1

Welcome to Visual Basic .NET

The goal of this third edition is to help you come up to speed with the Visual Basic .NET language even if you have never programmed anything before. You will start slowly, and build on what you learn. So take a deep breath, let it out slowly, and tell yourself you can do this. No sweat! No kidding!

Programming a computer is a lot like teaching a child to tie their shoes. Until you find the correct way of giving the instructions, not much gets accomplished. Visual Basic .NET is a language in which you can tell your computer how to do things. But like a child, the computer will only understand if you explain things very clearly. If you have never programmed before, this sounds like an arduous task, and sometimes it is. However, Visual Basic .NET gives you a simple language to explain some complex things. Although it never hurts to have an understanding of what is happening at the lowest levels, Visual Basic .NET frees the programmer from having to deal with the mundane complexities of writing Windows programs. You are free to concentrate on solving problems.

Visual Basic .NET helps you create solutions that run on the Microsoft Windows operating system. If you are looking at this book, you might have already felt the need or the desire to create such programs. Even if you have never written a computer program before, as you progress through the *Try It Outs* in this book, you will become familiar with the various aspects of the Visual Basic .NET language, as well as its foundation in Microsoft's .NET Framework. You will find that it is not nearly as difficult as you have been imagining. Before you know it, you will be feeling quite comfortable creating a variety of different types of programs with Visual Basic .NET. Also, as the name implies, Visual Basic .NET can be used to create applications for use over the Internet and newly incorporated into Visual Studio .NET 2003 for its ability to create smart device applications (Pocket PCs and SmartPhones). However, while learning any new technology, you have to walk before you can run, so you begin by focusing on Windows applications before extending your boundaries to other platforms.

In this chapter, we will cover the following subjects:

- The installation of Visual Basic .Net 2003
- A tour of the Visual Basic .Net Integrated Development Environment (IDE)
- How to create a simple Windows program
- How to use and leverage the integrated help system

Chapter 1

Windows Versus DOS Programming

A Windows program is quite different from its ancient relative, the MS-DOS program. A DOS program follows a relatively strict path from beginning to end. Although this does not necessarily limit the functionality of the program, it does limit the road the user has to take to get to it. A DOS program is like walking down a hallway; to get to the end you have to walk down the hallway, passing any obstacles that you may encounter. A DOS program would only let you open certain doors along your stroll.

Windows on the other hand, opened up the world of event-driven programming. Events in this context include, for example, clicking on a button, resizing a window, or changing an entry in a text box. The code that you write responds to these events. To go back to the hallway analogy: in a Windows program to get to the end of the hall, you just click on the end of the hall. The hallway can be ignored. If you get to the end and realize that is not where you wanted to be, you can just set off for the new destination without returning to your starting point. The program reacts to your movements and takes the necessary actions to complete your desired tasks (Visual Basic .NET).

Another big advantage in a Windows program is the abstraction of the hardware; which means that Windows takes care of communicating with the hardware for you. You do not need to know the inner workings of every laser printer on the market, just to create output. You do not need to study the schematics for graphics cards to write your game. Windows wraps up this functionality by providing generic routines that communicate with the drivers written by hardware manufacturers. This is probably the main reason that Windows has been so successful. The generic routines are referred to as the Windows Application Programming Interface (API).

Before Visual Basic 1.0 was introduced to the world in 1991, developers had to be well versed in C++ programming, as well as the rudimentary building blocks (Windows API) of the Windows system itself. This complexity meant that only the dedicated and properly trained individuals were capable of turning out software that could run on Windows. Visual Basic changed all of that, and it has been estimated that there are now as many lines of production code written in Visual Basic as in any other language.

Visual Basic changed the face of Windows programming by removing the complex burden of writing code for the user interface (UI). By allowing programmers to *draw* their own UI, it freed them to concentrate on the business problems they were trying to solve. Once the UI is drawn, the programmer can then add the code to react to events.

Visual Basic has also been extensible from the very beginning. Third-party vendors quickly saw the market for reusable modules to aid developers. These modules, or controls, were originally referred to as VBXs (named after their file extension). If you did not like the way a button behaved you could either buy or create your own. However, these controls had to be written in C or C++. Database access utilities were some of the first controls available. Version 5 of Visual Basic introduced the concept of ActiveX that allowed developers to create their own ActiveX controls.

When Microsoft introduced Visual Basic 3.0, the programming world changed again. Now you could build database applications directly accessible to users (so-called front-end applications) completely with Visual Basic. There was no need to rely on third-party controls. Microsoft accomplished this task with the introduction of the Data Access Objects (DAO), which allowed programmers to manipulate data with the same ease as manipulating the user interface.

Versions 4.0 and 5.0 extended the capabilities of version 3.0 in order to allow developers to target the new Windows 95 platform. Crucially they also made it easier for developers to write code, which could then

be manipulated to make it usable to other language developers. Version 6.0 provided a new way to access databases with the integration of ActiveX Data Objects (ADO). ADO was developed by Microsoft to aid Web developers using Active Server Pages to access databases. With all of the improvements to Visual Basic over the years, it ensured its dominant place in the programming world. It helps developers write robust and maintainable applications in record time.

With the release of Visual Basic .NET in February 2002, many of the restrictions that used to exist have been obliterated. In the past, Visual Basic has been criticized and maligned as a "toy" language, as it did not provide all of the features of more sophisticated languages such as C++ and Java. Now, Microsoft has removed these restrictions and made Visual Basic .NET a very powerful development tool. This trend continues with Visual Basic .NET 2003. Although not as drastic a change as from Visual Basic 6 to Visual Basic .NET, there are enough improvements in the language (including support for the .NET Framework 1.1) that Visual Basic .NET 2003 is a welcome upgrade and is a great choice for programmers of all levels.

Installing Visual Basic .NET

You may own Visual Basic .NET:

- As part of Visual Studio .NET, a suite of tools and languages that also includes C# (pronounced C-sharp) and Visual C++ .NET. Visual Studio comes in three flavors: Professional, Enterprise Developer, and Enterprise Architect. Each of these versions comes with progressively more tools for building and managing the development of larger enterprise wide applications.
- As the Standard Edition, which includes a cut down set of the tools and languages available with Visual Studio .NET.

Both enable you to create your own applications for the Windows platform. The installation procedure is straightforward. In fact, the Visual Basic .NET Install is smart enough to figure out exactly what your computer requires to make it work.

The descriptions that follow are based on installing Visual Studio .NET Professional. However, all of Visual Studio .NET's languages use the same screens and windows (and hence look very similar), so you would not be seeing much that you would not see anyway.

Try It Out — Installing Visual Basic .NET

1. The Visual Basic .NET CD has an auto-run feature, but if the Setup screen does not appear after inserting the CD, you have to run setup.exe from the root directory of the CD. To do this, go to your Windows Start menu (usually found right at the bottom of your screen) and select Run. Then type d:\setup.exe into the Open box, where d is the drive letter of your CD drive. After the setup program initializes you will see the screen as shown in Figure 1-1.
2. This dialog box shows the order in which the installation takes place. To function properly, Visual Basic .NET requires that several components and updates be installed on your machine. Step 1 is the Windows component update, so click on the Windows Component Update link; you will then be prompted to insert the Component Update CD that came with your Visual Studio .NET disks.

Depending on how your operating system is configured you may receive the following message like the one shown in Figure 1-2 before you install the pre-requisites.

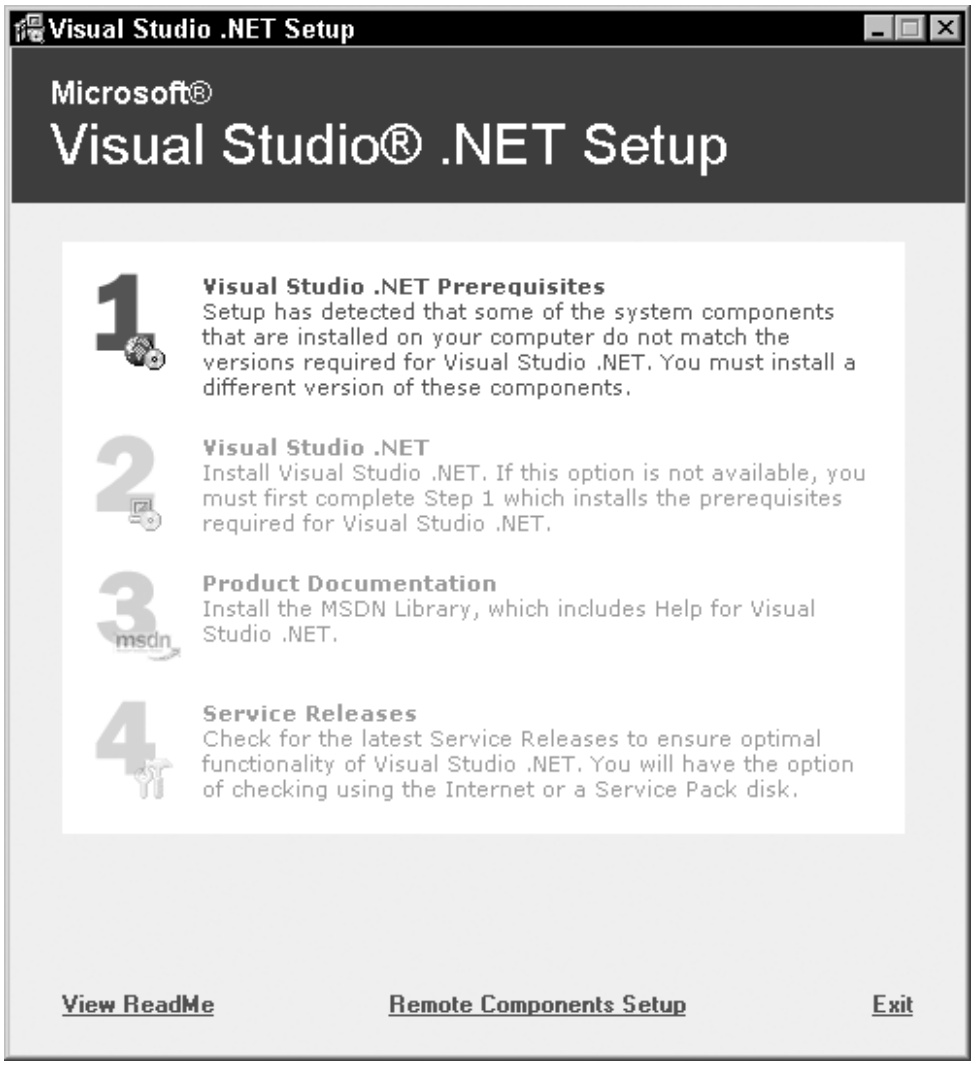

Figure 1-1

If you want to build Web applications locally you must install Internet Information Services (IIS) and Front Page Server Extensions. Clicking Setup Instructions takes you to a Web page with explicit instructions on how to install these components. You will have to restart the Visual Studio .NET / Visual Basic .NET installation after installing IIS. If you skip this step you will have to use a remote Web server to host your application.

3. The installation program then examines your system to see exactly which components have to be installed. Depending on the current state of your machine, this list could include any of the following items:

   - Windows NT 4.0 Service Pack 6.0a
   - Windows Installer 2.0

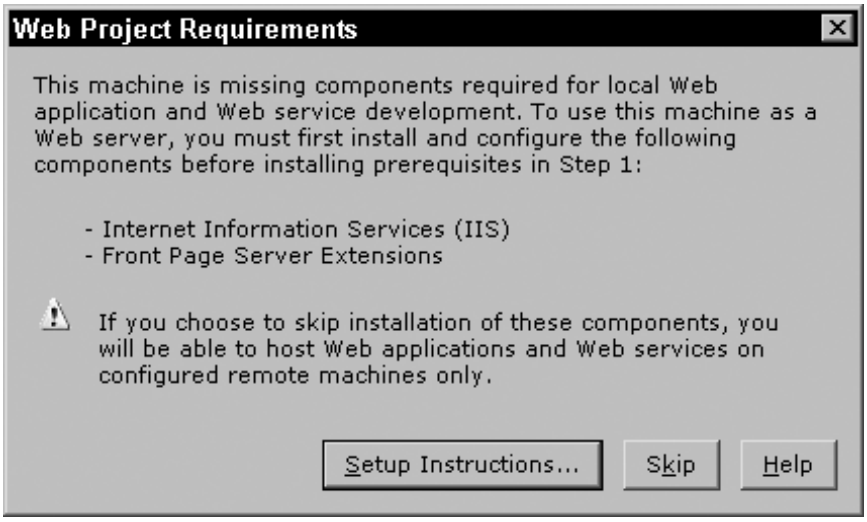

Figure 1-2

- Microsoft FrontPage 2000 Web Extensions Client
- Setup Runtime Files
- Microsoft Internet Explorer 6.0 with Service Pack 1
- Microsoft Data Access Components 2.7 with Service Pack 1
- Microsoft .NET Framework 1.1
- Microsoft Visual J# .Net Redistributable Package 1.1

If you don't know what some of those things are, don't worry about it. They are just Windows components that Visual Studio .NET or Visual Basic .NET requires.

4. After agreeing to the End User License agreement for the Prerequisite components, click Continue and the list of needed prerequisite components will be displayed.

Depending on what components you have already installed on your machine, your list of components that require updating may be different. For reference, these are the options on a completely patched version of Windows XP Professional that includes Internet Information Services.

5. Click Install Now! to begin the installation of the Prerequisites. After the Prerequisite install has finished, you will be returned to the initial Setup screen and step 2 will be enabled. You will now be able to install Visual Studio .NET, so click on Visual Studio .NET.
6. As with most installations you will be presented with an option list of components to install (see Figure 1-3). You can choose to install only the features that you need. For example, if your drive space is limited and you have no immediate need for Visual C++ .NET, you can exclude it from the installation. You will also be given the chance to select the location of items (although the defaults should suffice unless your particular machine has special requirements). Any option that is not chosen at the initial setup can always be added later as your needs or interests change.

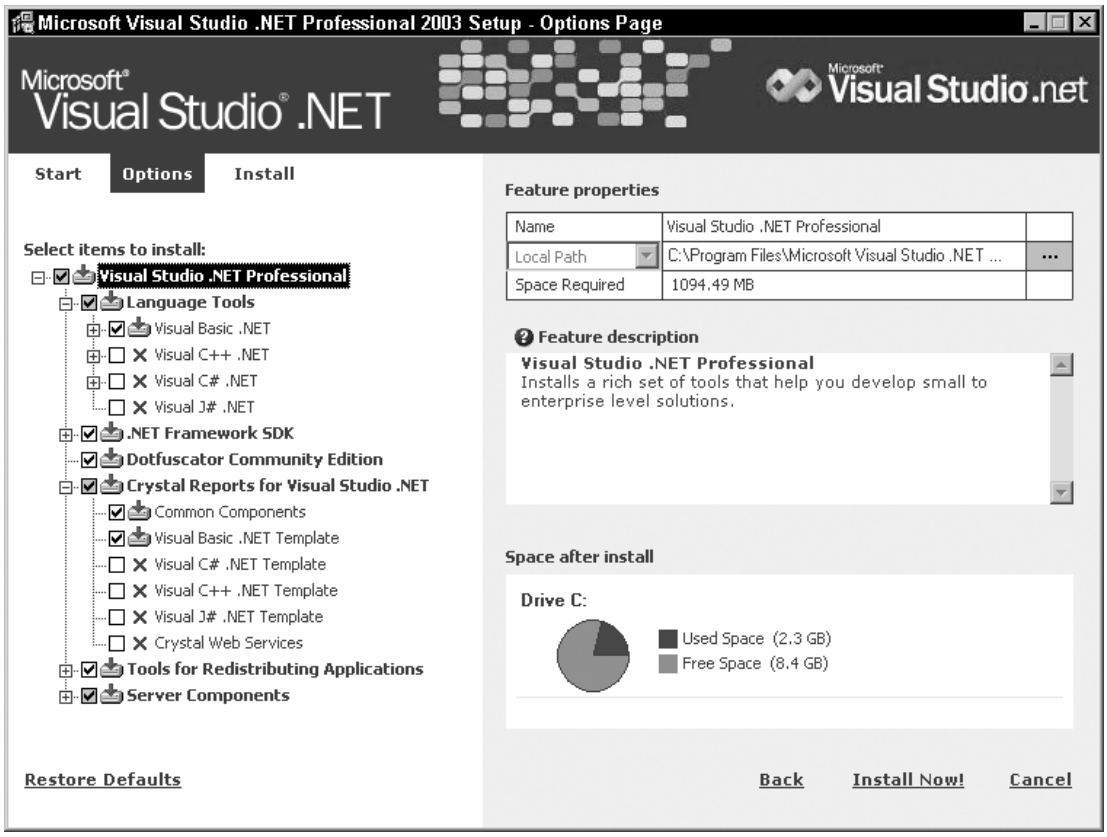

Figure 1-3

There are three sections of information given for each feature:

- The Feature properties section outlines where the required files will be installed and how much space will be needed to do this.
- The Feature description box gives you an outline of each feature and its function.
- Finally, the Space Allocation section illustrates how the space on your hard drive will be affected by the installation as a whole.

When you are running Visual Basic .NET, a lot of information is swapped from the disk to memory and back again. Therefore, it is important to have some free space on your disk. There is no exact rule for determining how much free space you will need, but if you use your machine for development as well as other tasks, anything less than 100MB free space should be considered a full disk.

7. Once you have chosen all of the features you want, click Install Now! Installation will begin and you can sit back and relax for a bit. The setup time varies depending on how many features you chose to install. As a reference, the installation process took over an hour on a 650 MHz laptop computer with 256MB RAM, a 12GB hard drive, and running Windows XP Professional.

7. When installation is completed, you will see a dialog informing you that the installation has completed.

   Here you will see any problems that Setup encountered along the way. You are also given the chance to look at the installation log. This log provides a list of all actions taken during the installation process. Unless your installation reported errors, the installation log can safely be ignored. The Visual Studio .NET setup is nearly complete. Click Done to move on to installing the documentation.

8. Visual Studio .NET no longer includes the MSDN documentation as part of the installation. Instead it uses the separate MSDN Library installation routine. The big advantage in this is that one can always install the most current documentation regardless of what came out of your Visual Basic .NET box.

9. The MSDN Library installation is simple and straightforward and this section covers the highlights. After inserting disk one of the MSDN Library (assuming you are installing from CD) you will see the initial welcome screen.

10. This wizard interface guides you through the installation process. After gathering your License and User Information you will see the screen shown in Figure 1-4.

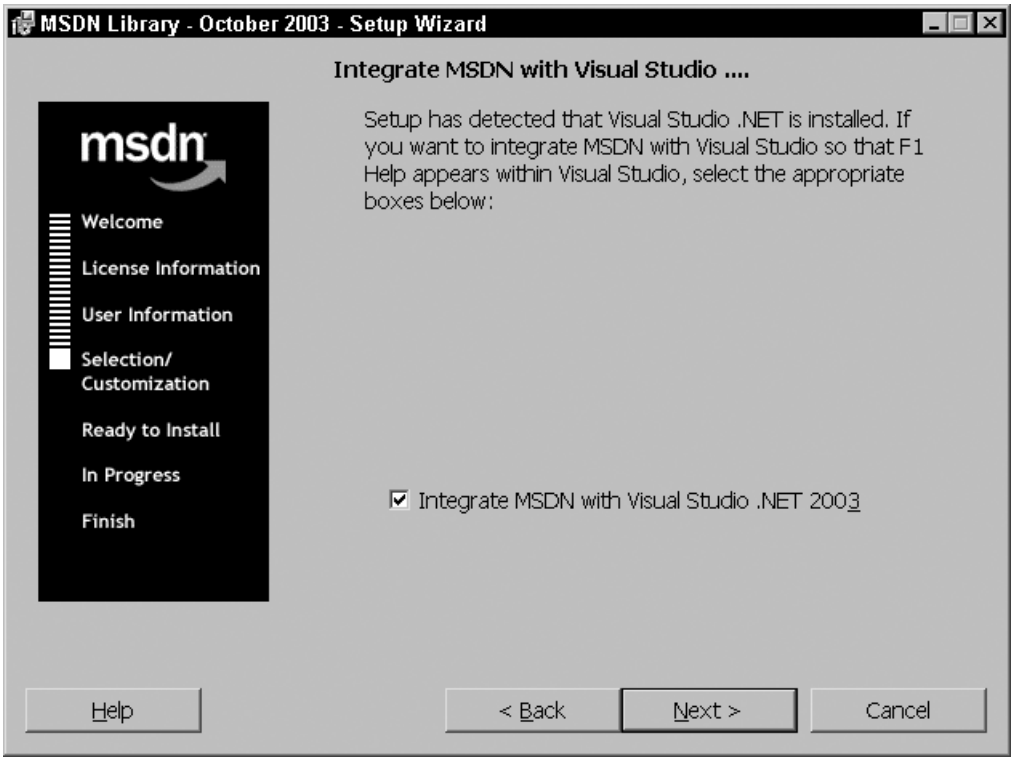

Figure 1-4

Make sure that the Integrate MSDN with Visual Studio .Net 2003 check box is checked so that Visual Studio .NET /Visual Basic .NET can find the MSDN documentation.

11. After clicking Next, you will be allowed to select the amount of the documentation you want to install. For example, if you did not install C++ there is probably little reason to install that documentation.

If you have the spare hard drive space, it is a very good idea to install the full documentation. MSDN does not always include specific examples or documentation for Visual Basic .NET; therefore you may find what you are looking for under SQL Server documentation, or even C#.

12. After the MSDN documentation has been installed you are returned to the initial setup screen again and the Service Releases option is now available.

It is a good idea to select Service Releases to check for updates. Microsoft has done a good job of making software updates available through the Internet. These updates can include anything from additional documentation to bug fixes. You will be given the choice to install any updates via a Service Pack CD or the Internet. Obviously, the Internet option requires an active connection. Since updates can be quite large, a fast connection is highly recommended.

Once you have performed the update process, Visual Basic .NET is ready to use. Now the real fun can begin! So get comfortable, relax, and let us enter the world of Visual Basic .NET.

The Visual Basic .NET IDE

You don't actually need the Visual Basic .NET product to write applications in the Visual Basic .NET language. The actual ability to run Visual Basic .NET code is included with the .NET Framework. You could actually just write all of your Visual Basic .NET using a text editor such as Notepad. You could also hammer nails using your shoe as a hammer, but that slick pneumatic nailer sitting there is probably a lot more efficient.

However, by far the easiest way to write in Visual Basic .NET is by using the Visual Studio .NET Integrated Development Environment, also known as the IDE. This is what you actually see when working with Visual Basic .NET—the windows, boxes, and so on. The IDE provides a wealth of features that are unavailable in ordinary text editors—such as code checking, visual representations of the finished application, and an explorer that displays all of the files that make up your project.

The Profile Setup Page

An IDE is a way of bringing together a suite of tools that make developing software a lot easier. Fire up Visual Basic .NET and see what you've got. If you used the default installation, go to your Windows Start menu and then Programs (All Programs on Windows XP and Windows Server 2003) ➪ Microsoft Visual Studio.NET 2003 ➪ Microsoft Visual Studio.NET 2003. A splash screen will briefly appear and then you should find yourself presented with the Start screen's My Profile tab, as shown in Figure 1-5.

This screen allows you to do some basic configuration of the IDE so that it serves you better. Since this IDE serves all the Visual Studio .NET languages, there are some settings to tailor it to our particular development interests. However, the default settings are acceptable for most users. Make any changes

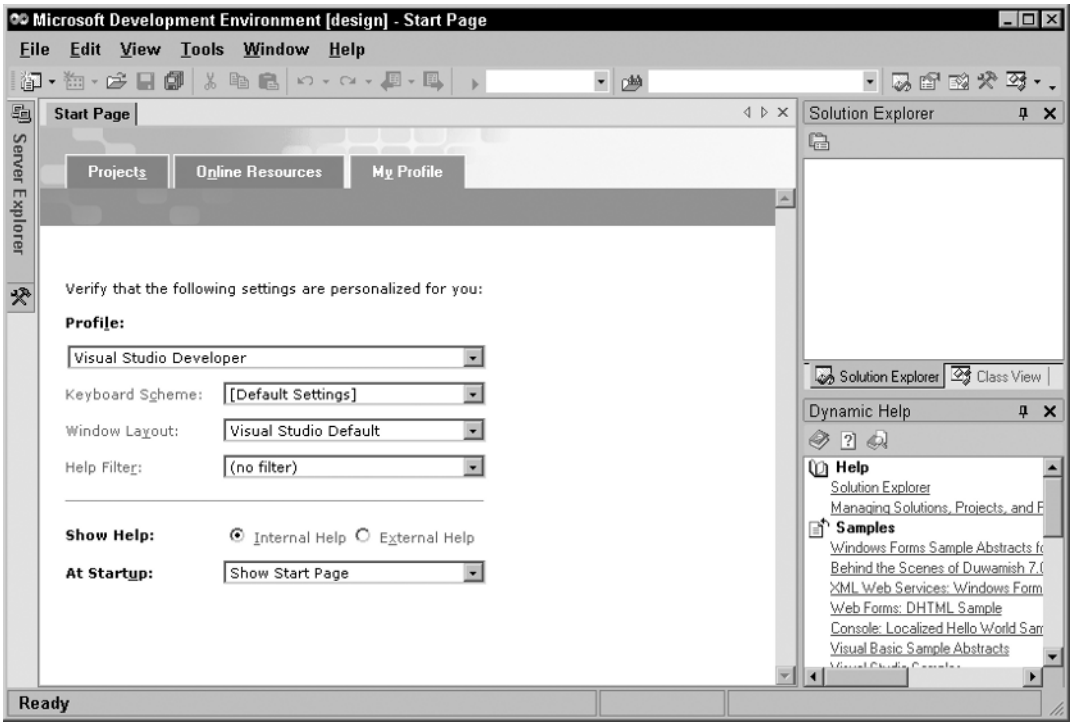

Figure 1-5

that you want and then click the Projects tab to be taken to that view. There you can create new projects and open existing projects.

The Projects Tab

By now, you may be a bit anxious to start writing some code. But first, take a look at the Projects tab on the Start Page and see what is there. Assuming that you have been following along while setting up Visual Studio .NET, your screen should now look something like Figure 1-6. Of course, since you have not created any projects yet, your project list will be empty. As you start creating projects, this list will grow and as you can see, the list contains the project name and the modified date. The project names are hyperlinks and clicking on a hyperlink for that project will open it up in the IDE.

Begin your exploration of the Visual Basic .NET IDE by looking at the toolbar and menu, which as you will learn are not really that different from toolbars and menus you have seen in other Microsoft software such as Word, Excel, and PowerPoint.

The Menu

Visual Studio .NET's menu is dynamic, meaning that items will be added or removed depending on what you are trying to do. While you are still looking at the Projects page, the menu bar will only consist of the

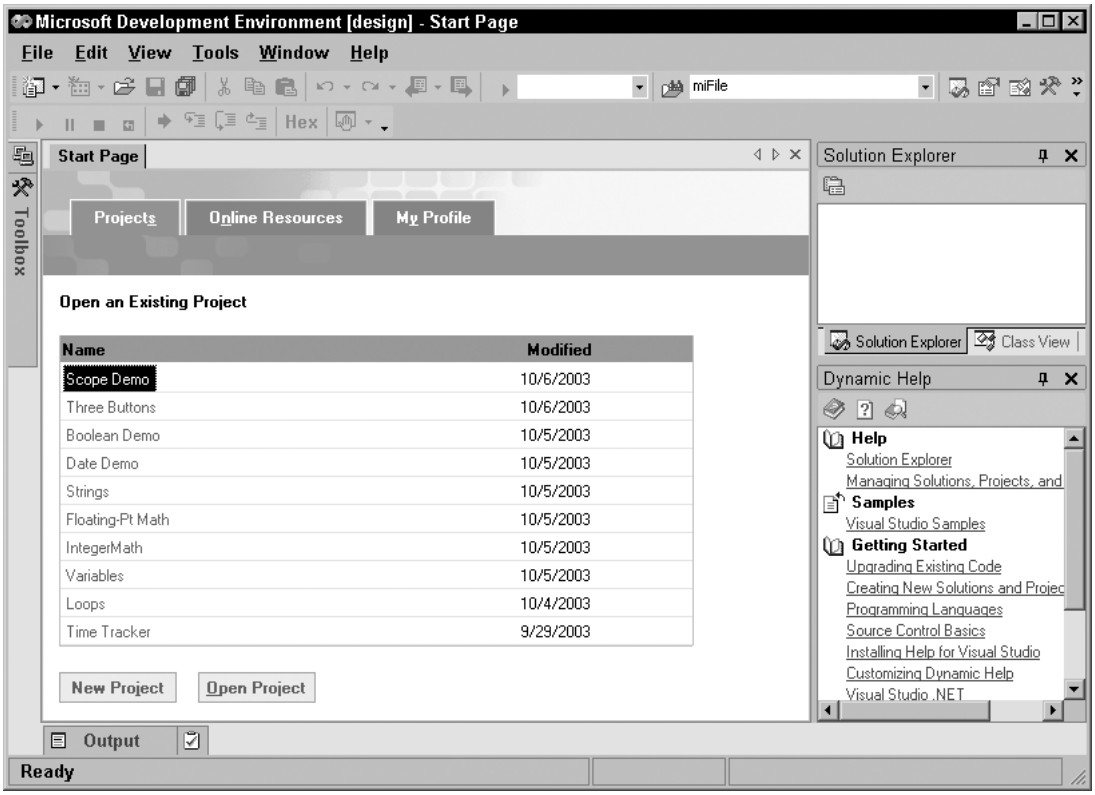

Figure 1-6

File, Edit, View, Tools, Window, and Help menus. However, when you start working on a project, the full Visual Studio .NET menu appears as shown in Figure 1-7.

Figure 1-7

At this point, there is no need to cover each menu topic in great detail. You will become familiar with each of them as you progress through the book. Here is a quick rundown of what activities each menu item pertains to:

- **File:** It seems every Windows program has a File menu. It has become the standard where you should find, if nothing else, a way to exit the application. In this case, you can also find ways of opening and closing single files and whole projects.
- **Edit:** The Edit menu provides access to the items you would expect: Undo, Redo, Cut, Copy, Paste, and Delete.
- **View:** The View menu provides quick access to the windows that make up the IDE, such as the Solution Explorer, Properties window, Output window, Toolbox, and so on.

- **Project:** The Project menu allows you to add various files to your application such as forms and classes.
- **Build:** The Build menu becomes important when you have completed your application and want to run it without the use of the Visual Basic .NET environment (perhaps running it directly from your Windows Start menu as you would any other application such as Word or Access).
- **Debug:** The Debug menu allows you to start and stop running your application within the Visual Basic .NET IDE. It also gives you access to the Visual Studio .NET debugger. The debugger allows you to step through your code while it is running to see how it is behaving.
- **Data**: The Data menu helps you to use information that comes from a database. It only appears when you are working with the visual part of your application (the [Design] tab will be the active one in the main window), not when you are writing code. Chapters 15 and 16 will introduce you to working with databases.
- **Format**: The Format menu also appears only when you are working with the visual part of your application. Items on the Format menu allow you to manipulate how the controls you create will appear on your forms.
- **Tools:** The Tools menu has commands to configure the Visual Studio .NET IDE, as well as links to other external tools that may have been installed.
- **Window:** The Window menu has become standard for any application that allows more than one window to be open at a time, such as Word or Excel. The commands on this menu allow you to switch between the windows in the IDE.
- **Help**: The Help menu provides access to the Visual Studio .NET documentation. There are many different ways to access this information (for example, via the help contents, an index, or a search). The Help menu also has options that connect to the Microsoft Web site to obtain updates or report problems.

The Toolbars

There are many toolbars available within the IDE, including Formatting, Image Editor, and Text Editor, which you can add to and remove from the IDE via the View ➪ Toolbars menu option. Each one provides quick access to often-used commands, preventing you from having to navigate through a series of menu options. For example, the leftmost icon on the toolbar shown in Figure 1-8 (New Project) is available from the menu by navigating to File ➪ New ➪ Project.

The default toolbar (called Standard) appears at the top of the IDE as:

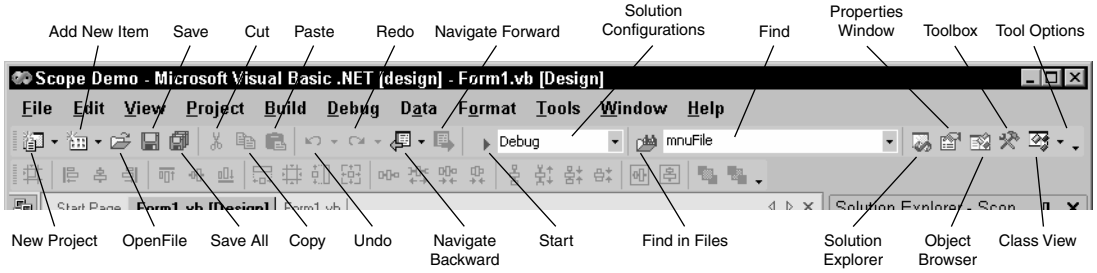

Figure 1-8

The toolbar is segmented into groups of related options, which are separated by a vertical bar. The first five icons provide access to the commonly used project and file manipulation options available through the File and Project menus, such as opening and saving files.

The next group of icons is for editing (Cut, Copy, and Paste). The third group of icons is for editing and navigation. The navigation buttons replicate functionality found in the View menu and allow you to cycle through the tabs at the top of the main window.

The fourth group of icons provides the ability to start your application running (via the blue triangle) and to specify build configurations. There are times when you want certain parts of your code only to appear in a debug version, a bit like a rough draft version of your application. For example, you may have code in your application that is only useful for tracking down problems in the application. When it is time to release your application to the world, you will want to exclude this code by setting the Solution Configurations settings to Release. You can also access the functionality offered by this group via the Build and Debug menus.

The next section allows you to locate parts of your code quickly. The simplest way to search is to type some text into the Find text box and press Enter. If the text is found, it will be highlighted in the central window. The Find in Files option allows you to specify more sophisticated searches, including matching the case of the text, looking in specific files or projects, and replacing the found text with new text. The search functionality can also be accessed via the Edit ➪ Find and Replace menu option.

The next group of icons provides quick links back to the Solution Explorer, Properties window, Object Browser, Toolbox, and Class view. If any of these windows are closed, clicking the appropriate icon will bring it back into view.

If you forget what a particular icon does, you can hover your mouse pointer over it so that a tooltip appears displaying the name of the toolbar option.

You could continue to look at each of the other windows directly from the Start Page. But, as you can see they are all empty at this stage, and therefore not too revealing. The best way to look at the capabilities of the IDE is to use it while writing some code.

Creating a Simple Application

To finish your exploration of the Visual Basic .NET IDE you need to create a project, so that the windows shown earlier in Figure 1-6 actually have some interesting content for you to look at. You are now going to create a very simple application called HelloUser that will allow you to enter a person's name and display a greeting to that person in a message box.

Try It Out Creating a HelloUser Project

1. Click on the New Project button on the Projects tab of the Start Page.
2. The New Project dialog box will open. Make sure you have Visual Basic Projects selected in the Project Types tree-view box to the left. Next, select Windows Application in the Templates box on the right. If you need to save this project to a location other than the default, be sure to enter it into the Location box. Finally, type HelloUser in the Name text box and click OK. Your New Project screen should look like Figure 1-9.

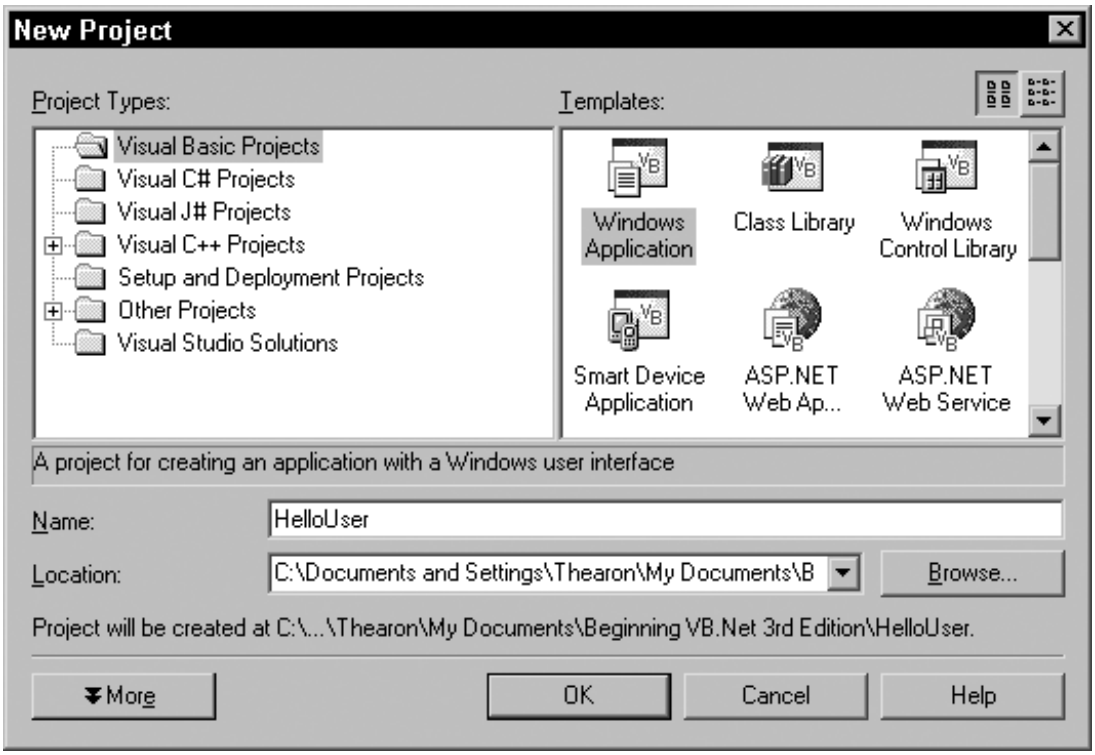

Figure 1-9

3. Visual Basic .NET will then create an empty Windows application for you. So far, your HelloUser program consists of one blank window called a Windows Form (or sometimes just a form), with the default name of Form1.vb, as shown in Figure 1-10.

Whenever Visual Studio .NET creates a new file, either as part of the project creation process or when you create a new file, it will use a name that describes what it is (in this case, a form) followed by a number.

Windows in the Visual Studio .NET IDE

At this point, you can see that the various windows in the IDE are beginning to show their purposes, and you should take a brief look at them now before you come back to the *Try It Out*. Note that if any of these windows are not visible on your screen, you can use the View menu to select and show them. Also if you do not like the location of any particular window you can move it by clicking on its title bar (the blue bar at the top) and dragging it to a new location. The windows in the IDE can float (stand out on their own) or be docked (as they appear in Figure 1-10). The following list introduces the available windows:

- **Server Explorer:** The Server Explorer gives you management access to the servers on your network. Here you can create database connections and view the services provided by the available servers. In Figure 1-10, the Server Explorer is a tab at the bottom of the Toolbox window.

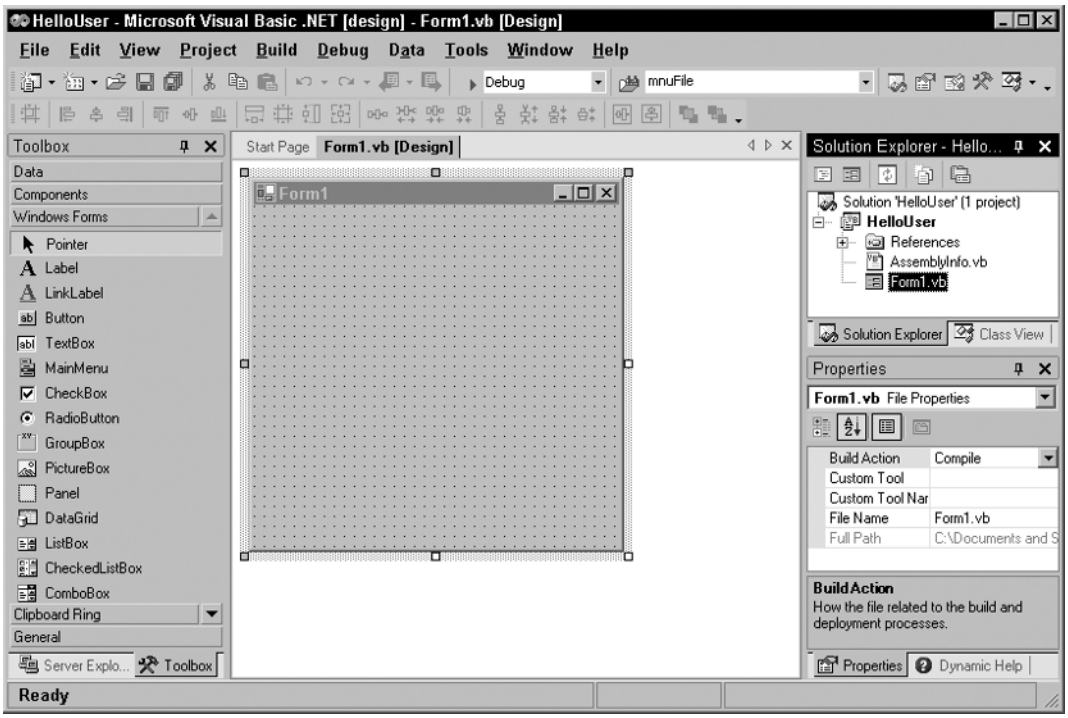

Figure 1-10

- **Toolbox:** The Toolbox contains reusable components that can be added to your application. These can range from buttons to data connectors to customized controls either purchased or developed by you.
- **Design Window:** The Design window is where a lot of the action takes place. This is where you will draw your user interface on your forms. This window is sometimes referred to as the Designer.
- **Solution Explorer:** The Solution Explorer window contains a hierarchical view of your solution. A solution can contain many projects while a project contains forms, classes, modules and components that solve a particular problem.
- **Class View:** The Class View window (shown as a tab at the bottom the Solution Explorer window) gives you a tree view of the classes in your program and shows the properties and methods that each contains. A class is a code file that groups data and the functions that manipulate it together into one unit. A property is data and a method is a function or subroutine.
- **Properties:** The Properties window shows what properties the selected object makes available. Although you can set these properties in your code, sometimes it is much easier to set them while you are designing your application (for example, drawing the controls on your form). You will notice that the File Name property has the value Form1.vb. This is the physical file name for the form's code and layout information.
- **Dynamic Help:** The Dynamic Help window (shown as a tab at the bottom of the Properties window) displays a list of help topics that relate to what has focus in the IDE. If you click on the

form in the Design window and then open Dynamic Help, you will see a list of help topics relating to forms.

Try It Out Creating a HelloUser Project (cont.)

1. Change the name of your form to something more indicative of what your application is. Click on Form1.vb in the Solution Explorer window. Then, in the Properties window, change the File Name property from Form1.vb to HelloUser.vb and press Enter, as shown in Figure 1-11. When changing properties you must either hit Enter or click off the property for it to take effect.

Figure 1-11

2. Notice that the form's file name has also been updated in the Solution Explorer to read HelloUser.vb.

3. Now click on the form displayed in the Design window; the Properties window will change to display the form's Form properties (instead of the File properties, which we have just been looking at). You will notice that the Properties window is dramatically different. The difference is the result of two different views of the same file. When the form name is highlighted in the Solution Explorer window, the physical file properties of the form are displayed. When the form in the Design window is highlighted, the visual properties and logical properties of the form are displayed.

   The Properties window allows you to easily set a control's properties. Remember properties are a particular object's set of internal data. Properties usually describe appearance or behavior. In Figure 1-12, you can see that properties are grouped together in categories—Appearance (Header is not shown), Behavior, Configurations, Data, and Design are the ones shown here.

You can see that under the Appearance category (header not shown), even though we changed the file name of the form to HelloUser.vb, the text or caption of the form is still Form1. Also notice that the (Name) property under the Design category is still set to Form1. Unless this is changed any reference in the code to this form must refer to it as Form1.

4. Right now, the title (Text property) of your form (displayed in the bar at the top) is Form1. This is not very descriptive, so change it to reflect the purpose of this application. Locate the Text

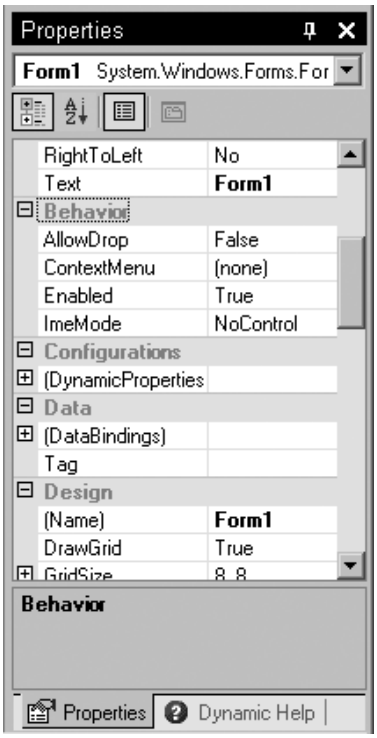

Figure 1-12

property in the Appearance section of the Properties window and change its value to Hello from Visual Basic .NET and press Enter. Notice that the form's title has been updated to reflect the change (see Figure 1-13).

If you have trouble finding properties, click the little AZ button on the toolbar toward the top of the Properties window. This changes the property listing from being ordered by category to being ordered by name.

5. You are now finished with the procedure. Click on the Start button on the Visual Studio .NET toolbar (the blue triangle) to run the application. As you work through the book, whenever we say "run the project" or "start the project," just click on the Start button. An empty window with the title Hello from Visual Basic .NET is displayed.

Notice how the grid patterns of dots have disappeared. These are displayed at design time to help place controls such as boxes, labels, and radio buttons onto your form. They're not needed (or even particularly desirable) at run time, so they're not shown.

That was simple, but your little application isn't doing much at the moment. Let us make it a little more interactive. To do this you are going to add some controls—a label, textbox, and two buttons to the form. This will let you see how the Toolbox makes adding functionality quite simple. You may be wondering at this point when you will actually look at some code. Soon! The great thing about Visual Basic .NET is that you can develop a fair amount of your application *without* writing any code. Sure, the code is still there, behind the scenes, but as you will see, Visual Basic .NET writes a lot of it for you.

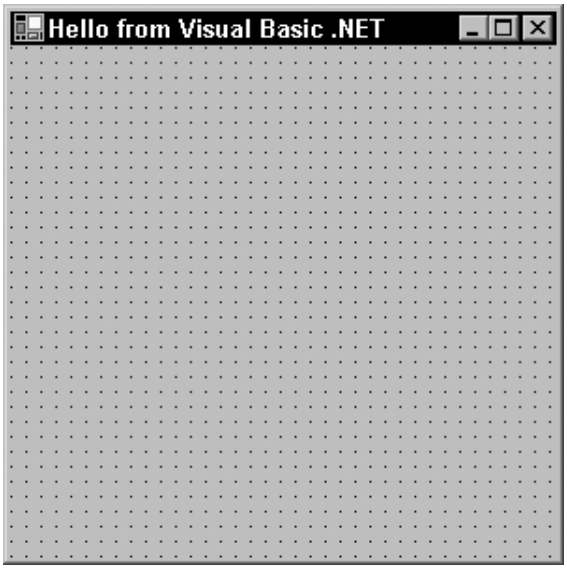

Figure 1-13

The Toolbox

The Toolbox is accessed via the View ➪ Toolbox menu option, the Toolbox icon on the Standard menu bar, or by pressing *Ctrl+Alt+X*. Alternatively, the Toolbox tab is displayed on the left of the IDE and hovering your mouse over this tab will cause the Toolbox window to fly out, partially covering your form.

The Toolbox contains a tabbed view of the various controls and components that can be placed onto your form. Controls such as text boxes, buttons, radio buttons, and combo boxes can be selected and then *drawn* onto your form. For the HelloUser application, you will only be using the controls on the Windows Forms tab.

In Figure 1-14 you can see a partial listing of standard .NET controls for Windows Forms. The down arrow button to the right of the Clipboard Ring tab title actually scrolls the Windows Forms control list down as there are too many controls to fit. The up arrow button to the right of the Windows Forms tab scrolls the list up. Note that the order in which your controls appear may be different.

Controls can be added to your forms in any order, so it does not matter if you add the label control after the textbox or the buttons before the label.

Try It Out Adding Controls to the HelloUser Application

1. Stop the project if it is still running, as you now want to add some controls to your form. The simplest way to do this is to click the X button in the top-right corner of the form. Alternatively, you can click on the blue square in the Visual Studio .NET IDE (which displays a tool tip of Stop Debugging if you hover over it with your mouse pointer).
2. Add a Label control to the form. Click Label in the Toolbox to select it. Move the cursor over the form's Designer. You'll notice that the cursor looks like a crosshair with a little floating letter A beneath it. Click and hold the mouse button where you want the top-left corner of the label and

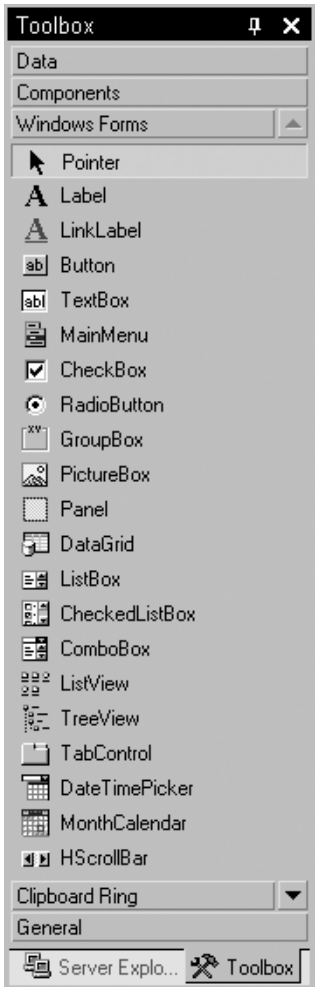

Figure 1-14

drag the mouse to where you want the bottom right. (Placing controls on your form can also be accomplished by double-clicking on the required control in the Toolbox.)

3. If the Label control you have just drawn is not in the desired location or is too big or too small, it really isn't a problem. Once the control is on the form you can resize it or move it around. Figure 1-15 shows what the control looks like after you place it on the form. To move it, click on the gray dotted border and drag it to the desired location. To resize it, click and drag on one of the white box "handles" and stretch the control in the needed direction. The corner handles resize both the horizontal and vertical dimension at the same time.

4. After drawing a control on the form, we should at least configure its name and the text that it will display. You will see that the Properties window to the right of the Designer has changed to Label1, telling you that you are currently examining the properties for it. In the Properties window, set your new label's Text property to Enter Your Name and its (Name) property to lblName.

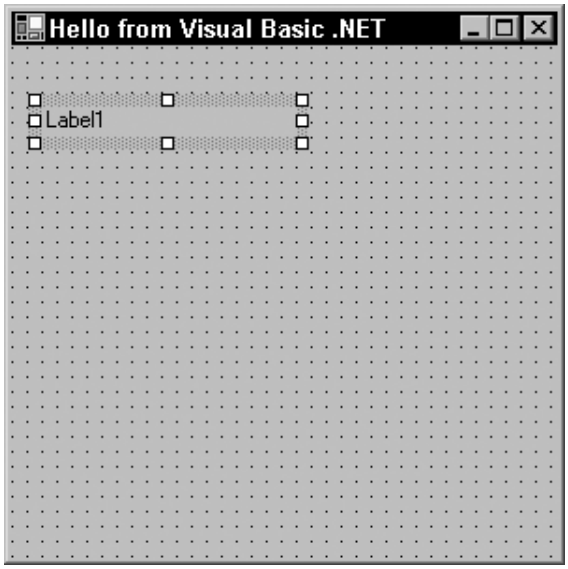

Figure 1-15

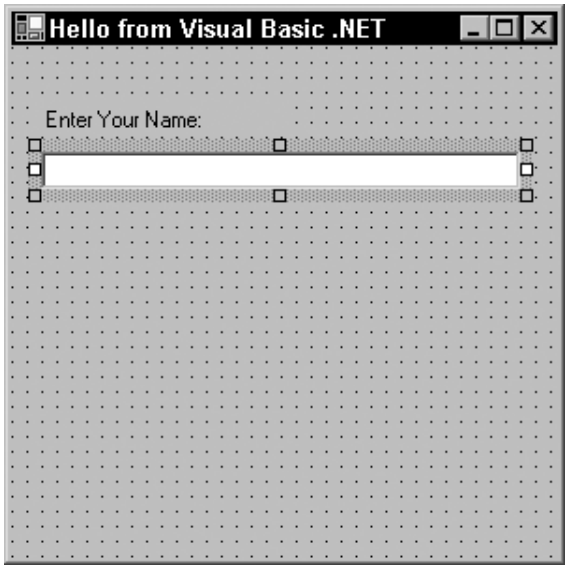

Figure 1-16

5. Now, directly beneath the label, you want to add a textbox, so that you can enter a name. You are going to repeat the procedure you followed for adding the label, but this time make sure you select the TextBox from the toolbar. Once you have dragged-and-dropped (or double-clicked) the control into the appropriate position, use the Properties window to set its Name property to txtName and clear the Text property so that the textbox now appears to be blank as shown in Figure 1-16.

Notice how, out of the eight sizing handles surrounding the control, only two are shown in white. By default, the TextBox control cannot be made any taller than the absolute height necessary to contain the font that it will use to draw the text.

6. In the bottom left corner of the form, add a Button control in exactly the same manner as you added the label and textbox. Set its Name property to btnOK, and its Text property to & OK. Your form should now look similar to the one shown in Figure 1-17.

The ampersand (&) is used in the Text property of buttons to create a keyboard shortcut (known as a hot key). The letter with the & sign placed in front of it will become underlined (as shown in Figure 1-17) to signal users that they can select that button by pressing the Alt-letter key combination, instead of using the mouse (on some configurations the underline doesn't appear to the user until they press ALT). In this particular instance, pressing Alt+O would be the same as clicking directly on the OK button. There is no need to write code to accomplish this.

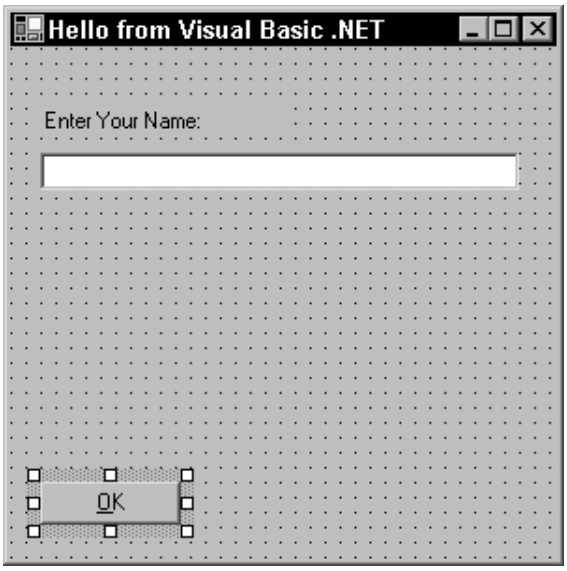

Figure 1-17

7. Now add a second Button control to the bottom right corner of the form and set the Name property to btnExit and the Text property to E&xit. Your form should look similar to Figure 1-18.

Now before you finish your sample application, let us briefly discuss some coding practices that you should be using.

Modified Hungarian Notation

You may have noticed that the names given to the controls look a little funny. Each name is prefixed with a shorthand identifier describing the type of control it is. This makes it much easier to understand what type of control you are working with when you are looking through the code. For example, say you had a control called simply Name, without a prefix of lbl or txt, you would not know whether you were working with a textbox that accepted a name or a label that displayed a name. Imagine if, in the previous

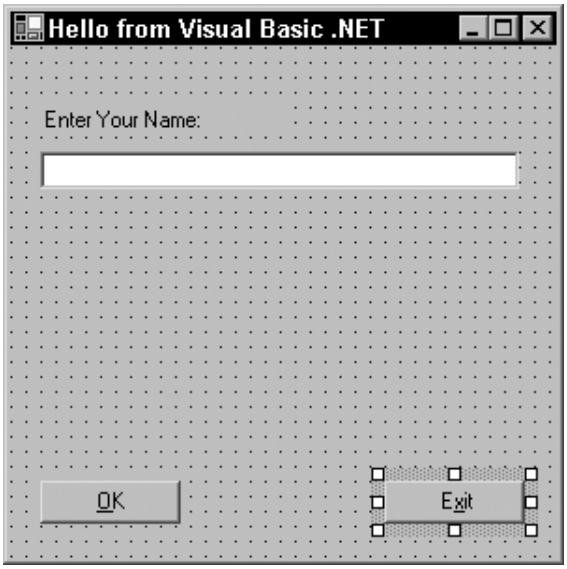

Figure 1-18

Try It Out, you had named your label Name1 and your textbox Name2—you would very quickly become confused. How about if you left your application for a month or two and then came back to it to make some changes?

When working with other developers, it is very important to keep the coding style consistent. One of the most commonly used styles used for controls within application development in many languages is Modified Hungarian notation. The notion of prefixing control names to identify their use was brought forth by Dr. Charles Simonyi. He worked for the Xerox Palo Alto Research Center (XPARC) before joining Microsoft. He came up with short prefix mnemonics that allowed programmers to easily identify the type of information a variable might contain. Since Dr. Simonyi is Hungarian, and the prefixes make the names look a little foreign, the name Hungarian Notation stuck. Because the original notation was used in C/C++ development, the notation for Visual Basic .NET is termed Modified. Table 1.1 shows some of the commonly used prefixes that you shall be using in this book.

Hungarian Notation can be a real time-saver when looking at code someone else wrote, or at code that you have written months past. However, by far the most important thing is to be consistent in your naming. When you start coding, pick a convention for your naming. It is recommended that you use the de facto standard Modified-Hungarian for Visual Basic .NET, but it is not required. Once you pick a convention, stick to it. When modifying someone else's code, use theirs. A standard naming convention followed throughout a project will save countless hours when the application is maintained. Now let's get back to the application. It's now time to write some actual code.

The Code Editor

Now that you have the HelloUser form defined, you have to add some code to actually make it do something interesting. You have already seen how easy it is to add controls to a form. Providing the

Table 1-1 Common prefixes in Visual Basic .Net

Control	Prefix
Button	`cmd` or `btn`
ComboBox	`cbo`
CheckBox	`chk`
Label	`lbl`
ListBox	`lst`
MainMenu	`mnu`
RadioButton	`rdb`
PictureBox	`pic`
TextBox	`txt`

functionality behind those on-screen elements is no more difficult. To add the code for a control, you just double-click on it. This will open the code editor in the main window, shown in Figure 1-19:

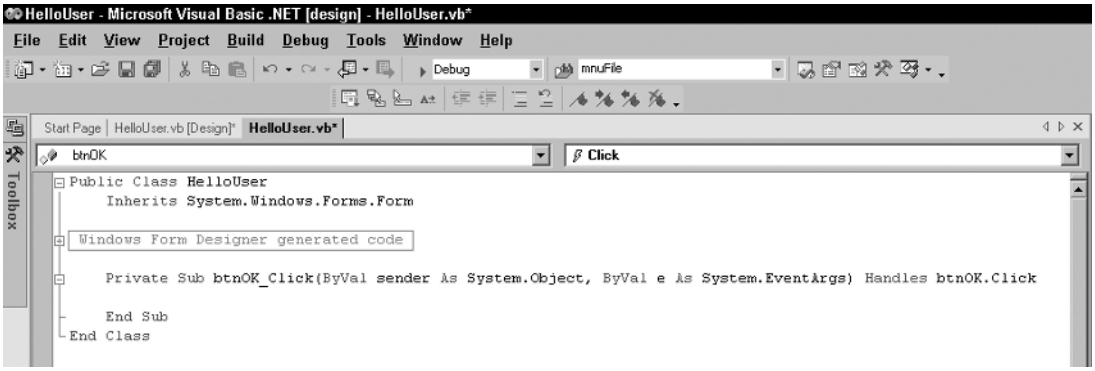

Figure 1-19

Notice that an additional tab has been created in the main window. Now you have the Design tab and the Code tab. You draw the controls on your form in the Design tab and you write code for your form in the Code tab. One thing to note here is that you have not created a separate file for the code. The visual definition and the code behind it both exist in the same file: HelloUser.vb. This is actually the reason why building applications with Visual Basic .NET is so slick and easy. Using the Design view you can visually lay out your application, and then using the Code view you add just the bits of code to implement your desired functionality.

You will also notice that there are two combo boxes at the top of the window. These provide shortcuts to the various parts of your code. If you pull down the one on the left you will see a list of all the objects within your application. If you pull down the one on the right you will see a list of all defined functions

or subroutines for the object selected in the left combo box. If this particular form had a lot of code behind it, these pull-downs would make navigating to the desired area very quick—jumping to the selected area. However, since all of the code fits in the window, there are not a lot of places to get lost.

Now look at the code in the window. The code in Visual Studio .NET is set up into regions designated by the plus (+) and minus (–) buttons along the left side. These regions can be collapsed and expanded in order to simplify what you are looking at. If you expand the region labeled Windows Form Designer generated code, you will see a lot of code that Visual Basic .NET has automatically generated for you, which takes care of defining each of the controls on the form and how the form itself should behave. You do not have to worry about the code in this region, so collapse the Windows Form Designer generated code region once more and concentrate on the code you have to write.

Try It Out Adding Code to the HelloUser Project

1. To begin adding the necessary code, click the Design tab to show the form again. Then double-click on the OK button. The code window will reopen with the following code. This is the shell of button's Click event and is the place where we enter the code that we want to run when we click on the button. This code is known as an event handler and sometimes is also referred to as an event procedure:

```
  Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnOK.Click

  End Sub
```

Due to the typographic constraints in publishing, it is not possible to put the Sub declaration on one line. Visual Basic .NET allows you to break up lines of code by using the underscore character (_) to signify a line continuation. The space before the underscore is required. Any whitespace preceding the code on the following line is ignored.

   Sub is an example of a keyword. In programming terms, a keyword is a special word that is used to tell Visual Basic .NET to do something special. In this case, it tells Visual Basic .NET that this is a procedure. Anything that you type between the lines `Private Sub` and `End Sub` will make up the event procedure for the OK button.

2. Now add the highlighted code into the procedure:

```
  Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnOK.Click
  'Display a message box greeting the user
  MessageBox.Show("Hello," & txtName.Text & _
         "! Welcome to Visual Basic .NET.", _
         "HelloUser Message")
End Sub
```

Throughout this book, you will be presented with a code that you should enter into your program if you are following along. Usually, we will make it pretty obvious where you put the code, but as we go we will explain anything that looks out of the ordinary.

3. After you have added the code, go back to the Design tab, and double-click on the Exit button. Add the highlighted code to the btnExit _Click event procedure.

```
    Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnExit.Click
    'End the program and close the form
    Me.Close()
End Sub
```

You may be wondering what `Me` is. `Me` refers to the form. Just like the pronoun *me*, it is just a shorthand for referring to oneself.

4. Now that the code is finished, the moment of truth has arrived and you can see your creation. First though, save your work by using File ➪ Save from the menu, or by clicking the disk icon on the toolbar.

5. Now click on the Start button on the toolbar. You will notice a lot of activity in the Output window at the bottom of your screen. Provided you have not made any mistakes in entering the code, this information just lets you know what files are being loaded to run your application.

   It is at this point that Visual Studio .NET will compile the code. Compiling is the activity of taking the Visual Basic .NET source code that you have written and translating it into a form that the computer understands. After the compilation is complete, Visual Studio .NET will run (also known as execute) the program and we'll be able to see the results.

If Visual Basic .NET encounters any errors, they will be displayed as tasks in the Task List window. Double-clicking on a task will transport you to the offending line of code. We will learn more about how to debug the errors in our code in Chapter 9.

6. When the application loads you will see the main form. Enter a name and click OK (or press the *Alt+O* key combination) (see Figure 1-20).

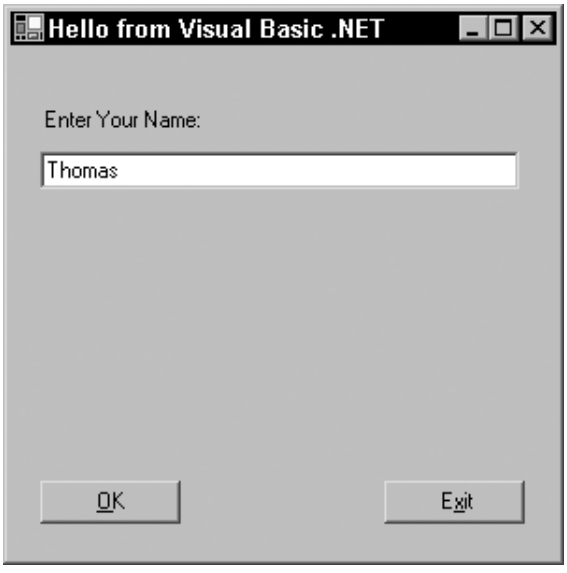

Figure 1-20

7. A window known as a message box appears, welcoming the person whose name was entered in the textbox on the form—in this case Thomas (see Figure 1-21).

Figure 1-21

8. After you close the message box by clicking the OK button, click on the Exit button on your form. The application will close and you will be brought back to the Visual Basic .NET IDE.

How It Works

The code that you added to the click event for the OK button will take the name that was entered in the text box and use it as part of the message that was displayed in Figure 1-21.

The first line of text entered in this procedure is actually a comment. Comments in Visual Basic .Net begin with a single quote (') and everything following is considered a comment.

The `MessageBox.Show` method displays a message box that accepts various parameters. As used in your code, you have passed the string text to be displayed in the message box. This is accomplished through the concatenation of string constants defined by text enclosed in quotes. Concatenation of strings is performed through the use of the ampersand (&) character.

In the code that follows, you have concatenated a string constant of "Hello," followed by the value contained in the Text property of the `txtName` text box control followed by a string constant of `"! Welcome to Visual Basic .NET."`. The second parameter being passed to the `MessageBox.Show` method is the caption to be used in the title bar of the Message Box dialog box.

Finally, the underscore (_) character used at the end of the lines in your code below enables you to split your code onto separate lines. This tells the compiler that the rest of the code for the parameter is continued on the next line. This is really useful when building long strings as it allows you to view the entire code fragment in the code editor without having to scroll the code editor window to the right to view the entire line of code.

```
  Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnOK.Click
  'Display a message box greeting the user
  MessageBox.Show("Hello, " & txtName.Text & _
          "! Welcome to Visual Basic .NET", _
          "HelloUser Message")
End Sub
```

The next procedure that you added code for was the Click event of the Exit button. Here you simply enter the code: `Me.Close()`. As explained earlier, the `Me` keyword refers to the form itself. The `Close` method of the form closes the form and releases all resources associated with it thus ending the program.

```
Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnExit.Click
'End the program and close the form
Me.Close()
End Sub
```

Using the Help System

The Help system included in Visual Basic .NET is an improvement over Help systems in previous versions. As you begin to learn Visual Basic .NET, you will probably become very familiar with the Help system. However, it is worthwhile to give you an overview, just to help speed your searches for information.

The Help menu contains the menu items shown in Figure 1-22:

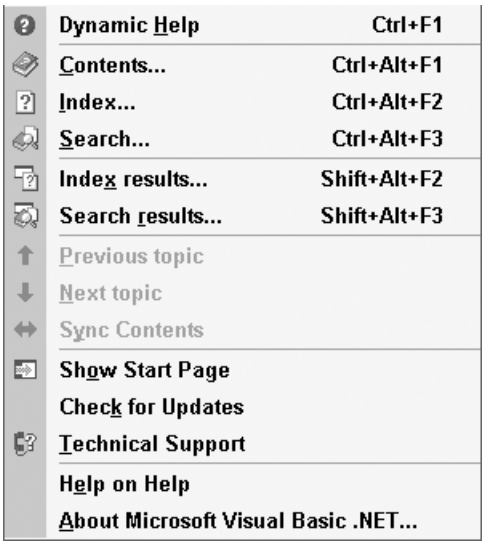

Figure 1-22

As you can see this menu contains many more entries than the typical Windows application. The main reason for this is the vastness of the documentation. Few people could keep it all in their heads—but luckily, that is not a problem, as you can always quickly and easily refer to the Help system. Think of it as a safety net for your brain.

One really fantastic new feature is Dynamic Help. When you select the Dynamic Help menu item from the Help menu, the Dynamic Help window is displayed with a list of relevant topics for whatever you may be working on. If you followed the default installation and have not rearranged the IDE, the Dynamic Help is displayed as a tab behind the Properties window.

Let us say, for example, that you are working with a text box (perhaps the text box in the HelloUser application) and want to find out some information; you just select the textbox on our form or in the code window and you can see all the help topics that pertain to text boxes, as shown in Figure 1-23

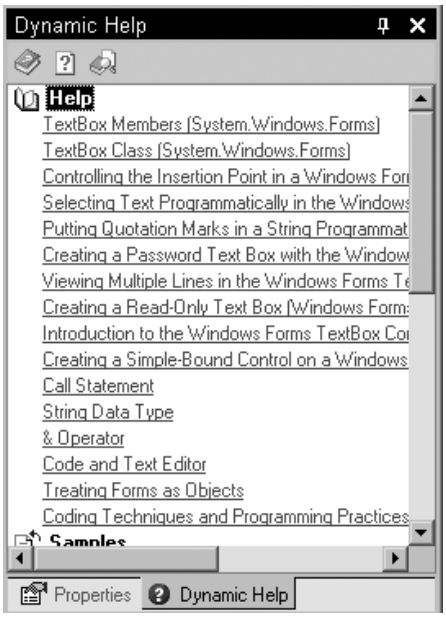

Figure 1-23

The other help commands in the Help menu (Contents, Index, and Search), function just as they would in any other Windows application.

Summary

Hopefully, you are beginning to see that developing basic applications with Visual Basic .NET is not that difficult. You have taken a look at the IDE and saw how it can help you put together software very quickly. The Toolbox allows you to add controls quickly and easily to your programs. The Properties window makes configuring those controls a snap, while the Solution Explorer gives you a bird's eye view of the files that make up your project. You even wrote a little code.

In the coming chapters you will go into even more detail and get comfortable writing code. Before you go too far into Visual Basic .NET itself the next chapter will give you an introduction to the Microsoft .NET Framework. This Framework is what gives all of the .NET languages their ease of use, ease of interoperability, and simplicity in learning.

To summarize, you should now be familiar with:

- The Integrated Development Environment (IDE)
- Adding controls to your form in the Designer
- Setting the properties of your controls
- Adding code to your form in the code window

Chapter 1

Exercises

At this point, you have not covered much about Visual Basic .NET, the language. So these exercises wouldn't be too difficult.

1. What Modified-Hungarian prefix should you use for a combo box? A label? A textbox?
2. (*This assumes you set the Help Filter to Visual Basic and Related.*) Open the Help System and search for MessageBox. Notice how many topics are returned. Change the Help Filter option on the My Profile screen, to No Filter. Repeat the search for MessageBox. Did the Help System return more or fewer topics?
3. When creating a button, how would you make the button respond to a keyboard hot key?

The answers for these questions and those at the end of all the other chapters can be found in Appendix D.