Chapter 1

Getting to Know VBA

In This Chapter

- Finding uses for Visual Basic for Applications (VBA) programs
- Discovering where VBA appears other than in Microsoft Office
- Using the VBA Integrated Development Environment (IDE)
- Writing a one-line program

Have you ever talked with someone about an application that you're using and said that you thought the vendor who created the application was clueless? The application is just too hard or too time consuming to use because the features are difficult to access. In a few cases, I'll bet you saw a feature that *almost* does what you want it to do . . . but not quite. Something that almost works is frustrating to use, and many of us have wished for a solution to the problem.

At some point, someone at Microsoft made something that fixes all these problems and more: Visual Basic for Applications (VBA). VBA is a simple programming language. By using VBA, you can have things your way — you can customize your applications to meet your needs and expectations. No longer are you a slave to what the vendors want. If you use an application that supports VBA, you can add new features — such as automated letter writing and special equation handling — to change things around to the way that you want. In short, it becomes your custom application and not something that the vendor thinks that you want.

VBA works with many applications, including the Microsoft Office applications. You use VBA to write programs to accomplish tasks automatically or change the application environment. Many people think that they can't write even simple programs. This book helps you understand that anyone can write a program. In fact, you write your first program in this chapter. Of course, first you find out the secret handshake for starting the VBA Editor. Using the VBA editor is just a little different from the word processors you've used in the past. Along the way, you see some interesting uses for VBA and just how many applications you can modify by using it.

Batteries Included — VBA Comes with Office

A good many people have written to ask me whether VBA really does come with Office. The answer is yes. All Office products support VBA, and you can use VBA to perform a wealth of tasks, many of which will seem impossible now. Older versions of Office provide a convenient method for accessing the VBA editor. Simply use the Tools➪Macro➪Visual Basic Editor command to display the VBA editor where you type your VBA commands and store them for later use.

One of the reasons for this section is that Microsoft no longer feels that the average user is smart enough to work with VBA. I find it amazing that the company keeps dumbing down its products and making them more difficult to use in the process, but it does. Newer versions of Office hide VBA from view. If you're using a product such as Word 2007, you actually need to look for VBA before you can use it. Don't bother to scour the new Ribbon interface because you won't find it there. The following steps help you reveal the VBA hidden in your copies of Word, Excel, and PowerPoint.

1. **Choose the Word, Excel, or PowerPoint button and click Word Options, Excel Options, or PowerPoint Options.**

   You see the Word Options (see Figure 1-1), Excel Options, or PowerPoint Options dialog box. All three dialog boxes are similar and have the VBA option in the same place.

2. **Check Show Developer Tab in the Ribbon.**

3. **Click OK.**

   Word, Excel, or PowerPoint displays the Developer tab, shown in Figure 1-2, which contains VBA options described in this book.

Depending on which Office 2007 product you use, you'll find the VBA options in different places. You already know that Word, Excel, and PowerPoint place these buttons on the Developer tab of the Ribbon. When working with Access, you'll find the VBA buttons located on the Database Tools tab of the Ribbon. The actual buttons look the same as those shown in Figure 1-2. Even though Outlook does use the new Ribbon interface, you'll find VBA on the Tools➪Macro menu, just as you always have.

Another way in which the Ribbon changes things is that you can no longer right-click a toolbar (because the toolbars don't exist) and choose Customize to add new menu entries. The Ribbon doesn't allow any changes without some programming on your part. Chapter 12 describes the process you use to add new buttons to the Ribbon. Any toolbars you created programmatically with VBA in the past now appear on the Add-Ins tab of the Ribbon, so even programmatically created toolbars have lost some of their effectiveness in Office 2007.

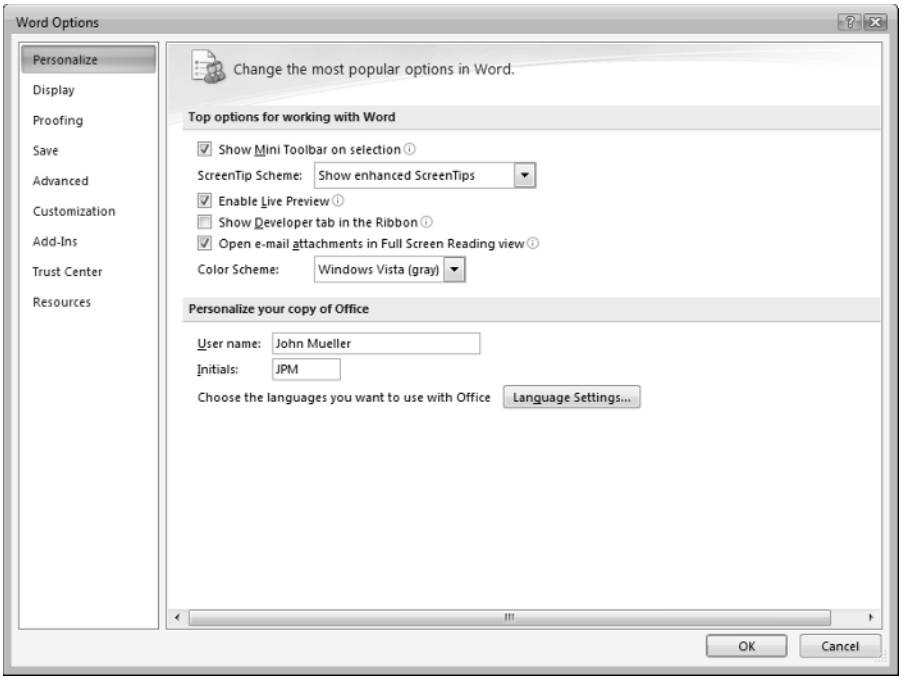

Figure 1-1: The Word Options dialog box helps you configure Word for specific needs.

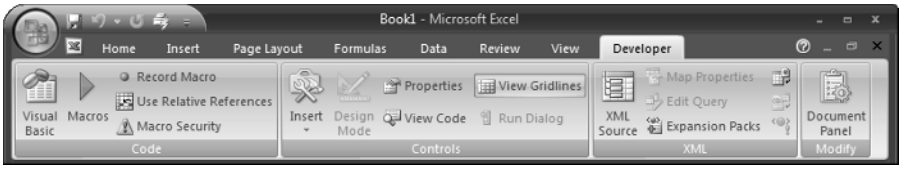

Figure 1-2: The Developer tab of the Ribbon contains the features you used to find in the Tools menu.

Interestingly enough, Microsoft didn't upgrade OneNote, Publisher, Visio, and Project to use the new Ribbon interface. Consequently, you access VBA using the same method you always have on the Tools➪Macro menu. In addition, you'll find that these products lack many of the new features that Microsoft is touting for its core Office products.

VBA: It's Not Just for Programmers

One of the things that you should think about is why you want to use VBA. I know that some of you are probably just interested in using VBA, but most of you need a good reason for taking time out of your busy schedules. It's important to think about what tasks you can use VBA to do. It won't take out the garbage or fold your laundry, but you can use it to write some types of letters automatically. With this in mind, you find out about a few things in this section that I've done with VBA. Knowing you, I'm sure you'll come up with more.

Automating documents

I hate writing letters, especially if the letter contains most of the same information that I wrote for the last letter. Sometimes you can automate letters by using mail merge, but that generally doesn't work too well for individualized letters. In these situations, I set up a form that contains the common information that I include in some letters but not in others. I check off the items that I need for the current letter, and VBA automatically writes it for me. You can see my automated letter secrets in Chapter 13.

Document automation isn't limited to word processing. You can also automate a spreadsheet. I have several programs that I've created for Excel. For example, whenever I get a new client for my business, I click a button, and VBA creates all the required client entries in Excel for me. Because Excel performs the task the same way every time, I can't forget anything and each client receives the same level of high-quality service. You can see techniques for creating automated Excel worksheets in Chapter 14.

If you have to move the data that you create in your word processor or spreadsheet to the Internet, VBA can help make the process nearly automatic. Chapter 16 contains everything that you need to know to move information from one Microsoft Office product to another without the usual modification and reformatting. In Bonus Chapter 1 on the Web site (at `http://www.dummies.com/go/vbafd5e`), you see how to create automated documents in FrontPage. Bonus Chapter 2 shows how to work with Visio. The Visio applications focus on automating drawing tasks, but you'll see other examples as well.

Customizing an application's interface

Sometimes an application feature just bugs you. You could turn it off if it bugs you that much, but that might not be an option if you need that function in

your work. Use VBA to create a new version of the feature with everything that you need and nothing that you don't. For example, I never liked how Word performs a word count, so I created my own program to perform the task. Chapter 12 shows you some of my secrets for taming unruly interfaces.

Changing an application interface to your liking is easy. You can create a customized menu system or toolbars. You can move some interface elements out to a form or get rid of them completely. In addition, any interface change that you want to make is probably doable by using VBA. In addition, you don't necessarily have to use just one interface. You can create programs to change the interface as needed for the task that you're performing. For example, I have a program to switch between book, article, and client document-writing modes. Chapter 7 shows a number of interesting ways to use forms.

Performing calculations

One of the most common uses of special applications is to perform complex calculations. You can create many types of equations by using any of the Microsoft Office products. Sometimes, however, you need to change the data before you can use it or perform the calculation differently depending on the value of one or more inputs. Whenever a calculation becomes too complicated for a simple equation, use VBA to simplify things by solving the calculation problem using small steps rather than one big step. Chapters 4 and 14 show a number of ways to work with calculations.

Sometimes the number that you create using a calculation doesn't mean much — it's just a number until someone makes a decision. Some decisions are easy to make yet repetitive. Chapter 5 shows the methods that your application can use to make decisions automatically with VBA. Smart applications save you more time for playing that game of Solitaire.

Getting stuff from a database

I use Access to store a variety of information — everything from my movie collection to a list of clients that I work with regularly. You use databases to store information, although that doesn't help much if you can't get it out. Use VBA to get the information from your database in the form that you need it. For example, you can display that information on a form so that you can review it, or use that same data to create a report.

I love databases because they provide the most flexible method for storing repetitive information, such as a client list or any other kind of list that you can imagine. Don't assume that databases are so complicated that you'll

never understand how they work. Most productivity databases are actually quite simple to use. All you need is a little easily understood VBA code to gain access to them. Chapter 15 shows you everything you need to know to work with productivity databases.

VBA even includes ways of creating temporary databases for those lists that you need only today. This can save you a lot of time and still force the computer to do the work for you. You can see these alternatives in Chapter 9.

Adding new application features

With all the features that vendors have stuffed into applications, you'd think that every possible need would be satisfied. However, I'm convinced that vendors never actually use the applications that they build. (A nifty new screen saver for Windows is not my idea of a necessary feature.) However, the window-sizing program that I really needed came from a third-party vendor.

Most of this book covers adding new application features. Discover how to add specific features by reading specific chapters. (See the preceding sections to find where.) If you read this book from cover to cover, you'll be able to use VBA to add just about any feature to any product that supports VBA. Your friends will be impressed and think that you're a genius. Maybe your boss will become convinced that you're the most valuable employee in the world and give you a large bonus. Reading this book could make you famous, but more importantly, it will make you less frustrated.

Making special tools

If you have to send information to other people who might not have Microsoft Office and they need the information formatted, you might have to work a long time to find a solution. Chapters 10 and 11 contain two methods for storing information in alternative formats. Chapter 10 uses the trusty text file, and Chapter 11 relies on eXtensible Markup Language (XML) files.

Having things your way

Sometimes I'd just like to scream. Microsoft seems to think that it knows precisely what I want — based on what people tell it. Who these other people are remains a mystery, but I wouldn't trust the person in the dark suit sitting next to you.

Fortunately, you can use VBA to help customize Microsoft's well-intentioned application features. If Word decides that it absolutely must display the information you don't want on startup, store your settings to disk and restore

them every time you launch Word. The use of automatically executing programs (see Chapter 2) can help you have things your own way. Chapter 10 shows you how to store your settings in text format, and Chapter 11 shows you how to store them in XML format.

Other Products Use VBA, Too

Don't assume that VBA is good only if you're using Microsoft Office or a few other Microsoft products. With VBA at your command, you can control a lot of different applications. Go to the Microsoft site `http://msdn.microsoft.com/vba/companies/company.asp` to see a list of companies that have licensed VBA. You'll be amazed at the number of applications that you can work with using VBA. Here are a few of my favorites:

- **Corel products (`http://www.corel.com/`):** Corel makes WordPerfect and Draw. *WordPerfect* is a word processing program that many legal offices still use. One of my first professional writing jobs required the use of WordPerfect. *CorelDRAW,* a drawing program that many professionals enjoy using, supports a wealth of features. All the line art in this book was originally drawn using CorelDRAW, and all my drawing setups are performed automatically by using VBA programs.
- **Micrografx iGrafx series (`http://www.micrografx.com/`):** This product can help you create flowcharts or organizational charts. Unlike a lot of drawing tasks, both flowcharts and organizational charts are extremely repetitive, making them a perfect place to use VBA.
- **IMSI TurboCad (`http://www.turbocad.com/`):** I love to work with wood, which means that I have to draw plans for new projects from time to time. TurboCad is the drawing program that I prefer to use. It's relatively inexpensive, and the VBA programs I've created for it automate many of the drawing tasks, such as creating ¾" boards.

VBA hasn't been around forever. If you drag out that old, dusty copy of WordPerfect for DOS, you'll be disappointed because it doesn't support VBA. The Microsoft vendor participant list doesn't tell you which version of a product supports VBA for the most part, so you either have to check the product packaging or ask the vendor.

A Room with a View

Many people approach VBA with the same enthusiasm and clarity of thought with which the condemned person faces the gallows. When you work with an application, you see what the developer wants you to see and not much more. You're in the user room — the one without a view. Approach using VBA like

entering a new room: You now have a room *with* a view — you're the one who sees what will happen and when.

Looking at the Integrated Development Environment (IDE)

VBA is a visual programming environment. That is, you see how your program will look before you run it. Its editor is very visual, using various windows to make your programming experience easy and manageable. You'll notice slight differences in the appearance of the editor when you use it with Vista as compared to older versions of Windows. In addition, you might notice slight differences when using the editor with a core Office application — one that uses the new Ribbon interface. Figure 1-3 shows what this Integrated Development Environment (IDE) looks like when it's opened using Excel in Vista. No matter which Office product and version of Windows you use, the editor has essentially the same appearance (and some small differences), the same menu items, and the same functionality.

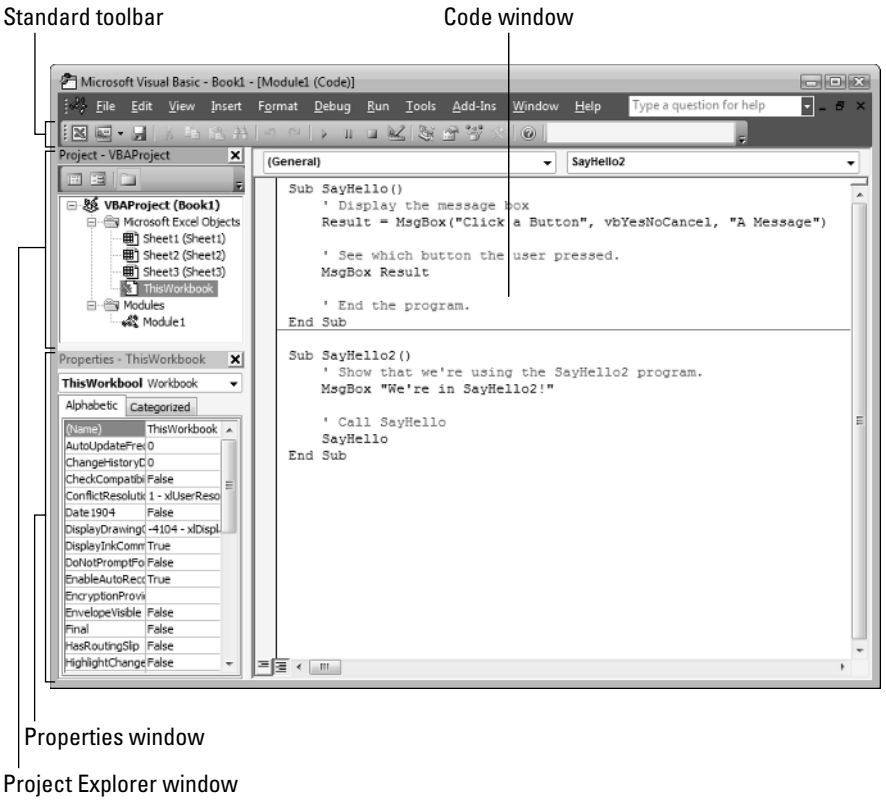

Figure 1-3: The VBA IDE is an editor for writing VBA applications.

An *IDE* is an editor, just like your word processor, spreadsheet, or database form. Just as application editors have special features that make them especially useful for working with data, an IDE is a programming editor with special features that make it useful for writing instructions that the application should follow. These instructions are *procedural code* — a set of steps.

As you can see from Figure 1-3, the VBA IDE consists of a menu system, toolbars, a Project Explorer window, a Properties window, and a Code window, to start with. The IDE can show other windows when it needs to, but these are the three windows that you see when you start VBA. Here's a brief summary of what each of the windows does. (The upcoming "Starting the Visual Basic Editor" section shows how to use them.)

- **Project Explorer:** This window contains a list of the items in your project, which contains all the document elements in a single file. Your application exists within a file that appears in the Project Explorer window.
- **Properties:** Whenever you select an object, the Properties window tells you about it. For example, this window tells you whether the object is blue or whether it has words on it.
- **Code:** Eventually, you have to write some code to make your application work. This window contains the special words that tell your application what to do. Think of it as a place to write a specialized to-do list.

Looking at the VBA Toolbox

You won't have to write code for every task in VBA. The IDE also supports forms, just like the forms that you use to perform other tasks. In this case, you decide what appears on the form and how the form acts when the user works with it. To make it easier to create forms, VBA provides the Toolbox, like the one shown in Figure 1-4, which contains controls used to create forms.

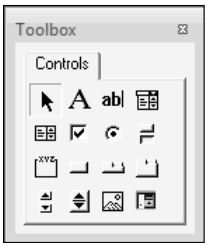

Figure 1-4: Use the VBA Toolbox to add controls to forms you create.

Each Toolbox button performs a unique task. For example, clicking one button displays a text box, but clicking another displays a command button. The form features that these buttons create are *controls.* Chapter 7 shows you

how to use all these controls, as well as how to add other controls when the controls that the Toolbox provides don't meet a particular need.

Looking at objects

You see the term *object* quite a bit while you read this book and use VBA to create your own applications. An object used in a program is very much like an object in real life. Programmers came up with this term to make programs easier to understand. Read on while I use the real-world example of an apple to explain what an object is in VBA — and to understand why objects are such an important part of VBA and how they make things easier.

Property values are up

When you look at an apple, you can see some of its properties: The apple is red, green, or yellow. VBA objects also have properties — for example, a button can have a *caption* (the text that users see when they look at the button). Some of the apple's properties are hidden. You don't know what the apple will taste like until you bite into it. Likewise, some VBA objects have hidden properties.

There's a method to my madness

You can do a number of things with an apple. For example, picking an apple from a tree is a method of interacting with the apple. Likewise, VBA objects have methods. You can move a button from one place to another with the `Move` method. *Methods* let the developer do something to the object.

And now, for a special event!

An apple usually changes color when it ripens. No one did anything to the apple; it turned ripe because it reached maturity. This is an *event.* Likewise, VBA objects can experience events. A user clicks a command button, and the command button generates a `Click` event. As a developer, you didn't do anything to the command button. The command button decides when to generate the event. In short, *events* let the developer react to changing object conditions.

Starting the Visual Basic Editor

How you start the Visual Basic Editor depends on the application that you're using. Newer versions of Office use a different approach than older versions.

In all cases, you see a Visual Basic Editor window, similar to the one shown in Figure 1-3. This section describes each of these variations.

Word 2007, Excel 2007, and PowerPoint 2007

Make sure that you enable the use of VBA by using the procedure in the "Batteries Included — VBA Comes with Office" section, earlier in this chapter. After you have the Developer tab displayed on the Ribbon, select it. Click Visual Basic on the left side of the Developer tab (refer to Figure 1-2). You'll see the Visual Basic Editor.

Access 2007

Access 2007 displays the Database Tools tab of the Ribbon whenever it's possible to use the Visual Basic Editor. Because you must have a database open and meet certain other conditions, you won't always see the Database Tools tab. When you do see this tab, select it and click Visual Basic. You'll see the Visual Basic Editor.

OneNote 2007, Publisher 2007, Visio 2007, Project 2007, and all older versions of Office

If you're using any of the products listed in the heading to this section, start the Visual Basic Editor by choosing Tools➪Macro➪Visual Basic Editor. When you execute this command, you'll see the Visual Basic Editor.

Security under Vista

Vista places extra security constraints on Office products. The User Access Control (UAC) makes it impossible to run some macros that would ordinarily work under previous versions of Windows. Even setting the macro security won't help, in some cases, depending on the security policies set by the administrator, your personal security settings, and the task the macro

performs. In general, you want to sign your macros before you use them under Vista. See the "Adding a Digital Signature to Your Creation" section of Chapter 8 for details.

Setting macro security for Word 2007, Excel 2007, PowerPoint 2007, and Access 2007

Office 2007 sets the security bar very high. It's unlikely that you'll be able to run most of the macros in this book without changing your security settings. The following steps help you make the required changes:

1. **Select the Developer or Database Tools tab on the Ribbon.**
2. **Click Macro Security.**

   You see the Trust Center dialog box, shown in Figure 1-5.

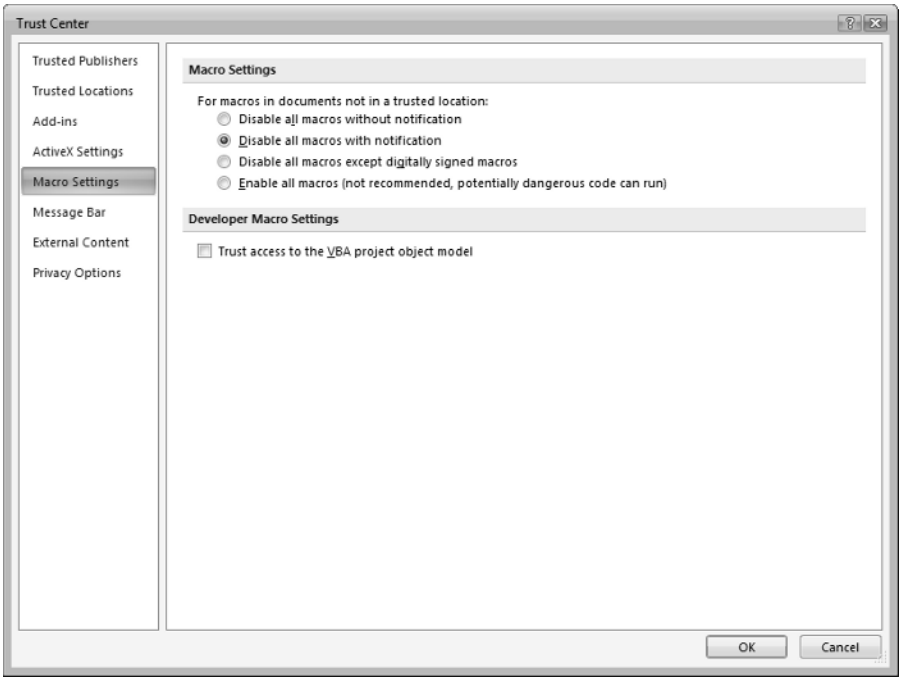

Figure 1-5: Use the Trust Center to adjust the security settings for your Office product.

3. **Select Enable All Macros unless you plan to sign each of the macros in this book before running them.**
4. **Check Trust Access to the VBA Project Object Model.**
5. **Click OK.**

   You can now run macros, but with greatly reduced security. Make sure you change the settings back as soon as possible.

Setting macro security for OneNote 2007, Publisher 2007, Visio 2007, Project 2007, and all older versions of Office

Depending on which version of Microsoft Office you use and how you set it up at the beginning, the macro security feature might be set too high to allow you to use the examples in this book. To change the macro security level, use the following procedure.

1. **Choose the Tools➪Options command.**

   The Microsoft Office application displays the Options dialog box.
2. **Select the Security tab.**
3. **Click Macro Security.**

   The Microsoft Office application displays the Security dialog box.
4. **Select the Security Level tab and choose the Low option.**
5. **Click OK twice to close the Security and Options dialog boxes.**

Using Project Explorer

Project Explorer appears in the Project Explorer window. You use it to interact with the objects that make up a project. A *project* is an individual file used to hold your program, or at least pieces of it. The project resides within the Office document that you're using, so when you open the Office document, you also open the project. See Chapter 3 for a description of how projects and programs interact. Project Explorer works much like how the left pane of Windows Explorer does. Normally, you see just the top-level objects, like the Excel objects shown in Figure 1-6.

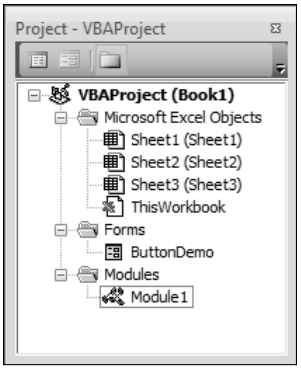

Figure 1-6: Use Project Explorer to work with project objects.

The objects listed in Project Explorer depend on the kind of application that you're working with. For example, if you're working with Word, you see documents and document templates. Likewise, if you're working with Excel, you see worksheets and workbooks. However, no matter what kind of application you work with, the way that you use Project Explorer is the same.

Figure 1-6 also shows some special objects. A project can contain forms, modules, and class modules. Here's a description of these special objects:

- **Forms:** Contain user interface elements and help you interact with the user. Chapter 7 shows how to work with forms.
- **Modules:** Contain the nonvisual code for your application. For example, you can use a module to store a special calculation. Most of this book contains modules.
- **Class modules:** Contain new objects that you want to build. You can use a class module to create a new data type. Chapter 8 shows how to work with objects.

To select an object so that you can see and change its properties, highlight it in Project Explorer. To open the object so that you can modify it, double-click the object.

Right-clicking everything

Project Explorer has a number of hidden talents, which you can find by right-clicking objects to see what you can do with them. For example, right-click the `VBAProject (Book1)` entry at the top of Figure 1-6 to see the context menu shown in Figure 1-7.

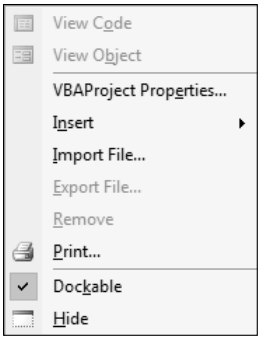

Figure 1-7: Right-click VBA objects to display context menus.

It's amazing to see what's hidden on this menu. Don't worry about using all of the menu entries now. Each of the menu entries appears at least once and probably more often in the book. For example, Chapter 3 shows how to use the `VBAProject` Properties entry. The important thing to remember now is that most objects have context menus that you can access by right-clicking or using the Context Menu button on your keyboard.

Working with special entries

Sometimes you see a special entry in Project Explorer. For example, when you work with a Word document, you might see a References folder, which contains any references that the Word document makes. Normally, it contains a list of templates that the document relies upon for formatting.

In many cases, you can't modify the objects in the special folders. This is the case with the References folder used by Word document objects. The References folder is there for information only. To modify the referenced template, you need to find its object in Project Explorer. In this book, I don't discuss special objects because you normally don't need to work with them.

Using the Properties window

Most of the objects that you click in the VBA IDE have properties that describe the object in some way. The earlier "Property values are up" section of this chapter tells about properties if you haven't worked with them before. The following sections provide details about the Properties window (refer to Figure 1-3).

Understanding property types

A property needs to describe the object. When you look at an object, you naturally assume something about the information provided by a particular

property. For example, when describing the color of an apple, you expect to use *red, yellow,* or *green.* Likewise, VBA object properties have specific types.

One of the most common property types is `text`. The `Caption` property of a form is text. The text appears at the top of the form when the user opens it.

Another common property type is a logic, or Boolean, value. For example, if a control has a `Visible` property and this property is set to `True`, the control appears onscreen. Set this property to `False`, and the control won't appear onscreen even though it still exists as part of the application.

Object properties can also have numeric values. For example, to describe where to place a control onscreen, set the `Top` and `Left` properties to specific numeric values. These values tell how many pixels are between the top and left corner of the screen and the top-left corner of the control.

In some cases, a property can display a drop-down list box from which you can choose the correct value. Other properties display a dialog box like the one for color, shown in Figure 1-8.

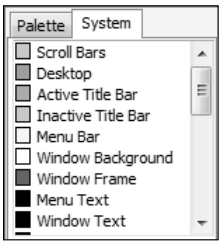

Figure 1-8: Some properties display a dialog box to select the correct value.

Getting help with properties

Don't expect to memorize every property for every object that VBA applications can create. Not even the gurus can do that. To determine what a particular property will do for your application, just highlight the property and press F1, and, in most cases, VBA displays a Help window similar to the one shown in Figure 1-9.

The older versions of Office Help don't include quite as many features as shown in Figure 1-9. For example, you won't find an option to tell Microsoft whether the information is helpful. Notice also that the bottom of the Help window now contains a status bar that tells you whether the information you're seeing is static or taken directly from Microsoft's Web site. Finally, the Standard toolbar now includes a button that looks like a thumbtack. When placed in one position, the Help window always remains on top so that you can see it no matter what you might be doing. When placed in the second position, the Help window hides (like any other window) when you cover it with another window.

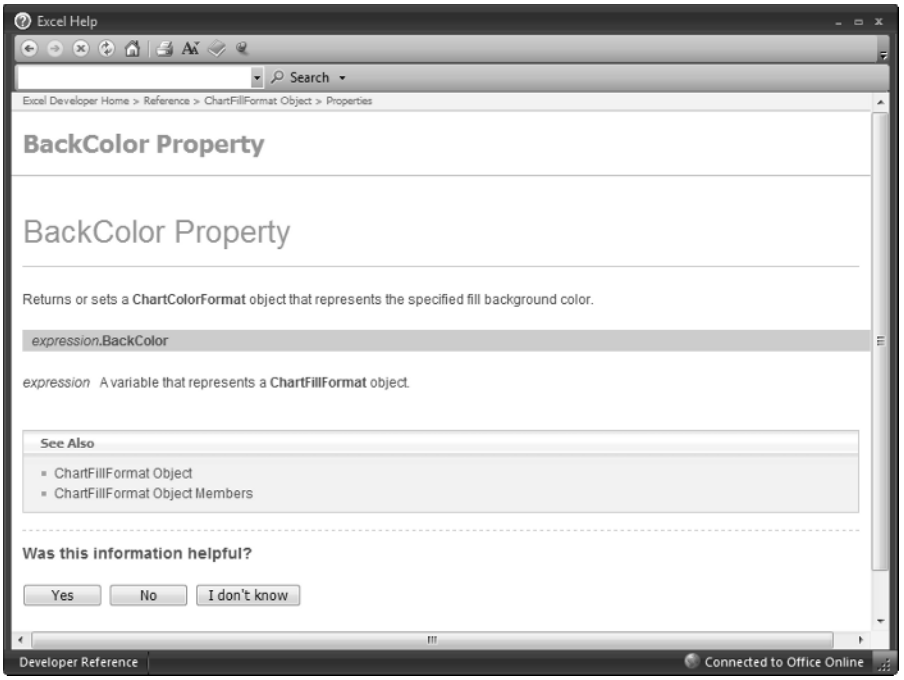

Figure 1-9: Help documents the properties that VBA supports.

Such help screens tell you about the property and how it's used as well as provide you with links for additional information. The additional information is especially important when you start changing the property values in your application code. For example, click the Example link, and the help system shows how to write code that uses that property. (You don't have to click the Example link when working with newer versions of Office — the example appears at the bottom of the help screen.)

Click the See Also link on help screens for more information about a topic, such as info about objects, properties, methods, and events associated with the topic. In some cases, you also get recommended ways to work with an object, property, method, or event. (You don't have to click the See Also link when working with newer versions of Office — the additional information links appear in the middle or bottom of the help screen.)

Using the Code window

The *Code window* is where you write your application code. It works like any other editor that you've used, except that you type in a special language: VBA. Figure 1-10 shows a typical example of a Code window with some code loaded. Notice that the Project Explorer window and the Properties window are gone — you can display them again by using the View➪Project Explorer

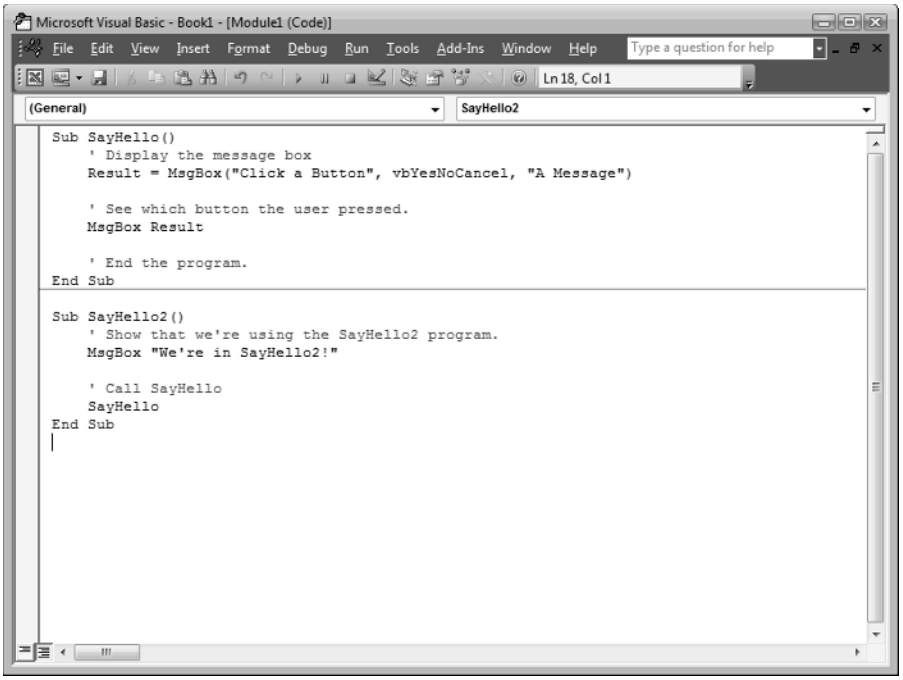

Figure 1-10: Use the Code window to modify your program.

and View➪Properties Window commands. As an alternative, press Ctrl+R to display Project Explorer or F4 to display the Properties window.

Opening an existing Code window

Sometimes you won't be able to complete an application and need to work on it later. To open an existing Code window, find the module that you want to open in Project Explorer. Double-click the module entry, and the IDE displays the code within it with your code loaded.

The Code window also appears when you perform other tasks. For example, if you double-click one of the controls on a form, the Code window appears so that you can add code to the default event handler. VBA calls the *event handler* (special code that responds to the event) every time that the specified event occurs.

Creating a new Code window

When you start a new module within an existing document or template, open a new Code window by using either the Insert➪Module or Insert➪Class Module command. After you save this module or class module, it appears in Project Explorer (refer to Figure 1-3) with the other modules and class modules in your project.

Typing text in the Code window

When you type code, VBA checks what you type. If you make a major error, such as typing a word that VBA doesn't understand, you see an error message explaining what you did wrong (see Figure 1-11). If you don't understand the error, click the Help button for additional information.

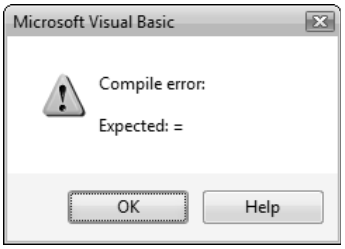

Figure 1-11: VBA displays an error message when you make a mistake.

While you type the code for your application, VBA also formats it. For example, if you type a keyword in lowercase letters, VBA changes it so it appears as shown in the help file. ***Hint:*** Keywords also appear in a different color so that you can easily identify them. This book contains examples of the common VBA keywords.

Finding more Code window features

The Code window has a context menu, just like other objects in VBA. When you right-click the Code window, you see a list of optional actions that you can perform. For example, you can obtain a list of properties and methods that apply to the object that you're currently using in the window. Chapter 3 shows how to use many of the special Code window features.

Getting help with code

Because it's hard to remember precisely how to use every function and method that VBA supports, use the VBA help feature. For any keyword that you type in the Code window, highlight the keyword and press F1, and VBA will look for help on the keyword that you selected.

Make sure that you select the entire keyword, or VBA might not find the information that you need. Double-click the keyword to ensure that you highlight the entire word.

Using the Immediate window

Although you can use the Immediate window for debugging applications, this window can actually help you learn about VBA and save you from having to

write reams of code. You can execute statements one at a time. Use the View➪Immediate Window command to display the Immediate window. This window normally appears at the bottom of the IDE, and it won't contain any information until you type something in it.

Creating a variable in the Immediate window

Most developers spend their days using the Immediate window to check their applications for errors. You can use the Immediate window to ask VBA about the value of a variable, for example. (A *variable* acts as a storage container for a value, such as `Hello World`.) This feature is always available in the VBA IDE, even if you aren't using VBA for anything at the moment. To try this feature, type **MyVal = "Hello World"** (don't forget the double quotes) in the Immediate window and then press Enter. Now type **? MyVal** and then press Enter. Figure 1-12 shows the output of this little experiment.

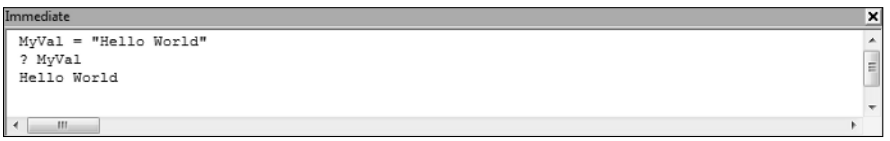

Figure 1-12: Use the Immediate window to check the value of a variable.

You asked VBA to create a variable named `MyVal` and assign it a value of `Hello World`. The next step is to ask VBA what `MyVal` contains by using the `?` operator. Figure 1-12 shows that `MyVal` actually does contain `Hello World`.

Creating a one-line program

Experimenting with the Immediate window is one of the fastest ways to learn how to use VBA because you get instant results. You can also copy successful experiments from the Immediate window and paste them into the Code window. Using this method ensures that your code contains fewer errors than if you type it directly into the Code window.

If you've read the earlier section "Creating a variable in the Immediate window," you created a variable named `MyVal`. The variable still exists in memory unless you closed VBA. You can use this variable for a little experiment — your first program. Type **MsgBox MyVal** into the Immediate window and then press Enter. You see a message box like the one shown in Figure 1-13.

Congratulations! You just completed your first VBA application! The code that you typed asked VBA to use the `MsgBox` function to display the text in the `MyVal` variable. Click OK to clear the message box.

Figure 1-13: The `MsgBox` function produces a message box like this.

Using Object Browser

VBA provides access to a lot of objects, more than you'll use for any one program. With all the objects that you have at your disposal, you might forget the name of one or more of them at some time. Object Browser helps you find the objects that you need. In fact, you can use it to find new objects that could be useful for your next project. Use the View➪Object Browser command to display Object Browser, as shown in Figure 1-14. Normally, you need to filter the information in some way.

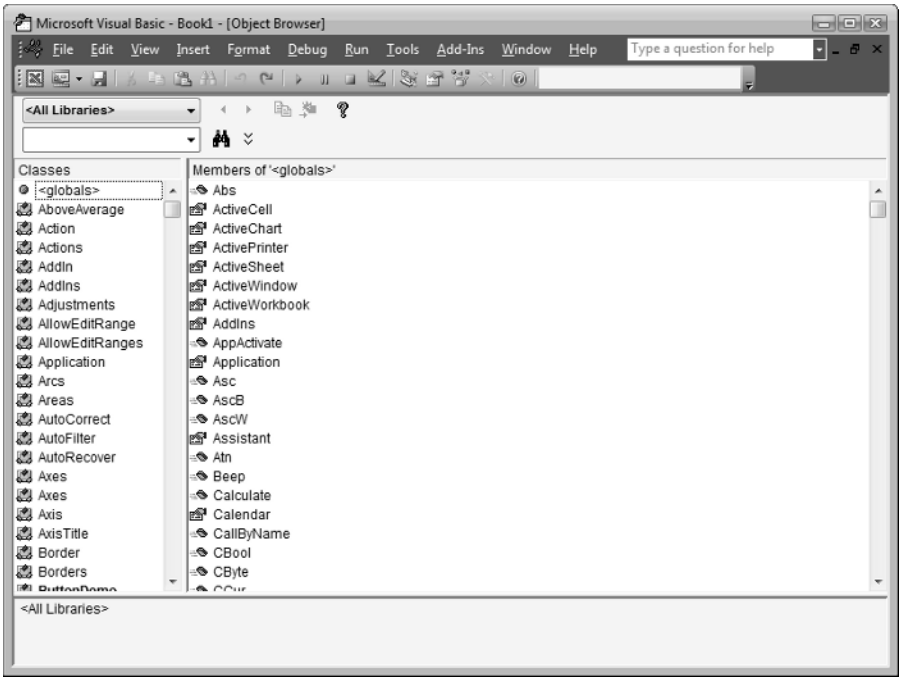

Figure 1-14: View the objects that VBA makes available via Object Browser.

Browsing objects

Object Browser contains a list of the contents of all projects and libraries loaded for the VBA IDE. You can view the list of projects and libraries by using the Project/Library drop-down list box. When you start Object Browser, this list box reads `<All Libraries>`, which means that you're viewing everything that VBA has to offer — usually too much for someone to make sense of it all.

Projects and libraries are different, but you won't normally need to worry about them to use the objects that they contain. A *project* is the VBA code contained in one of the files that you load into the application. In most cases, you use a project to store the code that you create. A *library* is external code contained in a Dynamic Link Library (DLL) file. The DLL contains support routines used by the application or VBA. This code is normally written by a developer using a language such as Visual Basic or Visual C++. You can't easily edit the code in a DLL.

The list of projects and libraries might look complicated at first, but you can narrow it to a few types of entries. Of course, you always see your project templates. In addition to project templates, you find these libraries in the list:

- **Application:** This library has the name of the application, such as Excel or Word. It also includes the features that the application provides for VBA users. For example, the Excel library has a `Chart` object, which contains a list of chart-related methods, properties, and events that Excel supports.
- **Office:** This library contains a list of objects that Microsoft Office supports. For example, this is where you find the objects used to support Office Assistant. Of course, if you're using an application other than Microsoft Office, you won't see this library. Your application might provide an alternative.
- **StdOLE:** This library contains some of the Object Linking and Embedding (OLE) features that you use in the application. For example, when you embed a picture into a Word document, this library provides the required support. You can use this library in your VBA applications, too, but the Office or application-related library usually provides access to objects that are easier and faster to use.
- **VBA:** This library contains special utility objects that VBA developers need. For example, it contains the `MsgBox` function, which I demonstrate in the earlier "Using the Immediate window" section of this chapter.

Whenever you want to browse the libraries for a specific object, limit the amount of material that you have to search by using the options in the Project/Library drop-down list box (*filtering* the content). This is a helpful technique when you perform searches as well.

Looking for names and features in Object Browser

When you remember . . . almost, but not quite . . . the name of a method or other programming feature that you want to use, using the search feature of Object Browser can make your life easier. Simply type the text that you want to look for in the Search Text field (the empty box beneath All Libraries), and then click the Search button (the one with a symbol that looks like binoculars) in Object Browser. The Search Results field shown in Figure 1-15 shows what happens when you look for `MsgBox`.

Whenever you choose (highlight) one of the entries in the Search Results field, the bottom two panes change to show that entry. This feature helps you locate specific information about the search result and see it in context with other methods, properties, and events. Notice that the bottom pane tells you more about the selection item. In this case, it tells you how to use the `MsgBox` function.

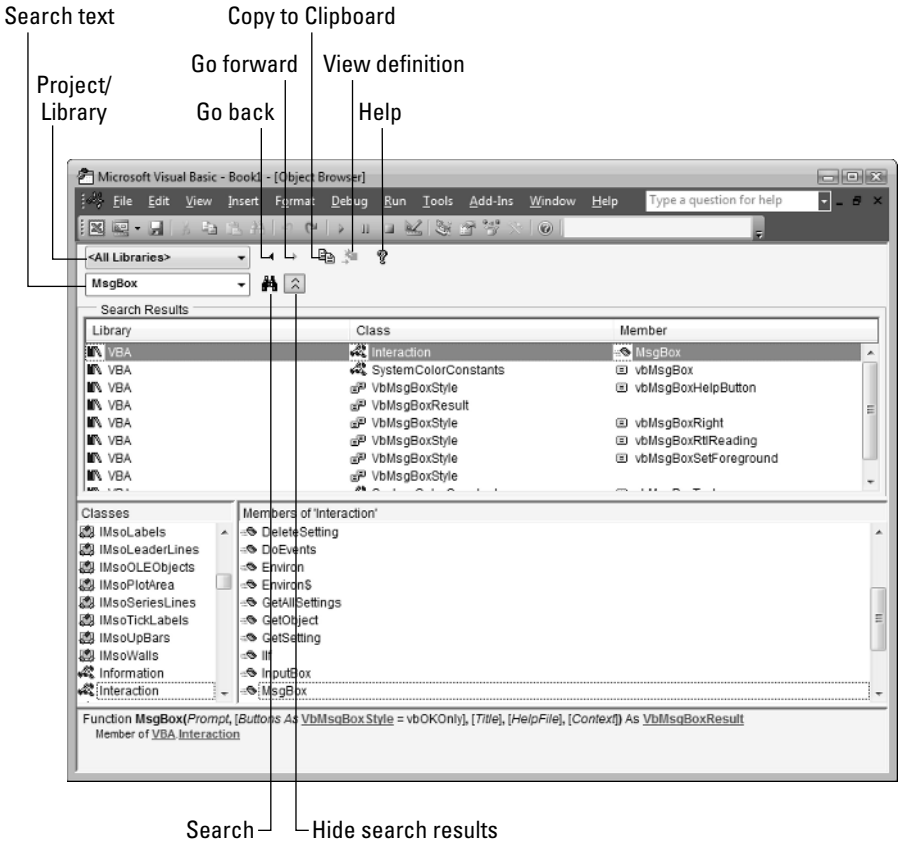

Figure 1-15: Search for the method that you want to use.

Cutting and pasting in Object Browser

Whenever you find a method, property, or event that you want to use in Object Browser, you can copy the information to the Clipboard by clicking the Copy to Clipboard button (the one with a symbol that looks like two documents) and then pasting that information directly into your application code. Using this feature means not only that you type less code, but also that you have fewer errors to consider.

Getting help in Object Browser

Sometimes the information at the bottom of the Object Browser display isn't enough to tell you about the element that you're viewing. When this happens, highlight the element that you want to know more about and press F1, and VBA displays the help screen for that element.