1

Welcome to Visual Basic 2012

WHAT YOU WILL LEARN IN THIS CHAPTER

- Using event-driven programming
- Installing Visual Basic 2012
- Touring the Visual Basic 2012 integrated development environment (IDE)
- Creating a simple Windows program
- Using the integrated Help system

WROX.COM CODE DOWNLOADS FOR THIS CHAPTER

The wrox.com code downloads for this chapter are found at `www.wrox.com/remtitle.cgi?isbn=1118311813` on the Download Code tab. The code is in the `311813 C01.zip` download.

This is an exciting time to enter the world of programming with Visual Basic 2012 and Windows 8. Windows 8 represents the latest Windows operating system from Microsoft and is packed with a lot of new features to make Windows programming fun. Much has changed in the Windows user interface, and Visual Basic 2012 makes it easy to write professional-looking Windows applications as well as web applications and web services. Haven't upgraded to Windows 8 yet? No worries; Visual Basic 2012 also enables you to write professional-looking applications for previous versions of Windows as well.

The goal of this book is to help you use the Visual Basic 2012 programming language, even if you have never programmed before. You will start slowly and build on what you have learned in subsequent chapters. 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 his shoes. Until you find the correct way of giving the instructions, not much is accomplished. Visual Basic 2012 is a language you can use to tell your computer how to do things; but like a child, the computer will understand

only if you explain things very clearly. If you have never programmed before, this sounds like an arduous task, and sometimes it can be. However, Visual Basic 2012 offers an easy-to-use language to explain some complex tasks. Although it never hurts to have an understanding of what is happening at the lowest levels, Visual Basic 2012 frees the programmer from having to deal with the mundane complexities of writing Windows applications. You are free to concentrate on solving real problems.

Visual Basic 2012 helps you create solutions that run on the Microsoft Windows operating systems, such as Windows 8, Windows Server 2008, and Windows Phone. If you are looking at this book, you might have already felt the need or desire to create such programs. Even if you have never written a computer program before, as you progress through the Try It Out exercises in this book, you will become familiar with the various aspects of the Visual Basic 2012 language, as well as its foundations in the Microsoft .NET Framework. You will find that it is not nearly as difficult as you imagined. Before you know it, you will feel quite comfortable creating a variety of different types of programs with Visual Basic 2012.

Visual Basic 2012 can also be used to create web applications and web services, as well as mobile applications that can run on Tablet PCs or smartphones. However, you will begin by focusing on Windows applications before extending your boundaries to other platforms.

EVENT-DRIVEN PROGRAMMING

A Windows program is quite different from yesteryear's 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 entire hallway, passing any obstacles that you may encounter. A DOS program would let you open only certain doors along your stroll.

Windows, on the other hand, opened up the world of *event-driven programming*. *Events* in this context include clicking a button, resizing a window, or changing an entry in a text box. The code that you write responds to these events. In terms of the hallway analogy: In a Windows program, to get to the end of the hall you just click the end of the hall. The hallway itself 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.

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 own 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 why Windows has been so successful. The generic routines are referred to as the Windows *application programming interface* (*API*), and most of the classes in the .NET Framework take care of communicating with those APIs.

Before Visual Basic 1 was introduced to the world in 1991, developers had to be well versed in C and C++ programming, as well as the building blocks of the Windows system itself, the Windows API. This complexity meant that only dedicated and properly trained individuals were capable of turning out software that could run on Windows. Visual Basic changed all 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. When 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). Prior to Visual Basic 5, if you did not like the way a button behaved, you could either buy or create your own, but those 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*, which enabled developers to create their own *ActiveX controls*.

When Microsoft introduced Visual Basic 3, the programming world changed significantly. 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 *Data Access Objects* (*DAOs*), which enabled programmers to manipulate data with the same ease as manipulating the user interface.

Versions 4 and 5 extended the capabilities of version 3 to enable developers to target the new Windows 95 platform. 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 provided a new way to access databases with the integration of *ActiveX Data Objects* (*ADOs*). The ADO feature was developed by Microsoft to aid web developers using *Active Server Pages* (*ASP*) to access databases. All the improvements to Visual Basic over the years have 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, most of the restrictions that used to exist were obliterated. In the past, Visual Basic was criticized and maligned as a "toy" language because it did not provide all the features of more sophisticated languages such as C++ and Java. Microsoft removed these restrictions with Visual Basic .NET, which was rapidly adopted as a very powerful development tool. This trend has continued with the release of Visual Basic 2003; Visual Basic 2005; Visual Basic 2008; Visual Basic 2010; and the latest release, Visual Basic 2012. Each new release of the Visual Basic .NET programming language offers many new features, improvements, and trends, making it a great choice for programmers of all levels.

INSTALLING VISUAL BASIC 2012

You may own Visual Basic 2012 in one of the following forms:

- As part of Visual Studio 2012, a suite of tools and languages that also includes C# (pronounced "C-sharp") and Visual C++. The Visual Studio 2012 product line includes Visual Studio Professional Edition or Visual Studio Tools Team Editions. The Team Edition versions come with progressively more tools for building and managing the development of larger, enterprise-wide applications.

- As Visual Basic 2012 Express Edition (a free edition for students and beginners), which includes the Visual Basic 2012 language and a smaller set of the tools and features available with Visual Studio 2012.

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

The descriptions in the following Try It Out exercise are based on installing Visual Studio 2012 Professional Edition Beta 1. Most of the installation processes are straightforward, and you can accept the default installation options for most environments. Therefore, regardless of which edition you are installing, the installation process should be smooth when accepting the default installation options.

TRY IT OUT Installing Visual Basic 2012

There are two common ways to install Visual Studio. You can burn a DVD from a downloaded image or use the web installer. To use the web installer, just follow the on-screen prompts. If you choose to burn a DVD, follow these steps.

The Visual Studio 2012 DVD has an auto-run feature, but if the Setup screen does not appear after inserting the DVD, you need to run `vs_professional.exe` from the root directory of the DVD. To do this, follow these steps:

1. Click the Windows Start menu at the bottom left of your screen and then select Run or browse to the Setup program on the DVD. In the Run dialog, you can click the Browse button to locate the `vs_professional.exe` program on your DVD. Then click the OK button to start the setup program. After the setup program initializes, you will see the initial screen, as shown in Figure 1-1.

FIGURE 1-1

2. The dialog shown in Figure 1-1 shows the location and size of the installation. You just need to agree to the terms and conditions by selecting the check box for the terms. Then you will see a link to Install Visual Studio. After selecting the box, click Install to begin.

> ***NOTE*** *If you are a Windows Vista, Windows 7, or Windows 8 user, you may be prompted that the setup program needs to run, in which case you will need to grant permission to let the setup program continue. After the setup program continues, you can sit back and relax while all the features are being installed. The setup program can take 20 minutes or more depending on the installation features chosen and the speed of your computer.*

3. Once the installation has been completed, you are presented with a dialog informing you of the status of the installation, shown in Figure 1-2. From here, you can launch Visual Studio. Click LAUNCH to begin.

FIGURE 1-2

Now the real fun can begin—so get comfortable, relax, and enter the world of Visual Basic 2012.

THE VISUAL STUDIO 2012 IDE

You don't need Visual Basic 2012 to write applications in the Visual Basic .NET language. The capability to run Visual Basic .NET code is included with the .NET Framework. You could write all your Visual Basic .NET code 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 a lot more efficient. In the same way, by far the easiest way to write in Visual Basic .NET code is by using the Visual Studio 2012 Integrated Development Framework, known as the "IDE". This is what you see when working with Visual Basic 2012: the windows, boxes, and so on. The IDE provides a wealth of features unavailable in ordinary text editors—such as code checking, visual representations of the finished application, and an explorer that displays all the files that make up your project.

The Profile Setup Page

An IDE is a way of bringing together a suite of tools that makes developing software a lot easier. If Visual Studio is not running, fire it up and see what you've got. If you used the default installation, go to your Windows Start menu and then select Visual Studio 11. A splash screen briefly appears, and then you see the Choose Default Environment Settings dialog. Select the Visual Basic Development Settings option and your choice for the documentation installation; then click Start Visual Studio. After Visual Studio configures the environment based on the chosen settings, the Microsoft Development Environment will appear, as shown in Figure 1-3.

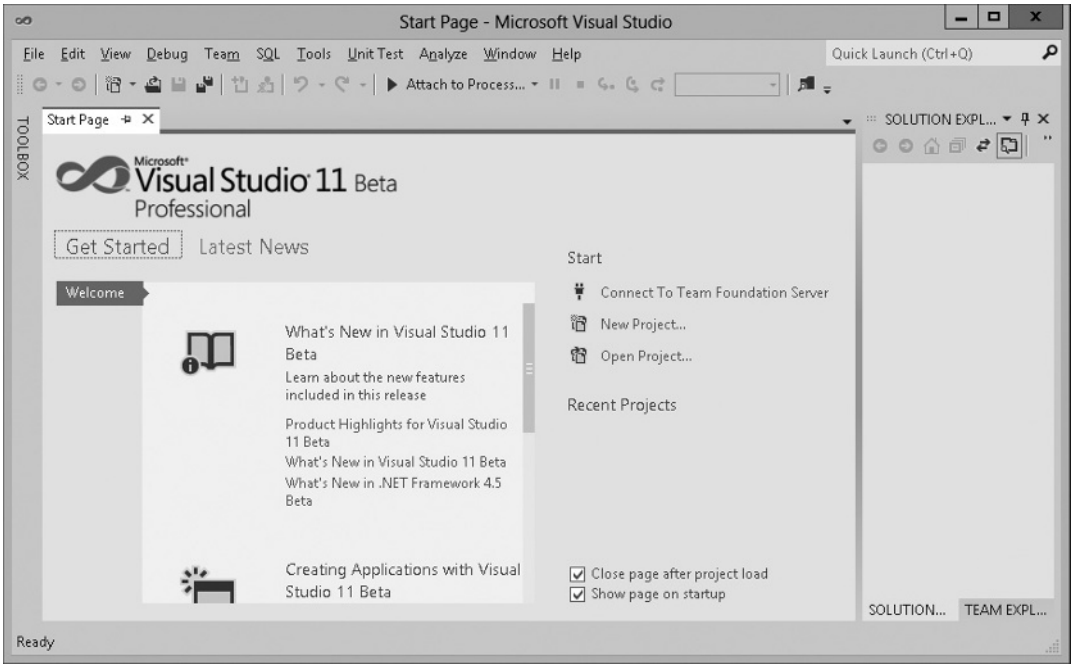

FIGURE 1-3

The Menu

By now, you might be eager to start writing some code; but hold off and begin your exploration of the IDE by looking at the menu and toolbar, which are not really all that different from the toolbars and menus available in other Windows applications (although they differ from the Ribbon in Microsoft Office 2007 and some of the newer Windows applications).

The Visual Studio 2012 menu is dynamic, which means items are added or removed depending on what you are trying to do. When looking at the blank IDE, the menu bar consists only of the File, Edit, View, Build, Debug, Team, SQL, Data, Format, Tools, Unit Test, Analyze, Window, and Help menus. When you start working on a project, however, the full Visual Studio 2012 menu appears, as shown in Figure 1-4.

FIGURE 1-4

At this point, there is no need to cover each menu topic in 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:** Most software programs have 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 common items you would expect: Undo, Redo, Cut, Copy, Paste, and Delete.
- **View:** The View menu provides quick access to the windows that exist in the IDE, such as the Solution Explorer, Properties window, Output window, Toolbox, and so on.
- **Project:** The Project menu enables 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 2012 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 enables you to start and stop running your application within the Visual Basic 2012 IDE. It also gives you access to the Visual Studio 2012 debugger. The debugger enables you to step through your code while it is running to see how it is behaving.
- **Team:** This menu connects to Team Foundation Server. You use this when working with a team to build software.
- **SQL:** The SQL menu allows you to work with a database and create SQL inside of the IDE.

- **Data:** This menu enables you to use information that comes from a database. You can view and add data sources, and preview data. Chapters 15 and 16 introduce you to working with databases.
- **Format:** This menu formats controls. When designing a windows form, you use this menu to finalize the form design, align controls, and resize controls so the form looks perfect.
- **Tools:** The Tools menu has commands to configure the Visual Studio 2012 IDE, as well as links to other external tools that may have been installed.
- **Unit Test:** The Test menu provides options that enable you to create and view unit tests for your application to exercise the source code in various scenarios.
- **Analyze:** The Analyze menu will help you check your code. Use this menu to run performance and code analysis tools to help you write better code.
- **Window:** This 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 enable you to switch between the windows in the IDE.
- **Help:** The Help menu provides access to the Visual Studio 2012 documentation. There are many different ways to access this information (e.g., through the Help contents, an index, or a search). The Help menu also has options that connect to the Microsoft website to obtain updates or report problems.

The Toolbars

Many toolbars are available within the IDE, including Formatting, Image Editor, and Text Editor, which you can add to and remove from the IDE through the View ➪ Toolbars menu option. Each one provides quick access to frequently used commands, preventing you from having to navigate through a series of menu options. For example, the leftmost icon (New Project) on the default toolbar (called the Standard toolbar), shown in Figure 1-5, is available from the menu by navigating to File ➪ New Project.

FIGURE 1-5

The toolbar is segmented into groups of related options, which are separated by vertical bars:

- **Navigation:** The first group of icons is for navigating through code. Use these to go backward and forward as you move through your code.
- **Project and file options:** The next four 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.
- **Code commenting:** The third group of icons is for commenting out and un-commenting sections of code. This process can be useful in debugging when you want to comment out a

section of code to determine which results the program might produce by not executing those lines of code.

- **Managing code edits:** The fourth group of icons is for undoing and redoing edits and for navigating through your code.
- **Code step through:** The fifth group of icons provides the ability to start (via the green triangle), pause, and stop your application. You can also use three icons in this group to step into your code line by line, step over entire sections of code, and step out of a procedure. The solution configuration is used to build your project so you can debug and step into your code or so you can produce a build that you can release to customers. These icons will be covered in depth in Chapter 10.
- **Find in files dialog:** The final icon gives you access to the Find In Files dialog. This is an important feature that you will use often. You can also access this dialog by the shortcut keys Ctrl+F.

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 windows in the IDE by clicking the View menu and choosing the appropriate window; but as you can see, they are all empty at this stage and therefore not very 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 IDE, you need to create a project so that the windows shown earlier in Figure 1-3 have some interesting content for you to look at.

TRY IT OUT Creating a Hello User Project

In this Try It Out exercise, you will create a very simple application called Hello User that will allow you to enter a person's name and display a greeting to that person in a message box:

1. Click the New Project button on the toolbar.
2. In the New Project dialog, select Visual Basic in the Installed Templates tree-view box to the left and then select Windows beneath it. The Templates pane on the right will display all the available templates for the project type chosen. Select the Windows Forms Application template.
3. Type **Hello User** in the Name text box and click OK. Your New Project dialog should look like Figure 1-6.

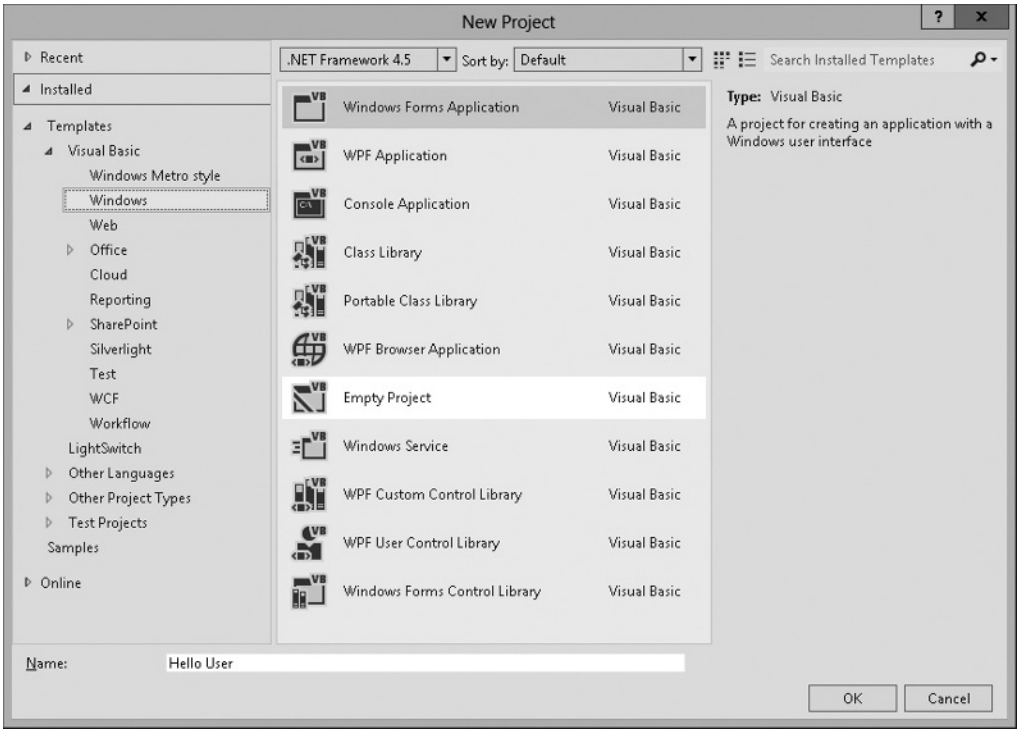

FIGURE 1-6

Visual Studio 2012 allows you to target your application to a specific version of the Microsoft .NET Framework. The combo box at the top of the Templates pane in the New Project dialog has version 4.5 selected, but you can target your application to earlier versions of the .NET Framework.

The IDE then creates an empty Windows application for you. So far, your Hello User 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 1-7.

Whenever Visual Studio 2012 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 2012 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 exercise.

> ***NOTE*** *If any of these windows are not visible on your screen, you can use the View menu to show them. Also, if you do not like the location of any particular window, you can move it by clicking its title bar 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 1-7).*

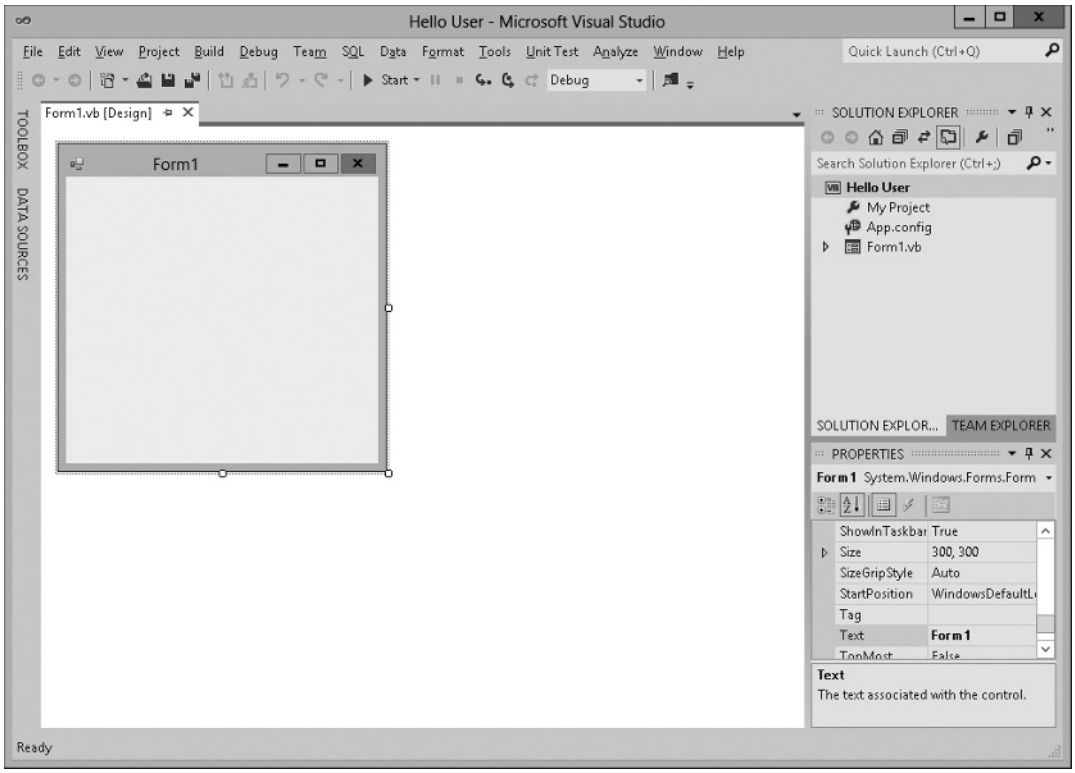

FIGURE 1-7

The following list introduces the most common windows:

- **Toolbox:** This contains reusable controls and components that can be added to your application. These range from buttons to data connectors to customized controls that you have either purchased or developed.
- **Design window:** This 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:** This window contains a hierarchical view of your solution. A *solution* can contain many projects, whereas a *project* contains forms, classes, modules, and components that solve a particular problem.
- **Properties:** This 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 filename for the form's code and layout information.

TRY IT OUT Creating a Hello User Project (*continued*)

Next, you'll give your form a name and set a few properties for it:

1. Change the name of your form to something more indicative of your application. Click `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 1-8.

FIGURE 1-8

2. When changing properties you must either press Enter or click another property for it to take effect. Note that the form's filename has also been updated in the Solution Explorer to read `HelloUser.vb`.
3. Click 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 you have just been looking at).

> ***NOTE*** *The Properties window is dramatically different. This 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 set a control's properties easily. Properties are a particular object's set of internal data; they usually describe appearance or behavior. In Figure 1-9 you can see that properties are displayed alphabetically. The properties can also be grouped together in categories: Accessibility, Appearance, Behavior, Data, Design, Focus, Layout, Misc, and Window Style.

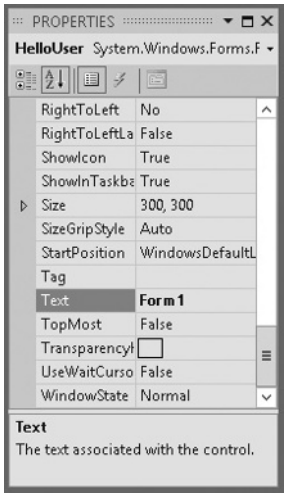

FIGURE 1-9

4. Right now, the title (`Text` property) of your form (displayed in the bar at the top of the form) is `Form1`. This is not very descriptive, so change it to reflect the purpose of this application. Locate the `Text` property in the Properties window. Change the `Text` property's value to **`Hello from Visual Basic 2012`** and press Enter. Note that the form's title has been updated to reflect the change.

> ***NOTE*** *If you have trouble finding properties, click the little AZ icon 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 this procedure. Click the Start button on the Visual Studio 2012 toolbar (the green triangle) to run the application. As you work through the book, whenever we say "run the project" or "start the project," just click the Start button. An empty window with the title Hello from Visual Basic 2012 is displayed.

That was simple, but your little application isn't doing much at the moment. Let's make it a little more interactive. To do this, you will add some controls—a label, a text box, and two buttons—to the form. This will enable you to 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 2012 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 2012 writes a lot of it for you.

The Toolbox

The Toolbox is accessed through the View ➪ Toolbox menu option, by clicking 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; hovering your mouse over this tab will cause the Toolbox window to fly out, partially covering your form.

The Toolbox contains a Node-type 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'll use only the controls in the Common Controls node. Figure 1-10 shows a listing of common controls for Windows Forms.

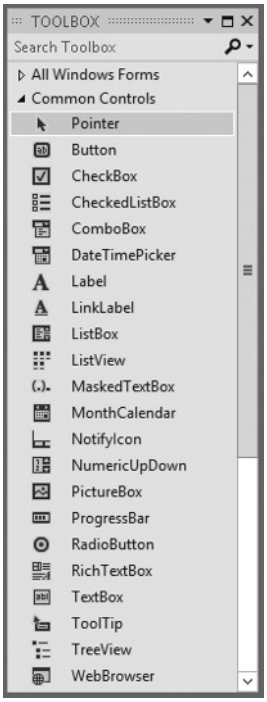

FIGURE 1-10

Controls can be added to your forms in any order, so it doesn't matter whether you add the label control after the text box or the buttons before the label.

TRY IT OUT Adding Controls to the HelloUser Application

In the following Try It Out exercise, you start adding controls.

1. Stop the project if it is still running because you now want to add some controls to your form. The simplest way to stop your project is to click the close (X) button in the top-right corner of the form. Alternatively, you can click the blue square on the toolbar (which displays a ToolTip that says "Stop Debugging" if you hover over it with your mouse pointer).
2. Add a Label control to the form. Click Label in the Toolbox, drag it over to the form's Designer, and drop it in the desired location. (You can also place controls on your form by double-clicking the required control in the Toolbox or clicking the control in the Toolbox and then drawing it on the form.)
3. If the Label control you have just drawn is not in the desired location, no problem. When the control is on the form, you can resize it or move it around. Figure 1-11 shows what the control looks like after you place it on the form. To move it, click the control and drag it to the desired location. The label will automatically resize itself to fit the text that you enter in the `Text` property.

FIGURE 1-11

4. After drawing a control on the form, you 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 the label. In the Properties window, set your new label's `Text` property to **Enter Your Name**. Note that after you press Enter or click another property, the label on the form has automatically resized itself to fit the text in the `Text` property. Now set the `Name` property to **lblName**.
5. Directly beneath the label, you want to add a text box so that you can enter a name. You will repeat the procedure you followed for adding the label, but this time make sure you select the TextBox control from the toolbar. After you have dragged and dropped (or double-clicked) the control into the appropriate position as shown in Figure 1-12, use the Properties window to set its `Name`

property to **txtName**. Notice the sizing handles on the left and right side of the control. You can use these handles to resize the text box horizontally.

FIGURE 1-12

6. In the bottom-left corner of the form, add a Button control in exactly the same manner as you added the label and text box. Set its `Name` property to **btnOK** and its `Text` property to **&OK**. Your form should now look similar to the one shown in Figure 1-13.

FIGURE 1-13

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 1-13) 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 until the user presses Alt). In this particular instance, pressing Alt+O would be the same as clicking the OK button. There is no need to write code to accomplish this.

7. Now add a second Button control to the bottom-right corner of the form by dragging the Button control from the Toolbox onto your form. Notice that as you get close to the bottom right of the form, a blue snap line appears, as shown in Figure 1-14. This snap line enables you to align this

new Button control with the existing Button control on the form. The snap lines assist you in aligning controls to the left, right, top, or bottom of each other, depending on where you are trying to position the new control. The light blue line provides you with a consistent margin between the edge of your control and the edge of the form. Set the `Name` property to **btnExit** and the `Text` property to **E&xit.** Your form should look similar to Figure 1-15.

FIGURE 1-14

FIGURE 1-15

Now, before you finish your sample application, the following section briefly discusses some coding practices that you should be using.

Modified Hungarian Notation

You might 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 as you look 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 text box that accepted a name or with a label that displayed a name. Imagine if, in the previous Try It Out exercise, you had named your label Name1 and your text box Name2—you would very quickly become confused. What 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 for control names within application development in many languages was designed by Dr. Charles Simonyi, who 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. Because Simonyi is from Hungary, and the prefixes make the names look a little foreign, this naming system became known as *Hungarian Notation*. The original notation was used in C/C++ development, so the notation for Visual Basic 2012 is termed *Modified Hungarian Notation*. Table 1-1 shows some of the commonly used prefixes that you'll utilize in this book.

TABLE 1-1 Common Prefixes in Visual Basic 2012

CONTROL	PREFIX
Button	`btn`
ComboBox	`cbo`
CheckBox	`chk`
Label	`lbl`
ListBox	`lst`
MainMenu	`mnu`
RadioButton	`rdb`
PictureBox	`pic`
TextBox	`txt`

Hungarian Notation can be a real time-saver when you are looking at either code someone else wrote or code that you wrote months earlier. However, by far the most important thing is to be consistent in your naming. When you start coding, choose a convention for your naming. It is recommended that you use the de facto standard Modified-Hungarian for Visual Basic 2012, but it is not required. After you pick a convention, stick to it. When modifying others' 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 time to write some code.

The Code Editor

Now that you have the HelloUser form defined, you have to add some code to make it actually do something interesting. You have already seen how easy it is to add controls to a form. Providing the functionality behind those on-screen elements is no more difficult. To add the code for a control, you just double-click the control in question. This opens the Code Editor in the main window, shown in Figure 1-16.

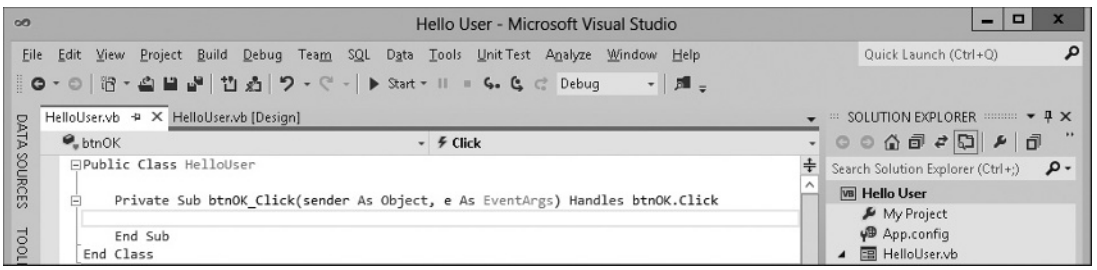

FIGURE 1-16

Note that an additional tab has been created in the main window. Now you have the Design tab and the Code tab, each containing the name of the form you are working on. 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 Visual Studio 2012 has created a separate file for the code. The visual definition and the code behind it exist in separate files: `HelloUser.Designer.vb` and `HelloUser.vb`. This is actually the reason why building applications with Visual Basic 2012 is so slick and easy. Using the design mode you can visually lay out your application; then, using Code view, you add just the bits of code to implement your desired functionality.

Note also the two combo boxes at the top of the window. They provide shortcuts to the various parts of your code. The combo box on the left is the Class Name combo box. If you expand this combo box, you will see a list of all the objects within your form. The combo box on the right is the Method Name combo box. If you expand this combo box, you will see a list of all defined functions and events for the object selected in the Class Name combo box. If this particular form had a lot of code behind it, these combo boxes would make navigating to the desired code area very quick—jumping to the selected area in your code. However, all the code for this project so far fits in the window, so there aren't a lot of places to get lost.

TRY IT OUT Adding Code to the Hello User Project

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

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

End Sub
```

As a result of the typographic constraints in publishing, it may not be possible to put the `Sub` declaration on one line. Visual Basic 2012 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. In some cases, you can break a line of code without the underscore as shown in the code that follows. You will learn more about the rules of breaking lines later.

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

End Sub
```

Sub is an example of a keyword. In programming terms, a *keyword* is a special word that is used to tell Visual Basic 2012 to do something special. In this case, it tells Visual Basic 2012 that this is a *subroutine*, a procedure that does not return a value. 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 bolded code to the procedure:

```
Private Sub btnOK_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnOK.Click
    'Display a message box greeting to the user
    MessageBox.Show("Hello, " & txtName.Text & _
        "! Welcome to Visual Basic 2010.", _
        "Hello User Message")
End Sub
```

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

3. After you have added the preceding code, go back to the Design tab and double-click the Exit button. Add the following bolded code to the `btnExit_Click` event procedure:

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

4. Now that the code is finished, the moment of truth has arrived and you can see your creation. First, however, save your work by using File ➪ Save All from the menu or by clicking the Save All button on the toolbar. The Save Project dialog is displayed, as shown in Figure 1-17, prompting you for a name and location for saving the project.

   By default, a project is saved in a folder with the project name; in this case, Hello User. Because this is the only project in the solution, there is no need to create a separate folder for the solution, which contains the same name as the project, thus the "Create directory for solution" check box is unselected.

5. Now click the Start button on the toolbar. At this point, Visual Studio 2012 will compile the code. *Compiling* is the activity of taking the Visual Basic 2012 source code that you have written and translating it into a form that the computer understands. After the compilation is complete, Visual Studio 2012 *runs* (also known as *executes*) the program, and you'll be able to see the results.

   Any errors that Visual Basic 2012 encounters will display as tasks in the Error List window. Double-clicking a task transports you to the offending line of code. You will learn more about how to debug the errors in your code in Chapter 10.

6. When the application loads, you see the main form. Enter a name and click OK or press the Alt+O key combination (see Figure 1-18).

A window known as a message box appears as shown in Figure 1-19, welcoming the person whose name was entered in the text box on the form—in this case, Chris.

7. After you close the message box by clicking the OK button, click the Exit button on your form. The application closes and you will be returned to the Visual Studio 2012 IDE.

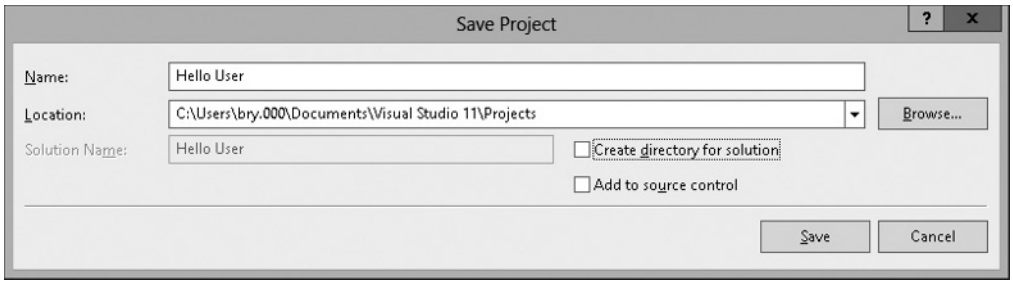

FIGURE 1-17

FIGURE 1-18

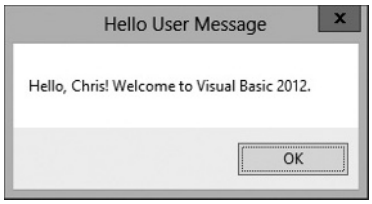

FIGURE 1-19

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 1-19.

The first line of text you entered in this procedure (`'Display a message box greeting to the user`) is actually a *comment*, text that is meant to be read by the human programmer who is writing or maintaining the code, not by the computer. Comments in Visual Basic 2012 begin with a single quote (`'`), and everything following on that line is considered a comment and ignored by the compiler. Comments are discussed in detail in Chapter 3.

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 into one long string is performed through the use of the ampersand (`&`) character.

The code that follows concatenates 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 2012.`" The second parameter passed to the `MessageBox.Show` method is the caption to be used in the title bar of the Message Box dialog.

Finally, the underscore (`_`) character used at the end of the lines in the following code 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 very useful when building long strings because it enables 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(sender As Object,
    e As EventArgs) Handles btnOK.Click
    'Display a message box greeting to the user
    MessageBox.Show("Hello, " & txtName.Text & _
        "! Welcome to Visual Basic 2010.", _
        "Hello User Message")
End Sub
```

The next procedure that you added code for was the Exit button's `Click` event. Here you simply enter this code: `Me.Close()`.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(sender As Object,
    e As 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 2012 is an improvement over the Help systems in the earliest versions of Visual Basic. As you begin to learn Visual Basic 2012, you will probably become very familiar with the Help system. However, a brief overview would be useful, just to help speed your searches for information.

The Help menu contains the items shown in Figure 1-20.

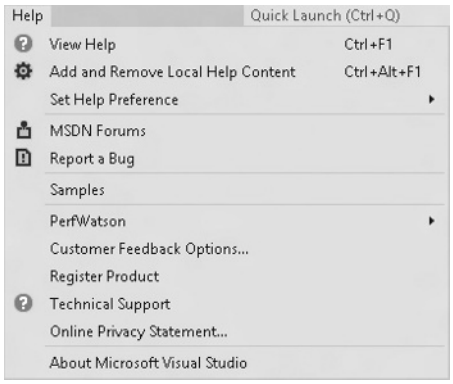

FIGURE 1-20

As you can see, this menu contains a few more items 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, because you can always quickly and easily refer to the Help system or search the forums for people who are experiencing or have experienced a similar programming task. Think of it as a safety net for your brain.

You can also quickly access the Help documentation for a particular subject by simply clicking on a keyword in the Code Editor and pressing the F1 key.

SUMMARY

Hopefully, you are beginning to see that developing basic applications with Visual Basic 2012 is a breeze. You have taken a look at the IDE and have seen how it can help you put together software very quickly. The Toolbox enables you to add controls to your form and design a user interface very quickly and easily. 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 get too far into Visual Basic 2012 itself, however, the next chapter provides an introduction to the Microsoft .NET Framework, which is what gives all the .NET languages their ease of use, ease of interoperability, and simplicity in learning.

EXERCISES

The answers for this exercise and those at the end of each chapter in this book can be found in Appendix A.

1. Create a Windows application with a Textbox control and a Button control that will display whatever is typed in the text box when the user clicks the button.

▶ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	CONCEPTS
The integrated development environment (IDE)	How to create projects in the IDE, how to navigate between Design view and Code view, and how to run and debug projects.
Adding controls to your form in the Designer	How to use the Toolbox to drag and drop controls onto your form, and how to move and resize controls on your form.
Setting the properties of your controls	How to display text in the control and to name the controls something meaningful.
Adding code to your form in the code window	How to add code to control what your program does.