Writing Excel Macros with VBA, 2nd Edition

By Steven Roman, Ph.D.

Publisher : O'Reilly
Pub Date : June 2002
ISBN : 0-596-00359-5
Pages : 560

Table of Contents

To achieve the maximum control and flexibility from Microsoft Excel often requires careful custom programming using the VBA (Visual Basic for Applications) language. *Writing Excel Macros with VBA, 2nd Edition* offers a solid introduction to writing VBA macros and programs, and will show you how to get more power at the programming level: focusing on programming languages, the Visual Basic Editor, handling code, and the Excel object model.

TEAMFLY

Team-Fly®

Table of Content

Copyright © 2002, 1999 O'Reilly & Associates, Inc. All rights reserved.

Originally published under the title *Writing Excel Macros*.

Printed in the United States of America.

Published by O'Reilly & Associates, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly & Associates books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (http://safari.oreilly.com). For more information contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly & Associates, Inc. Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly & Associates, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps. The association between the image of a blue jay and the topic of Excel macros is a trademark of O'Reilly & Associates, Inc.

While every precaution has been taken in the preparation of this book, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Preface

As the title suggests, this book is for those who want to learn how to program Microsoft Excel Version 8 (for Office 97) and Version 9 (for Office 2000).

We should begin by addressing the question, "Why would anyone want to program Microsoft Excel?" The answer is simple: to get more power out of this formidable application. As you will see, there are many things that you can do at the programming level that you cannot do at the user-interface level—that is, with the menus and dialog boxes of Excel. Chapter 1 provides some concrete examples of this.

This book provides an introduction to programming the Excel object model using Visual Basic for Applications (VBA). However, it is not intended to be an encyclopedia of Excel programming. The goal here is to acquaint you with the main points of Excel programming—enough so that you can continue your education (as we all do) on your own. The goal is that after reading this book you should not need to rely on any source other than the Excel VBA Help file or a good Excel VBA reference book and a nice object browser (such as my Enhanced Object Browser, a coupon for which is included in the back of this book).

It has been my experience that introductory programming books (and, sadly, most trade computer books) tend to do a great deal of handholding. They cover concepts at a very slow pace by padding them heavily with overblown examples and irrelevant anecdotes that only the author could conceivably find amusing, making it difficult to ferret out the facts. Frankly, I find such unprofessionalism incredibly infuriating. In my opinion, it does the reader a great disservice to take perhaps 400 pages of information and pad it with another 600 pages of junk.

There is no doubt in my mind that we need more professionalism from our authors, but it is not easy to find writers who have both the knowledge to write about a subject and the training (or talent) to do so in a pedagogical manner. (I should hasten to add that there are a number of excellent authors in this area—it's just that there are not nearly enough of them.) Moreover, publishers tend to encourage the creation of 1000-plus page tomes because of the general feeling among the publishers that a book must be physically wide enough to stand out on the bookshelf! I shudder to think that this might, in fact, be true. (I am happy to say that O'Reilly has not succumbed to this opinion.)

By contrast, *Writing Excel Macros with VBA* is not a book in which you will find much handholding (nor will you find much handholding in any of my books). The book proceeds at a relatively rapid pace from a general introduction to programming through an examination of the Visual Basic for Applications programming language to an overview of the Excel object model. Given the enormity of the subject, not everything is covered, nor should it be. Nevertheless, the essentials of both the VBA language and the Excel object model are covered so that, when you have finished the book, you will know enough about Excel VBA to begin creating effective working programs.

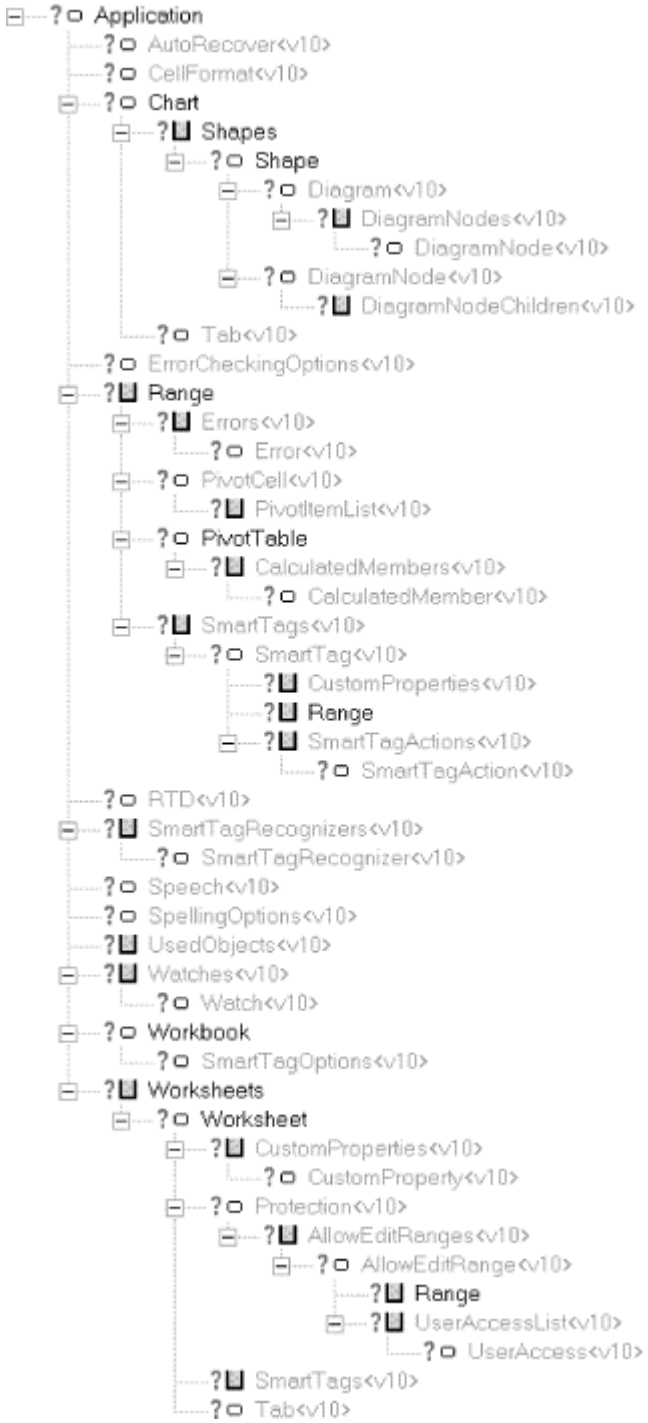
I have tried to put my experience as a professor (about 20 years) and my experience writing books (about 30 of them) to work here to create a true learning tool for my readers. Hopefully, this is a book that can be read, perhaps more than once, and can also serve as a useful reference.

Preface to the Second Edition

With the recent release of Excel 10 (also called Excel XP), it was necessary to update my book. Excel XP is mostly an evolutionary step forward from Excel 2000, but does have some interesting new features worth special attention, such as support for text-to-speed and smart tags.

The Excel object model has 37 new objects, containing 266 new members. There are also 180 new members of preexisting objects. In this book, I cover most of the central objects. Figure P-1 shows most of the new objects in the Excel XP object hierarchy and where these objects occur in the Excel XP object model. (This figure is taken from my program Object Model Browser. For more information on this program, please visit my web site at http://www.romanpress.com.)

Figure P-1. New objects in the Excel XP object hierarchy



The Book's Audience

As an introduction to programming in Excel VBA, the book is primarily addressed to two groups of readers:

- Excel users who are not programmers but who would like to be. If you fall into this category, it is probably because you have begun to appreciate the power of Excel and want to take advantage of its more advanced features or just accomplish certain tasks more easily.
- Excel users who are programmers (in virtually any language—Visual Basic, Visual Basic for Applications, BASIC, C, C++, and so on) but who are not familiar with the Excel object model. In this case, you can use *Writing Excel Macros* to brush up on some of the details of the VBA language and learn about the Excel object model and how to program it.

Organization of This Book

Writing Excel Macros consists of 21 chapters that can informally be divided into four parts (excluding the introductory chapter). In addition, there are five appendixes.

Chapter 1 examines why you might want to learn programming and provides a few examples of the kinds of problems that can best be solved through programming. Chapter 2 introduces programming and the Visual Basic for Applications language.

Chapter 2 through Chapter 4 form the first part of the book. Chapter 3 and Chapter 4 examine the Visual Basic Integrated Development Environment (IDE), which is the programming environment used to develop Excel VBA applications.

The second part of the book consists of Chapter 5 through Chapter 8, which form an introduction to the VBA language, the language component that is common to Microsoft Visual Basic and to many of Microsoft's major applications, including Word, Excel, PowerPoint, and Access, as well as to software from some other publishers. Individual chapters survey VBA's variables, data types, and constants (Chapter 5), functions and subroutines (Chapter 6), intrinsic functions and statements (Chapter 7), and control statements (Chapter 8).

The third part of the book is devoted to some general topics that are needed to create usable examples of Excel applications and to the Excel object model itself. We begin with a discussion of object models in general (Chapter 9). The succeeding chapters discuss what constitutes an Excel application (Chapter 10), Excel events (Chapter 11), Excel menus and toolbars (Chapter 12), and Excel dialog boxes, both built-in and custom (Chapter 13 and Chapter 14). (Those who have read my book *Learning Word Programming* might notice that these topics came at the end of that book. While I would have preferred this organization here as well, I could not construct meaningful Excel examples without covering this material *before* discussing the Excel object model.)

The last chapters of the book are devoted to the Excel object model itself. This model determines which elements of Excel (workbooks, worksheets, charts, cells, and so on) are accessible through code and how they can be controlled programmatically. Chapter 15 gives an overview of the Excel object model. Subsequent chapters are devoted to taking a closer look at some of the main objects in the Excel object model, such as the Application object (Chapter 16), which represents the Excel application itself; the Workbook object (Chapter 17), which represents an Excel workbook; the

Worksheet object (Chapter 18), which represents an Excel worksheet; the Range object (Chapter 19), which represent a collection of cells in a workbook; the PivotTable object (Chapter 20); and the Chart object (Chapter 21). Chapter 22 covers Smart Tags. I have tried to include useful examples at the end of most of these chapters.

The appendixes provide a diverse collection of supplementary material, including a discussion of the Shape object, which can be used to add some interesting artwork to Excel sheets, determining what printers are available on a user's system (this is not quite as easy as you might think), and how to program Excel from other applications (such as Word, Access, or PowerPoint). There is also an appendix containing a very brief overview of programming languages that is designed to give you a perspective on where VBA fits into the great scheme of things.

The Book's Text and Sample Code

When reading this book, you will encounter many small programming examples to illustrate the concepts. I prefer to use small coding examples, hopefully, just a few lines, to illustrate a point.

Personally, I seem to learn much more quickly and easily by tinkering with and tracing through short program segments than by studying a long, detailed example. The difficulty in tinkering with a long program is that changing a few lines can affect other portions of the code, to the point where the program will no longer run. Then you have to waste time trying to figure out why it won't run.

I encourage you to follow along with the code examples by typing them in yourself. (Nevertheless, if you'd rather save yourself the typing, sample programs are available online; see Section P.7 later in this Preface.) Also, I encourage you to experiment -- it is definitely the best way to learn. However, to protect yourself, I *strongly* suggest that you use a throw-away workbook for your experimenting.

One final comment about the sample code is worth making, particularly since this book and its coding examples are intended to teach you how to write VBA programs for Microsoft Excel. Generally speaking, there is somewhat of a horse-before-the-cart problem in trying to write about a complicated object model, since it is almost impossible to give examples of one object and its properties and methods without referring to other objects that may not yet have been discussed. Frankly, I don't see any way to avoid this problem completely, so rather than try to rearrange the material in an unnatural way, it seems better to simply proceed in an orderly fashion. Occasionally, we will need to refer to objects that we have not yet discussed, but this should not cause any serious problems, since most of these forward references are fairly obvious.

About the Code

The code in this book has been carefully tested by at least three individuals—myself, my editor Ron Petrusha, and the technical reviewer, Matt Childs. Indeed, I have tested the code on more than one machine (with different operating systems) and at more than one time (at least during the writing of the book and during the final preparation for book production).

Unfortunately, all three of us have run into some deviations from expected behavior (that is, the code doesn't seem to work as advertised, or work at all) as well as some inconsistencies in code

TEAM FLY

TeamFly®

behavior (that is, it works differently on different systems or at different times). Indeed, there have been occasions when one of us did not get the same results as the others with the same code and the same data. Moreover, I have personally had trouble on occasion duplicating my own results after a significant span of time!

I suppose that this shouldn't be entirely surprising considering the complexity of a program like Excel and the fallibility of us all, but the number of such peccadilloes has prompted me to add this caveat.

Offhand, I can think of two reasons for this behavior—whether it be real or just apparent—neither of which is by any means an excuse:

- The state of documentation being what it is, there may be additional *unmentioned* requirements or restrictions for some code to work properly, or even at all. As an example, nowhere in the vast documentation—at least that I could find—does it say that we cannot use the HasAxis method to put an axis on a chart before we have set the location of the data for that axis! (This seems to me to be putting the cart before the horse, but that is not the issue.) If we try to do so, the resulting error message simply says "Method 'HasAxis' of object '_Chart' has failed." This is not much help in pinpointing the problem. Of course, without being privy to this kind of information from the source, we must resort to experimentation and guesswork. If this does not reveal the situation, it will *appear* that the code simply does not work.
- Computers are not static. Whenever we install a new application, whether it be related to Excel or not, there is a chance that a DLL or other system file will be replaced by a newer file. Sadly, newer files are not always better. This could be the cause, but certainly not the excuse, for inconsistent behavior over time.

The reason that I am bringing this up is to let you know that you may run into some inconsistencies or deviations from expected behavior as well. I have tried to point out some of these problems when they occur, but you may encounter others. Of course, one of our biggest challenges (yours and mine) is to determine whether it is we who are making the mistake and not the program. I will hasten to add that when I encounter a problem with code behavior, I am usually (but not always) the one who is at fault. In fact, sometimes I must remind myself of my students, who constantly say to me, "There is an error in the answers in the back of the textbook." I have learned over 20 years of teaching that 99% of the time (but not 100% of the time), the error is *not* in the book! Would that the software industry had this good a record!

I hope you enjoy this book. Please feel free to check out my web site at http://www.romanpress.com.

Conventions in this Book

Throughout this book, we have used the following typographic conventions:

`Constant width`

indicates a language construct such as a language statement, a constant, or an expression. Lines of code also appear in constant width, as do functions and method prototypes.

Italic

represents intrinsic and application-defined functions, the names of system elements such as directories and files, and Internet resources such as web documents and email addresses. New terms are also italicized when they are first introduced.

`Constant width italic`

in prototypes or command syntax indicates replaceable parameter names, and in body text indicates variable and parameter names.

Obtaining the Sample Programs

The sample programs presented in the book are available online from the Internet and can be freely downloaded from our web site at http://www.oreilly.com/catalog/exlmacro2.

How to Contact Us

We have tested and verified all the information in this book to the best of our ability, but you may find that features have changed (or even that we have made mistakes!). Please let us know about any errors you find, as well as your suggestions for future editions, by writing to:

O'Reilly & Associates
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the U.S. or Canada)
(707) 829-0515 (international/local)
(707) 829-0104 (fax)

There is a web page for this book, where we list any errata, examples, and additional information. You can access this page at:

http://www.oreilly.com/catalog/exlmacro2

To ask technical questions or comment on the book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, software, Resource Centers, and the O'Reilly Network, see our web site at:

http://www.oreilly.com

Acknowledgments

I would like to express my sincerest thanks to Ron Petrusha, my editor at O'Reilly. As with my other books, Ron has been of considerable help. He is one of the best editors that I have worked with over the last 17 years of book writing.

Also, I would like to thank Matt Childs for doing an all-important technical review of the book.

Chapter 1. Introduction

Microsoft Excel is an application of enormous power and flexibility. But despite its powerful feature set, there is a great deal that Excel either does not allow you to do or does not allow you to do easily through its user interface. In these cases, we must turn to Excel programming.

Let me give you two examples that have come up in my consulting practice.

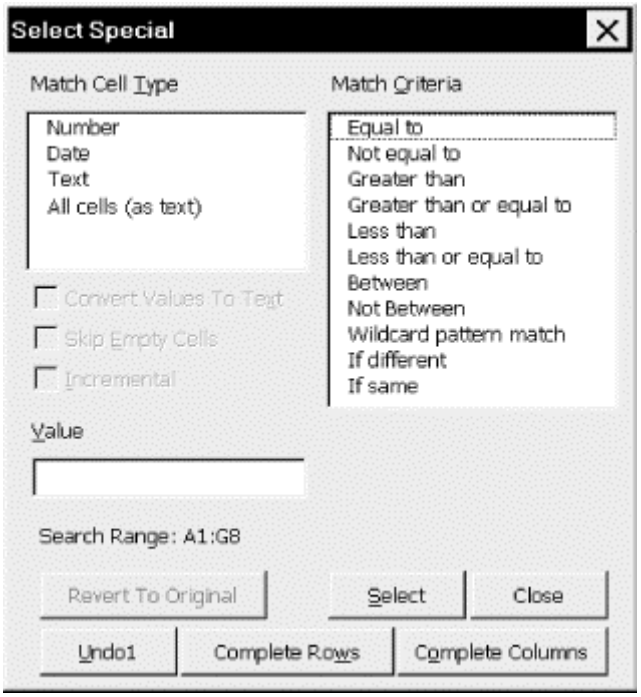
1.1 Selecting Special Cells

The Excel user interface does not have a built-in method for selecting worksheet cells based on various criteria. For instance, there is no way to select all cells whose value is between 0 and 100 or all cells that contain a date later than January 1, 1998. There is also no way to select only those cells in a given column that are different from their immediate predecessors. This can be very useful when you have a sorted column and want to extract a set of unique values, as shown in Figure 1-1.

Figure 1-1. Selecting unique values

I have been asked many times by clients if Excel provides a way to make such selections. After a few such questions, I decided to write an Excel utility for this purpose. The dialog for this utility is shown in Figure 1-2. With this utility, the user can select a match type (such as number, date, or text) and a match criterion. If required, the user supplies one or two values for the match. This has proven to be an extremely useful utility.

Figure 1-2. The Select Special utility



In this book, we will develop a simpler version of this utility, whose dialog is shown in Figure 1-3. This book will also supply you with the necessary knowledge to enhance this utility to something similar to the utility shown in Figure 1-2.

Figure 1-3. Select Special dialog

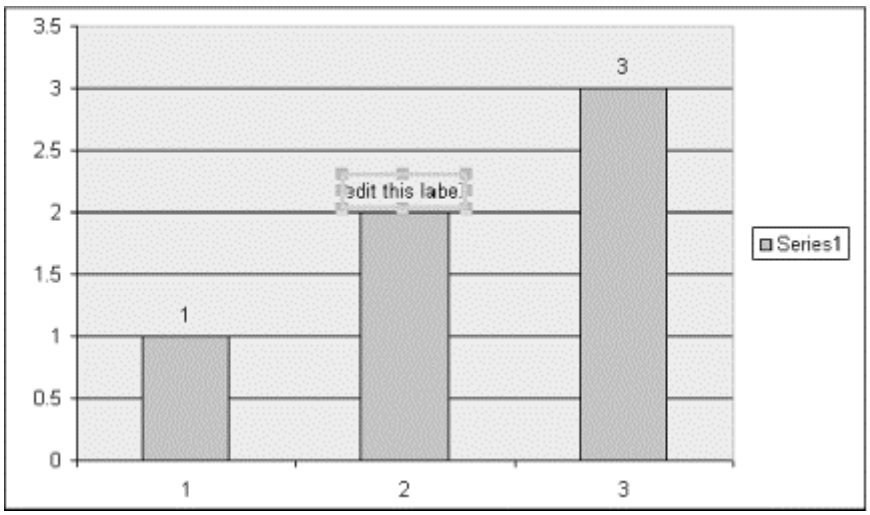
1.2 Setting a Chart's Data Point Labels

As you may know, data labels can be edited individually by clicking twice (pausing in between clicks) on a data label. This places the label in edit mode, as shown in Figure 1-4. Once in edit mode, we can change the text of a data label (which breaks any links) or set a new link to a worksheet cell. Accomplishing the same thing programmatically is also very easy. For instance, the code:

```
ActiveChart.SeriesCollection(1).DataLabels(2).Text =
"=MyChartSheet!R12C2"
```

sets the data label for the second data point to the value of cell B12. Note that the formula must be in R1C1 notation. (We will explain the code in Chapter 21, so don't worry about the details now.)

Figure 1-4. A data label in edit mode



Unfortunately, however, Excel does not provide a simple way to link all of the data labels for a data series with a worksheet range, beyond doing this one data label at a time. In Chapter 21, we will create such a utility, the dialog for which is shown in Figure 1-5. This dialog provides a list of all the data series for the selected chart. The user can select a data series and then define a range to which the data labels will be linked or from which the values will be copied. If the cell values are copied, no link is established, and so changes made to the range are not reflected in the chart. There is also an option to control whether formatting is linked or copied.

Figure 1-5. Set Data Labels dialog

I hope that these illustrations have convinced you that Excel programming can at times be very useful. Of course, you can do much more mundane things with Excel programs, such as automating the printing of charts, sorting worksheets alphabetically, and so on.

1.3 Topics in Learning Excel Programming

In general, the education of an Excel programmer breaks down into a few main categories, as follows.

The Visual Basic Editor

> First, you need to learn a bit about the environment in which Excel programming is done. This is the so-called *Visual Basic Editor* or Excel VBA *Integrated Development Environment* (IDE for short). We take care of this in Chapter 3 and Chapter 4.

The Basics of Programming in VBA

> Next, you need to learn a bit about the basics of the programming language that Excel uses. This language is called *Visual Basic for Applications* (VBA). Actually, VBA is used not only by Microsoft Excel, but also by the other major components in the Microsoft Office application suite: Access, Word, and PowerPoint. Any application that uses VBA in this way is called a *host application* for VBA. (There are also a number of non-Microsoft products that use VBA as their underlying programming language. Among the most notable is Visio, a vector-based drawing program.) It is also used by the standalone programming environment called Visual Basic (VB).
>
> We will discuss the basics of the VBA programming language in Chapter 5 through Chapter 8.

Object Models and the Excel Object Model

> Each VBA host application (Word, Access, Excel, PowerPoint, Visual Basic) supplements the basic VBA language by providing an *object model* to deal with the objects that are particular to that application.
>
> For instance, Excel VBA includes the *Excel object model*, which deals with such objects as workbooks, worksheets, cells, rows, columns, ranges, charts, pivot tables, and so on. On the other hand, the *Word object model* deals with such objects as documents, templates, paragraphs, fonts, headers, tables, and so on. Access VBA includes two object models, the *Access object model* and the *DAO object model*, that allow the programmer to deal with such objects as database tables, queries, forms, and reports. (To learn more about the Word, Access, and DAO object models, see my books *Learning Word Programming* and *Access Database Design and Programming*, also published by O'Reilly.)
>
> Thus, an Excel programmer must be familiar with the general notion of an object model and with the Excel object model in particular. We discuss object models in general in Chapter 9, and our discussion of the Excel object model takes up most of the remainder of the book.

Incidentally, the Excel object model is quite extensive—a close second to the Word object model in size and complexity, with almost 200 different objects.

Lest you be too discouraged by the size of the Excel object model, I should point out that you only need to be familiar with a handful of objects to program meaningfully in Excel VBA. In fact, as we will see, the vast majority of the "action" is related to just seven objects: Application, Range, WorksheetFunction, Workbook, Worksheet, PivotTable, and Chart.

To help you get an overall two-dimensional picture of the Excel object model, as well as detailed local views, I have written special object browser software. (The object browser comes with over a dozen other object models as well.) For more information, please visit http://www.romanpress.com.

Whether you are interested in Excel programming to be more efficient in your own work or to make money writing Excel programs for others to use, I think you will enjoy the increased sense of power that you get by knowing how to manipulate Excel at the programming level. And because Excel programming involves accessing the Excel object model by using the Visual Basic for Applications programming language—the same programming language used in Microsoft Word, Access, and PowerPoint—after reading this book, you will be half-way to being a Word, Access, and PowerPoint programmer as well!

Part I: The VBA Environment

Chapter 2. Preliminaries

We begin with some general facts related to programming and programming languages that will help to give the main subject matter of this book some perspective. After all, VBA is just one of many programming languages, and anyone who wants to be a VBA programmer should have some perspective on where VBA fits into the greater scheme of things. Rest assured, however, that we will not dwell on side issues. The purpose of this chapter is to give a very brief overview of programming and programming languages that will be of interest to readers who have not had any programming experience, as well as to those who have.

2.1 What Is a Programming Language?

Simply put, a programming language is a very special and very restricted language that is understood by the computer at some level. We can roughly divide programming languages into three groups, based on the purpose of the language:

- Languages designed to manipulate the computer at a low level, that is, to manipulate the operating system (Windows or DOS) or even the hardware itself, are called *low-level languages*. An example is assembly language.
- Languages designed to create standalone applications, such as Microsoft Excel, are *high-level languages*. Examples are BASIC, COBOL, FORTRAN, Pascal, C, C++, and Visual Basic.
- Languages that are designed to manipulate an application program, such as Microsoft Excel, are *application-level* languages. Examples are Excel VBA, Word VBA, and PowerPoint VBA.

Those terms are not set in concrete and may be used differently by others. However, no one would disagree that some languages are intended to be used at a lower level than others.

The computer world is full of programming languages—hundreds of them. In some cases, languages are developed for specific computers. In other cases, languages are developed for specific types of applications. Table 2-1 gives some examples of programming languages and their general purposes.

Table 2-1. Some Programming Languages

Language	General Purpose
ALGOL	An attempt to design a universal language
BASIC	A simple, easy-to-learn language designed for beginners
C, C++	A very powerful languages with excellent speed and control over the computer
COBOL	A language for business programming
FORTRAN	A language for scientific programming and number crunching
Lisp	A language for list processing (used in artificial intelligence)
Pascal	A language to teach students how to program "correctly"
SIMULA	A language for simulating (or modeling) physical phenomena
Smalltalk	A language for object-oriented programming
Visual Basic	A version of BASIC designed for creating Windows applications
Visual C++	A version of C++ designed for creating Windows applications

Team-Fly®

Programming languages vary quite a bit in their syntax. Some languages are much easier to read than others (as are spoken languages). As a very simple example, Table 2-2 shows some ways that different programming languages assign a value (in this case, 5) to a variable named X. Notice the variation even in this simple task.

Table 2-2. Assignment in Various Languages

Language	Assignment Statement
APL	`X <- 5`
BASIC	`LET X = 5 or X = 5`
BETA	`5 -> X`
C, C++	`X = 5;`
COBOL	`MOVE 5 TO X`
FORTRAN	`X = 5`
J	`X =. 5`
LISP	`(SETQ X 5)`
Pascal	`X := 5`
Visual Basic	`X = 5`

If you're interested in how Visual Basic compares with some of the other major programming languages, Appendix F contains a short description of several languages, along with some programming examples.

2.2 Programming Style

The issue of what constitutes good programming style is, of course, subjective, just as is the issue of what constitutes good writing style. Probably the best way to learn good programming style is to learn by example and to always keep the issue somewhere in the front of your mind while programming.

This is not the place to enter into a detailed discussion of programming style. However, in my opinion, the two most important maxims for good programming are:

- When in doubt, favor readability over cleverness or elegance.
- Fill your programs with lots of *meaningful* comments.

2.2.1 Comments

Let us take the second point first. It is not possible to overestimate the importance of adding meaningful comments to your programs—at least any program with more than a few lines.

The problem is this: good programs are generally used many times during a reasonably long lifetime, which may be measured in months or even years. Inevitably, a programmer will want to return to his or her code to make changes (such as adding additional features) or to fix bugs. However, despite all efforts, programming languages are not as easy to read as spoken languages. It is just inevitable that a programmer will not understand (or perhaps not even recognize!) code that was written several months or years earlier, and must rely on carefully written comments to help reacquaint himself with the code. (This has happened to me more times that I would care to recall.)

Let me emphasize that commenting code is almost as much of an art as writing the code itself. I have often seen comments similar to the following:

```
' Set x equal to 5
x = 5
```

This comment is pretty useless, since the actual code is self-explanatory. It simply wastes time and space. (In a teaching tool, such as this book, you may find some comments that would otherwise be left out of a professionally written program.)

A good test of the quality of your comments is to read just the comments (not the code) to see if you get a good sense not only of what the program is designed to do, but also of the steps that are used to accomplish the program's goal. For example, here are the comments from a short BASIC program that appears in Appendix F:

```
' BASIC program to compute the average
' of a set of at most 100 numbers

' Ask for the number of numbers

' If Num is between 1 and 100 then proceed
   ' Loop to collect the numbers to average
      ' Ask for next number
      ' Add the number to the running sum
   ' Compute the average
   ' Display the average
```

2.2.2 Readability

Readability is also a subjective matter. What is readable to one person may not be readable to another. In fact, it is probably fair to say that what is readable to the author of a program is likely to be less readable to *everyone else*, at least to some degree. It is wise to keep this in mind when you start programming (that is, assuming you *want* others to be able to read your programs).

One of the greatest offenders to code readability is the infamous `GOTO` statement, of which many languages (including VBA) have some variety or other. It is not my intention to dwell upon the `GOTO` statement, but it will help illustrate the issue of good programming style.

The `GOTO` statement is very simple—it just redirects program execution to another location. For instance, the following BASIC code asks the user for a positive number. If the user enters a nonpositive number, the `GOTO` portion of the code redirects execution to the first line of the program (the label `TryAgain`). This causes the entire program to be executed again. In short, the program will repeat until the user enters a positive number:

```
TryAgain:
INPUT "Enter a positive number: ", x
IF x <= 0 THEN GOTO TryAgain
```

While the previous example may not be good programming style, it is at least readable. However, the following code is much more difficult to read:

```
TryAgain:
INPUT "Enter a number between 1 and 100: ", x
IF x > 100 THEN GOTO TooLarge
IF x <= 0 THEN GOTO TooSmall
PRINT "Your number is: ", x
```

```
GOTO Done
TooLarge:
PRINT "Your number is too large"
GOTO TryAgain
TooSmall:
PRINT "Your number is too small"
GOTO TryAgain
Done:
END
```

Because we need to jump around in the program in order to follow the possible flows of execution, this type of programming is sometimes referred to as *spaghetti code*. Imagine this style of programming in a program that was thousands of lines long! The following version is much more readable, although it is still not the best possible style:

```
TryAgain:
INPUT "Enter a number between 1 and 100: ", x
IF x > 100 THEN
  PRINT "Your number is too large"
  GOTO TryAgain
ELSEIF x <= 0 THEN
  PRINT "Your number is too small"
  GOTO TryAgain
END IF
PRINT "Your number is: ", x
END
```

The following code does the same job, but avoids the use of the `GOTO` statement altogether, and would no doubt be considered better programming style by most programmers:

```
DO
  INPUT "Enter a number between 1 and 100: ", x
  IF x > 100 THEN
    PRINT "Your number is too large"
  ELSEIF x <= 0 THEN
    PRINT "Your number is too small"
  END IF
LOOP UNTIL x >= 1 AND x <= 100
PRINT "Your number is: ", x
END
```

Readability can also suffer at the hands of programmers who like to think that their code is especially clever or elegant but, in reality, just turns out to be hard to read and error-prone. This is especially easy to do when programming in the C language. For instance, as a very simple example, consider the following three lines in C:

```
x = x + 1;
x = x + i;
i = i - 1;
```

The first line adds `1` to *`x`* , the second line adds `i` to *`x`* , and the third line subtracts `1` from *`i`*. This code is certainly readable (if not terribly meaningful). However, it can also be written as:

```
x = ++x+i--;
```

This may be some programmer's idea of clever programming, but to me it is just obnoxious. This is why a sagacious programmer always favors readability over cleverness or elegance.

sample content of Writing Excel Macros with VBA, 2nd Edition

- *click Hanns and Rudolf: The True Story of the German Jew Who Tracked Down and Caught the Kommandant of Auschwitz book*
- The Blood of Olympus (The Heroes of Olympus, Book 5) pdf, azw (kindle)
- read Common Stocks and Uncommon Profits and Other Writings (2nd Edition) (Wiley Investment Classics)
- click Turning to One Another: Simple Conversations to Restore Hope to the Future pdf
- Sweet Land Stories pdf, azw (kindle), epub, doc, mobi

- http://korplast.gr/lib/Operation-Chaos--Operation-Otherworld--Book-1-.pdf
- http://growingsomeroots.com/ebooks/Ostatnie---yczenie.pdf
- http://betsy.wesleychapelcomputerrepair.com/library/Writing-for-Video-Games.pdf
- http://jaythebody.com/freebooks/Stealing-the-General--The-Great-Locomotive-Chase-and-the-First-Medal-of-Honor.pdf
- http://damianfoster.com/books/Sweet-Land-Stories.pdf