FIFTH EDITION

VBA FOR MODELERS

Developing Decision Support Systems with Microsoft® Office Excel®

S. CHRISTIAN ALBRIGHT

VBA FOR MODELERS

DEVELOPING DECISION SUPPORT SYSTEMS WITH MICROSOFT® OFFICE EXCEL®

VBA FOR MODELERS

DEVELOPING DECISION SUPPORT SYSTEMS WITH MICROSOFT® OFFICE EXCEL®

FIFTH EDITION

S. Christian Albright

Kelley School of Business, Indiana University

CENGAGE Learning®

Australia • Brazil • Mexico • Singapore • United Kingdom • United States

CENGAGE Learning®

VBA for Modelers: Developing Decision Support Systems with Microsoft® Office Excel®, Fifth Edition
S. Christian Albright

Vice President, General Manager Science, Math, and Quantitative Business: Balraj Kalsi

Product Director: Joe Sabatino

Product Manager: Aaron Arnsparger

Associate Content Developer: Brad Sullender

Manufacturing Planner: Ron Montgomery

Marketing Manager: Heather Mooney

Art and Cover Direction, Production Management, and Composition: Lumina Datamatics, Inc.

Cover Image: © Awstok/Shutterstock

Intellectual Property

Analyst: Christina Ciaramella

Project Manager: Betsy Hathaway

Library of Congress Control Number: 2014958175

ISBN: 978-1-285-86961-2

Cengage Learning
20 Channel Center Street
Boston, MA 02210
USA

Cengage Learning is a leading provider of customized learning solutions with employees residing in nearly 40 different countries and sales in more than 125 countries around the world. Find your local representative at **www.cengage.com**.

Cengage Learning products are represented in Canada by Nelson Education, Ltd.

To learn more about Cengage Learning Solutions, visit **www.cengage.com**

Purchase any of our products at your local college store or at our preferred online store **www.cengagebrain.com**

Printed in the United States of America
Print Number: 01 Print Year: 2015

To my wonderful wife, Mary—she is my best friend and constant companion. To our talented son, Sam, his equally talented wife, Lindsay, and our two amazing grandsons, Teddy and Archer. And to Bryn, our dear Welsh corgi who still just loves to play ball.

About the Author

S. Christian Albright

Chris Albright got his B.S. degree in Mathematics from Stanford in 1968 and his Ph.D. degree in Operations Research from Stanford in 1972. Until his retirement in 2011, he taught in the Operations & Decision Technologies Department in the Kelley School of Business at Indiana University. His teaching included courses in management science, computer simulation, and statistics to all levels of business students: undergraduates, MBAs, and doctoral students. He has published over 20 articles in leading operations research journals in the area of applied probability and he has authored several books, including *Practical Management Science, Data Analysis and Decision Making, Data Analysis for Managers, Spreadsheet Modeling and Applications*, and *VBA for Modelers.* He jointly developed *StatTools*, a statistical add-in for Excel, with the Palisade Corporation. In "retirement," he continues to revise his books, he works as a consultant for Palisade, and he has developed a commercial product, Excel Now!, an Excel tutorial.

On the personal side, Chris has been married to his wonderful wife Mary for 43 years. They have a special family in Philadelphia: their son Sam, his wife Lindsay, and their two sons, Teddy and Archer. Chris has many interests outside the academic area. They include activities with his family (especially traveling with Mary), going to cultural events at Indiana University, power walking, and reading. And although he earns his livelihood from statistics and management science, his *real* passion is for playing classical music on the piano.

Contents

Preface

I wrote *VBA for Modelers* for students and professionals who want to create decision support systems (DSSs) using Microsoft Excel–based spreadsheet models. The book does *not* assume any prior programming experience. It contains two parts. Part I covers the essentials of VBA (Visual Basic for Applications) programming, and Part II provides many applications with their associated programming code. This part assumes that readers are either familiar with spreadsheet modeling or are taking a concurrent course in management science or operations research. There are many excellent books available for VBA programming, many others covering decision support systems, and still others for spreadsheet modeling. However, I have not found a book that attempts to unify these subjects in a practical way. *VBA for Modelers* is designed for this purpose, and I hope you will find it to be an important resource and reference in your own work.

Why This Book?

The original impetus for this book began about 20 years ago. Wayne Winston and I were experimenting with the spreadsheet approach to teaching management as we were writing the first edition of our *Practical Management Science (PMS)* book. Because I have always had an interest in computer programming, I decided to learn VBA, the relatively new macro language for Excel, and use it to a limited extent in my undergraduate management science modeling course. My intent was to teach the students how to wrap a given spreadsheet model, such as a product mix model, into an *application* with a "front end" and a "back end" by using VBA. The front end would enable a user to provide inputs to the model, usually through one or more dialog boxes, and the back end would present the user with a nontechnical report of the results. I found it to be an exciting addition to the usual modeling course, and my students overwhelmingly agreed.

The primary problem with teaching this type of course was the lack of an appropriate VBA textbook. Although there are many good VBA trade books available, they usually go into much more technical VBA details than I have time to cover, and their objective is usually to teach VBA programming as an end in itself. I expect that many adopters of our *Practical Management Science* book will decide to use parts of *VBA for Modelers* to supplement their management science courses, just as I have been doing. For readers who have already taken a management science course, there is more than enough material in this book to fill an entire elective course or to be used for self-study.

However, even for readers with no background or interest in management science, the first part of this book has plenty of value. We are seeing an increasing

number of our business students and graduates express interest in automating Excel with macros. In short, they want to become Excel "power users." After the first edition of this book appeared, I taught a purely elective MBA course covering the first part of the book. To my surprise and delight, it regularly attracted about 40 MBA students per year. Yes, it attracted *MBA students*, not computer science majors! (Since I have retired from teaching, the VBA course is still being taught, and it continues to attract these types of audiences.). The students see real value in knowing how to program for Excel. And it is amazing and gratifying to see how far these students can progress in a short 7-week course. Many find programming, especially for Excel, to be as addictive as I find it.

Objectives of the Book

VBA for Modelers shows how the power of spreadsheet modeling can be extended to the masses. Through VBA, complex management science models can be made accessible to nontechnical users by providing them with simplified input screens and output reports. The book illustrates, in complete detail, how such applications can be developed for a wide variety of business problems. In writing the book, I have always concerned myself with the following questions: How much will readers be able to do on their own? Is it enough for readers to see the completed applications, marvel at how powerful they are, and possibly take a look at the code that runs in the background? Or should they be taken to the point where they can develop their *own* applications, code and all? I suspect this depends on the audience, but I know I *can* get students to the point where they can develop modest but useful applications on their own and, importantly, experience the thrill of programming success.

With these thoughts in mind, I have written this book so that it can be used at several levels. For readers who want to learn VBA from scratch and then apply it, I have provided a "VBA primer" in Part I of the book. It is admittedly not as complete as some of the thick Excel VBA books available, but I believe it covers the basics of VBA quite adequately. Importantly, it covers coding methods for working with Excel ranges in Chapter 6 and uses these methods extensively in later chapters, so that readers will not have to use trial and error or wade through online help, as I had to do when I was learning VBA. Readers can then proceed to the applications in Chapters 19 through 35 and apply their skills. In contrast, there are probably many readers who do not have time to learn all of the details, but they can still *use* the applications in Part II of the book for demonstration purposes. Indeed, the applications have been developed for generality. For example, the transportation model in Chapter 24 is perfectly general and can be used to solve *any* transportation model by supplying the appropriate input data.

Approach

I like to teach (and learn) through examples. I have found that I can learn a programming language only if I have a strong motivation to learn it. I suspect that

most of you are the same. The applications in the latter chapters are based on many interesting management science models. They provide the motivation for you to learn the material. The examples illustrate that this book is not about programming for the sake of programming. Instead, it is about developing useful applications for business. You probably already realize that Excel modeling skills make you more valuable in the workplace. This book will help you develop VBA skills that make you much *more* valuable.

Contents of the Book

The book is written in two parts. Part I, Chapters 1–18, is a VBA primer for readers with little or no programming experience in VBA (or any other language). Although all of these chapters are geared to VBA, some are more about general programming concepts, whereas others deal with the unique aspects of programming for Excel. Specifically, Chapters 7, 9, and 10 discuss control logic (If-Then-Else constructions), loops, arrays, and subroutines, topics that are common to all programming languages. In contrast, Chapters 6 and 8 explain how to work with some of the most common Excel objects (ranges, workbooks, worksheets, and charts) in VBA. In addition, several chapters discuss aspects of VBA that can be used with Excel and any other applications (Access, Word, PowerPoint, and so on) that use VBA as their programming language. Specifically, Chapter 3 explains the Visual Basic Editor (VBE), Chapter 4 illustrates how to record macros, Chapter 11 explains how to build user forms (dialog boxes), and Chapter 12 discusses the important topic of error handling.

The material in Part I is reasonably complete, but it is available, in greater detail and with a somewhat different emphasis, in several other books. The unique aspect of *this* book is Part II, Chapters 19–35. (Due to length, the last two chapters, Chapter 34, An AHP Application for Choosing a Job, and Chapter 35, A Poker Simulation Application, are available online only. You can find them at www.CengageBrain.com.) Each chapter in this part discusses a specific application. Most of these are optimization and simulation applications, and many are quite general. For example, Chapter 21 discusses a general product mix application, Chapter 23 discusses a general production scheduling application, Chapter 24 discusses a general transportation application, Chapter 25 discusses a stock-trading simulation, Chapter 29 discusses a multiple-server queue simulation, Chapter 30 discusses a general application for pricing European and American options, and Chapter 32 discusses a general portfolio optimization application. (Many of the underlying models for these applications are discussed in *Practical Management Science*, but I have attempted to make these applications stand-alone here.)

The applications can be used as they stand to solve real problems, or they can be used as examples of VBA application development. All of the steps in the development of these applications are explained, and all of the VBA source code is included. Using an analogy to a car, you can simply get in and drive, or you can open the hood and see how everything works.

Chapter 19 gets the process started in a "gentle" way. It provides a general introduction to application development, with an important list of guidelines. It then illustrates these guidelines in a car loan application. This application should be within the grasp of most readers, even if they are not yet great programmers. By tackling this application first, readers get to develop a simple model, with dialog boxes, reports, and charts, and then tie everything together. This car loan application illustrates an important concept that I stress throughout the book. Specifically, applications that really *do* something are often long and have a lot of details. But this does not mean that they are *difficult*. With perseverance—a word I use frequently—readers can fill in the details one step at a time and ultimately experience the thrill of getting a program to work correctly.

Virtually all management science applications require input data. A very important issue for VBA application development is how to get the required input data into the spreadsheet model. I illustrate a number of possibilities in Part II. If only a small amount of data is required, dialog boxes work well. These are used for data input in many of the applications. However, there are many times when the data requirements are much too large for dialog boxes. In these cases, the data are usually stored in some type of database. I illustrate some common possibilities. In Chapter 21, the input data for a product mix model are stored in a separate worksheet. In Chapter 31, the stock price data for finding the betas of stocks are stored in a separate Excel workbook. In Chapter 33, the data for a DEA model are stored in a text (.txt) file. In Chapter 24, the data for a transportation model are stored in an Access database (.mdb) file. Finally, in Chapter 32, the stock price data required for a portfolio optimization model are located on a Web site and are imported into Excel, *at runtime*. In each case, I explain the VBA code that is necessary to import the data into the Excel application.

New to the Fifth Edition

The impetus for writing the fifth edition was the release of Excel 2013. In terms of VBA, there aren't many changes from Excel 2010 to Excel 2013 (or even from Excel 2007 to Excel 2013), but I used the opportunity to incorporate changes that were made in Excel 2013, as well as to modify a lot of the material throughout the book.

- Programmers can never let well enough alone. We are forever tinkering with our code, not just to make it work better, but often to make it more elegant and easier to understand. So users of previous editions will see minor changes to much of the code throughout the book.
- The biggest change, which has nothing to do with the version of Excel, is the way information is passed between modules and user forms. In previous editions, I did this with global variables, a practice frowned upon by many professional programmers. In this edition, I pass the required information through arguments to "ShowDialog" functions in the user forms. This new method is explained in detail in Chapter 11 and is then used in later chapters where user forms appear.

- Chapter 15 contains a brief discussion of the new PowerPivot tool introduced in Excel 2013. This tool can actually be automated with VBA, but because of its advanced nature, I don't discuss the details. Maybe this will appear in the *next* edition of the book, by which time Excel's online help will hopefully be improved.

How to Use the Book

I have already discussed several approaches to using this book, depending on how much you want to learn and how much time you have. For readers with very little or no computer programming background who want to learn the fundamentals of VBA, Chapters 1–12 should be covered first, in approximately that order. (I should point out that it is practically impossible to avoid "later" programming concepts while covering "early" ones. For example, I admit to using a few If statements and loops in early chapters, *before* discussing them formally in Chapter 7. I don't believe this should cause problems. I use plenty of comments, and you can always look ahead if you need to.) After covering VBA fundamentals in the first 12 chapters, the next six optional chapters can be covered in practically any order.

Chapter 19 should be covered next. Beyond that, the applications in the remaining chapters can be covered in practically any order, depending on your interests. However, some of the details in certain applications will not make much sense without the appropriate training in the management science models. For example, Chapter 34 discusses an AHP (Analytical Hierarchy Process) application for choosing a job. The VBA code is fairly straightforward, but it will not make much sense unless you have some knowledge of AHP. I assume that the knowledge of the models comes from a separate source, such as *Practical Management Science*; I cover it only briefly here.

Finally, readers can simply use the Excel application files to solve problems. Indeed, the applications have been written specifically for nontechnical end users, so that readers at all levels should have no difficulty opening the application files in Part II of the book and using them appropriately. In short, readers can decide how much of the material "under the hood" is worth their time.

Premium Web Site Content

The companion Web site for this book can be accessed at www.cengagebrain.com. There you will have access to all of the Excel (.xlsx and .xlsm) and other files mentioned in the chapters, including those in the exercises. The Excel files require Excel 97 or a more recent version, but they are realistically geared to Excel 2007 and later versions. Many of the files from Chapter 17 and later chapters "reference" Excel's Solver. They will not work unless the Solver add-in is installed and loaded. Chapters 14 and 24 uses Microsoft's ActiveX Data Object (ADO)

model to import the data from an Access database into Excel. This will work only in Excel 2000 or a more recent version. Finally, Chapter 13 uses the Office FileDialog object. This works only in Excel XP (2002) or a more recent version.

The book is also supported by a Web site at www.kelley.iu.edu/albrightbooks. The Web site contains errata and other useful information, including information about my other books.

Acknowledgments

I would like to thank all of my colleagues at Cengage Learning. Foremost among them are my current editor, Aaron Arnsbarger, and my former editors, Curt Hinrichs and Charles McCormick. The original idea was to develop a short VBA manual to accompany our *Practical Management Science* book, but Curt persuaded me to write an entire book. Given the success of the first four editions, I appreciate Curt's insistence. I am also grateful to many of the professionals who worked behind the scenes to make this book a success:

- Brad Sullender, Content Developer; Heather Mooney, Marketing Manager; Kristina Mose-Libon, Art Director; and Sharib Asrar as the Project Manager at Lumina Datamatics.

Next, I would like to thank the reviewers of past editions of the book. Thanks go to

- Gerald Aase, Northern Illinois University; Ravi Ahuja, University of Florida; Grant Costner, University of Oregon; R. Kim Craft, Rollins College; Lynette Molstad Gorder, Dakota State University; and Jim Hightower, California State University-Fullerton; Don Byrkett, Miami University; Kostis Christodoulou, London School of Economics; Charles Franz, University of Missouri; Larry LeBlanc, Vanderbilt University; Jerry May, University of Pittsburgh; Jim Morris, University of Wisconsin; and Tom Schriber, University of Michigan.

Finally, I want to thank my wife, Mary. She continues to support my book-writing activities, even when it requires me to work evenings and weekends in front of a computer. I also want to thank our Welsh corgi Bryn, who faithfully accompanies her daddy when he goes upstairs to do his work. She doesn't add much technical assistance, but she definitely adds a lot of motivational assistance.

S. Christian Albright
(e-mail at albright@indiana.edu,
Web site at www.kelley.iu.edu/albrightbooks)
Bloomington, Indiana
January 2015

Part I

VBA Fundamentals

This part of the book is for readers who need an introduction to programming in general and Visual Basic for Applications (VBA) for Excel in particular. It discusses programming topics that are common to practically all programming languages, including variable types and declarations, control logic, looping, arrays, subroutines, and error handling. It also discusses many topics that are specific to VBA and its use with Excel, including the Excel object model; recording macros; working with ranges, workbooks, worksheets, charts, and other Excel objects; developing user forms (dialog boxes); and automating other applications, including Word, Outlook, Excel's Solver add-in, and Palisade's @RISK add-in, with VBA code.

Many of the chapters in Part I present a business-related exercise immediately after the introductory section. The objective of each such exercise is to motivate you to work through the details of the chapter, knowing that many of these details will be required to solve the exercise. The finished files are included in the online materials, but I urge you to try the exercises on your own, before looking at the solutions.

The chapters in this part should be read in approximately the order they are presented, at least up through Chapter 12. Programming is a skill that builds upon itself. Although it is not always possible to avoid referring to a concept from a later chapter in an earlier chapter, I have attempted to refrain from doing this as much as possible. The one small exception is in Chapters 6 (on ranges) and 7 (on control logic and loops). It is almost impossible to do any interesting programming in Excel without knowing about ranges, and it is almost impossible to do any interesting programming in general without knowing about control logic and loops. I compromised by putting the chapter on ranges first and using some simple control logic and loops in it. I don't believe this should cause any problems.

1 Introduction to VBA Development in Excel

1.1 Introduction

My books *Practical Management Science* (PMS) and *Business Analytics: Data Analysis and Decision Making* (DADM), both co-authored with Wayne Winston, illustrate how to solve a wide variety of business problems by developing appropriate Excel models. If you are familiar with this modeling process, you probably do not need to be convinced of the power and applicability of Excel. You realize that Excel modeling skills will make you a valuable employee in the workplace. This book takes the process one giant step farther. It teaches you how to develop applications in Excel by using Excel's programming language, Visual Basic for Applications (VBA).

In many Excel-modeling books, you learn how to model a particular business problem. You enter given inputs in a worksheet, you relate them with appropriate formulas, and you eventually calculate required outputs. You might also optimize a particular output with Solver, and you might create one or more charts to show outputs graphically. You do all of this through the Excel interface, using its ribbons (as of Excel 2007), menus, and toolbars, entering formulas into its cells, using the chart tools, using the Solver dialog box, and so on. If you are conscientious, you document your work so that other people in your company can understand your completed model. For example, you clearly indicate the input cells so that other users will know which cells they should use for their own inputs and which cells they should leave alone.

Now suppose that your position in a company is to *develop* applications for other less-technical people in the organization to use. Part of your job is still to develop spreadsheet models, but the details of these models might be incomprehensible to many users. These users might realize that they have, say, a product mix problem, where they will have to supply certain inputs, and then some computer magic will eventually determine a mix of products that optimizes company profit. However, the part in between is beyond their capabilities. Your job, therefore, is to develop a user-friendly application with a model (possibly hidden from the user) surrounded by a "front end" and a "back end." The front end will present the user with dialog boxes or some other means for enabling them to define their problem. Here they will be asked to specify input values and possibly other information. Your application will take this information, build the appropriate model, optimize it if necessary, and eventually present the back end to the user—a nontechnical report of the results, possibly with accompanying charts.

This application development is possible with VBA, as I will demonstrate in this book. I make no claim that it is easy or that it can be done quickly, but I do claim that it is within the realm of possibility for people like yourself, not just for professional programmers. It requires a logical mind, a willingness to experiment and take full advantage of online help, plenty of practice, and, above all, perseverance. Even professional programmers seldom accomplish their tasks without difficulty and plenty of errors; this is the nature of programming. However, they learn from their errors (and their colleagues), and they refuse to quit until they get their programs to work properly. Computer programming is essentially a process of overcoming one small hurdle after another. This is where perseverance is so important. But if you are not easily discouraged, and if you love the feeling of accomplishment that comes from getting something to work, you will love the challenge of application development described in the book.

1.2 VBA in Excel 2007 and Later Versions

As you are probably aware, Excel went through a major face lift in 2007. The look of Excel, especially its menus and toolbars, is now much different than in Excel 2003 and earlier. Unfortunately, some users have not converted to Excel 2007 or a later version, so book authors, including myself, are in the uncomfortable position of having to write simultaneously for several audiences. Fortunately, not much about VBA changed in the transition from 2003 to 2007 or from 2007 to 2010 or from 2010 to 2013. I will try to point out the differences as necessary throughout the book, hopefully without interrupting the flow too much.

Perhaps the main difference is in the file extensions you will see. In Excel 2003 and earlier, all Excel files (except for add-ins, not covered here) ended in .xls. It didn't matter whether they contained VBA code or not; they were still .xls files. In Excel 2007 and later versions, there are two new extensions. Files without VBA code now have .xlsx extensions, whereas files with VBA code *must* use .xlsm extensions. If you try to save a file with VBA code as an .xlsx file, you won't be allowed to do so. There is one exception: you can save your new files in the old Excel 2003 format, which is still an option (with Save As), in which case they will have .xls extensions. Why would you do this? The probable reason is that you want to share a file you created in Excel 2007 or a later version with a friend who still uses Excel 2003. Of course, if your file includes features new to Excel 2007 or a later version, your friend won't be able to see them.

I have been using Excel 2007, 2010, and now 2013 since their original releases, and I personally think they are great improvements over earlier versions, at least in most respects. So I will provide my example files in .xlsx and .xlsm formats. If you are using Excel 2003, you will be able to open these if you first install a free Office Compatibility Pack from Microsoft (just search the Web for it). Without this compatibility pack, Excel 2003 users cannot read files in the new .xlsx or .xlsm formats (although users of Excel 2007 and later versions can always read files in the old .xls format).

The fortunate part is that VBA has changed very little. I will usually *not* include new features of Excel 2007 or later versions in my example files that Excel 2003 users (even those with the compatibility pack) could not see. And in the few cases where I need to do so, I will make it clear that these examples are for users of Excel 2007 or later versions only.

1.3 Example Applications

If you have used my PMS or DADM books, you probably understand what a spreadsheet model is. However, you might not understand what I mean by spreadsheet *applications* with front ends and back ends. In other words, you might not understand what this book intends to teach you. The best way to find out is to run some of the applications that will be explained in Part II of the book. At this point, *you* can become the nontechnical user by opening any of the following files that accompany this book: **Product Mix.xlsm, Scheduling.xlsm, Stock Options.xlsm,** and **Transportation.xlsm**. Simply open any of these files and follow instructions. It should be easy. After all, the purpose of writing these applications is to make it easy for a nontechnical user to run them and get results they can understand. Now step back and imagine what must be happening in the background to enable these applications to do what they do. This is what you will be learning in the book. By running a few applications, you will become anxious to learn how to do it yourself. These sample applications illustrate just how powerful a tool VBA for Excel can be.

Security Settings and Trusted Locations

You might encounter annoying messages when you try to open these applications. Microsoft realizes that viruses can be carried in VBA code, so it tries to protect users. First, it sets a macro security level to High by default. This level disallows *any* VBA macros to run. Obviously, this is not good when you are trying to learn VBA programming. The fix is easy.

- For users of Excel 2010 and 2013, open Excel, click the File button, then Options, then the Trust Center tab, then Trust Center Settings, then the Macro Settings tab, and check the "Disable all macros with notification" option.
- For users of Excel 2007, it is the same as for Excel 2010 and 2013 except that you click the Office button, not the File button. (The Office button was replaced by the File button in 2010.)
- For users of Excel 2003 or earlier, open Excel, select the **Tools → Macro → Security** menu item, and select Medium.
- You should need to do this only once. However, even with this macro security setting, you are always asked whether you want to enable macros when you open a file that contains VBA code. Of course, you should typically enable macros. Otherwise, you will be safe from viruses, but none of the VBA code will run!

There is another option, at least in Excel 2007 and later versions, which avoids the security settings altogether. If you find that most of the Excel files with VBA code are in a particular folder on your hard drive, you can add this folder to the list of *trusted locations* on your computer. To do this, a one-time task on a given computer, go to the Trust Center Settings, as explained in the first bullet above, then Trusted Locations, then "Add new location," and browse for the folder you want to add. (In the resulting dialog box, you will probably want to check the "Subfolders of this location are also trusted" option.) From then on, any .xlsm files in this folder (or its subfolders) will open without any warning about enabling macros.

I will make one final comment about enabling macros that pertains to Excel 2007 or later versions only. If you open a file that contains macros, that is, an .xlsm file, and it isn't in a trusted location, you sometimes see the message in Figure 1.1 and you sometimes instead see the button in Figure 1.2 (right above the formula bar). Thanks to John Walkenbach, a fellow VBA author, I finally learned the pattern. If the VB editor (discussed in Chapter 3) is already open when you open the file, you will see the message in Figure 1.1. If it isn't open, you will see the button in Figure 1.2. Why did Microsoft do it this way? I have no idea.

Figure 1.1 Enable Macro Message with VB Editor Open

Figure 1.2 Enable Macro Button with VB Editor Not Open

1.4 Decision Support Systems

In many companies, programmers provide applications called **decision support systems (DSSs).** These are applications, based on Excel or some other package, that help managers make better decisions. They can vary from very simple to very complex, but they usually provide some type of user-friendly interface so that a manager can experiment with various inputs or decision variables to see their effect on important output variables such as profit or cost. Much of what you will be learning, especially in Part II of the book, is how to create Excel-based DSSs. In fact, if you ran the applications in the previous section, you should already understand what decision support means. For example, the Transportation application helps a manager plan the optimal shipping of a product in a logistics network, and the Stock Options application helps a financial analyst price various types of financial options. The value that you, the programmer, provide by developing these applications is that other people in your company can then run them—easily—to make better decisions.

1.5 Required Background

Readers of this book probably vary widely in their programming experience. At one extreme, many of you have probably never programmed in VBA or any other language. At the other extreme, a few of you have probably programmed in Visual Basic but have never used it to automate Excel and build Excel applications. In the middle, some of you have probably had some programming experience in another language such as C or Java but have never learned VBA. This book is intended to appeal to all such audiences. Therefore, a simplified answer to the question, "What programming background do I need?" is "None." You need only a willingness to learn and experiment.

If you ran the applications discussed in Section 1.2, you are probably anxious to get started developing similar applications. If you already know the fundamentals of VBA for Excel, you can jump ahead to Part II of the book. But most of you will have to learn how to walk before you can run. Therefore, the chapters in Part I go through the basics of the VBA language, especially as it applies to Excel. The coverage of this basic material will provide you with enough explanations and examples of VBA's important features to enable you to understand the applications in Part II—and to do some Excel development on your own.

If you want more detailed guidance in VBA for Excel, you can learn from Microsoft's online help or the many user groups on the Web. Indeed, this is perhaps the best way to learn, especially in the middle of a development project. If you need to know one specific detail to overcome a hurdle in the program you are writing, you can look it up quickly in online help or do an online search for it. A good way to do this will be demonstrated shortly.

Part II of the book does presume some modeling ability and general business background. For example, if you ran the Product Mix application, you probably realize that it develops and optimizes a product mix model, a classic

linear programming model. One (but not the only) step in developing this application is to develop a product mix model exactly as in Chapter 3 of PMS. As another example, if you ran the Stock Options application, you realize the need to understand option pricing, explained briefly in the second simulation chapter of PMS. Many of the applications in this book are based on examples (product mix, scheduling, transportation, and so on) from PMS or DADM. You can refer to these books as necessary.

1.6 Visual Basic Versus VBA

Before going any further, I want to clarify one common misconception. Visual Basic (VB) is *not* the same as VBA. VB is a software development language that you can buy and run separately, without the need for Excel or Office. Actually, there are several versions of VB available. The most recent is called VB.NET, which comes with Microsoft's Visual Studio software development suite. (The .NET version of VB has many enhancements to the VB language.) Before VB.NET, there was VB6, still in use in thousands of applications. In contrast, VBA comes with Office. If you own Microsoft Office, you own VBA. The VB language is very similar to VBA, but it is not the same. The main difference is that VBA is the language you need to manipulate Excel, as you will do here.

You can think of it as follows. The VBA language consists of a "backbone" programming language with typical programming elements you find in all programming languages: looping, logical If-Then-Else constructions, arrays, subroutines, variable types, and others. In this respect, VBA and VB are essentially identical. However, the "A" in VBA means that any application software package, such as Excel, Access, Word, or even a non-Microsoft software package, can "expose" its *object model* to VBA, so that VBA can manipulate it programmatically. In short, VBA can be used to develop applications for any of these software packages. This book teaches you how to do so for Excel.

Excel's objects are discussed in depth in later chapters, but a few typical Excel objects you will recognize immediately are ranges, worksheets, workbooks, and charts. VBA for Excel knows about these Excel objects, and it enables you to manipulate them with code. For example, you can change the font of a cell, name a range, add or delete a worksheet, open a workbook, and change the title of a chart. Part of learning VBA for Excel is learning the VB backbone language, the elements that have nothing to do with Excel specifically. But another part, the more challenging part, involves learning how to manipulate Excel's objects in code. That is, it involves learning how to write computer programs to do what you do every day through the familiar Excel interface. If you ever take a course in VB, you will learn the backbone elements of VBA, but you will not learn how to manipulate objects in Excel. This requires VBA, and you *will* learn it here.

By the way, there are also VBA for Access, VBA for Word, VBA for PowerPoint, VBA for Outlook, and others. The difference between them is that each

has its own specific objects. To list just a few, Access has tables, queries, and forms; Word has paragraphs and footnotes; PowerPoint has slides; and Outlook has mail. Each version of VBA shares the same VB backbone language, but each requires you to learn how to manipulate the objects in the specific application. There is certainly a learning curve in moving, say, from VBA for Excel to VBA for Word, but it is not nearly as difficult as if they were totally separate languages. In fact, the power of VBA, as well as the relative ease of programming in it, has prompted many third-party software developers to license VBA from Microsoft so that they can use VBA as the programming language for their applications. One example is Palisade, the developer of the @RISK and PrecisionTree add-ins for Excel, as will be discussed briefly in Chapter 17. In short, once you know VBA, you know a lot about what is happening in the programming world—and you can very possibly use this knowledge to obtain a valuable job in business.

1.7 Some Basic Terminology

Before proceeding, it is useful to clarify some very basic and important terminology that will be used throughout the book. First, whenever you program in any language, your basic building blocks are lines of **code**, short for **programming code**. Any line of code is intended to accomplish something, and it must obey the rules of syntax for the programming language being used. This book is all about coding in VBA.

Typically, a set of logically related lines of code that accomplishes a specific task is called a **subroutine**, a **procedure**, or a **macro**. In fact, one of the first keywords you will learn in VBA is `Sub`. This keyword begins all subroutines. The terms *subroutine*, *procedure*, and *macro* are essentially equivalent, although programmers tend to use the terms *subroutine* and *procedure*, whereas spreadsheet users tend to use the term *macro*. I tend to refer to any of these as a sub.

Finally, the term **program** is typically used to refer to all of the subs in an application. When you explore the more complex applications in Part II of the book, you will see that they often include many subs, where each sub is intended to perform one specific task in the overall program. (Chapter 10 discusses why this division of a program into multiple subs makes a lot of sense.)

1.8 Summary

VBA is the programming language of choice for an increasingly wide range of application developers. The main reason for this is that VBA uses the familiar Visual Basic programming language and then adapts it to many Microsoft and even non-Microsoft application software packages, including Excel. In addition, VBA is a relatively easy programming language to master. This makes it accessible to a large number of nonprofessional programmers in the business world—including you. By learning how to program in VBA, you will definitely enhance your value in the workplace.

2

The Excel Object Model

2.1 Introduction

This chapter introduces the Excel object model—the concept behind it and how it is implemented. Even if you have programmed in another language, this will probably be new material, even a new way of thinking, for you. However, without understanding Excel objects, you will not be able to proceed very far with VBA for Excel. This chapter provides just enough information to get you started. Later chapters focus on many of the most important Excel objects and how they can be manipulated with VBA code.

2.2 Objects, Properties, Methods, and Events

Consider the many things you see in the everyday world. To name a few, there are cars, houses, computers, people, and so on. These are all examples of **objects.** For example, let's focus on a car. A car has attributes, and there are things you can do to (or with) a car. Some of its attributes are its weight, its horsepower, its color, and its number of doors. Some of the things you can do to (or with) a car are drive it, park it, accelerate it, crash it, and sell it. In VBA, the attributes of an object are called **properties:** the size property, the horsepower property, the color property, the number of doors property, and so on. In addition, each property has a **value** for any *particular* car. For example, a particular car might be white and it might have four doors. In contrast, the things you can do to (or with) an object are called **methods:** the drive method, the park method, the accelerate method, the crash method, the sell method, and so on. Methods can also have qualifiers, called **arguments,** that indicate *how* a method is performed. For example, an argument of the crash method might be speed—how fast the car was going when it crashed.

The following analogy to parts of speech is useful. Objects correspond to *nouns,* properties correspond to *adjectives,* methods correspond to *verbs,* and arguments of methods correspond to *adverbs.* You might want to keep this analogy in mind as the discussion proceeds.

Now let's move from cars to Excel. Imagine all of the things—objects—you work with in Excel. Some of the most common are ranges, worksheets, charts, and workbooks. (A workbook is really just an Excel file.) Each of these is an object in the Excel object model. For example, consider the single-cell range B5.

This cell is a Range object.[1] Like a car, it has properties. It has a Value property: the value (either text or number) displayed in the cell. It has a HorizontalAlignment property: left-, center-, or right-aligned. It has a Formula property: the formula (if any) in the cell. These are just a few of the many properties of a range.

A Range object also has methods. For example, you can copy a range, so Copy is a method of a Range object. You can probably guess the argument of the Copy method: the Destination argument (the paste range). Another range method is the ClearContents method, which is equivalent to selecting the range and pressing the Delete key. It deletes the contents of the range, but it does not change the formatting. If you want to clear the formatting as well, there is also a Clear method. Neither the ClearContents method nor the Clear method has any arguments.

Learning the various objects in Excel, along with their properties and methods, is a lot like learning vocabulary in English—especially if English is not your native language. You learn a little at a time and generally broaden your vocabulary through practice and experience. Some objects, properties, and methods are naturally used most often, and you will learn quickly. Others you will never need, and you will probably remain unaware that they even exist. However, there are many times when you *will* need to use a particular object or one of its properties or methods that you have not yet learned. Fortunately, there is excellent online help available—a type of dictionary—for learning about objects, properties, and methods. It is called the **Object Browser** and is discussed in the next chapter.

There is one other important feature of objects: **events.** Some Excel objects have events that they can respond to. A good example is the Workbook object and its Open event. This event happens—we say it **fires**—when the workbook is opened in Excel. In fact, you might not realize it, but the Windows world is full of events that fire constantly. Every time you click or double-click a button, press a key, move your mouse over some region, or perform a number of other actions, various events fire. Programmers have the option of responding to events by writing **event handlers.** An event handler is a section of code that runs whenever the associated event fires. In later chapters, particularly Chapter 11, you will learn how to write your own event handlers. For example, it is often useful to write an event handler for the Open event of a Workbook object. Whenever the workbook is opened in Excel, the event handler then runs automatically. It could be used, for example, to ensure that the user sees a certain worksheet when the workbook opens.

2.3 Collections as Objects

Continuing the car analogy, imagine that you enter a used car lot. Each car in the lot is a particular car object, but it also makes sense to consider the *collection* of all

[1] From here on, "proper" case, such as Range or HorizontalAlignment, will be used for objects, properties, and methods. This is the convention used in VBA. Also, they appear in this book in a different font.

cars in the lot as an object. This is called a Collection object. Clearly, the collection of cars is not conceptually the same as an individual car. Rather, it is an object that includes all of the individual car objects.

Collection objects also have properties and methods, but they are not the same as the properties and methods of the objects they contain. Generally, there are many *fewer* properties and methods for collections. The two most common are the Count property and the Add method. The Count property indicates the number of objects in the collection (the number of cars in the lot). The Add method adds a new object to a collection (a new car joins the lot).

It is easy to spot collections and the objects they contain in the Excel object model. Collection objects are plural, whereas a typical object contained in a collection is singular. A good example involves worksheets in a given workbook. The Worksheets collection (note the plural) is the collection of all worksheets in the workbook. Any one of these worksheets is a Worksheet object (note the singular). Again, these must be treated differently. You can count worksheets in the Worksheets collection, or you can add another worksheet to the collection. In contrast, typical properties of a Worksheet object are its Name (the name on the sheet tab) and Visible (True or False) properties, and a typical method of a Worksheet object is the Delete method. (Note that this Delete method reduces the Count of the Worksheets collection by one.)

The main exception to this plural/singular characterization is the Range object. There is no "Ranges" collection object. A Range object cannot really be considered singular *or* plural; it is essentially some of each. A Range object can be a single cell, a rectangular range, a union of several rectangular ranges, an entire column, or an entire row. Range objects are probably the most difficult to master in all of their varied forms. This is unfortunate because they are the most frequently used objects in Excel. Think of your own experience in Excel, and you will realize that you are almost always doing something with ranges. An entire chapter, Chapter 6, is devoted to Range objects so that you can master some of the techniques for manipulating these important objects.

2.4 The Hierarchy of Objects

Returning one last time to cars, what is the status of a car's hood, a car's trunk, or a car's set of wheels? These are also objects, with their own properties and methods. In fact, the set of wheels is a collection object that contains individual wheel objects. The point, however, is that there is a natural hierarchy, as illustrated in Figure 2.1. The Cars collection is at the top of the hierarchy. It contains a set of individual cars. The notation Cars (Car) indicates that the collection object is called Cars and that each member of this collection is a Car object. Each car "contains" a number of objects: a Wheels collection of individual Wheel objects, a Trunk object, a Hood object, and others not shown. Each of these can have its own properties and methods. Also, some can contain objects farther down the hierarchy. For example, the figure indicates that an object down the hierarchy from Hood is the HoodOrnament object. Note that each of the rectangles in this

Figure 2.1 Object Model for Cars

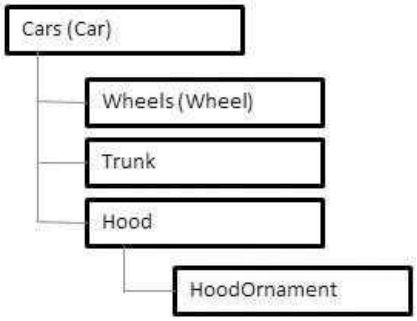


figure represents an *object*. Each object has properties and methods that could be shown emanating from its rectangle, but this would greatly complicate the figure.

The same situation occurs in Excel. The full diagram of the Excel object model appears in Figure 2.2. (This is the Excel 2003 version; versions for Excel 2007 or later are only slightly different.[2]) This figure shows how all objects, including collection objects, are arranged in a hierarchy. At the top of the hierarchy is the Application object. This refers to Excel itself. One object (of several) one step down from Application is the Workbooks collection, the collection of all open Workbook objects. This diagram is admittedly quite complex. All you need to realize at this point is that Excel has a very rich object model—a lot of objects; fortunately, you will need only a relatively small subset of this object model for most of your applications. This relatively small subset is the topic of later chapters.

2.5 Object Models in General

Although the Excel object model is used in this book, you should now have some understanding of what it would take to use VBA for other applications such as Word, Access, or even non-Microsoft products. In short, you would need to learn *its* object model. You can think of each application "plugging in" its object model to the underlying VB language. Indeed, third-party software developers who want to license VBA from Microsoft need to *create* an object model appropriate for their application. Programmers can then use VBA to manipulate the objects in this model. This is a powerful idea, and it is the reason why VBA is the programming language of choice for so many developers—regardless of whether they are working in Excel or any other application.

Figures 2.3 and 2.4 illustrate two other object models. (Again, these are the Office 2003 versions.) The object model in Figure 2.3 is for Word. A few of these objects are probably familiar, such as Sentence, Paragraph, and Footnote. If you

[2] For example, if you perform a Web search for "Excel 2013 object model diagram," you will see a number of such diagrams.

Figure 2.2 Excel Object Model

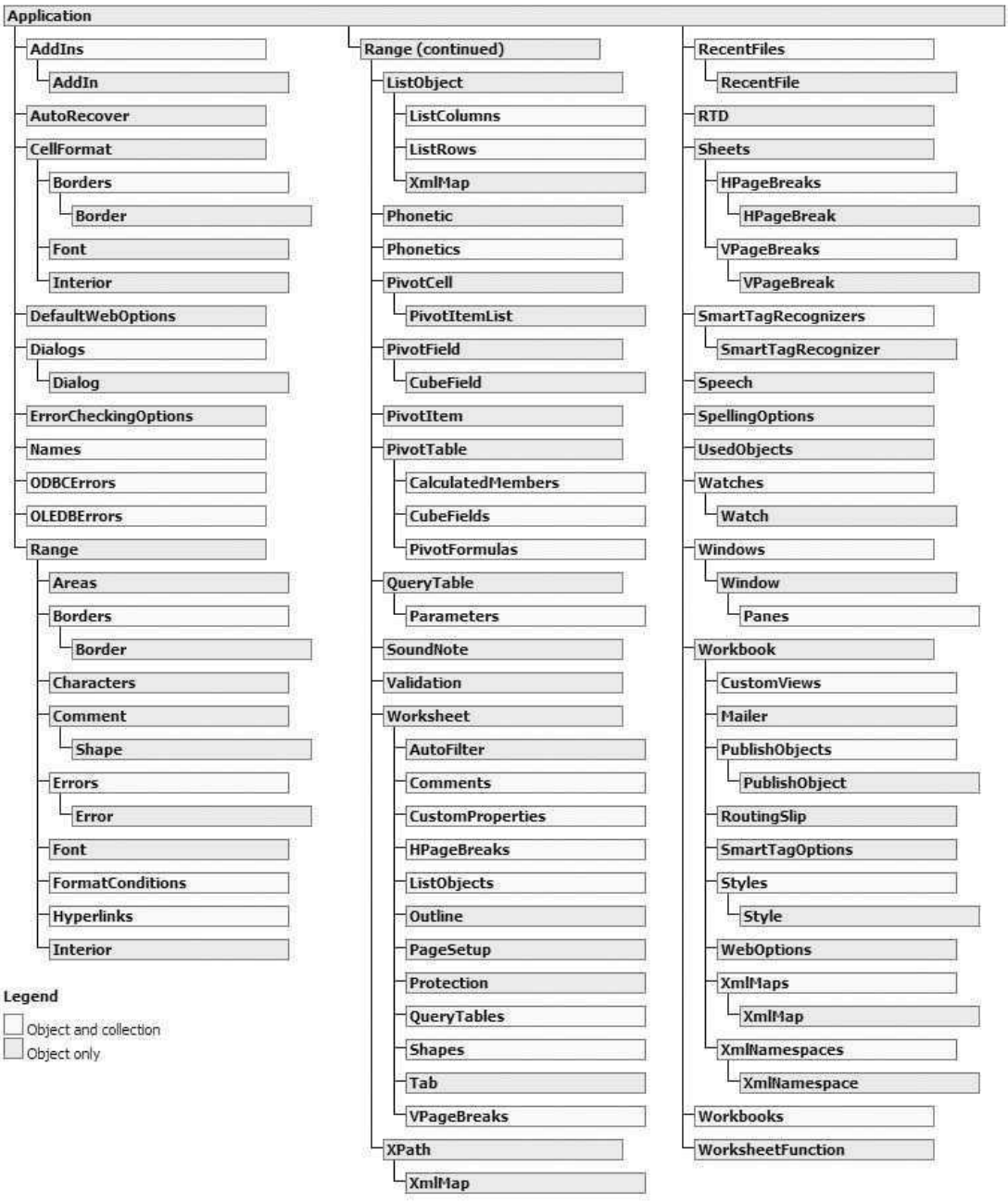
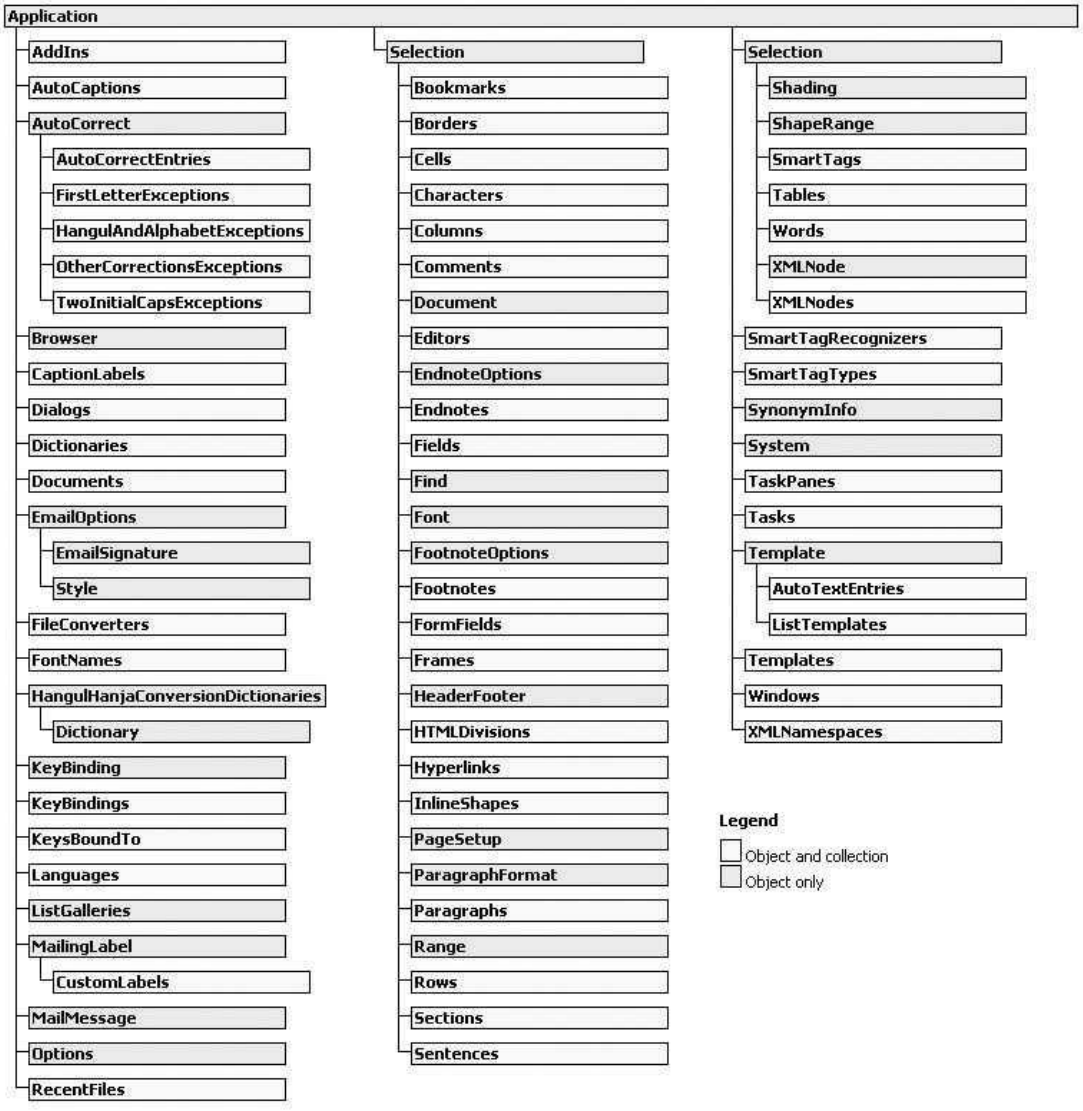
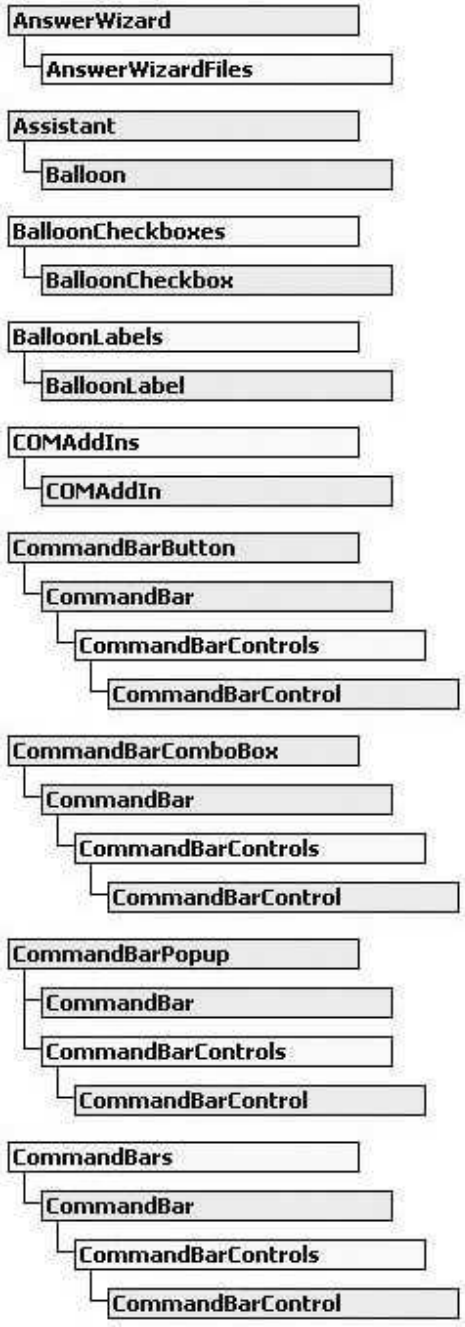


Figure 2.3 Word Object Model



were learning VBA for Word, you would need to learn the most common elements of this object model. Figure 2.4 shows part (about 40%) of the object model for Microsoft Office as a whole. You might wonder why Office has a separate object model from Excel or Word. The reason is that Office is an integrated suite, where all of its programs—Excel, Word, PowerPoint, Outlook, and the rest—share a number of features. For example, they all have menus and toolbars, referred to collectively as the CommandBars collection in the object model. Therefore, if you

Figure 2.4 Part of Office Object Model



want to use VBA to manipulate toolbars or menus in Excel, as many programmers do, you have to learn part of the Office object model. But then this same knowledge would enable you to manipulate menus and toolbars in Word, PowerPoint, and the others. (Actually, menus and toolbars were replaced for the most part by ribbons in Excel 2007 and later versions, but the CommandBar object is still present. This topic is discussed in Chapter 16.)

The Excel object model continues to evolve as new versions of Excel are released. Sometimes new objects, properties, or methods are added. Other times, some are dropped from the official object model but still continue to work, for backward compatibility. Occasionally, some are dropped completely, so that programs written in an earlier version no longer work. Fortunately, these are the rare exceptions. If you are working in Excel 2007 or later versions and are interested in seeing the types of changes that have been made, open the Visual Basic Editor (Alt+F11 from Excel), press the F1 key for help, and search for "object model changes." Although the list is fairly long, not much in terms of VBA code has changed since this book was originally written for Excel 2003.

2.6 Summary

This chapter has introduced the concept of an object model, and it has briefly introduced the Excel object model that is the focus of the rest of the book. If you have never programmed in an object-oriented environment, you can look forward to a whole new experience. However, the more you do it, the more natural it becomes. It is certainly the dominant theme in today's programming world, so if you want to be part of this world, you have to start thinking in terms of objects. You will get plenty of chances to do so throughout the book.

3 The Visual Basic Editor

3.1 Introduction

At this point, you might be asking where VBA lives. I claimed in Chapter 1 that if you own Excel, you also own VBA, but many of you have probably never seen it. You do your VBA work in the **Visual Basic Editor** (**VBE**), which you can access easily from Excel by pressing the **Alt+F11** key combination. The VBE provides a very user-friendly environment for writing your VBA programs. This chapter walks you through the VBE and shows you its most important features. It also helps you write your first VBA program. By the way, you might also hear the term **Integrated Development Environment** (**IDE**). This is a general term for an environment where you do your programming, regardless of the programming language. The VBE is the IDE for programming with VBA in Excel.

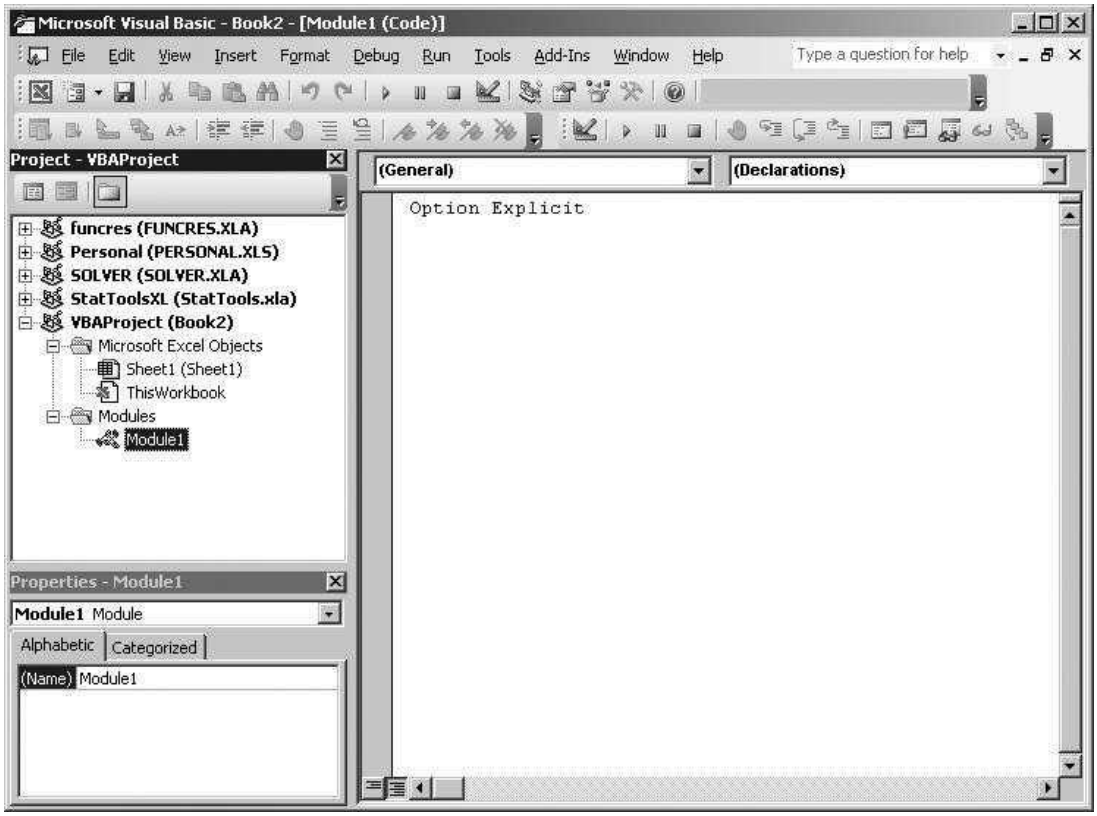
3.2 Important Features of the VBE

To understand this section most easily, you should follow along at your computer. Open Excel and press **Alt+F11** to get into the VBE.[1] It should look something like Figure 3.1, although the configuration you see might be somewhat different. By the time this discussion is completed, you will be able to make your screen look like that in Figure 3.1 or change it according to your own preferences. This is your programming workspace, and you have a lot of control over how it appears. This chapter provides some guidance, but the best way to learn is by experimenting.

The large blank pane on the right is the **Code** window. This is where you write your code. (If any of the windows discussed here are *not* visible on your screen, you can select the View menu from the VBE and then select the window you want to make visible.) The rest of the VBE consists of the top menu, one or more toolbars, and one or more optional windows. Let's start with the windows.

[1] In Excel 2003 and earlier, the **Tools → Macro → Visual Basic Editor** menu item also gets you into the VBE, but Alt+F11 is quicker. In Excel 2007 and later versions, you should first make the **Developer** ribbon visible. To do this in Excel 2007, click the Office button and then Excel Options. Under the Popular tab, select the third option at the top: Show Developer tab in the Ribbon. In Excel 2010 and later versions, right-click any ribbon and select Customize the Ribbon. Then check the Developer item in the right pane. You need to do this only once. The Developer tab is a must for programmers. Among other things, you can get to the VBE by clicking on its Visual Basic button, but again, Alt+F11 is quicker.

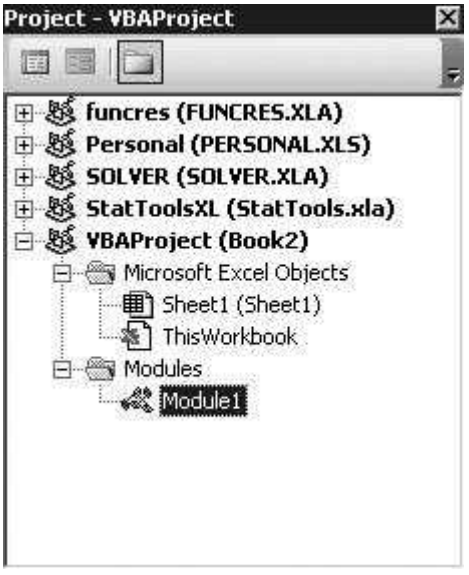
Figure 3.1 Visual Basic Editor (VBE)



The **Project Explorer** window, repeated in Figure 3.2, shows an Explorer-type list of all open projects. (Your list will probably be different from the one shown here. It depends on the files you have open and the add-ins that are loaded.) For example, the active project shown here has the generic name VBAProject and corresponds to the workbook Book2—that is, the file **Book2.xlsx**.[2] Below a given project, the Project Explorer window shows its "elements." In the Microsoft Excel Objects folder, these elements include any worksheets or chart sheets in the Excel file and an element called ThisWorkbook, which refers to the workbook itself. There can also be folders for modules (for VBA code), user forms (for dialog boxes), and references (for links to other libraries of code you need), depending on whether you have any of these in your project. Modules, user forms, and references are discussed in detail in later chapters.

[2] For our purposes, there is no difference between a project and a workbook. However, VBA allows them to have separate names: VBAProject and Book2, for example. If you save Book2 as **Practice.xlsm**, say, the project name will still be VBAProject. Admittedly, it is somewhat confusing, but just think of projects as Excel files.

Figure 3.2 Project Explorer Window



The **Properties** window, shown in Figure 3.3, lists a set of properties. This list depends on what is currently selected. For example, the property list in Figure 3.3 is relevant for the project itself. It indicates a single property only—the project's name. Therefore, if you want to change the name of the project from the generic VBAProject to something more meaningful, such as MyFirstProgram, this is the place to do it. Chapter 11 discusses the use of the Properties window in much more detail. At this point, you don't really need the Properties window, so you can close it by clicking on its close button (the upper right X).

The VBE also has at least three toolbars that are very useful: **Standard, Edit,** and **Debug.** They appear in Figures 3.4, 3.5, and 3.6, where some of the most useful buttons are indicated. (If any of these toolbars are not visible on your

Figure 3.3 Properties Window

Figure 3.4 Standard Toolbar

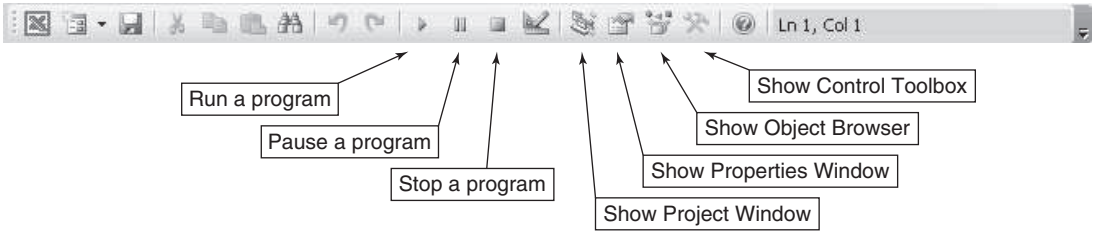


Figure 3.5 Edit Toolbar

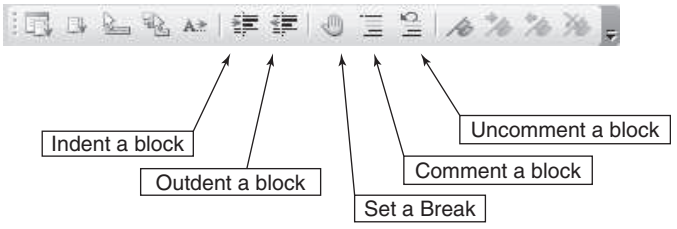
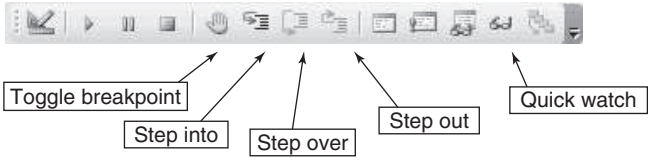


Figure 3.6 Debug Toolbar



computer, you can make them visible through the View menu.) From the Standard toolbar, you can run, pause, or stop a program you have written. You can also display the Project or Properties window (if it is hidden), and you can display the Object Browser or the Control Toolbox (more about these later). From the Edit toolbar, you can perform useful editing tasks, such as indenting or outdenting (the opposite of indenting), and you can comment or uncomment blocks of code, as is discussed later. Finally, although the Debug toolbar will probably not mean much at this point, it is invaluable when you need to debug your programs—as you will undoubtedly need to do. It is discussed in more detail in Chapter 5. For future reference, here are a few menu items of particular importance.

- You usually need at least one module in a project. This is where you will typically store your code. To insert a module, use the **Insert → Module** menu item. If you ever have a module you do not need, highlight the module in the Project Explorer window and use the **File → Remove Module** menu item. (Answer No to whether you want to export the module.)

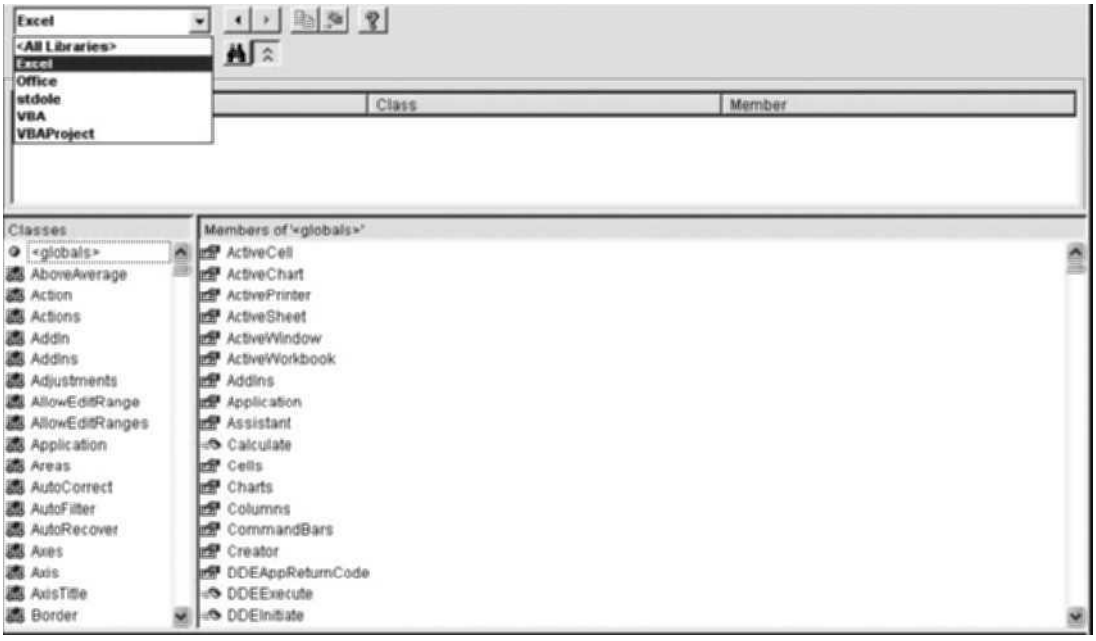
- Chapter 11 explains how to build your own dialog boxes. VBA calls these **user forms.** To insert a new user form into a project, use the **Insert → User Form** menu item. You can delete an unwanted user form in the same way you delete a module.
- Under the Insert menu, there is also a **Class Module** item. You can usually ignore this. It is more advanced, but it is discussed briefly in Chapter 18.
- The **Tools → Options** menu item allows you to change the look and feel of the VBE in a variety of ways. You should probably leave the default settings alone—with one important exception. Try it now. Select **Tools → Options,** and make sure the **Require Variable Declarations** box under the Editor tab *is* checked. The effect of this is explained in Chapter 5. You might also want to uncheck the **Auto Syntax Check** box, as I always do. If it is checked, the editor beeps at you each time you make a syntax error in a line of code and then press Enter. This can be annoying. Even if this box is unchecked, the editor will still warn you about a syntax error by coloring the offending line red.
- If you ever want to password-protect your project so that other people cannot see your code, use the **Tools → VBA Properties** menu item and click the **Protection** tab. This gives you a chance to enter a password. (Just don't forget it, or you will not be able to see your *own* code.)
- If you click the familiar **Save** button (or use the **File → Save** menu item), this saves the project currently highlighted in the Project Explorer window. It saves your code *and* anything in the underlying Excel workbook. (It is all saved in the .xlsm file.) You can achieve the same objective by switching back to Excel and saving in the usual way from there. (Note, however, that in Excel 2007 and later versions, if your file started as an .xlsx file without any VBA code, you will have to save it as an .xlsm file once it contains code.)

3.3 The Object Browser

VBA's **Object Browser** is a wonderful online help tool. To get to it, open the VBE and click the Object Browser button on the Standard toolbar (see Figure 3.4). If you prefer keyboard shortcuts, you can press the F2 key. Either way, this opens the window shown in Figure 3.7. At the top left, there is a dropdown list of *libraries* that you can get help on. Our main interest is in the Excel library, the VBA library, and, to a lesser extent, the Office library. The **Excel** library provides help on all of the objects and their properties and methods in the Excel object model. The **VBA** library provides help on the VBA elements that are common to *all* applications that can use VBA: Excel, Access, Word, and others. The **Office** library provides help on objects common to all Office programs, such as CommandBars objects (menus and toolbars).

For now, select the Excel library. In the bottom left pane, you see a list of all objects in the Excel object model, and in the right pane, you see a list of all properties and methods for any object selected in the left pane. A property is designated by a hand icon, and a method is designated by a green rectangular icon. A few objects, such as the Workbook object, also have events they can respond to. An event is designated by a lightning icon.

Figure 3.7 Object Browser

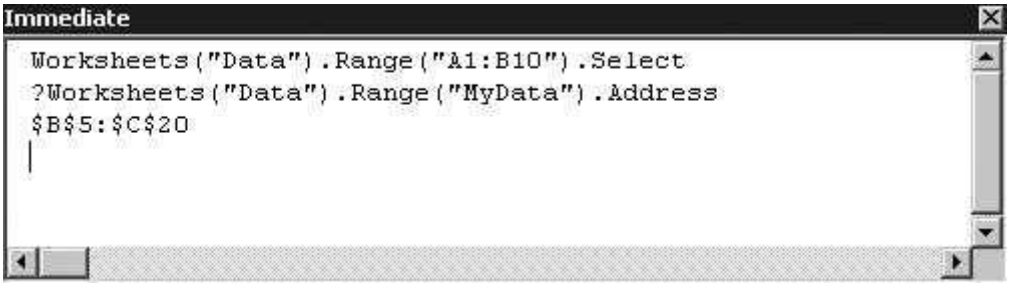


To get help on any of these items, select it and then click the question mark button at the top. It is too early in our VBA discussion to be asking for online help, but you should not forget about the Object Browser. It can be invaluable as you develop your projects. I use it constantly, and you should too. Of course, you can get similar help by performing online searches for specific items, but the Object Browser stores everything in one place.

3.4 The Immediate and Watch Windows

There are two other windows in the VBE that you should be aware of: the **Immediate** and **Watch** windows. Each can be opened through the View menu or the Debug toolbar. (The Immediate window can also be opened quickly with the **Ctrl+g** key combination.) The Immediate window, shown in Figure 3.8, is useful for issuing one-line VBA commands. If you type a command and press Enter, the command takes effect immediately. For example, the first line in Figure 3.8 selects the range A1:B10 of the Data worksheet (assuming there is a Data worksheet in the active workbook). If you type this, press Enter, and switch back to Excel, you will see that the range A1:B10 has been selected. If you precede the command by a question mark, you can get an immediate answer to a question. For example, if you type the second line in the figure, which asks for the address of the range named MyData, and then press Enter, you immediately get the answer on the third line.

Figure 3.8 Immediate Window



Many programmers send information to the Immediate window through their code. If you see the command Debug.Print, followed by something to be printed, the programmer is asking for this to be printed to the Immediate window. This is not a permanent copy of the printed information. It is usually a quick check to see whether a program is working properly.

The Watch window is used for debugging. Programs typically include several variables that change value as the program runs. If the program does not appear to be working as it should, you can put a watch on one or more key variables to see how they change as the program progresses. Debugging in this way is discussed in some detail in Chapter 5.

3.5 A First Program

Although you do not yet know much about VBA programming, you know enough to write a simple program and run it. Besides, sooner or later you will have to stop reading and do some programming on your own. Now is a good time to get started. Although the example in this section is very simple, there are a few details you probably won't understand completely, at least not yet. Don't worry. Later chapters will clarify the details. For now, just follow the directions and realize the thrill of getting a program to work.

This example is based on a simple data set in the file **First Program.xlsx.** It shows sales of a company by region and by month for a 3-year period. (See Figure 3.9, where some rows have been hidden. The range B2:G37 has the range name SalesRange.) Your boss wants you to write a program that scans the sales of each region and, for each, displays a message that indicates the number of months that sales in that region are above a user-selected value such as $150,000. To do this, go through the following steps. (In case you get stuck, the finished version is stored in the file **First Program Finished.xlsm.)**

1. **Open the file.** Get into Excel and open the **First Program.xlsx** file. Because it is going to contain VBA code, save it as **First Program.xlsm.**
2. **Get into the VBE.** Press Alt+F11 to open the VBE. Make sure the Project Explorer Window is visible. If it isn't, open it with the **View → Project Explorer** menu item.

Figure 3.9 Sales by Region and Month

	A	B	C	D	E	F	G
1	***Month***	***Region 1***	***Region 2***	***Region 3***	***Region 4***	***Region 5***	***Region 6***
2	Jan-08	144770	111200	163140	118110	105010	167350
3	Feb-08	155180	155100	129850	133940	140880	104110
4	Mar-08	86230	162310	142950	131490	150160	158720
5	Apr-08	148800	165160	123840	141050	175870	108100
6	May-08	157140	130300	114990	128220	147790	167470
7	Jun-08	126150	163240	149360	152240	167320	181070
31	Jun-10	124320	148410	162310	186440	147200	146200
32	Jul-10	135100	131520	151780	153920	121200	141430
33	Aug-10	150790	151970	168800	144170	140360	139990
34	Sep-10	93740	168100	142040	126440	113500	130500
35	Oct-10	124160	148560	120190	155600	132590	155510
36	Nov-10	109840	189790	127460	135160	149470	163330
37	Dec-10	127100	108640	145300	127920	151130	122900

3. **Add a module.** In the Project Explorer window, make sure the **First Program.xlsm** project is highlighted (select it if necessary), and use the **Insert → Module** menu item to add a module to this project. This module is automatically named Module1, and it will hold your VBA code.
4. **Start a sub.** Click anywhere in the Code window, type Sub CountHighSales, and press Enter. You should immediately see the following code. You have started a program called CountHighSales. (Any other descriptive name could be used instead of CountHighSales, but it shouldn't contain any spaces.) The keyword Sub informs VBA that you want to write a **subroutine** (also called a **procedure** or a **macro),** so it adds empty parentheses next to the name CountHighSales and adds the keywords End Sub at the bottom—two necessary elements of any subroutine. The rest of your code will be placed between the Sub and End Sub lines. Chapters 5 and 10 discuss subroutines in more detail, but for now, just think of a subroutine as a section of code that performs a particular task. For this simple example, there is only one subroutine.

```
Sub CountHighSales( )
End Sub
```

5. **Type the code.** Type the code exactly as shown below between the Sub and End Sub lines. It is important to indent properly for readability. To indent as shown, press the Tab key. Also, note that there is no word wrap in the VBE. To finish a line and go to the next line, you need to press the Enter key. Other than this, the Code window is much like a word processor. You will note that keywords such as Sub and End Sub are automatically colored blue by the VBE. This is a great feature for helping you program. Also, spaces are often inserted for you to make your code more readable. For example, if you

type nHigh=nHigh+1, the editor will automatically insert spaces on either side of the equals and plus signs.

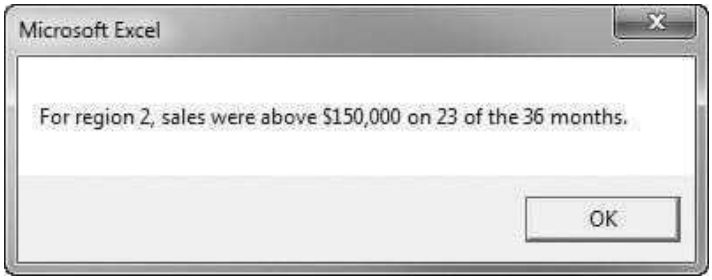
```
Sub CountHighSales( )
    Dim i As Integer
    Dim j As Integer
    Dim nHigh As Integer
    Dim cutoff As Currency

    cutoff = InputBox("What sales value do you want to check for?")
    For j = 1 To 6
        nHigh = 0
        For i = 1 To 36
            If wsData.Range("Sales").Cells(i, j) >= cutoff Then _
                nHigh = nHigh + 1
        Next i
        MsgBox "For region " & j & ", sales were above " & Format(cutoff, "$0,000") _
            & " on " & nHigh & " of the 36 months."
    Next j
End Sub
```

6. **Avoid syntax errors.** Two special characters in this code are the ampersand, &, and the underscore, _. Make sure each ampersand has a space on either side of it, and make sure each line-ending underscore has a space before it. (These spaces are *not* added automatically for you.) There are other syntax errors you could make, but these are the most likely in this short subroutine. Be sure to check your spelling carefully and fix any errors before proceeding.
7. **Run the program from the VBE.** Your program is now finished. The next step is to run it. There are several ways to do so, two of which are demonstrated here. For the first method, make sure the cursor is anywhere within your subroutine and select the **Run → Run Sub/UserForm** menu item. (Alternatively, click the "green triangle" button on the Standard toolbar, or press the F5 key.) If all goes well, you should see the input box in Figure 3.10, where you can enter a value such as 150000. The program will then search for all values greater than or equal to $150,000 in the data set. Next, you will see a series of message boxes such as the one in Figure 3.11. Each message box tells you how many months the sales in some region are above the sales cutoff value you entered. This is exactly what you wanted the program to do.
8. **Run the program with a button.** The run method in the previous step is fine for you, the programmer, but your users won't want to get into the VBE to run the program. They probably don't even want to *see* the VBE. They will instead want to run the program directly from the Excel worksheet that contains the data. You can make this easy for them. First, switch back to Excel (click the Excel button on the taskbar of your screen). Then click the Insert dropdown list on the Developer ribbon (see footnote 1 of this chapter for how to make the Developer tab visible), click the upper left "button" control, and drag a rectangular button somewhere on your worksheet, as

Figure 3.10 InputBox for Sales Cutoff Value

Figure 3.11 MessageBox for Region 2



shown in Figure 3.12.[3] You will immediately be asked to assign a macro to this button. This is because the only purpose of a button is to run a macro. You should assign the CountHighSales macro you just wrote. Then you can type a more meaningful caption on the button itself. (Again, see Figure 3.12 for a possible caption.) At this point, the button is "selected"—there is a dotted border around it. To deselect it, just click anywhere else on the worksheet. Now your button is ready to go. To run your program, just click the button.

9. **Save the file.** In case you haven't done so already, save the file under the original (or a new) name. This will save your code and the button you created. Again, make sure you save it with the .xlsm extension.

A note on saving. You have undoubtedly been told to save frequently in all of your computer-related courses. Frequent saving is at least as important in a programming environment. After all the effort you expend to get a program working correctly, you don't want that sinking feeling when your unsaved work is wiped out by a sudden power outage or some other problem. So I will say it, too—save, save, save!

[3] In Excel 2003 and earlier, the button control is on the Forms toolbar, which you can make visible by right-clicking any toolbar and checking the Forms option. Although buttons are ready-made for running macros, Excel shapes can also be used. Give it a try. From the Insert menu, select and then drag a shape such as a rectangle. Then right-click, and you will see an Assign Macro menu item.

Figure 3.12 Button on the Worksheet

	A	B	C	D	E	F	G
1	***Month***	***Region 1***	***Region 2***	***Region 3***	***Region 4***	***Region 5***	***Region 6***
2	Jan-08	144770	111200	163140	118110	105010	167350
3	Feb-08	155180	155100	129850	133940	140880	104110
4	Mar-08	86230	162310	142950	131490	150160	158720
5	Apr-08	148800	165160	123840	141050	175870	108100
6	May-08	157140	130300	114990	128220	147790	167470
7	Jun-08	126150	163240	149360	152240	167320	181070
8	Jul-08	174010	183360	122120	149730	134220	135530
9	Aug-08	171780	130050	124130	134510	175590	122230
35	Oct-10	124160	148560	120190	155600	132590	155510
36	Nov-10	109840	189790	127460	135160	149470	163330
37	Dec-10	127100	108640	145300	127920	151130	122900

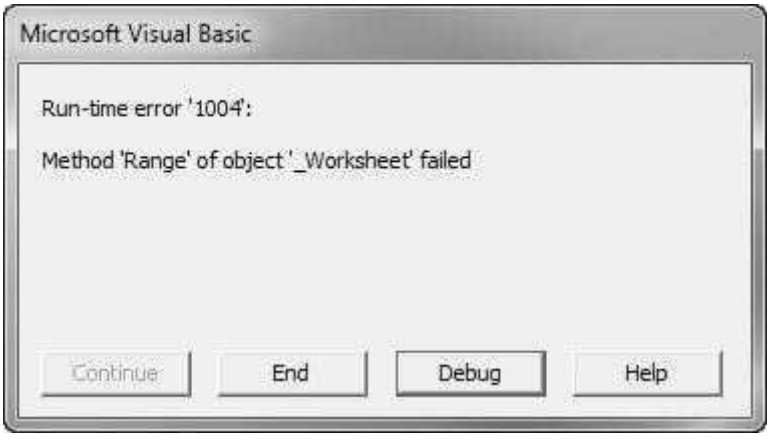
Count High Sales Values

Troubleshooting

What if you get an error message when you run your program? First, read your program carefully and make sure the code is exactly like the code shown here. Again, the underscores at the ends of the `If` and `MsgBox` lines must be preceded by a space. (Their purpose is to extend long lines of code to the next line.) Also, the ampersand (&) characters in the `MsgBox` line should have a space on either side of them. If you have any lines colored red, this is a sure sign you have typed something incorrectly. (This is another feature of the VBE that helps programmers. Red lines signify syntax errors.) In any case, if you get some version of the dialog box in Figure 3.13, click the End button. This stops a program with bugs and lets you fix any errors. Alternatively, click the Debug button, and you will see a line of code in yellow. This line is typically the offending line, or close to it. (Again, debugging is discussed in some detail in Chapter 5.)

If your typing is correct and you still get an error, check steps 7 and 8. If you are using step 7 to run the program, make sure your cursor is somewhere *inside* the subroutine. If you are using the button method in step 8, make sure you have assigned the `CountHighSales` macro to the button. (Right-click the button

Figure 3.13 A Typical Error Dialog Box



and select the Assign Macro menu item.) There are not too many things that can go wrong with this small program, so you should eventually get it to work. Remember, perseverance is the key.

Brief Analysis of the Program

I could not expect you to write this program without my help at this point. But you can probably understand the gist of it. The four lines after the Sub line declare variables that are used later on. The next line displays an InputBox (see Figure 3.12) that asks for a user's input. The section starting with For j = 1 To 6 and ending with Next j is a loop that performs a similar task for each sales region. As you will learn in Chapter 7, loops are among the most powerful tools in a programmer's arsenal. For example, if there were 600 regions rather than 6, the only required change would be to change 6 to 600 in the For j = 1 To 6 line. Computers are excellent at performing repetitive tasks.

Within the loop on regions, there is another loop on months, starting with For i = 1 To 36 and ending with Next i. Within this loop there is an If statement that checks whether the sales value for the region in that month is at least as large as the cutoff value. If it is, the variable nHigh is increased by 1. Once this inner loop has been completed, the results for the region are reported in a MessageBox.

Again, the details might be unclear at this point, but you can probably understand the overall logic of the program. And if you typed everything correctly and ran the program as instructed, you now know the thrill of getting a program to work as planned. I hope you experience this feeling frequently as you work through this book.

3.6 Intellisense

A lot of things are advertised to be the best thing since sliced bread. Well, one of the features of the VBE really is. It is called **Intellisense**. As you were writing the program in the previous section, you undoubtedly noticed how the editor gave you hints and tried to complete certain words for you. You see Intellisense in the following situations:

- Every time you type the first line of a sub and then press Enter, Intellisense adds the End Sub line automatically for you.[4]
- Whenever you start declaring a variable in a Dim statement, Intellisense helps you with the variable type. For example, if you type Dim nHigh As In, it will

[4] There are many other VBA constructs that are bracketed with a beginning line and an ending line: If and End If, For and Next, Do and Loop, and others. You might imagine that if VBA is smart enough to add End Sub for you after you type the Sub line, it is smart enough to add an End If line after an If line, a Next line after a For line, and so on. However, it isn't that smart, at least not yet. My guess is that Microsoft simply hasn't gotten around to it yet. Interestingly, the Visual Studio editor for .NET *is* that smart. It even indents automatically for you.

guess that you want In to be Integer. All you have to do at this point is press the Tab key, and Integer will appear.

- Intellisense helps you with properties and methods of objects. For example, if you type Range("A1:C10"). (including the period), you will see all of the properties and methods of a Range object. At this point you can scroll through the list and choose the one you want.
- Intellisense helps you with arguments of methods. For example, if you type Range("A1:C10").Copy and then a space, you will see all of the arguments (actually, only one) of the Copy method. (Any arguments shown in square brackets in this list are optional. All others are required.)
- Intellisense helps you with hard-to-remember constants. For example, if you type Range("A1").End(, you will see that there are four constants to choose from: xlDown, xlUp, xlToRight, and xlToLeft. (This corresponds to pressing the End key and then one of the arrow keys in Excel. You will learn more about it in Chapter 6.)
- Sometimes you create fairly long variable names like productCost or firstCustomer. Then you need to use them repeatedly in your code. If you start typing one of them, like firs, and then press **Ctrl+Space**, you will get a list of all variables that start with these letters, and you can choose the one you want. In fact, if there is only one variable that starts with these letters, it will be inserted automatically. This can save a lot of typing—and typing errors.

In short, Intellisense is instant online help. It doesn't necessarily help you with the *logic* of your program, but it speeds up your typing, and it helps ensure that you get the syntax and spelling correct. After you get used to Intellisense, you will find that it is absolutely indispensable.

3.7 Color Coding and Case

Another feature of the VBE that enhances readability and helps you get your code correct is color coding.

- All keywords, such as Sub, End, For, and many others, are automatically colored blue.
- All comments (discussed in Chapter 5) are colored green.
- All of the rest of your code is colored black.
- If you make a syntax error in a line of code and then press Enter, the offending line is colored red. This is a warning that you should fix the line before proceeding.

Besides coloring, the editor corrects case for you.

- All keywords start with a capital letter. Therefore, if you type sub and press Enter, the editor changes it to Sub.
- If you declare a variable with the spelling unitCost and then type it as UNITCost later on in the program, the editor automatically changes it to unitCost.

(Whatever spelling you use in the Dim statement is the one used subsequently, even if it is something weird like uNitCost.) Actually, case doesn't matter at all to VBA—it treats unitCost the same as uNitCost or any other variation, but the editor at least promotes consistency.

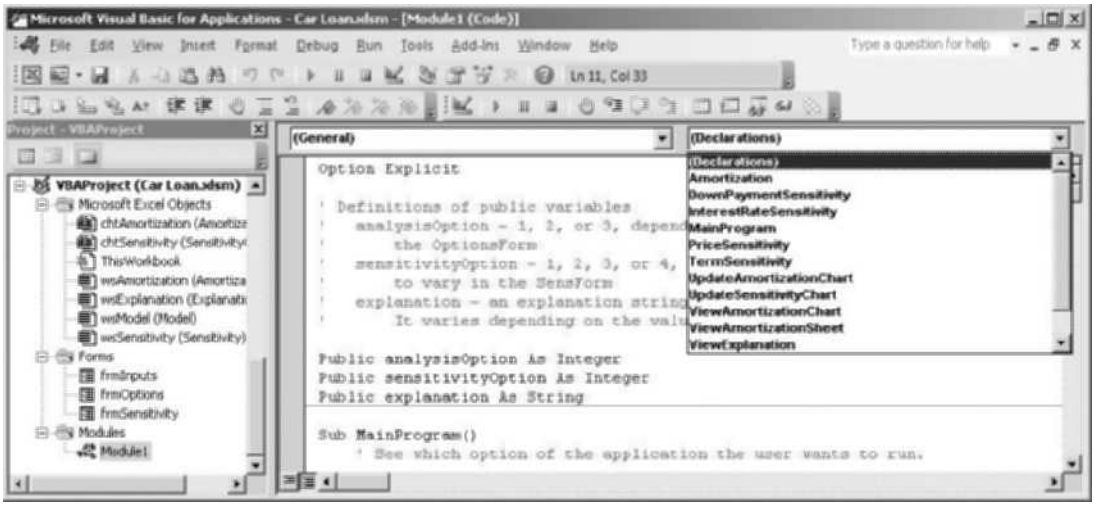
3.8 Finding Subs in the VBE

For the next few chapters, each of your programs will consist of a single sub, so when you select the file's module in the VBE's Project Explorer, your sub will appear in the Code window. However, in the programs in Part II of the book, there are multiple subs, and not all of them are in modules. In this case, it can be tedious to locate them in the Code window. Fortunately, the VBE provides tools to make this easy.

To follow along, open the **Car Loan.xlsm** file from Chapter 19. It not only has multiple subs in its module, but it has code in other locations, including code behind user forms (discussed in Chapter 11). The point is that it has multiple subs in various places. For now, double-click Module1 in the Project Explorer. You will see the MainProgram sub in the Code window. Now click the right dropdown arrow above the Code window. (See Figure 3.14.) You will see a list of all subs in Module1. To go quickly to any of them, just select the one you want.

Next, right-click the first form, frmInputs, in the Project Explorer and select View Code. This shows the code behind this form. (Again, all of this is explained in Chapter 11.) Now click the left dropdown arrow above the Code window. (See Figure 3.15.) You will see a list of all the controls on the form. Any of these can have associated code that responds to its events. For example, select

Figure 3.14 List of Subs in a Module



Source: Microsoft Corporation