HOUR 2

Navigating Visual Basic .NET

The key to expanding your knowledge of Visual Basic .NET is to become as comfortable as possible—as quickly as possible—with the Visual Basic .NET design environment. Just as a carpenter doesn't think much about hammering a nail into a piece of wood, performing actions such as saving projects, creating new forms, and setting object properties should become second nature to you. The more comfortable you are with the tools of Visual Basic .NET, the more you can focus your energies on what you're creating with the tools.

In this hour, you'll learn how to customize your design environment by moving, docking, floating, hiding, and showing design windows, as well as how to customize menus and toolbars. You'll even create a new toolbar from scratch. After you've gotten acquainted with the environment, I'll teach you about projects and the files that they're made of (taking you beyond what was briefly discussed in Hour 1, "Jumping In with Both Feet: A Visual Basic .NET Programming Tour"), and I'll introduce you to the design windows with which you'll work most frequently. Finally, I'll show you how to get help when you're stuck.

The highlights of this hour include the following:

- Navigating Visual Basic .NET
- Using the Visual Studio .NET Start Page to open and create projects
- Showing, hiding, docking, and floating design windows
- Customizing menus and toolbars
- Adding controls to a form using the toolbox
- Viewing and changing object attributes using the Properties window
- Working with the many files that make up a project
- How to get help

Using the Visual Studio .NET Start Page

By default, the Visual Studio .NET Start Page shown in Figure 2.1 is the first thing you see when you start Visual Basic (if Visual Basic isn't running, start it now). The Visual Studio .NET Start Page is a gateway for performing tasks with Visual Basic .NET. From this page, you can open previously edited projects, create new projects, and edit your user profile.

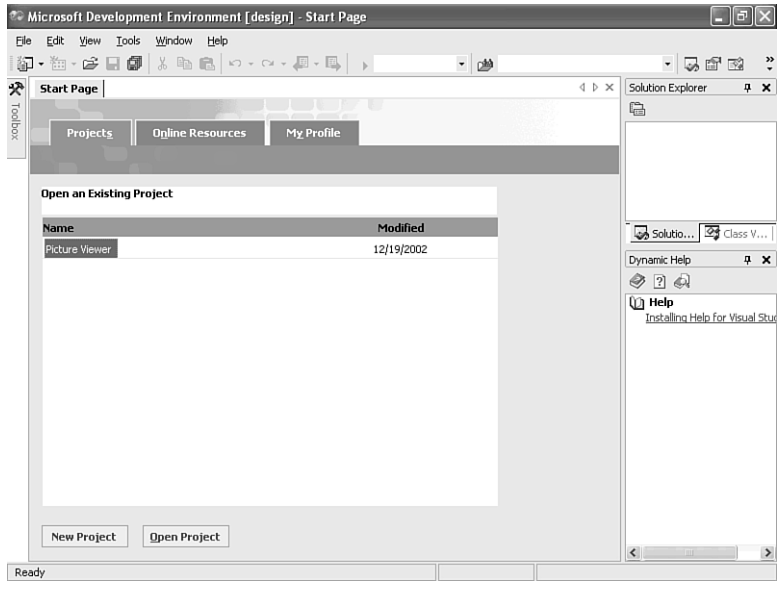


FIGURE 2.1
The Visual Studio .NET Start Page is the default entry point for all .NET languages.

From this page, you can have Visual Basic .NET load the last solution you edited, show the Open Project dialog box, show the New Project dialog box, or show an empty design environment. To view or edit the startup options, choose Options from the Tools menu to

display the Options dialog box shown in Figure 2.2. The General section of the Environment folder is selected when the Options dialog box first appears. This section happens to contain the At Startup option. If the Visual Studio .NET Start Page doesn't appear when you start Visual Studio .NET, verify the settings on the Options dialog box; you might need to change At Startup to Show Start Page.

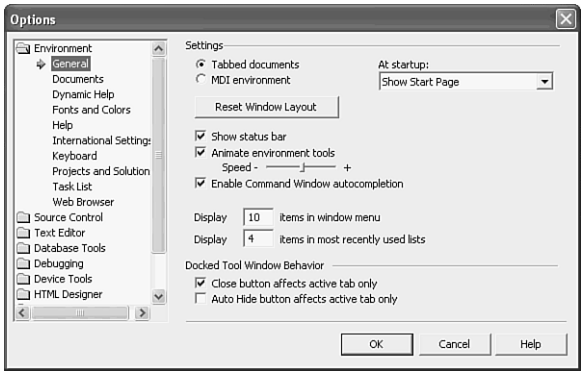


FIGURE 2.2
Use the At Startup setting to control the first thing you see when Visual Studio starts.

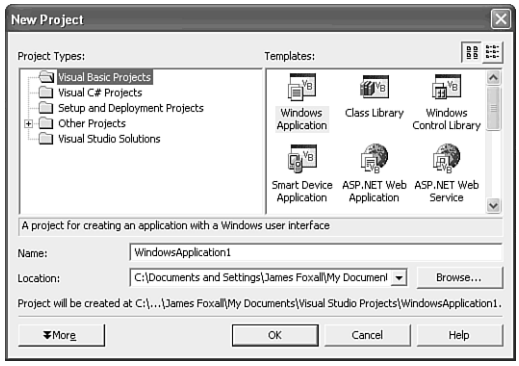
Creating New Projects

To create new projects, click the New Project button in the lower left of the Visual Studio .NET Start Page. This shows the New Project dialog box shown in Figure 2.3. The Project Types list varies from machine to machine, depending on which products of the Visual Studio .NET family are installed. Of course, we're interested only in the Visual Basic Project types in this book.

You can create many types of projects with Visual Basic .NET, but this book focuses mostly on creating Windows Applications, perhaps the most common of the project types. You will learn about some of the other project types as well, but when you're told to create a new project, make sure that the Windows Application icon is selected unless you're told otherwise.

When you create a new project, be sure to enter a name for it in the Name text box before clicking OK or double-clicking a project type icon. This ensures that the project is created with the proper path and filenames, eliminating work you would otherwise have to do to change these values later. After you specify a name, you can create the new project either by double-clicking the project type template icon or by clicking an icon once to select it and then clicking OK. After you've performed either of these actions, the New Project dialog box closes and a new project of the selected type is created.

FIGURE 2.3
Use the New Project dialog box to create Visual Basic .NET projects from scratch.



By default, Visual Studio saves all your projects in subfolders of your My Documents folder. The hierarchy used by Visual Basic .NET is

```
\My Documents\Visual Studio Projects\<Project Name>
```

Notice how the name you give your project is used as its folder name. This makes it easy to find the folders and files for any given project and is one reason that you should always give your projects descriptive names. You can use a path other than the default by specifying it on the New Project dialog box, but you probably won't often need to do this if you develop alone. If you're on a team of developers, however, you might choose to locate your projects on a shared drive so that others can access the source files.

You can create a new project at any time (not just when starting Visual Studio .NET) by opening the New submenu on the File menu and choosing Project. You can't have multiple projects open at once, however, so if you choose to open a project, Visual Basic .NET asks whether you want to save changes for the current project and then closes it.

After you enter a project name and choose a location, you click OK to create the project. Visual Studio .NET then creates the necessary folders and source files, as well as displaying the project in the IDE for you to begin working with it.

Opening an Existing Project

Over time, you'll open more projects than you create. There are essentially two ways to open projects from the Visual Studio .NET Start Page.

If it's a project you've recently opened, the project name will appear in a list within a rectangle in the middle of the Start Page (as Picture Viewer does in Figure 2.1). Because

the name displayed for the project is the one given when it was created, this is yet another reason to give your projects descriptive names. Clicking a project name opens the project. I'd venture to guess that you'll use this technique 95% of the time.

To open a project for the first time (such as when opening sample projects), click the Open Project button on the Visual Studio Start Page. This displays a standard dialog box that you can use to locate and select a project file.

As with creating new projects, you can open an existing project at any time, not just when starting Visual Basic, by selecting File, Open. Remember, you can have only one project open at a time, so opening a project causes the current project to be closed. Again, if you've made changes to the current project, you'll get a chance to save them before it's closed.

Navigating and Customizing the Visual Basic .NET Environment

Visual Basic .NET lets you customize many of its interface elements, such as windows and toolbars, enabling you to be more efficient in the work that you do. Create a new Windows Application now by opening the File menu, clicking New, and then choosing Project. This project will be used to illustrate manipulating the design environment. Name this project **Environment Tutorial**. (This exercise won't create anything reusable, but it will help you learn how to navigate the design environment.) Your screen should look like the one shown in Figure 2.4.

Your screen might not look exactly like that shown in Figure 2.4, but it'll be close. By the time you've finished this hour, you'll be able to change the appearance of the design environment to match this figure—or to any configuration you prefer.

Working with Design Windows

Design windows, such as the Properties window and Solution Explorer shown in Figure 2.4, provide functionality for building complex applications. Just as your desk isn't organized exactly like that of your co-workers, your design environment doesn't have to be the same as anyone else's either.

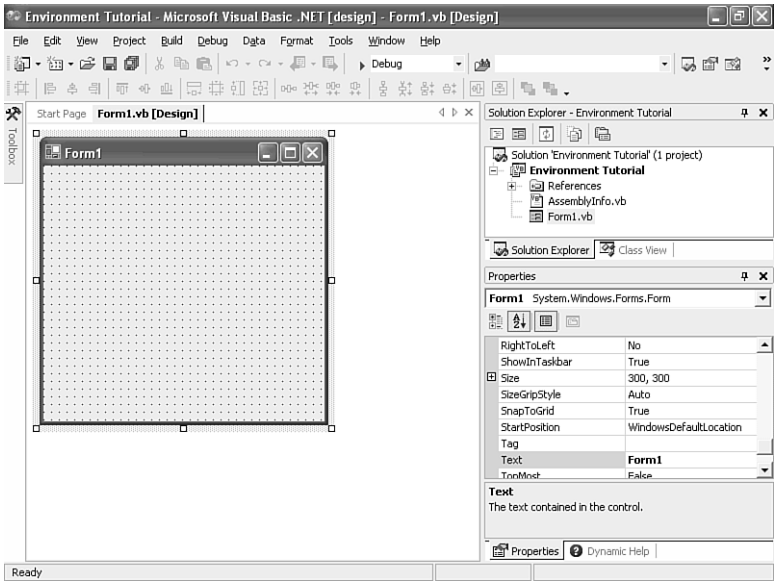


FIGURE 2.4
This is pretty much how the integrated development environment (IDE) appears when you first install Visual Studio.

A design window can be placed into one of four primary states:

- Closed—The window is not visible.
- Floating—The window appears floating over the IDE.
- Docked—The window appears docked to an edge of the IDE.
- Automatically hidden—The window is docked, but it hides itself when not in use.

Showing and Hiding Design Windows

When a design window is closed, it doesn't appear anywhere. There is a difference between being closed and being automatically hidden, as you'll learn shortly. To display a closed or hidden window, choose the corresponding menu item from the View menu. For example, if the Properties window isn't displayed in your design environment, you can display it by choosing Properties Window on the View menu (or press its keyboard shortcut—F4). Whenever you need a design window and can't find it, use the View menu to display it. To close a design window, click its Close button (the button on the right side of the title bar with the X on it), just as you would to close an ordinary window.

Floating Design Windows

Floating design windows are visible windows that float over the workspace, as shown in Figure 2.5. Floating windows are like typical application windows in that you can drag them around and place them anywhere you please, even on other monitors when you're using a multiple-display setup. In addition to moving a floating window, you can also change its size by dragging a border.

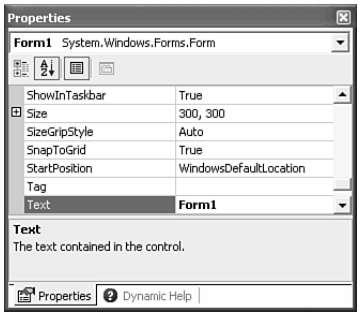


FIGURE 2.5
Floating windows appear over the top of the design environment.

Docking Design Windows

Visible windows appear docked by default. A *docked* window is a window that appears attached to the side, top, or bottom of the work area or to some other window. The Properties window in Figure 2.4, for example, is docked to the right side of the design environment. To make a floating window become a docked window, drag the title bar of the window toward the edge of the design environment to which you want to dock the window. As you drag the window, you'll drag a rectangle that represents the outline of the window. When you approach an edge of the design environment, the rectangle will change shape and "stick" in a docked position. If you release the mouse while the rectangle appears this way, the window will be docked. Although this is hard to explain, it's very easy to do.

You can size a docked window by dragging its edge opposite the side that's docked. If two windows are docked to the same edge, dragging the border between them enlarges one while shrinking the other.

To try this, you'll need to float a window that's already docked. To float a window, you "tear" the window away from the docked edge by dragging the title bar of the docked window away from the edge to which it's docked. Note that this technique won't work if a window is set to Auto Hide (which is explained next). Try docking and floating windows now by following these steps:

1. Ensure that the Properties window is currently displayed (if it's not, show it by pressing F4). Make sure that the Properties window isn't set to Auto Hide by right-clicking its title bar and deselecting Auto Hide from the shortcut menu (if it's selected), as shown in Figure 2.6.
2. Drag the title bar of the Properties window away from the docked edge. When the rectangle representing the border of the window changes shape, release the mouse button. The Properties window should now float.

3. Dock the window once more by dragging the title bar toward the right edge of the design environment. Again, release the mouse button when the rectangle changes shape. If you dropped the window directly on top of an existing window, you'll create a tabbed window. This is discussed shortly.

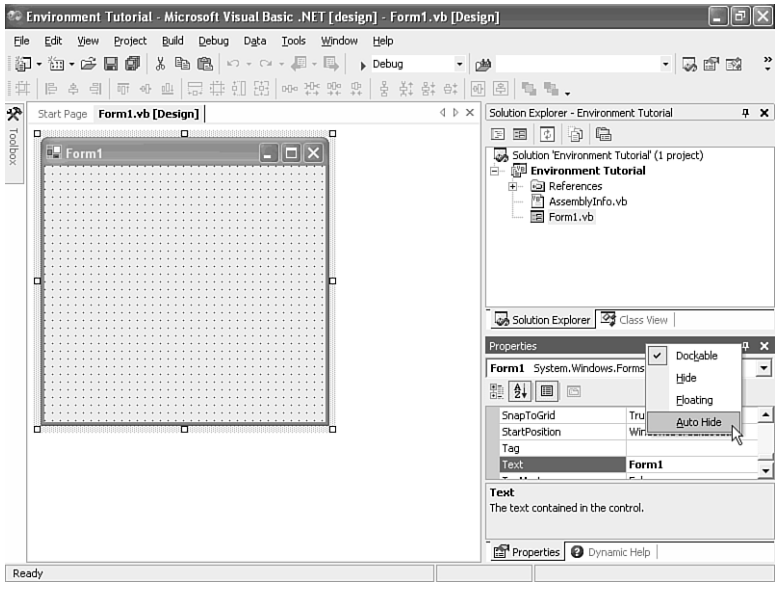


FIGURE 2.6
You can't float a window that's set to Auto Hide.

If you don't want a floating window to dock, regardless of where you drag it to, right-click the title bar of the window and choose Floating from the context menu. To allow the window to be docked again, right-click the title bar and choose Dockable.

Auto Hiding Design Windows

A relatively new feature of Visual Basic .NET's design environment is the capability to auto hide windows. Although you might find this a bit disconcerting at first, after you get the hang of things, this is a very productive way to work because your workspace is freed up, yet design windows are available by simply moving the mouse. Windows that are set to Auto Hide are always docked; you can't set a floating window to Auto Hide. When a window auto hides, it appears as a vertical tab on the edge to which it's docked—much like minimized applications are placed in the Windows taskbar.

Look at the left edge of the design environment in Figure 2.6. Notice the vertical tab titled Toolbox. This tab represents an auto-hidden window. To display an auto-hidden

window, move the pointer over the tab representing the window. When you move the pointer over a tab, Visual Basic displays the design window so that you can use its features. When you move the pointer away from the window, the window automatically hides itself—hence the name. To make any window hide itself automatically, right-click its title bar and select Auto Hide from its shortcut menu. You can also click the little picture of a pushpin appearing in the title bar next to the Close button to toggle the window's Auto Hide state.

Performing Advanced Window Placement

The techniques discussed so far in this section have been basic methods for customizing your design environment. Things can become a bit more complicated if you want them to. Such complication presents itself primarily as the capability to create tabbed floating windows like the one shown in Figure 2.5. Notice that at the bottom of the floating window is a set of tabs. Clicking a tab shows its corresponding design window, replacing the window currently displayed. These tabs are created much the same way in which you dock and undock windows: by dragging and dropping.

For instance, to make the Solution Explorer window a floating window of its own, you would drag the Solution Explorer window tab away from the floating window. As you do so, a rectangle appears, showing you the outline of the new window. Where you release the rectangle determines whether you dock the design window being dragged or whether you make the window float. To change a design window into a new tab of an already floating window, drag its title bar and drop it in the title bar of a window that's already floating.

In addition to creating tabbed floating windows, you can dock two floating windows together. To do this, drag the title bar of one window over another window (other than over the title bar) until the shape changes, and then release the mouse. Figure 2.7 shows two floating windows that are docked to one another and a third window that's floating by itself.

Using all the techniques discussed so far, you can tailor the appearance of your design environment in all sorts of ways. There is no one best configuration. You'll find that different configurations work better for different projects and in different stages of development. For instance, when I'm designing the interface of a form, I want the toolbox to stay visible but out of my way, so I tend to make it float, or I turn off its Auto Hide property and leave it docked to the left edge of the design environment. However, after the majority of the interface elements have been added to a form, I want to focus on code. Then I dock the toolbox and make it Auto Hide itself; it's there when I need it, but it's out of the way when I don't. Don't be afraid to experiment with your design windows, and don't hesitate to modify them to suit your changing needs.

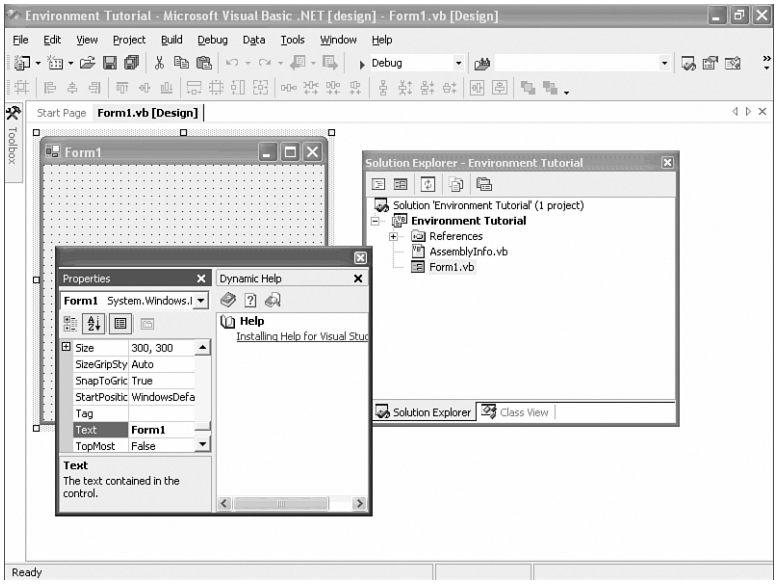


FIGURE 2.7
Floating, docked, floating and docked—there are a lot of possibilities!

Working with Toolbars

Toolbars are the mainstay for performing functions quickly in almost every Windows program (you'll probably want to add them to your own programs at some point, and in Hour 9, "Adding Menus and Toolbars to Forms," you'll learn how). Every toolbar has a corresponding menu item, and buttons on toolbars are essentially shortcuts to their corresponding menu items. To maximize your efficiency when developing with Visual Basic, you should become familiar with the available toolbars. As your Visual Basic skills progress, you can customize existing toolbars and even create your own toolbars to more closely fit the way you work.

Showing and Hiding Toolbars

Visual Basic .NET includes a number of built-in toolbars you can use when creating projects. Two toolbars are visible in most of the figures shown so far in this hour. The one on the top is the Standard toolbar, which you'll probably want displayed all the time. The second toolbar is the Layout toolbar, which provides useful tools for building forms.

Visual Basic 6 had approximately 5 toolbars; Visual Basic .NET, on the other hand, has more than 20! The toolbars you'll use most often as a new Visual Basic developer are the Standard, Text Editor, and Debug toolbars. Therefore, this hour discusses each of these. In addition to these predefined toolbars, you can create your own custom toolbars to contain any functions you think necessary. You'll learn how to do this later in this hour.

To show or hide a toolbar, choose View, Toolbars to display a list of available toolbars. Toolbars currently displayed appear selected (see Figure 2.8). Click a toolbar name to toggle its visible state.

Right-click any visible toolbar to quickly access the list of available toolbars.

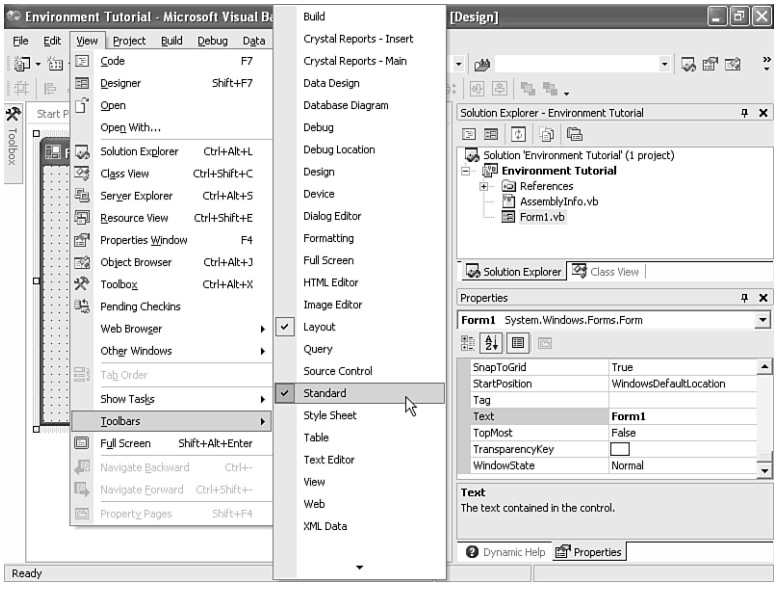


FIGURE 2.8
Hide or show toolbars to make your work more efficient.

Docking and Resizing Toolbars

Just as you can dock and undock Visual Basic's design windows, you can dock and undock the toolbars. Unlike the design windows, however, Visual Basic .NET's toolbars don't have a title bar that you can click and drag when they're in a docked state. Instead, each docked toolbar has a *drag handle* (a stack of short horizontal lines along its left edge)**.** To float (undock) a toolbar, click and drag the grab handle away from the docked edge. Once a toolbar is floating, it has a title bar. To dock a floating toolbar, click and drag its title bar to the edge of the design environment to which you want it docked. This is the same technique you use to dock design windows.

A shortcut for docking a floating toolbar is to double-click its title bar.

Although you can't change the size of a docked toolbar, you can resize a floating toolbar (a floating toolbar behaves like any other normal window). To resize a floating toolbar, move the pointer over the edge you want to stretch and then click and drag to the border to change the size of the toolbar.

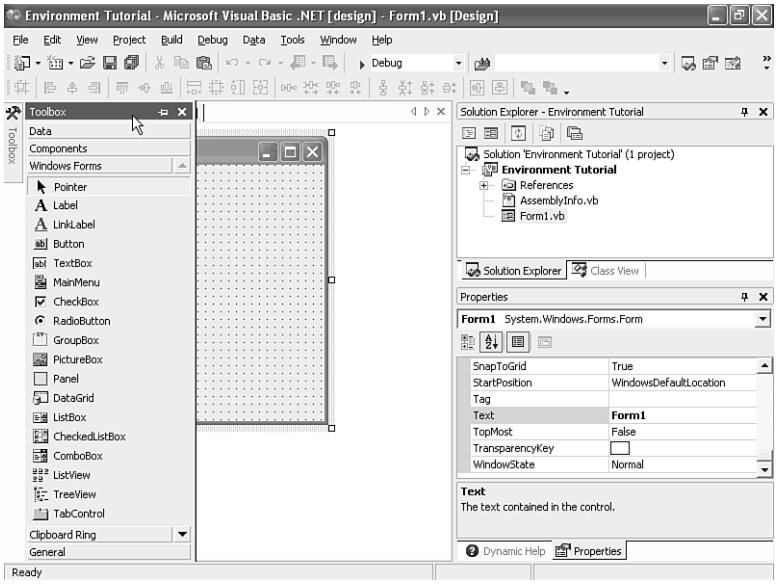
Adding Controls to a Form Using the Toolbox

The IDE offers some fantastic tools for building a graphical user interface (GUI) for your applications. Most GUIs consist of one or more forms (Windows) with various elements on the forms, such as text boxes and list boxes. The toolbox is used to place controls onto a form. The default toolbox you see when you first open or create a Visual Basic project is shown in Figure 2.9. The buttons labeled Data, Components, Windows Forms, and so on are actually tabs, although they don't look like standard tabs. Clicking any of these tabs causes a related set of controls to appear. The default tab is the Windows Forms tab, and it contains many great controls you can place on Windows forms (the forms used to build Windows applications, in contrast to Web applications discussed in Hour 23, "Introduction to Web Development"). All the controls that appear by default on the tabs are included with Visual Basic, and these controls are discussed in detail in Hour 7, "Working with Traditional Controls," and Hour 8, "Using Advanced Controls."

You can add a control to a form in one of three ways:

- In the toolbox, click the tool representing the control that you want to place on a form, and then click and drag on the form where you want the control placed (you're essentially drawing the border of the control). The location at which you start dragging is used for the upper-left corner of the control, and the point at which you release the mouse button and stop dragging becomes the lower-right corner.
- Double-click the desired control type in the toolbox. When you double-click a control in the toolbox, a new control of the selected type is placed in the upper-left corner of the form. The control's height and width are set to the default height and width of the selected control type.
- Drag a control from the toolbox and drop it on a form.

FIGURE 2.9
The standard toolbox contains many useful controls you can use to build robust user interfaces.



If you prefer to draw controls on your forms by clicking and dragging, I strongly suggest that you dock the toolbox to the right or bottom edge of the design environment or float it. The toolbar tends to interfere with drawing controls when it's docked to the left edge because it obscures part of the form.

The very first item on the Windows Forms tab, titled Pointer, isn't actually a control. When the pointer item is selected, the design environment is placed in a select mode rather than in a mode to create a new control. With the pointer item selected, you can select a control by clicking it to display all its properties in the Properties window—this is the default behavior.

Setting Object Properties Using the Properties Window

When developing the interface of a project, you'll spend a lot of time viewing and setting object properties using the Properties window (see Figure 2.10). The Properties window contains four items:

- An object drop-down list
- A list of properties

- A set of tool buttons used to change the appearance of the properties grid
- A section showing a description of the selected property

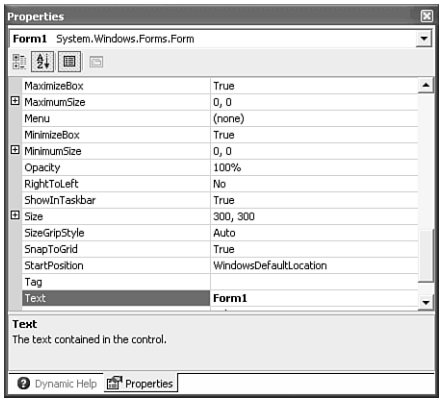


FIGURE 2.10
Use the Properties window to view and change properties of forms and controls.

Selecting an Object and Viewing Its Properties

The drop-down list at the top of the Properties window contains the name of the form with which you're currently working and all the controls (objects) on the form. To view the properties of a control, select it from the drop-down list or click the control on the form. You must have the pointer item selected in the toolbox to click an object to select it.

Viewing and Changing Properties

The first two buttons in the Properties window (Categorized and Alphabetic) enable you to select the format in which you view properties. When you select the Alphabetic button, the selected object's properties appear in the Properties window in alphabetical order. When you click the Categorized button, all the selected object's properties are listed by category. For example, the Appearance category contains properties such as BackColor and BorderStyle. When working with properties, select the view you're most comfortable with and feel free to switch back and forth between the views.

The Properties pane of the Properties window is used to view and set the properties of a selected object. You can set a property in one of the following ways:

- Type in a value
- Select a value from a drop-down list
- Click a Build button for property-specific options

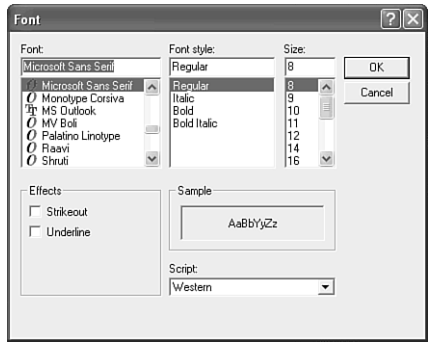
Many properties can be changed by more than one of these methods. For example, color properties supply a dropdown list of colors, but you can enter a numeric color value as well.

To better understand how changing properties works, follow these steps:

1. Start by creating a new Windows Application project. Name this project **Changing Properties**.
2. Add a new text box to a form by double-clicking the TextBox tool in the toolbox. You're now going to change a few properties of the new text box.
3. Select the Name property in the Properties window by clicking it. (If your properties are alphabetic, it will be at the top of the list, not with the Ns.) Type in a name for the text box—call it **txtComments**.
4. Click the BorderStyle property and try to type in the word **Big**. You can't; the BorderStyle property supports selecting values from a list only. You can, however, type a value that exists in the list. When you selected the BorderStyle property, a drop-down arrow appeared in the value column. Click this arrow now to display a list of the values that the BorderStyle property accepts. Select FixedSingle and notice how the appearance of the text box changes. To make the text box appear three-dimensional again, open the drop-down list and select Fixed3D.
5. Select the BackColor property, type in some text, and press the Tab key to commit your entry. Visual Basic .NET displays an Invalid Property Value error. This happens because although you can type in text, you're restricted to entering specific values. In the case of BackColor, the value must be a named color or a number that falls within a specific range. Click the drop-down arrow of the BackColor property and select a color from the drop-down list. (Selecting colors using the color palette is discussed later in this hour, and detailed information on using colors is provided in Hour 18, "Working with Graphics").
6. Select the Font property. Notice that a Build button appears (a small button with three dots on it). When you click the Build button, a dialog box specific to the property you've selected appears. In this instance, a dialog box that enables you to manipulate the font of the text box appears (see Figure 2.11). Different properties display different dialog boxes when you click their Build buttons.
7. Notice that the Size property has a plus sign next to it. This indicates that the property has one or more subproperties. Click the plus sign to expand the property, and you'll see that Size is composed of Width and Height.

By simply clicking a property in the Properties window, you can easily tell the type of input the property requires.

FIGURE 2.11
The Font dialog box gives you to change the appearance of text in a control.

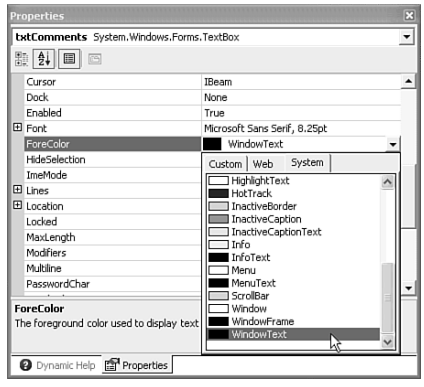


Working with Color Properties

Properties that deal with colors are unique in the way in which they accept values, yet all color-related properties behave the same way. In Visual Basic .NET, colors are expressed as a set of three numbers, each number having a value from 0 to 255. A given set of numbers represents the red, green, and blue (RGB) components of a color, respectively. The value 0,255,0, for example, represents pure green, whereas the value 0,0,0 represents black and 255,255,255 represents white. (See Hour 18 for more information on the specifics of working with color.)

A color rectangle is displayed for each color property in the Properties window; this color is the selected color for the property. Text is displayed next to the colored rectangle. This text is either the name of a color or a set of RGB values that defines the color. Clicking in a color property causes a drop-down arrow to appear, but the drop-down you get by clicking the arrow isn't a typical drop-down list. Figure 2.12 shows what the drop-down list for a color property looks like.

FIGURE 2.12
The color drop-down list enables you to select from three sets of colors.



The color drop-down list is composed of three tabs: Custom, Web, and System. Most color properties use a system color by default. Hour 5, "Building Forms—The Basics," goes into great detail on system colors, so I want to mention here only that system colors vary from computer to computer; they're determined by the user when she right-clicks the desktop and chooses Properties from the desktop's shortcut menu. Use a system color when you want a color to be one of the user's selected system colors. When a color property is set to a system color, the name of the system color appears in the property sheet.

The Custom tab shown in Figure 2.13 is used to specify a specific color, regardless of the user's system color settings; changes to system colors have no effect on the property. The most common colors appear on the palette of the Custom tab, but you can specify any color you desire.

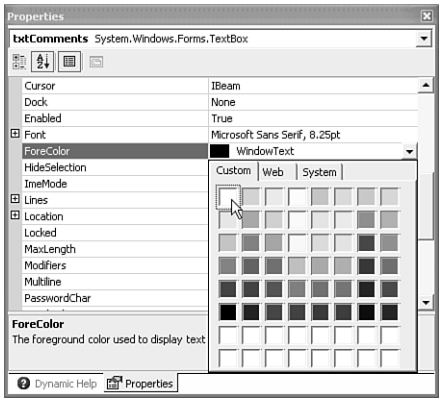


FIGURE 2.13
The Custom tab of the color drop-down list lets you specify any color imaginable.

The colors visible in the various palettes are limited by the number of colors that can be produced by your video card. If your video card doesn't support enough colors, some will appear *dithered*, which means they will appear as dots of colors rather than as a true, solid color. Keep this in mind as you develop your applications: What looks good on your computer mind turns to mush if a user's display isn't as capable.

The bottom two rows in the Custom color palette are used to mix your own colors. To assign a color to an empty color slot, right-click a slot in one of the two rows to access the Define Color dialog box (see Figure 2.14). Use the controls on the Define Color dialog box to create the color you want, and then click Add Color to add the color to the color palette in the slot you selected. In addition, the custom color is automatically assigned to the current property.

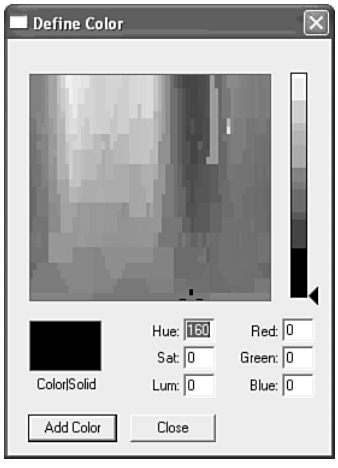


FIGURE 2.14
The Define Color dialog box enables you to create your own colors.

The Web tab is used in Web applications to pick from a list of browser-safe colors.

Viewing Property Descriptions

It's not always immediately apparent just exactly what a property is or does—especially for new users of Visual Basic .NET. The Description section at the bottom of the Properties window shows a simple description of the selected property (refer to Figure 2.10). To view a description, click a property or value area of a property. For a more complete description of a property, click it once to select it and then press F1 to display Help about the property.

You can hide or show the Description section of the Properties window at any time by right-clicking anywhere within the Properties window (other than in the value column or on the title bar) to display the Properties window shortcut menu and choosing Description. Each time you do this, you toggle the Description section between visible and hidden. To change the size of the Description box, click and drag the border between it and the Properties pane.

Managing Projects

Before you can effectively create an interface and write code, you need to understand what makes up a Visual Basic .NET project and how to add and remove various components from within your own projects. In this section, you'll learn about the Solution Explorer window and how it's used to manage project files. You'll also learn specifics about projects and project files, as well as how to change a project's properties.

Managing Project Files with the Solution Explorer

As you develop projects, they'll become more and more complex, often containing many objects such as forms and modules. Each object is defined by one or more files on your hard drive. In addition, you can build complex solutions composed of more than one project. The Solution Explorer window shown in Figure 2.15 is *the* tool for managing all the files in a simple or complex solution. Using the Solution Explorer, you can add, rename, and remove project files, as well as select objects to view their properties. If the Solution Explorer window isn't visible on your screen, show it now by choosing Solution Explorer from the View menu.

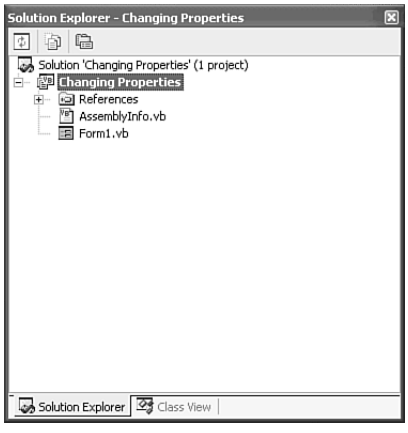


FIGURE 2.15
Use the Solution Explorer window to manage all the files that make up a project.

To better understand the Solution Explorer window, follow these steps:

1. Locate the Picture Viewer program you created in the Quick Tour by choosing File, Open, and then clicking Project.
2. Open the Picture Viewer project. The file you need to select is located in the Picture Viewer folder that Visual Basic created when the project was constructed. The file has the extension .sln (for solution). If you're asked whether you want to save the current project, choose No.
3. Select the Picture Viewer project item in the Solution Explorer (be sure not to click the Project node, not the Solution node). When you do, a button becomes visible toward the top of the window. This button has a picture of pieces of paper and has the ToolTip Show All Files (see Figure 2.16). Click this button and the Solution Explorer displays all files in the project.

Your Solution Explorer should now look like the one in Figure 2.16. Be sure to widen the Solution Explorer window so that you can read all the text it contains.

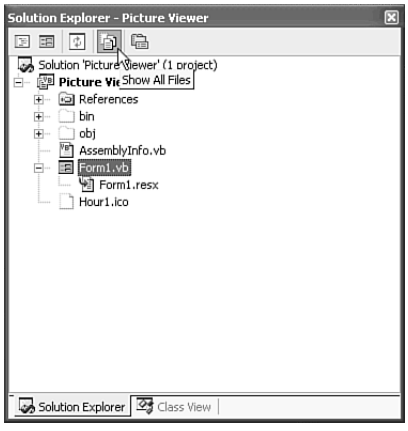


FIGURE 2.16
Notice that the form you defined appears as two files in the Solution Explorer.

Some forms and other objects might be composed of more than one file. By default, Visual Basic .NET hides project files that you don't directly manipulate. Click the plus sign (+) next to the form item and you'll see a sub-item titled Form1.resx. You'll learn about these additional files in Hour 5. For now, click the Show All Files button again to hide these related files.

You can view any object listed within the Solution Explorer using the object's default viewer by double-clicking the object. Each object has a default viewer, but might actually have more than one viewer. For instance, a form has a Form Design view as well as a Code view. By default, double-clicking a form in the Solution Explorer displays the form in Form Design view, where you can manipulate the form's interface.

You've already learned one way to access the code behind a form: double-click an object to access its default event handler. You'll frequently need to get to the code of a form without adding a new event handler. One way to do this is to use the Solution Explorer. When a form is selected in the Solution Explorer, buttons are visible at the top of the Solution Explorer window that enable you to display the code editor or the form designer, respectively.

You'll use the Solution Explorer window so often that you'll probably want to dock it to an edge and set it to Auto Hide, or perhaps keep it visible all the time. The Solution Explorer window is one of the easiest to get the hang of in Visual Basic .NET; navigating the Solution Explorer window will be second nature to you before you know it.

Working with Solutions

In truth, the Solution Explorer window is the evolution of the Project Explorer window from Visual Basic 6 and earlier versions, and the two windows are similar in many ways. Understanding solutions is easier to do when you understand projects.

A project is what you create with Visual Basic .NET. Often, the words *project* and *program* are used interchangeably, and this isn't much of a problem if you understand the important distinctions. A *project* is the set of source files that make up a program or component, whereas a *program* is the binary file that you build by compiling source files into something such as a Windows executable file (.exe). Projects always consist of a main project file and can be made up of any number of other files, such as form files, module files, or class module files. The main project file stores information about the project—all the files that make up the project, for example—as well as properties that define aspects of a project, such as the parameters to use when the project is compiled into a program.

What, then, is a solution? As your abilities grow and your applications increase in complexity, you'll find that you have to build multiple projects that work harmoniously to accomplish your goals. For instance, you might build a custom user control such as a custom data grid that you use within other projects you design, or you might isolate the business rules of a complex application into separate components to run on isolated servers. All the projects used to accomplish those goals are collectively called a *solution*. Therefore, a *solution* (at its most basic level) is really nothing more than a grouping of projects. In previous versions of Visual Basic, you created project groups; in Visual Basic .NET, you create solutions. Although the naming is different, the premise is the same.

You should group projects into a single solution only when the projects relate to one another. If you're working on a number of projects, but each of them is autonomous, work with each project in a separate solution.

Understanding Project Components

As I stated earlier, a project always consists of a main project file, and it might consist of one or more secondary files, such as files that make up forms or code modules. As you create and save objects within your project, one or more corresponding files are created and saved on your hard drive. Each file that's created for a Visual Basic .NET source object has the extension .vb, denoting that it defines a Visual Basic .NET object. Make sure that you save your objects with understandable names, or things will get confusing as the size of your project grows.

With previous editions of Visual Basic (version 6 and earlier), you could easily tell the type of object defined by project file by looking at the extension of the file. For example, form files had the extension .frm. Unfortunately, this is no longer the case and you need to be diligent about giving your files unique names.

All files that make up a project are text files. Some objects need to store binary information, such as a picture for a form's BackgroundImage property. Binary data is stored in an XML file (which is still a text file). Suppose that you had a form with an icon on it. You'd have a text file defining the form (its size, the controls on it, and the code behind it), and an associated resource file with the same name as the form file but with the extension .resx. This secondary file would be in XML format and would contain all the binary data needed to create the form.

If you want to see what the source file of a form file looks like, use Notepad to open one on your computer. Don't save any changes to the file, however, or it might never work again (</insert evil laugh here/>).

The following is a list of some of the components you might use in your projects:

- Modules Modules enable you to store code procedures without needing a specific form to attach them to.
- Class modules Class modules are a special type of module that enable you to create object-oriented applications. Throughout the course of this book, you're learning how to program using an object-oriented language, but you're mostly learning how to use objects supplied by Visual Basic .NET. In Hour 16, "Designing Objects Using Classes," you'll learn how to use class modules to create your own objects.
- Forms Forms are the visual windows that make up the interface of your application. Forms are defined using a special type of class module.
- User controls User controls (formerly ActiveX controls, which themselves are formerly OLE controls) are controls that can be used on the forms of other projects. For example, you could create a user control with a calendar interface for a contact manager. Creating user controls requires the skill of an experienced programmer; therefore, I won't be covering them in this book.

Setting Project Properties

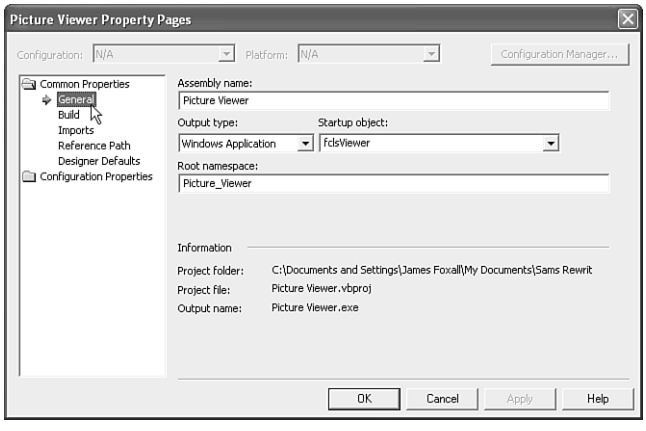
Visual Basic .NET projects have properties, just as other objects such as controls do. Projects have lots of properties, many of them relating to advanced functionality that I won't cover in this book. You need to be aware, however, of how to access project properties and how to change some of the more commonly used properties.

To access the properties for a project, right-click the project name (Picture Viewer) in the Solution Explorer window and choose Properties from the shortcut menu. Do this now.

If you select the project in the Solution Explorer, the last menu item on the Project menu will be Properties. If the last thing you selected was a form or other object, this menu item will read *<project name>* properties. These items display the same dialog box.

The Tree View control on the left side of the dialog box is used to display a property page (see Figure 2.17). When you first open the dialog box, the General page is visible. On this page, the setting you'll need to worry about most is the Startup Object property. The Startup Object setting determines the entry point to your application. This can be the name of a form or it can be the text Sub Main. If the name of a form is specified (as it is by default), that form will be loaded and displayed when the project first starts. If Sub Main is specified, you must create a standard module with a procedure called Sub Main, and the code in Sub Main will be the first code executed when the project starts (refer to Hour 10, "Creating and Calling Code Procedures," for more information).

FIGURE 2.17
Project properties let you tailor aspects of the project as a whole.



The Output Type option determines the type of compiled component defined by this source project. When you create a new project, you select the type of project to create (such as Windows Application), so this field is always filled in. There might be times when you have to change this setting after the project has been created, and this is the place to do so.

Notice that the project folder, project filename, and output name are displayed on this page as well. If you work with a lot of projects, you might find this information valuable; this is certainly the easiest place to find it.

The output name determines the filename created when you build an executable. Distributing applications is discussed in Hour 22, "Deploying a Visual Basic .NET Application."

As you work through the hours in this book, I'll refer to the Project Properties dialog box as necessary, explaining pages and items in context with other material.

Adding and Removing Project Files

When you first start Visual Basic .NET and create a new Windows Application project, Visual Basic .NET creates the project with a single form. You're not limited to having one form in a project, however; you can create new forms or add existing forms to your project at will (feeling powerful yet?). You can also create and add code modules and classes, as well as other types of objects.

You can add a new or existing object to your project in one of three ways:

- Choose the appropriate menu item from the Project menu.
- Click the small drop-down arrow that's part of the Add New Item button on the Standard toolbar, and then choose the object type from the drop-down list that displays (see Figure 2.18).
- Right-click the project name in the Solution Explorer window, and then choose Add from the shortcut menu to access a submenu from which you can select object types.

When you select Add *ObjectType* from any of these menus, a dialog box appears, showing you the objects that can be added to the project. Your chosen item is selected by default (see Figure 2.19). Simply name the object and click Open to create a new object of the selected type. To create an object of a different type, click the type to select it, name it, and then click Open.

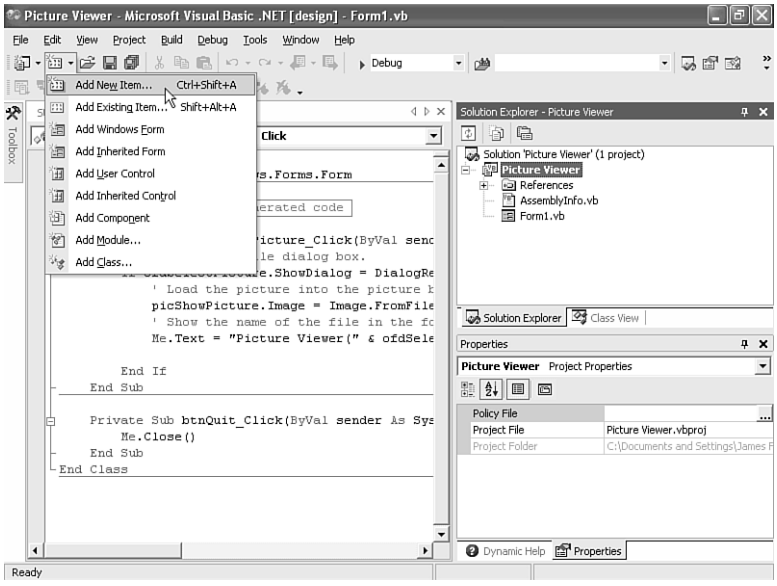


FIGURE 2.18
This tool button drop-down is one of three ways to add objects to a project.

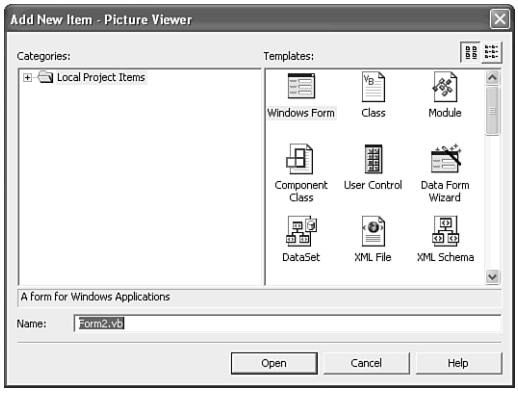


FIGURE 2.19
Regardless of the menu option you select, you can add any type of object you want using this dialog box.

Adding new forms and modules to your project is easy, and you can add as many as you like. You'll come more and more to rely on the Solution Explorer to manage all the objects in the project as the project becomes more complex.

Although it won't happen as often as adding project files, you might sometimes need to remove an object from a project. Removing objects from your project is even easier than adding them. To remove an object, right-click the object in the Solution Explorer window and select Exclude from Project. This removes the object from the file, but doesn't delete the source file from the disk. Selecting Delete, on the other hand, removes the file from the project and deletes it from the disk. Don't select Delete unless you want to totally destroy the file and you're sure that you'll never need it again in the future.

A Quick-and-Dirty Programming Primer

Programming is a complicated task. Everything is so interrelated that it's difficult, if not impossible, to isolate every programming concept and then present the material in a linear fashion. Instead, while learning one subject, you often have to touch on elements of another subject before you've had a chance to learn about the secondary topic. I've made every effort to avoid such forward references, but there are some concepts with which you'll need to be at least slightly familiar with before proceeding. You'll learn the guts of each of these topics in their respective lessons, but you'll need to have at least heard of them before digging any deeper in this book.

Storing Values in Variables

A *variable* is an element in code that holds a value. You might, for example, create a variable that holds the name of a user or the perhaps the user's age. Each variable (storage entity) must be created before it can be used. The process of creating a variable is known as declaring a variable. In addition, each variable is declared to hold data of a specific type, such as text (called a *string*) for a person's name or a number for a person's age. An example of a variable declaration is

```
Dim strFirstName As String
```

This statement creates a variable called strFirstName. This variable is of type String, which means it can hold any text that you choose to put into it. The contents of a variable can be changed as often as desired.

The key primer point to remember: Variables are storage locations that must be declared before use and that hold a specific type of information.

Using Procedures to Write Functional Units of Code

When you write Visual Basic .NET code, you place the code in a procedure. A procedure is a group of code statements that perform a specific function. You can call a procedure from code in another procedure. For example, you might create a procedure that totals the items on an order and another procedure that calculates the tax on the entire sale. There are two types of procedures: procedures that don't return values and procedures that do return values. Some procedures allow data to be passed to them. For example, the tax calculation procedure mentioned previously might allow a calling statement to pass a monetary total into the procedure, and then use that total to calculate tax. When a procedure accepts data from the calling code, the data is called a parameter. Procedures don't have to accept parameters.

A procedure that doesn't return a value is declared using the keyword Sub, and looks like this:

```
Public Sub MyProcedure()
   ' The procedure's code goes here.
End Sub
```

A procedure that returns a value is declared using the keyword Function. In addition, it has a data type specified at the end of the procedure, which denotes the type of data returned by the procedure:

```
Public Function MyProcedure() As String
   ' The procedure's code goes here.
End Function
```

Notice the words *As String*. The keyword As is used to specify a data type. In this example, the function returns a string, which is text.

If a procedure accepts a parameter, it is enclosed in the parentheses. Again, notice how the word As is used to denote the type of data being accepted:

```
Public Function CalculateTax(dblItemTotal As Double) As String
   ' The procedure's code goes here.
End Function
```

MessageBox.Show()

You're almost certainly familiar with the Windows message box—it's the little dialog box that's used to display text to a user (see Figure 2.20). Visual Basic .NET provides a way to display such messages using a single line of code: the MessageBox.Show statement. The following is a MessageBox.Show statement in its most basic form:

```
MessageBox.Show("Text to display goes here")
```

You'll use message boxes throughout this book, and you'll learn about them in detail in Hour 17, "Interacting with Users."

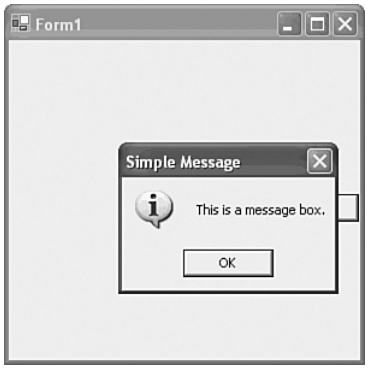


FIGURE 2.20
Visual Basic .NET makes it easy to display simple message boxes like this.

Getting Help

Although Visual Basic was designed to be as intuitive as possible, you'll find that you occasionally need assistance in performing a task. Honestly, Visual Basic .NET is nowhere near as intuitive as its predecessors—with all the additional power and flexibility came complexity. It doesn't matter how much you know, Visual Basic .NET is so complex and contains so many features that you'll have to use Help at times. This is particularly true when writing Visual Basic .NET code; you won't always remember the command you need or the syntax of a command. Fortunately, Visual Basic includes a comprehensive Help feature.

To access Help from within the design environment, press F1. Generally speaking, when you press F1, Visual Basic .NET shows you a Help topic directly related to what you're doing. This is known as *context-sensitive help*, and when it works, it works well. For example, you can display help for any Visual Basic .NET syntax or keyword (functions, objects, methods, properties, and so on) when writing Visual Basic code by typing the word into the code editor, positioning the cursor anywhere within the word (including before the first letter or after the last), and pressing F1. You can also get to Help from the Help menu on the menu bar.

If your project is in Run mode, Visual Basic .NET's Help won't be displayed when you press F1. Instead, the Help for your application will appear—if you've created Help.

You can have Help display topics directly within the design environment instead of in a separate window. This is a new feature of .NET. Personally, I think this method is considerably inferior to the old style of having Help float above the design environment. When Help is displayed within the design environment, you can't necessarily see the code, form, or other object with which you're working. To make Help float above the design environment, choose Options from the Tools menu to display the Options dialog box, click Help in the Tree view on the left, and select External Help.

Visual Basic .NET includes a new Help feature called Dynamic Help. To display the Dynamic Help window, choose Dynamic Help from the Help menu. The Dynamic Help window shows Help links related to what it is you're working on (see Figure 2.21). For instance, if you select a form, the contents of the Dynamic Help window show you Help links related to forms. If you click a text box, the contents of the Dynamic Help window adjust to show you Help links related to text boxes. This is an interesting feature, and you might find it valuable.

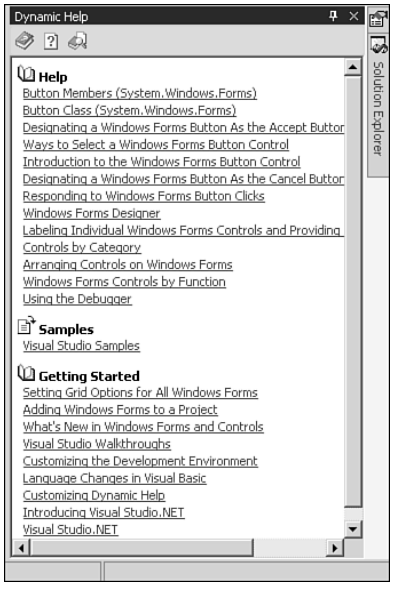


FIGURE 2.21
Dynamic Help gives you a list of Help links related to the task you are performing.

Summary

In this hour, you learned how to use the Visual Studio home page—your gateway to Visual Basic .NET. You learned how to create new projects and how to open existing projects. The Visual Basic environment is your workspace, toolbox, and so much more. You learned how to navigate the environment, including how to work with design windows (hide, show, dock, and float).

You'll use toolbars constantly, and now you know how to modify them to suit your specific needs. You learned how to create new toolbars and how to modify existing toolbars. This is an important skill that shouldn't be overlooked.

Visual Basic .NET has many different design windows, and in this hour, you began learning about some of them in detail. You learned how to get and set properties using the Properties window, how to manage projects using the Solution Explorer, and how to add controls to a form using the toolbox. You'll use these skills often, so it's important to get familiar with them right away. Finally, you learned how to access Visual Basic .NET's Help feature, which I guarantee you'll find very important as you learn to use Visual Basic .NET.

Visual Basic .NET is a vast and powerful development tool—far more powerful than any version that's come before it. Don't expect to become an expert overnight; that's simply impossible. However, by learning the tools and techniques presented in this hour, you've begun your journey. Remember, you'll use most of what you learned in this hour each and every time you use Visual Basic. Get proficient with these basics and you'll be building cool programs in no time!

Q&A

Q **How can I easily get more information about a property when the Description section of the Properties window just doesn't cut it?**

A Click the property in question to select it, and then press F1—context-sensitive help applies to properties in the Properties window, as well.

Q **I find that I need to see a lot of design windows at one time, but I can't find that "magic" layout. Any suggestions?**

A Run at a higher resolution. Personally, I won't develop in less than 1024×768. As a matter of fact, all my development machines have two displays, both running at this resolution (or higher). You'll find that any investment you make in having more screen real estate will pay you big dividends.

Workshop

The Workshop is designed to help you anticipate possible questions, review what you've learned, and get you thinking about how to put your knowledge into practice. The answers to the quiz are in Appendix B, "Answers to the Quizzes."

Quiz

1. How can you make the Visual Studio Start Page appear at startup if this feature has been disabled?
2. Unless instructed otherwise, you're to create what type of project when building examples in this book?
3. To make a docked design window appear when you hover over its tab and disappear when you move the mouse away from it, you change what setting of the window?
4. How do you access the Toolbars menu?
5. What design window do you use to add controls to a form?
6. What design window is used to change the attributes of an object?
7. To modify the properties of a project, you must select the project in what design window?
8. Which Help feature adjusts the links it displays to match what it is you are doing?

Exercises

1. Create a custom toolbar that contains Save All, Start, and Stop Debugging—three buttons you'll use often throughout this book.
2. Use the Custom Color dialog box to create a color of your choice, and then assign the color to the BackColor property of a form.