Toolbox so that you can remember where they are. Once on a Toolbox tab, they look just like regular controls in Visual Studio .NET.
3. Click the Windows Forms tab again with the right mouse button, and then click the Customize Toolbox command on the pop-up menu.

   The Customize Toolbox dialog box opens.
4. Click the COM Components tab in the Customize Toolbox dialog box.

   You'll see a long list of ActiveX components and controls that is unique to your system. The components appear in alphabetical order.
5. Scroll to the Microsoft Office Chart 9.0 control, or another control that looks interesting, and then click the check box next to the control.

   Your dialog box will look similar to the illustration on the following page.



6 To add additional controls to the Toolbox, simply click check boxes next to the controls that you want.

note

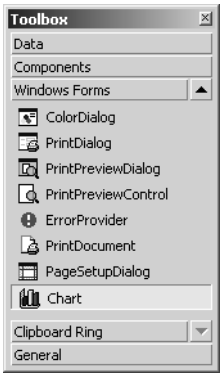
Not all the components listed in the Customize Toolbox dialog box are designed to work in the Toolbox, so they may not work properly when you try to use them.

7 Click OK to add the selected ActiveX control (or controls) to this project's Toolbox.

The control will appear at the bottom of the Toolbox list. The control will appear for this project only and won't appear in other projects.

Down scroll arrow

8 Click the down scroll arrow in the Toolbox to see the newly added control as shown here:



The Microsoft Office Chart ActiveX control is available in several different versions and is included with applications such as Office 2000 and Office XP. You'll find that most ActiveX controls work just like the standard controls in the Visual Basic .NET Toolbox. In fact, if you didn't know they were ActiveX controls, it would be difficult to tell them apart from the default controls in Visual Studio .NET. Each ActiveX control has adjustable property settings and methods that you can call just like the other controls you've used in this chapter.

Chapter 3 Quick Reference

To	Do this
 Create a text box	Click the TextBox control, and draw the box.
 Create a button	Click the Button control, and draw the button.
Change a property at runtime	Change the value of the property by using program code. For example: `Label1.Text = "Hello!"`
 Create a radio button	Use the RadioButton control. To create multiple radio buttons, place more than one button object inside a box that you create by using the GroupBox control.
 Create a check box	Click the CheckBox control, and draw a check box.
 Create a list box	Click the ListBox control, and draw a list box.
 Create a drop-down list box	Click the ComboBox control, and draw a drop-down list box.
Add items to a list box	Include statements with the Add method in the Form1_Load event procedure of your program. For example: `ListBox1.Items.Add "Printer"`
 Display a Web page	Create a link to the Web page using the LinkLabel control, and then open the link in a browser using the Process.Start method in program code.
Install ActiveX controls	Right click the Toolbox tab that you want to place the control in, and then click the Customize Toolbox command. Click the COM Components tab, place a check mark next to the ActiveX control that you want to install, and then click OK.

CHAPTER

4

Working with Menus and Dialog Boxes

In this chapter you will learn how to:

- ✔ *Add menus to your programs by using the MainMenu control.*
- ✔ *Process menu choices by using program code.*
- ✔ *Use the OpenFileDialog and ColorDialog controls to display special-purpose dialog boxes.*

In Chapter 3, you used several Microsoft Visual Studio controls to gather input from the user while he or she was using a program. In this chapter, you'll learn to present choices to the user by using professional-looking menus and dialog boxes. A *menu* is located on the menu bar and contains a list of related commands. When you click a menu title, a list of the menu commands appears in a drop-down list. Most menu commands are executed immediately after they are clicked; for example, when the user clicks the Copy command on the Edit menu, information is copied to the Clipboard immediately. If a menu command is followed by an ellipsis (…), however, Visual Basic displays a dialog box requesting more information before the command is carried out. In this chapter, you'll learn to use the MainMenu control and two dialog box controls to add menus and standard dialog boxes to your programs.

Upgrade Notes: What's New in Visual Basic .NET?

If you're experienced with Visual Basic 6, you'll notice some new features in Visual Basic .NET, including the following:

- Menus are no longer created using the Visual Basic 6 Menu Editor tool. Instead, you create a main menu object on your form using the MainMenu control, and then customize the object using property settings and the Menu Designer. However, menu choices are still processed with program code.
- Standard dialog boxes are no longer created using the CommonDialog control. Instead, you use one of seven Windows Forms controls that add standard dialog boxes to your project. These controls include OpenFileDialog, SaveFileDialog, FontDialog, ColorDialog, PrintDialog, PrintPreviewDialog, and PageSetupDialog.
- Forms now feature the ShowDialog method and the DialogResult property, making it easier to create custom forms that look and act like standard dialog boxes.

Adding Menus Using the MainMenu Control

The MainMenu control is a tool that adds menus to your programs and allows you to customize them with property settings in the Properties window. With the control, you can add new menus, modify and reorder existing menus, and delete old menus. You also can add special effects to your menus, such as access keys, check marks, and keyboard shortcuts. After you have added menus to your form, you can use event procedures to process the menu commands. In the following exercise, you'll use the MainMenu control to create a Clock menu containing commands that display the current date and time.

Create a menu

1. Start Visual Studio.
2. On the File menu, point to New, and then click Project.

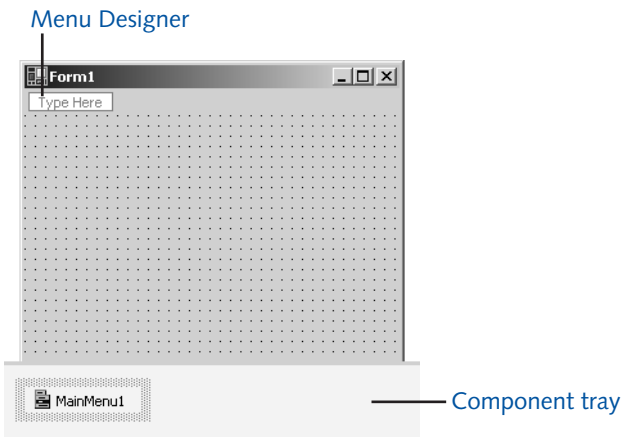
   The New Project dialog box appears.

3 Create a new Visual Basic Windows Application project named **MyMenu** in the c:\vbnetsbs\chap04 folder.

MainMenu control

4 Click the MainMenu control on the Windows Forms tab of the Toolbox, and draw a menu control on your form.

Don't worry about the location—Visual Studio will move the control and resize it automatically. Your screen will look like this:



The main menu object doesn't appear on your form, but below it. That's different from previous versions of Visual Basic, which in one way or another displayed all objects on the form itself—even ones that didn't have a visual representation when the program ran, such as the Timer control. But in Visual Studio .NET, non-visible objects such as menus and timers are displayed on a separate pane, named the component tray, in the development environment, and you can select them, set their properties, or delete them right from this pane.

In addition to the main menu object in the component tray, Visual Studio .NET displays a visual representation of the menu you created at the top of the form. The tag "Type Here" encourages you to click the tag and enter the title of your menu right now. After you enter the first menu title, you can enter submenu titles and other menu names by pressing the arrow keys and typing additional names. Best of all, you can come back to this in-line Menu Designer later and edit what you have done or add additional menu items—the main menu object is fully customizable and allows you to create an exciting menu-driven user interface like the ones you've seen in the best Windows applications.

5 Click the Type Here tag, type **Clock**, and then press Enter.

The word "Clock" is entered as the name of your first menu, and two additional Type Here tags appear, allowing you to create submenu items below the new Clock menu or additional menu titles. The submenu item is currently selected.

tip

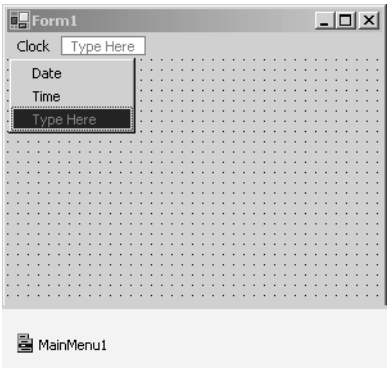
If the menu disappears from the form, click MainMenu1 in the component tray and the menu on the form should reappear.

6 Type **Date** to create a new Date command for the Clock menu, and then press Enter.

Visual Studio adds the Date command to the menu and selects the next submenu item.

7 Type **Time** to create a new Time command for the menu, and then press Enter.

You now have a Clock menu with two menu commands, Date and Time. You could continue to create additional menus or commands, but what you've done is sufficient for this example program. Your form will look like this:



8 Click the form to close the Menu Designer.

The Menu Designer closes, and your form appears in the development environment with no Clock menu. So where did the Clock menu go? The Clock menu is there, but it isn't currently visible—you need to click the Clock menu again to see the menus and work with them.

9 Click the Clock menu on the form.

The Clock menu appears again, along with the familiar Type Here tags. You're ready to start customizing the menu now.

Adding Access Keys to Menu Commands

Most applications allow you to access and execute menu commands using the keyboard. For example, in Visual Studio you can open the File menu by pressing the Alt key and then pressing the F key. Once the File menu is open, you can execute the Print command by pressing the P key. The key that you press in addition to the Alt key and the key that you press to execute a command in an open menu is called the *access key*. You can identify the access key in a menu item because it is underlined.

You define an access key by placing an ampersand (&) before the letter.

Visual Studio makes it easy to provide access key support. To add an access key to a menu item, activate the Menu Designer and type an ampersand (&) before the appropriate letter in the menu name. When you open the menu at runtime (when the program is running), your program will automatically support the access key.

tip

By default, Windows 2000 doesn't display the underline for access keys for a program until you press the Alt key for the first time. You can turn off this option on the Effects tab of the Display control panel.

Try adding access keys to the Clock menu now.

Menu Conventions

By convention, each menu title and menu command in an application for Microsoft Windows has an initial capital letter. File and Edit are often the first two menu names on the menu bar, and Help is the last. Other common menu names are View, Format, and Window. No matter what menus and commands you use in your applications, take care to be clear and consistent with them. Menus and commands should be easy to use and should have as much in common with those in other Windows-based applications as possible. As you create menu items, use the following guidelines:

- Use short, specific captions consisting of one or two words at most.
- Assign each menu item an access key. Use the first letter of the item if possible.
- Menu items at the same level must have a unique access key.
- If a command is used as an on/off toggle, place a check mark next to the item when it is active. You can add a check mark by setting the Checked property of the menu command to True using the Properties window.
- Place an ellipsis (…) after a menu command that requires the user to enter more information before the command can be executed. The ellipsis indicates that you will open a dialog box if the user selects this item.

Add access keys

1. Click the Clock menu name on the form, and then click it again.

   A blinking text-editing cursor appears in the Clock menu name. The text-editing cursor allows you to edit your menu name or add the ampersand character (&) for an access key.

2. Press the Left Arrow key to move the text-editing cursor to the beginning of the Clock menu name.

   The cursor blinks before the letter "C" in *Clock*.

3. Type **&** to define the letter "C" as the access key for the Clock menu.

   An ampersand appears in the text box in front of the word Clock.

4. Click the Date command in the menu list, and then click Date a second time to display the text-editing cursor.

5 Type an ampersand before the letter "D".

The letter "D" is now defined as the access key for the Date command.

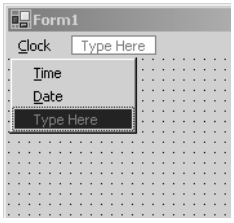
6 Click the Time command in the menu list, and then click the command a second time to display the text-editing cursor.

7 Type an ampersand before the letter "T".

The letter "T" is now defined as the access key for the Time command.

8 Press Enter.

Pressing Enter locks in your text-editing changes. Your form will look like this:



Now you'll practice using the Menu Designer to switch the order of the Date and Time commands on the Clock menu. Changing the order of menu items is an important skill to have; at times you'll think of a better way to define your menus.

Change the order of menu items

1 Click the Clock menu on the form to display its menu items.

Changing the order of menu items is very easy. You simply drag the menu item that you want to move to a new location on the menu. Try it now.

2 Drag the Time menu on top of the Date menu, and then release the mouse button.

Dragging one menu item on top of a second menu item means that you want to place the first menu item ahead of the second menu item on the menu. As quick as that, Visual Studio moved the Time menu item ahead of the Date item.

You've finished creating the user interface for the Clock menu. Now you'll use the menu event procedures to process the user's menu selections in the program.

tip

To delete an unwanted menu item from a menu, click the unwanted item in the menu list and then press the Delete key.

Processing Menu Choices

After menus and commands are configured using the main menu object, they also become new objects in your program. To make the menu objects do meaningful work, you need to write event procedures for them. Menu event procedures typically contain program statements that display or process information on the user interface form and modify one or more menu properties. If more information is needed from the user to process the selected command, an event procedure will often display a dialog box by using one of the Windows Forms dialog box controls or one of the input controls you used in Chapter 3.

In the following exercise, you'll add a label object to your form to display the output of the Time and Date commands on the Clock menu.

Add a label object to the form

Label control

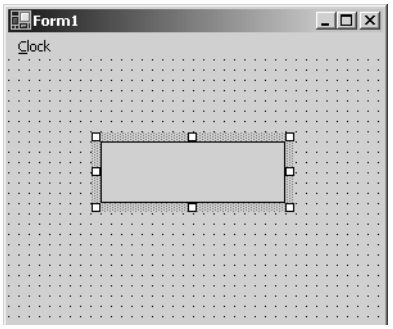
1. Click the Label control in the Toolbox.
2. Draw a medium-sized label in the middle area of the form.

   The label object appears on the form and will bear the name Label1 in the program code.
3. Set the following properties for the label:

Set Label1 properties by using the Properties window.

Object	Property	Setting
Label1	BorderStyle	FixedSingle
	Font	Microsoft Sans Serif, Bold, 14-point
	Text	(empty)
	TextAlign	MiddleCenter

Your form should look like this:



tip

In the following exercises, you'll enter program code to process menu choices. It's OK if you're still a bit hazy on what program codes does and how you use it—you'll learn much more about program statements in Chapters 5 through 7.

Now you'll add program statements to the Time and Date event procedures to process the menu commands.

Edit the menu event procedures

1. Click the Clock menu on the form to display its submenus.
2. Double-click the Time command in the menu to open an event procedure for the command in the Code Editor.

   The MenuItem3_Click event procedure appears in the Code Editor. The name MenuItem3_Click means that Time was the third menu item you created in this project (following Clock and Date), and the _Click syntax means that this is the event procedure that runs when a user clicks the menu item. We'll keep this menu name for now, but I wanted to point out to you that it isn't really that intuitive to use. You can create your own names for objects that describe their function in the program a little more specifically by using the Name property. Although I don't bother with that extra step in the first few exercises, later in the chapter you'll create menu names to establish more understandable and professional programming practices.
3. Type the following program statement:

   ```
   Label1.Text = TimeString
   ```

   This program statement displays the current time (from the system clock) in the Text property of the Label1 object, replacing the previous Label1 text (if any). TimeString is a property that contains the current time formatted for display or printing; you can use TimeString at any time in your programs to display the time accurately down to the second. (TimeString is essentially a replacement for the older QuickBASIC TIME$ statement.)

tip

Visual Basic's TimeString property returns the current system time. You can set the system time by using the Date/Time icon in Windows Control Panel; you can change the system time format by using Control Panel's Regional Settings (or Regional Options) icon.

4 Press the Down Arrow key.

Visual Basic interprets the line and adjusts capitalization and spacing, if necessary. (Visual Basic checks each line for syntax errors as you enter it. You can enter a line by pressing Enter, Up Arrow, or Down Arrow.)

View Designer button

5 Click the View Designer button in Solution Explorer, and then double-click the Date command on the Clock menu.

The MenuItem2_Click event procedure appears in the Code Editor. This event procedure is executed when the user clicks the Date command on the Clock menu.

6 Type the following program statement:

```
Label1.Text = DateString
```

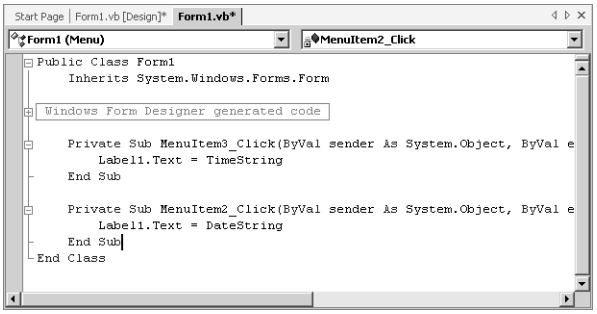
This program statement displays the current date (from the system clock) in the Text property of the Label1 object, replacing the previous Label1 text. The DateString property is also available for general use in your programs. Assign DateString to the Text property of an object whenever you want to display the current date on a form.

tip

Visual Basic's DateString property returns the current system date. You can set the system date by using the Date/Time icon in Control Panel; you can change the system date format by using Control Panel's Regional Settings (or Regional Options) icon.

7 Press the Down Arrow key to enter the line.

Your screen should look like this:



You've finished entering the menu demonstration program. Now you'll save your changes to the project and prepare to run it.

Save All button

Start button

The complete Menu program is located in the c:\vbnetsbs\chap04\menu folder.

8 Click the Save All button on the Standard toolbar.

Run the Menu program

1 Click the Start button on the Standard toolbar.

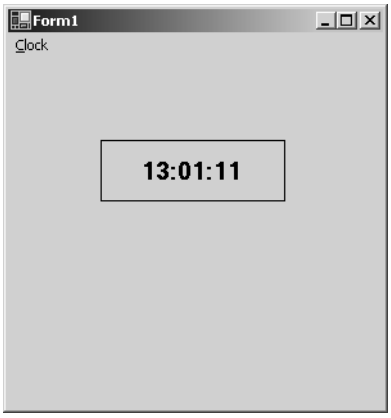
The Menu program runs in the development environment.

2 Click the Clock menu on the menu bar.

The Clock menu appears.

3 Click the Time command.

The current system time appears in the label box, as shown here:



Now you'll try displaying the current date by using the access keys on the menu.

4 Press and release the Alt key.

The Clock menu on the menu bar is highlighted.

5 Press C to display the Clock menu.

The contents of the Clock menu appear.

6 Press D to display the current date.

The current date appears in the label box.

7 Click the Close button on the program's title bar to stop the program.

Congratulations! You've created a working program that makes use of menus and access keys. In the next exercise, you'll learn how to use menus to display standard dialog boxes.

System Clock Properties and Functions

You can use various properties and functions to retrieve chronological values from the system clock. You can use these values to create custom calendars, clocks, and alarms in your programs. The following table lists the most useful system clock functions. For more information, check the Visual Studio online Help.

Property or Function	Description
TimeString	This property returns the current time from the system clock.
DateString	This property returns the current date from the system clock.
Now	This property returns an encoded value representing the current date and time. This property is most useful as an argument for other system clock functions.
Hour (*time*)	This function returns the hour portion of the specified time (0 through 23).
Minute (*time*)	This function returns the minute portion of the specified time (0 through 59).
Second (*time*)	This function returns the second portion of the specified time (0 through 59).
Day (*date*)	This function returns a whole number representing the day of the month (1 through 31).
Month (*date*)	This function returns a whole number representing the month (1 through 12).
Year (*date*)	This function returns the year portion of the specified date.
Weekday (*date*)	This function returns a whole number representing the day of the week (1 is Sunday, 2 is Monday, and so on).

Using Dialog Box Controls

Visual Studio contains seven standard dialog box controls on the Windows Forms tab of the Toolbox. These are provided ready-made so that you don't need to create your own custom dialog boxes for the most common tasks in

Windows applications, such as opening files, saving files, and printing. In many cases, you'll still need to write the program code that connects these dialog boxes to your program, but the user interfaces are built for you and conform to the standards for common use among Windows applications.

The seven standard dialog box controls available to you are listed in the following table. In many respects they are parallel to the objects provided by the CommonDialog control in Visual Basic 6, with a few important exceptions. The PrintPreviewControl control isn't listed here, but you'll find it useful if you use the PrintPreviewDialog control.

Control Name	Purpose
OpenFileDialog	Get the drive, folder name, and filename for an existing file
SaveFileDialog	Get the drive, folder name, and filename for a new file
FontDialog	Let the user choose a new font type and style
ColorDialog	Let the user select a color from a palette
PrintDialog	Let the user set printing options
PrintPreviewDialog	Display a print preview dialog box like Microsoft Word does
PageSetupDialog	Let the user control page setup options, such as margins, paper size, and layout

In the following exercises, you'll add a new menu to the Menu program and practice using the OpenFileDialog and ColorDialog controls. You may either use your existing Menu project or load the Menu project from the practice files folder if you didn't create it from scratch. (The completed version I've named Dialog, to preserve both projects on disk.)

Add OpenFileDialog and ColorDialog controls

1 If you didn't create the Menu project, click the File menu, point to Open, and then click Project, select the Menu project in the c:\vbnetsbs\chap04\menu folder, and then click Open. If the form isn't open, double-click Form1.vb in Solution Explorer to display the form.

Start your upgrades to this program by adding two dialog box controls to the component tray that contains the main menu object. The OpenFileDialog control will let your program open bitmap files, and the ColorDialog control will enable your program to change the color of the clock output. Dialog box controls appear in the component tray because they don't appear on the form at runtime.

OpenFileDialog control

2 Click the OpenFileDialog control on the Windows Forms tab of the Toolbox, and then click the component tray containing the main menu object.

Down scroll arrow

tip

If you don't see the OpenFileDialog in the Toolbox, it may be lower in the Toolbox list hidden from your view. To scroll down in the Toolbox list, click the down scroll arrow next to the Clipboard Ring tab.

An open file dialog object appears in the component tray.

ColorDialog control

3 Click the ColorDialog control on the Windows Forms tab of the Toolbox, and then click the component tray below the form again.

The component tray now looks like this:

Just like the main menu object, the open file dialog and color dialog objects can be customized with property settings.

Now you'll create a picture box object by using the PictureBox control. As you've seen, the picture box object displays artwork on a form. This time, you'll display artwork in the picture box by using the open file dialog box.

Add a picture box object

PictureBox control

1 Click the PictureBox control in the Toolbox.

2 Draw a picture box object on the form, below the label.

3 Use the Properties window to set the SizeMode property of the picture box to StretchImage.

Now you'll use the Menu Designer to add a File menu to the program.

Add a File menu

1 Click the Clock menu on the form, and then click the Type Here tag to the right of the menu.

Now you'll add to the program a File menu that includes Open, Close, and Exit commands.

2 Type **&File** to create a File menu with the letter "F" as an access key.

3. Press the Down Arrow key, and then type **&Open...** to create an Open... command with the letter "O" as an access key.

   The Open command will be used to open Windows bitmaps. Because the command will display a dialog box, you added an ellipsis to the command name.

4. Press the Down Arrow key, and then type **&Close** to create a Close command with the letter "C" as an access key.

   The Close command will be used to close open bitmap files in your program.

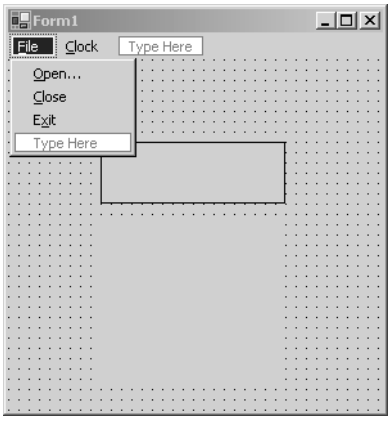
5. Press the Down Arrow key, and then type **E&xit** to create an Exit command with the letter "x" as an access key.

   The Exit command will be used to close the program. Notice that in this case the second letter of the Exit command was used as an access key, which matches how Exit is used in most Windows applications (such as Microsoft Word).

6. Drag the File menu on top of the Clock menu to move it to the first position.

   You can move entire menus, as well as menu commands, with the Menu Designer. It makes sense to have the File menu be the first menu in your program.

   Your form should look like this:



Changing the Object Names for Items on the File Menu

As I mentioned earlier in the chapter, you can change the name of objects on your form, including the names of menu item objects, by changing the Name property for each object that you want. The Name property doesn't change how an object looks at runtime, or what it displays for the user to see, but it does

make objects more readable and recognizable in your program code. Practice changing the names of the objects on the new File menu now so that you can recognize them more clearly in your program code. In future chapters, you'll routinely change the names of the objects you use often.

Change the object names

1. Click the File menu item, open the Properties window, and change the Name property to **mnuFile**. (The Name property appears in the Design category and is surrounded with parentheses.)

   Most programmers begin the names of their objects with a three-character extension that identifies what control or object library the object is related to or derived from. I've used *mnu* here to identify this object as a menu item, and I've used the word *File* to remind myself that the object is the File menu on the menu bar. Later you'll use other three-character extensions to identify different types of objects.

2. Click the Open… menu item, and change its Name property to **mnuOpenItem**.
3. Click the Close menu item, and change its Name property to **mnuCloseItem**.
4. Click the Exit menu item, and change its Name property to **mnuExitItem**.
5. Click the Save All button on the Standard toolbar to save your changes.

Save All button

These names will make your program easier to understand, and you can name the other objects in your project now if you like. (Good candidates would be the label object, the picture box object, and the other menu items.) But as I indicated, naming objects is optional, and the three-character prefix I recommended for menus is just an industry convention that some Visual Basic programmers follow.

Disabling a Menu Command

In a typical application for Windows, not all menu commands are available at the same time. In a typical Edit menu, for example, the Paste command is available only when there is data on the Clipboard. You can disable a menu item by setting the Enabled property for the menu object to False. When a command is disabled, it appears in dimmed type on the menu bar.

In the following exercise, you'll disable the Close command on the File menu. (Close is a command that can be used only after a file has been opened in the program.) Later in the chapter, you'll include a statement in the Open command event procedure that enables the Close command at the proper time.

Disable the Close command

1. Click the Close command on the Menu program's File menu.
2. Open the Properties window, and set the Enabled property for the mnuCloseItem object to False.

Now you'll add a Text Color command to the Clock menu to demonstrate how the color dialog box works. The color dialog box returns a color setting to the program through the Color property. You'll use that property to change the color of the text in the Label1 object.

Add the Text Color command to the Clock menu

1. Click the Clock menu, and then click the Type Here tag at the bottom of it.
2. Type **Text Co&lor...** to add a Text Color command to the menu with an access key of "l".

   The command Text Color is added to the Clock menu. The command contains a trailing ellipsis to indicate that it will display a dialog box when the user clicks it. The access key chosen for this command is "L" because "T" is already used in the menu, for Time. Your access keys won't behave correctly if you use duplicate keys at the same level within a particular menu or duplicate keys at the menu bar level.
3. Use the Properties window to change the Name property of the Text Color command to **mnuTextColorItem**.

Event Procedures That Manage Common Dialog Boxes

To display a dialog box in a program, you need to type the dialog box name with the ShowDialog method in an event procedure associated with the menu command. If necessary, you must also set one or more dialog box properties using program code before opening the dialog box. Finally, you need to use program code to respond to the user's dialog box selections after the dialog box has been manipulated and closed.

In the following exercise, you'll type in the program code for the mnuOpenItem_Click event procedure, the routine that executes when the Open command is clicked. You'll set the Filter property in the OpenFileDialog1 object to define the file type in the Open common dialog box. (You'll specify Windows bitmaps.) Then you'll use the ShowDialog method to display the open file dialog box. After the user has selected a file and closed this dialog box, you'll display the file he or

she selects in a picture box by setting the Image property of the picture box object to the filename the user selected. Finally you'll enable the Close command so that the user can unload the picture if he or she wants.

Edit the Open command event procedure

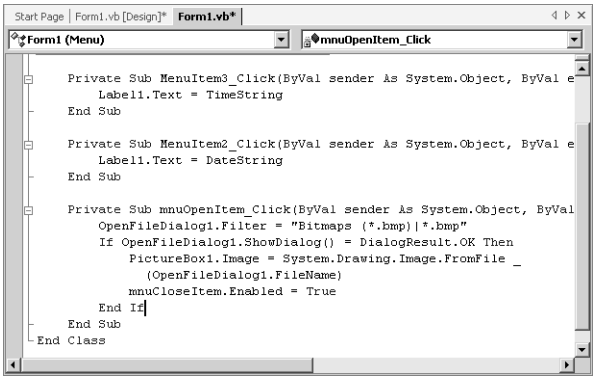
1. Double-click the Open command on the Menu project's File menu.

   The mnuOpenItem_Click event procedure appears in the Code Editor.

2. Type the following program statements in the event procedure, between the Private Sub and End Sub statements. Be sure to type each line exactly as it is printed here, and press the Down arrow key after the last line.

```
OpenFileDialog1.Filter = "Bitmaps (*.bmp)|*.bmp"
If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
    PictureBox1.Image = System.Drawing.Image.FromFile _
      (OpenFileDialog1.FileName)
    mnuCloseItem.Enabled = True
End If
```

Your screen should look like the illustration shown here:



The first three statements in the event procedure refer to three different properties of the open file dialog object. The first statement uses the Filter property to define a list of valid files. (In this case, the list has only one item: *.bmp.) This is important for the Open dialog box because a picture box object is designed for six types of files: bitmaps (.bmp files), Windows metafiles (.emf and .wmf files), icons (.ico files), Joint Photographic Experts Group format (.jpg and .jpeg files), Portable Network Graphics format (.png files), and Graphics Interchange Format (.gif files). (Attempting to display a .txt file in an image object would cause a runtime error, for example.)

The Filter property defines the file types that will be listed in the Open dialog box.

To add additional items to the Filter list, you can type a pipe symbol (|) between items. For example,

```
OpenFileDialog1.Filter = "Bitmaps (*.bmp)|*.bmp|Metafiles (*.wmf)|*.wmf"
```

allows both bitmaps and Windows metafiles to be chosen in the Open dialog box.

The second statement in the event procedure displays the Open dialog box in the program. ShowDialog is a new method in Visual Basic .NET—it is similar to the Show method in Visual Basic 6, but it can be used with any Windows Form. The ShowDialog method returns a result, named DialogResult, which indicates the button on the dialog box the user clicked. To determine whether the user clicked the Open button, an If…Then decision structure is used to check whether the returned result equals DialogResult.OK. If it does, then a valid bmp file path should be stored in the FileName property of the open file dialog object. (You'll learn more about the syntax of If…Then decision structures in Chapter 6.)

The third statement uses the filename selected in the dialog box by the user. When the user selects a drive, folder, and filename and then clicks Open, the complete path is passed to the program through the OpenFileDialog1.FileName property. The System.Drawing.Image.FromFile method, a method that loads electronic artwork, is then used to copy the specified Windows bitmap into the picture box object. (I wrapped this statement with the line continuation character because it was rather long.)

The fourth statement in the procedure enables the Close command on the File menu. Now that a file has been opened in the program, the Close command should be available so that users can close the file.

Now you'll type in the program code for the mnuCloseItem_Click event procedure, the routine that runs when the Close command on the File menu is clicked.

Edit the Close command event procedure

1. Display the form again, and then double-click the Close command on the File menu.

   The event procedure for the Close command appears in the Code Editor.

2. Type the following program statements in the event procedure, between the Private Sub and End Sub statements.

Use the Nothing keyword to clear the Image property of the picture box object.

```
PictureBox1.Image = Nothing
mnuCloseItem.Enabled = False
```

The first statement closes the open Windows bitmap by clearing the picture box object's Image property. The Nothing keyword is used here to disassociate the current bitmap object from the Image property—in other words, Nothing sets the property to zero, and the picture disappears. (You'll use Nothing later in this book to reset other object variables and properties too.) The second statement dims the Close command on the File menu because there is no longer an open file. This program statement is the equivalent to using the Properties window to change the Enabled property from True to False.

Now you'll type in the program code for the mnuExitItem_Click event procedure, the routine that stops the program when the Exit command on the File menu is clicked.

Edit the Exit command event procedure

1. Display the form again, and then double-click the Exit command on the File menu.

   The event procedure for the Exit command appears in the Code Editor.

2. Type the following program statement in the event procedure, between the Private Sub and End Sub statements:

   ```
   End
   ```

   The End statement stops the program when the user is finished. (It might look familiar by now.)

Edit the Text Color command event procedure

1. Display the form again, and then double-click the new Text Color command on the Clock menu.

   The event procedure for the Text Color command appears in the Code Editor.

2. Type the following program statements in the event procedure:

   ```
   ColorDialog1.ShowDialog()
   Label1.ForeColor = ColorDialog1.Color
   ```

tip

The Color dialog box can be used to set the color of any user interface element that supports color. Other possibilities include the form background color, the colors of shapes on the form, and the foreground and background colors of objects.

The first program statement uses the ShowDialog method to open the color dialog box. As you learned earlier in this chapter, ShowDialog is the method you use to open any form as a dialog box, including a form created by one of the standard dialog box controls that Visual Studio provides. The second statement in the event procedure assigns color that the user selected in the dialog box to the ForeColor property of the Label1 object. You might remember Label1 from earlier in this chapter—it's the label box you used to display the current time and date on the form. You'll use the color returned from the color dialog box to set the color of the text in the label.

Save All button

3 Click the Save All button on the Standard toolbar to save your changes.

Controlling Color Choices by Setting Color Dialog Box Properties

If you want to further customize the color dialog box, you can control just what color choices the dialog box presents to the user when the dialog box opens. You can adjust these color settings by using the Properties window, or by setting properties using program code before you display the dialog box with the ShowDialog method. The following table describes the most useful properties of the ColorDialog control. Each property should be set with a value of True to enable the option, or False to disable the option.

Property	Meaning
AllowFullOpen	Set to True to enable the Define Custom Colors button in the dialog box.
AnyColor	Set to True if the user can select any color shown in the dialog box.
FullOpen	Set to True if you want to display the Custom Colors area when the dialog box first opens.
ShowHelp	Set to True if you want to enable the Help button in the dialog box.
SolidColorOnly	Set to True if you only want the user to select solid colors (dithered colors will be disabled).

Now you'll run the Menu program and experiment with the menus and dialog boxes you've created.

Run the Menu program

Start button

The complete program is named Dialog, and is located in the c:\vbnetsbs\chap04\dialog folder.

1 Click the Start button on the Standard toolbar.

The program runs, and both the File and Clock menus appear on the menu bar.

2 On the form's File menu, click Open.

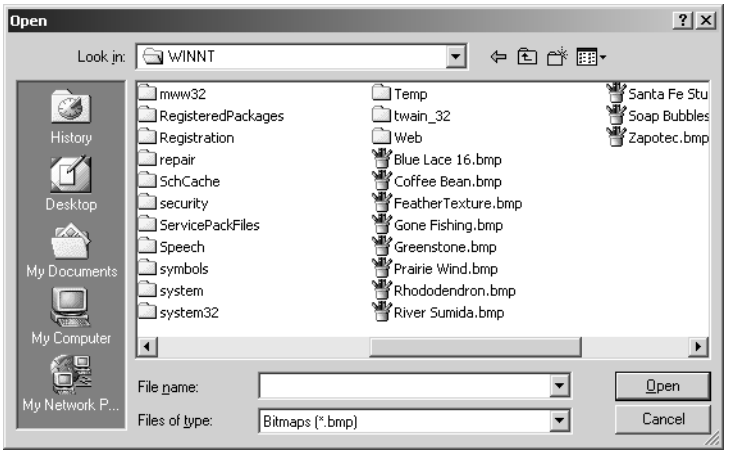
The Open dialog box appears. It looks great, doesn't it? Notice the Bitmaps (*.bmp) entry in the Files Of Type box. You defined this entry with the statement

```
OpenFileDialog1.Filter = "Bitmaps (*.bmp)|*.bmp"
```

in the mnuOpenItem_Click event procedure. The first part of the text in quotes—Bitmaps (*.bmp)—specifies which items are listed in the Files Of Type box. The second part—*.bmp—specifies the filename extension of the files that are to be listed in the dialog box.

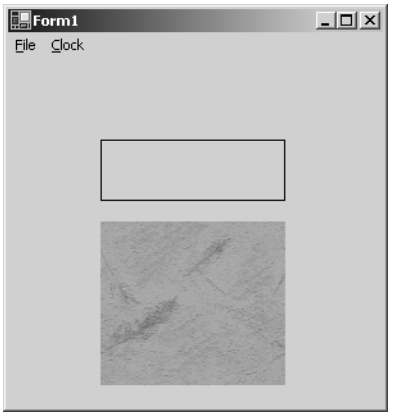
3 Open the c:\windows folder (or c:\winnt folder) on your hard disk, and browse through the long list of folders to get to the bitmap files.

A standard collection of bitmaps appears. Most of these files were included with Windows, and you may have added to the collection yourself.



4 Select one of the bitmap files and then click the Open button.

A picture of the bitmap appears in the picture box. (I've selected the FeatherTexture.bmp file.) Your form looks like this:



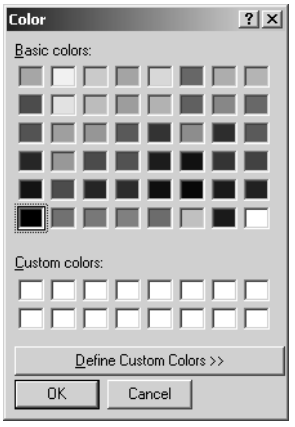
Now you'll practice using the Clock menu.

5 On the Clock menu, click the Time command.

The current time appears in the label box.

6 On the Clock menu, click the Text Color command.

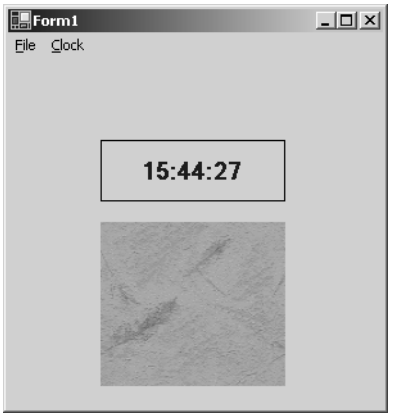
The Color dialog box appears, as shown here:



The Color dialog box contains elements that let you change the color of the clock text in your program. The current color setting, black, is selected.

7 Click the blue box, and then click the OK button.

The Color dialog box closes, and the color of the text in the clock label changes to blue.



8 On the Clock menu, click the Date command.

The current date is displayed in blue type. Now that the text color has been set in the label, it will remain blue until the color is changed again or the program closes.

9 Click the File menu.

Notice that the Close command is now enabled. (You enabled it in the mnuOpenItem_Click event procedure by using the statement mnuCloseItem.Enabled = True.)

10 Press **C** (the access key for Close) to close the bitmap image.

The file closes, and the Windows bitmap is removed. (This is the Nothing keyword at work.)

11 Click the File menu.

The Close command is now dimmed because there is no bitmap in the picture box.

12 Click the Exit command.

The program closes, and the Visual Studio development environment appears.

Adding Nonstandard Dialog Boxes to Programs

What if you need to add a dialog box to your program that isn't provided by one of the seven dialog box controls in Visual Studio? No problem—but you'll need to do a little extra design work. As you'll learn in future chapters, a Visual Basic program can use more than one form to receive and display information. To create nonstandard dialog boxes, you'll need to add new forms to your program, add input and output objects, and process the dialog box clicks in your program code. (These techniques will be discussed in Chapter 15.) In the next chapter, you'll learn how to use two handy dialog boxes that are specifically designed for receiving text input (InputBox) and displaying text output (MsgBox). These dialog boxes will help bridge the gap between the dialog box controls and the dialog boxes that you need to create on your own.

That's it! You've learned several important commands and techniques for creating menus and dialog boxes in your programs. After you learn more about program code, you'll be able to put these skills to work in your own programs.

One Step Further: Assigning Shortcut Keys to Menus

The MainMenu control also lets you assign *shortcut keys* to your menus. Shortcut keys are key combinations that a user can press to activate a command without using the menu bar. For example, on a typical Edit menu in an application for Windows (Microsoft Word), you can copy selected text to the Clipboard by pressing Ctrl+C. The MainMenu control's Shortcut property allows you to customize this setting. Try assigning two shortcut keys to the Clock menu in the Menu program now.

Assign shortcut keys to the Clock menu

1 Make sure that your program has stopped running, and is in design mode. You can modify a program only when it isn't running.

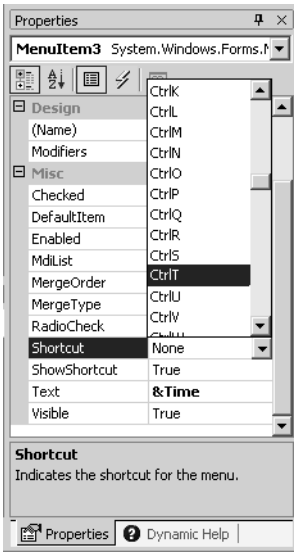
2 Click the Clock menu, and then click the Time command to highlight it.

Before you set the shortcut key for a menu command, you must select it. You assign a shortcut key by setting the Shortcut property for the command using the Properties window. The main menu object provides many standard shortcut key settings for you automatically.

You'll assign Ctrl+T as the shortcut key for the Time command.

3 Open the Properties window, click the Shortcut property, click the drop-down arrow in second column, scroll down the list of shortcut key settings, and then click CtrlT.

The Properties window will look like this:



tip

Visual Basic normally displays the shortcut key combination in the menu when you run the program, to give users a hint about which keys to press. To hide shortcut key combinations from the user (if you are running out of space), set the ShowShortcut property to False. The shortcut key will still work, but users won't see a visual reminder for it.

Start button

4. Click the Date command, and then change its Shortcut property setting to CtrlD.

   Now you'll run the program and try the shortcut keys.

5. Click the Start button on the Standard toolbar.
6. Press Ctrl+T to run the Time command.

   The current time appears in the program.

7. Press Ctrl+D to run the Date command.

   The current date appears in the program.

8. Click the Clock menu.

   The shortcut keys are listed beside the Time and Date commands, as shown in the following illustration. Visual Basic adds these key combinations when you define the shortcuts by using the Shortcut property.

9. On the program's File menu, click the Exit command.

   The program stops, and the development environment appears.

Chapter 4 Quick Reference

To	Do this
Create a menu item	Click the MainMenu control, and draw a menu on your form. Click the Type Here tag on your form, and type the name of the menu.
Add an access key to a menu item	Click the menu item twice to display the text editing cursor, and then type an ampersand (&) before the letter you want to use as an access key.
Assign a shortcut key to a menu item	Set the Shortcut property of the menu item using the Properties window. A list of common shortcut keys is provided.
Change the order of menu items	Drag the menu item you want to move to a new location.
Use a standard dialog box in your program	Add one of the seven standard dialog box controls to your form, and then customize it with property settings and program code.
Display an open file dialog box	Add the OpenFileDialog control to your form. Display the dialog box with the ShowDialog method. The FileName property contains the name of the file selected.
Display a Color dialog box	Add the ColorDialog control to your form. Display the dialog box with the ShowDialog method. The Color property contains the color the user selected.
Disable a menu	Set the Enabled property of a menu item to False using the Properties window.
Enable a menu command by using program code	Use the program statement `mnuCloseItem.Enabled = True` but substitute your command name for *mnuCloseItem*.
Clear an image from a picture box	Use the program statement `PictureBox1.Image = Nothing`