

Chapter 1

What Is VBA?

In This Chapter

- ▶ Gaining a conceptual overview of VBA
 - ▶ Finding out what you can do with VBA
 - ▶ Discovering the advantages and disadvantages of using VBA
 - ▶ Taking a mini-lesson on the history of Excel
-

This chapter is completely devoid of any hands-on training material. It does, however, contain some essential background information that assists you in becoming an Excel programmer. In other words, this chapter paves the way for everything else that follows and gives you a feel for how Excel programming fits into the overall scheme of the universe.

Okay, So What Is VBA?

VBA, which stands for *Visual Basic for Applications*, is a programming language developed by Microsoft — you know, the company run by the richest man in the world. Excel, along with the other members of Microsoft Office 2003, includes the VBA language (at no extra charge). In a nutshell, VBA is the tool that people like you and me use to develop programs that control Excel.



Don't confuse VBA with VB (which stands for Visual Basic). VB is a programming language that lets you create standalone executable programs (those EXE files). Although VBA and VB have a lot in common, they are different animals.



A few words about terminology

Excel programming terminology can be a bit confusing. For example, VBA is a programming language, but it also serves as a macro language. What do you call something written in VBA and executed in Excel? Is it a *macro* or is it a *program*? Excel's Help system often refers to VBA procedures as *macros*, so I use that terminology. But I also call this stuff a *program*.

I use the term *automate* throughout this book. This term means that a series of steps is completed automatically. For example, if you write a

macro that adds color to some cells, prints the worksheet, and then removes the color, you have *automated* those three steps.

By the way, *macro* does not stand for Messy And Confusing Repeated Operation. Rather, it comes from the Greek *makros*, which means large — which also describes your paycheck after you become an expert macro programmer.

What Can You Do with VBA?

You're probably aware that people use Excel for thousands of different tasks. Here are just a few examples:

- ✓ Keeping lists of things such as customer names, students' grades, or holiday gift ideas
- ✓ Budgeting and forecasting
- ✓ Analyzing scientific data
- ✓ Creating invoices and other forms
- ✓ Developing charts from data
- ✓ Yadda, yadda, yadda

The list could go on and on, but I think you get the idea. My point is simply that Excel is used for a wide variety of things, and everyone reading this book has different needs and expectations regarding Excel. One thing virtually every reader has in common is the *need to automate some aspect of Excel*. That, dear reader, is what VBA is all about.

For example, you might create a VBA program to format and print your month-end sales report. After developing and testing the program, you can execute the macro with a single command, causing Excel to automatically perform many time-consuming procedures. Rather than struggle through a tedious sequence of commands, you can grab a cup of joe and let your computer do the work — which is how it's supposed to be, right?

In the following sections, I briefly describe some common uses for VBA macros. One or two of these may push your button.

Inserting a text string

If you often need to enter your company name into worksheets, you can create a macro to do the typing for you. You can extend this concept as far as you like. For example, you might develop a macro that automatically types a list of all salespeople who work for your company.

Automating a task you perform frequently

Assume you're a sales manager and need to prepare a month-end sales report to keep your boss happy. If the task is straightforward, you can develop a VBA program to do it for you. Your boss will be impressed by the consistently high quality of your reports, and you'll be promoted to a new job for which you are highly unqualified.

Automating repetitive operations

If you need to perform the same action on, say, 12 different Excel workbooks, you can record a macro while you perform the task on the first workbook and then let the macro repeat your action on the other workbooks. The nice thing about this is that Excel never complains about being bored. Excel's macro recorder is similar to recording sound on a tape recorder. But it doesn't require a microphone.

Creating a custom command

Do you often issue the same sequence of Excel menu commands? If so, save yourself a few seconds by developing a macro that combines these commands into a single custom command, which you can execute with a single keystroke or button click.

Creating a custom toolbar button

You can customize the Excel toolbars with your own buttons that execute the macros you write. Office workers tend to be very impressed by this sort of thing.

Creating a custom menu command

You can also customize Excel's menus with your own commands that execute macros you write. Office workers are even more impressed by this.

Creating a simplified front end

In almost any office, you can find lots of people who don't really understand how to use computers. (Sound familiar?) Using VBA, you can make it easy for these inexperienced users to perform some useful work. For example, you can set up a foolproof data-entry template so you don't have to waste *your* time doing mundane work.

Developing new worksheet functions

Although Excel includes numerous built-in functions (such as SUM and AVERAGE), you can create *custom* worksheet functions that can greatly simplify your formulas. I guarantee you'll be surprised by how easy this is. (I show you how to do this in Chapter 21.) Even better, the Insert Function dialog box displays your custom functions, making them appear built in. Very snazzy stuff.

Creating complete, macro-driven applications

If you're willing to spend some time, you can use VBA to create large-scale applications complete with custom dialog boxes, onscreen help, and lots of other accoutrements.

Creating custom add-ins for Excel

You're probably familiar with some of the add-ins that ship with Excel. For example, the Analysis ToolPak is a popular add-in. You can use VBA to develop your own special-purpose add-ins. I developed my Power Utility Pak add-in using only VBA.

Advantages and Disadvantages of VBA

In this section I briefly describe the good things about VBA — and I also explore its darker side.

VBA advantages

You can automate almost anything you do in Excel. To do so, you write instructions that Excel carries out. Automating a task by using VBA offers several advantages:

- ✔ Excel always executes the task in exactly the same way. (In most cases, consistency is a good thing.)
- ✔ Excel performs the task much faster than you could do it manually (unless, of course, you're Clark Kent).
- ✔ If you're a good macro programmer, Excel always performs the task without errors (which probably can't be said about you or me).
- ✔ The task can be performed by someone who doesn't know anything about Excel.
- ✔ You can do things in Excel that are otherwise impossible — which can make you a very popular person around the office.
- ✔ For long, time-consuming tasks, you don't have to sit in front of your computer and get bored. Excel does the work, while you hang out at the water cooler.

VBA disadvantages

It's only fair that I give equal time to listing the disadvantages (or *potential* disadvantages) of VBA:

- ✔ You have to learn how to write programs in VBA (but that's why you bought this book, right?). Fortunately, it's not as difficult as you might expect.
- ✔ Other people who need to use your VBA programs must have their own copies of Excel. It would be nice if you could press a button that transforms your Excel/VBA application into a stand-alone program, but that isn't possible (and probably never will be).

A personal anecdote

Excel programming has its own challenges and frustrations. One of my earlier books, *Excel 5 For Windows Power Programming Techniques*, included a disk containing the examples from the book. I compressed these files so that they would fit on a single disk. Trying to be clever, I wrote a VBA program to expand the files and copy them to the appropriate directories. I spent a lot of time writing and debugging the code, and I tested it thoroughly on three different computers.

Imagine my surprise when I started receiving e-mail from readers who could not install the

files. With a bit of sleuthing, I eventually discovered that the readers who were having the problem had all upgraded to Excel 5.0c. (I developed my installation program using Excel 5.0a.) It turns out that the Excel 5.0c upgrade featured a very subtle change that caused my macro to bomb. Because I'm not privy to Microsoft's plans, I didn't anticipate this problem. Needless to say, this author suffered lots of embarrassment and had to e-mail corrections to hundreds of frustrated readers.

- ✔ Sometimes, things go wrong. In other words, you can't blindly assume that your VBA program will always work correctly under all circumstances. Welcome to the world of debugging.
- ✔ VBA is a moving target. As you know, Microsoft is continually upgrading Excel. You may discover that VBA code you've written doesn't work properly with a future version of Excel. Take it from me; I discovered this the hard way, as detailed in the "A personal anecdote" sidebar.

VBA in a Nutshell

A quick and dirty summary follows of what VBA is all about. Of course, I describe all this stuff in semiexcruciating detail later in the book.

- ✔ **You perform actions in VBA by writing (or recording) code in a VBA module.** You view and edit VBA modules using the Visual Basic Editor (VBE).
- ✔ **A VBA module consists of Sub procedures.** A Sub procedure has nothing to do with underwater vessels or tasty sandwiches. Rather, it's computer code that performs some action on or with objects (discussed in a moment). The following example shows a simple Sub procedure called Test. This amazing program displays the result of 1 plus 1.

```
Sub Test()  
    Sum = 1 + 1  
    MsgBox "The answer is " & Sum  
End Sub
```

- ✔ **A VBA module can also have Function procedures.** A Function procedure returns a single value. You can call it from another VBA procedure or even use it as a function in a worksheet formula. An example of a Function procedure (named AddTwo) follows. This Function accepts two numbers (called arguments) and returns the sum of those values.

```
Function AddTwo(arg1, arg2)
    AddTwo = arg1 + arg2
End Function
```

- ✔ **VBA manipulates objects.** Excel provides more than 100 objects that you can manipulate. Examples of objects include a workbook, a worksheet, a cell range, a chart, and a shape. You have many, many more objects at your disposal, and you can manipulate them using VBA code.
- ✔ **Objects are arranged in a hierarchy.** Objects can act as *containers* for other objects. At the top of the object hierarchy is Excel. Excel itself is an object called Application, and it contains other objects such as Workbook objects and CommandBar objects. The Workbook object can contain other objects, such as Worksheet objects and Chart objects. A Worksheet object can contain objects such as Range objects and PivotTable objects. The term *object model* refers to the arrangement of these objects. (See Chapter 4 for details.)
- ✔ **Objects of the same type form a collection.** For example, the Worksheets collection consists of all the worksheets in a particular workbook. The Charts collection consists of all Chart objects in a workbook. Collections are themselves objects.
- ✔ **You refer to an object by specifying its position in the object hierarchy, using a dot as a separator.** For example, you can refer to the workbook Book1.xls as

```
Application.Workbooks("Book1.xls")
```

This refers to the workbook Book1.xls in the Workbooks collection. The Workbooks collection is contained in the Application object (that is, Excel). Extending this to another level, you can refer to Sheet1 in Book1.xls as

```
Application.Workbooks("Book1.xls").Worksheets("Sheet1")
```

As shown in the following example, you can take this to still another level and refer to a specific cell (in this case, cell A1):

```
Application.Workbooks("Book1.xls").Worksheets("Sheet1").Range("A1")
```

- ✔ **If you omit specific references, Excel uses the *active* objects.** If Book1.xls is the active workbook, you can simplify the preceding reference as follows:

```
Worksheets("Sheet1").Range("A1")
```

If you know that Sheet1 is the active sheet, you can simplify the reference even more:

```
Range("A1")
```

✔ **Objects have properties.** You can think of a property as a *setting* for an object. For example, a Range object has such properties as Value and Address. A Chart object has such properties as HasTitle and Type. You can use VBA to determine object properties and to change properties.

✔ **You refer to a property of an object by combining the object name with the property name, separated by a period.** For example, you can refer to the value in cell A1 on Sheet1 as follows:

```
Worksheets("Sheet1").Range("A1").Value
```

✔ **You can assign values to variables.** A *variable* is a named element that stores things. You can use variables in your VBA code to store such things as values, text, or property settings. To assign the value in cell A1 on Sheet1 to a variable called *Interest*, use the following VBA statement:

```
Interest = Worksheets("Sheet1").Range("A1").Value
```

✔ **Objects have methods.** A *method* is an action Excel performs with an object. For example, one of the methods for a Range object is ClearContents. This method clears the contents of the range.

✔ **You specify a method by combining the object with the method, separated by a dot.** For example, the following statement clears the contents of cell A1:

```
Worksheets("Sheet1").Range("A1").ClearContents
```

✔ **VBA includes all the constructs of modern programming languages, including arrays and looping.**

Believe it or not, the preceding list pretty much describes VBA in a nutshell. Now you just have to find out the details. That's the purpose of the rest of this book.

An Excursion into Versions

If you plan to develop VBA macros, you should have some understanding of Excel's history. I know you weren't expecting a history lesson, but this is important stuff.

Here are all the major Excel for Windows versions that have seen the light of day, along with a few words about how they handle macros:

- ✔ **Excel 2:** The original version of Excel for Windows was called Version 2 (rather than 1) so that it would correspond to the Macintosh version. Excel 2 first appeared in 1987 and nobody uses it anymore, so you can pretty much forget that it ever existed.
- ✔ **Excel 3:** Released in late 1990, this version features the XLM macro language. A few people live in a time warp and still use this version.
- ✔ **Excel 4:** This version hit the streets in early 1992. It also uses the XLM macro language. A fair number of people still use this version. (They subscribe to the philosophy *if it ain't broke, don't fix it.*)
- ✔ **Excel 5:** This one came out in early 1994. It was the first version to use VBA (but it also supports XLM). Many people continue to use this version because they are reluctant to move up to Windows 95.
- ✔ **Excel 95:** Technically known as Excel 7 (there is no Excel 6), this version began shipping in the summer of 1995. It's a 32-bit version and requires Windows 95 or Windows NT. It has a few VBA enhancements, and it supports the XLM language. Excel 95 uses the same file format as Excel 5.
- ✔ **Excel 97:** This version (also known as Excel 8) was born in January 1997. It requires Windows 95 or Windows NT. It has *many* enhancements and features an entirely new interface for programming VBA macros. Excel 97 also uses a new file format (which previous Excel versions cannot open).
- ✔ **Excel 2000:** This version's numbering scheme jumped to four digits. Excel 2000 (also known as Excel 9) made its public debut in June 1999. It includes only a few enhancements from a programmer's perspective, with most enhancements being for users — particularly online users.
- ✔ **Excel 2002:** This version (also known as Excel 10 or Excel XP) appeared in late 2001. Perhaps this version's most significant feature is the ability to recover your work when Excel crashes. This is also the first version to use copy protection (known as *product activation*).
- ✔ **Excel 2003:** As I write this book, this is the current version and it is also known as Excel 11. Of all the Excel upgrades I've ever seen (and I've seen them all), Excel 2003 has the fewest new features. In other words, most hard-core Excel users (including yours truly) were very disappointed with Excel 2003.

So what's the point of this mini history lesson? If you plan to distribute your Excel/VBA files to other users, it's vitally important that you understand which version of Excel they use. People using an older version won't be able to take advantage of features introduced in later versions. For example, VBA's Split function was introduced in Excel 2000. If your VBA code uses this function, those running an earlier version of Excel will have problems. Specifically, they will see a "compile error" message, and nothing executes.

