MrExcel

Bill Jelen
Tracy Syrstad

VBA AND MACROS:

Microsoft® Excel® 2010

AUTOMATE REPORTS
BUILD FUNCTIONS
VISUALIZE DATA
WRITE FAST, RELIABLE SCRIPTS

MrExcel LIBRARY

VBA and Macros: Microsoft® Excel® 2010

Bill Jelen

Tracy Syrstad

que®

800 E. 96th Street
Indianapolis, Indiana 46240

Contents at a Glance

VBA and Macros: Microsoft® Excel® 2010

ISBN-13: 978-0-7897-4314-5
ISBN-10: 0-7897-4314-0

Library of Congress Cataloging-in-Publication Data:
Jelen, Bill.
VBA and macros : Microsoft Excel 2010 / Bill Jelen, Tracy Syrstad.
p. cm.
Includes index.
ISBN-13: 978-0-7897-4314-5
ISBN-10: 0-7897-4314-0
1. Microsoft Excel (Computer file) 2. Microsoft Visual Basic for applications. 3. Business—Computer programs. 4. Electronic spreadsheets. I. Syrstad, Tracy. II. Title.

HF5548.4.M523J46 2010
005.54—dc22

2010018831

Printed in the United States of America

Eighth Printing: August 2014

Associate Publisher
Greg Wiegand

Acquisitions Editor
Loretta Yates

Development Editor
Sondra Scott

Managing Editor
Sandra Schroeder

Senior Project Editor
Tonya Simpson

Copy Editor
Keith Kline

Indexer
Erika Millen

Proofreader
Language Logistics

Technical Editor
Bob Umlas

Publishing Coordinator
Cindy Teeters

Book Designer
Anne Jones

Compositor
Bronkella Publishing

Contents

About the Authors

Bill Jelen, Excel MVP and the host of MrExcel.com, has been using spreadsheets since 1985, and he launched the MrExcel.com website in 1998. Bill was a regular guest on Call for Help with Leo Laporte and has produced more than 1,200 episodes of his daily video podcast, Learn Excel from MrExcel. He is the author of 30 books about Microsoft Excel and writes the monthly Excel column for *Strategic Finance* magazine. You will most frequently find Bill taking his show on the road, doing half-day Power Excel seminars wherever he can find a room full of accountants or Excellers. Before founding MrExcel.com, Jelen spent 12 years in the trenches—working as a financial analyst for finance, marketing, accounting, and operations departments of a $500 million public company. He lives near Akron, Ohio, with his wife, Mary Ellen, and his sons, Josh and Zeke.

Tracy Syrstad is the project manager for the MrExcel consulting team. She was introduced to Excel VBA by a co-worker who encouraged her to learn VBA by recording steps and then modifying the code as needed. Her first macro was a simple lookup and highlight for a parts index, although it hardly seemed simple then. But she was encouraged by this success and others to follow. She'll never forget the day when it all clicked. She hopes this book will bring that click to its readers sooner and with less frustration. She lives near Sioux Falls, South Dakota, with her husband, John.

Dedication

To everyone in the MrExcel.com message board community.
—Bill Jelen

To John, who would only accept perfection, even if it took four coats of paint.
—Tracy Syrstad

Acknowledgments

Thanks to Tracy Syrstad for being a great co-author and for doing a great job of managing all the consulting projects at MrExcel.com.

Bob Umlas is the smartest Excel guy I know and is an awesome technical editor. At Pearson, Loretta Yates is an excellent acquisitions editor.

Along the way, I've learned a lot about VBA programming from the awesome community at the MrExcel.com message board. VoG and Richard Schollar and Jon von der Heyden all stand out as having contributed posts that lead to ideas in this book. Thanks to Pam Gensel for Excel macro lesson #1. Mala Singh taught me about creating charts in VBA, and Oliver Holloway brought me up to speed with accessing SQL Server.

At MrExcel.com, thanks to Barb Jelen, Wei Jiang, Tracy Syrstad, Schar Oswald, and Scott Pierson. Thanks also to Josh and Zeke Jelen, who have been picking up hours after school learning how to edit and produce the MrExcel podcast.

Finishing five Excel books for Excel 2010 simultaneously has been a monumental task. My family was incredibly supportive during this time. Thanks for Josh, Zeke, and Mary Ellen Jelen.

—Bill

Thanks to Bill Jelen, whose trust in me to run the consulting side of his business has done so much in building my self-confidence. And to LKH, whose blog I've learned so much from about writing and balancing working in the home and still having a personal life.

Richard Schollar and Joe Miskey: You've both been invaluable managing member issues at the forum and I feel I don't say thank you often enough. Thank you! And thanks to all the moderators who keep the board organized, despite the best efforts of the spammers.

There have been so many MrExcel.com clients whose projects have shown myriad ways that Excel can be used. Your excitement and appreciation over the solution we provide you has brightened my day as often as your unique projects have kept this job interesting.

—Tracy

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

As an associate publisher for Que Publishing, I welcome your comments. You can email or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that I cannot help you with technical problems related to the topic of this book. We do have a User Services group, however, where I will forward specific technical questions related to the book.

When you write, please be sure to include this book's title and author as well as your name, email address, and phone number. I will carefully review your comments and share them with the author and editors who worked on the book.

Email: feedback@quepublishing.com

Mail: Greg Wiegand
Associate Publisher
Que Publishing
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at quepublishing.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

INTRODUCTION

IN THIS INTRODUCTION

Getting Results with VBA

As corporate IT departments have found themselves with long backlogs of requests, Excel users have discovered they can produce the reports needed to run their business themselves using the macro language *Visual Basic for Applications* (VBA). VBA enables you to achieve tremendous efficiencies in your day-to-day use of Excel. This is both a good and bad thing. On the good side, without waiting for resources from IT, VBA helps you figure out how to import data and produce reports in Excel. On the bad side, you are now stuck importing data and producing reports in Excel.

What Is in This Book?

You have taken the right step by purchasing this book. I can help you reduce the learning curve so that you can write your own VBA macros and put an end to the burden of generating reports manually.

Reduce the Learning Curve

This Introduction provides a brief history of spreadsheets. Chapter 1 introduces the tools and confirms what you probably already know: The macro recorder does not work. Chapter 2 helps you understand the crazy syntax of VBA. Chapter 3 breaks the code on how to work efficiently with ranges and cells. By the time you get to Chapter 4, you will know enough to begin using the 25 sample user-defined functions in that chapter.

Chapter 5 covers the power of looping using VBA. The case study in this chapter creates a program to produce a department report, and then wraps that report routine in a loop to produce 46 reports.

Chapter 6 covers R1C1-style formulas. Chapter 7 looks at what changed in Excel VBA from Excel 2003 to Excel 2010. In the past, it was fairly straightforward to create VBA code that would run on any of the recent versions of Excel. Unfortunately, with the sweeping changes in Excel 2007 and Excel 2010, it became significantly more difficult to create this VBA code. Chapter 8 covers names. Chapter 9 includes some great tricks that use event programming. Chapter 10 introduces custom dialog boxes that you can use to collect information from the human using Excel.

Excel VBA Power

Chapters 11 through 13 provide an in-depth look at charting, Advanced Filter, and pivot tables. Any report automation tool will rely heavily on these concepts. Chapter 14 includes 25 code samples designed to exhibit the power of Excel VBA.

Chapters 15 through 18 handle data visualizations, web queries, sparklines, and automating another Office program such as Word.

Techie Stuff Needed to Produce Applications

Chapter 19 shows how to use arrays to build fast applications. Chapters 20 and 21 handle reading and writing to text files and Access databases. The techniques for using Access databases enable you to build an application with the multi-user features of Access while keeping the friendly front end of Excel.

Chapter 22, as it examines classes and collections, covers VBA from a Visual Basic programmer's point of view. Chapter 23 discusses advanced userform topics. Chapter 24 teaches some tricky ways to achieve tasks using the Windows application programming interface. Chapters 25 through 27 deal with error handling, custom menus, and add-ins.

Does This Book Teach Excel?

Microsoft believes the average Office user touches only 10 percent of the features in Office. I realize everyone reading this book is above average, and I have a pretty smart audience at MrExcel.com. Even so, a poll of 8,000 MrExcel.com readers shows that only 42 percent of smarter-than-average users are using any one of the top 10 power features in Excel.

I regularly present a Power Excel seminar for accountants. These are hard-core Excelers who use Excel 30 to 40 hours every week. Even so, two things come out in every seminar. First, half the audience gasps when they see how quickly you can do tasks with a particular feature such as automatic subtotals or pivot tables. Second, someone in the audience routinely trumps me. For example, someone asks a question, I answer, and someone in the second row raises a hand to give a better answer.

The point? You and I both know a lot about Excel. However, I will assume that in any given chapter, maybe 58 percent of the people have not used pivot tables before and maybe even fewer have used the "Top 10 Filter" feature of pivot tables. With this in mind, before I show how to automate something in VBA, I briefly cover how to do the same task in the

Excel interface. This book does not teach you how to do pivot tables, but it does alert you when you might need to explore a topic and learn more about it elsewhere.

CASE STUDY: MONTHLY ACCOUNTING REPORTS

This is a true story. Valerie is a business analyst in the accounting department of a medium-size corporation. Her company recently installed an overbudget $16 million ERP system. As the project ground to a close, there were no resources left in the IT budget to produce the monthly report that this corporation used to summarize each department.

However, Valerie had been close enough to the implementation process to think of a way to produce the report herself. She understood that she could export General Ledger data from the ERP system to a text file with comma-separated values. Using Excel, Valerie was able to import the G/L data from the ERP system into Excel.

Creating the report was not easy. Like many companies, there were exceptions in the data. Valerie knew that certain accounts in one particular cost center needed to be reclassed as an expense. She knew that other accounts needed to be excluded from the report entirely. Working carefully in Excel, Valerie made these adjustments. She created one pivot table to produce the first summary section of the report. She cut the pivot table results and pasted them into a blank worksheet. Then she created a new pivot table report for the second section of the summary. After about 3 hours, she had imported the data, produced five pivot tables, arranged them in a summary, and neatly formatted the report in color.

Becoming the Hero

Valerie handed the report to her manager. The manager had just heard from the IT department that it would be months before they could get around to producing "that convoluted report." When Valerie created the Excel report, she became the instant hero of the day. In 3 hours, Valerie had managed to do the impossible. Valerie was on cloud nine after a well-deserved "atta-girl."

More Cheers

The next day, Valerie's manager attended the monthly department meeting. When the department managers started complaining that they could not get the report from the ERP system, this manager pulled out his department report and placed it on the table. The other managers were amazed. How was he able to produce this report? Everyone was relieved to hear that someone had cracked the code. The company president asked Valerie's manager if he could have the report produced for each department.

Cheers Turn to Dread

You can probably see this coming. This particular company had 46 departments. That means 46 one-page summaries had to be produced once a month. Each report required importing data from the ERP system, backing out certain accounts, producing five pivot tables, and then formatting the reports in color. Even though it had taken Valerie 3 hours to produce the first report, after she got into the swing of things, she could produce the 46 reports in 40 hours. This is horrible. Valerie had a job to do before she became responsible of spending 40 hours a month producing these reports in Excel.

VBA to the Rescue

Valerie found my company, MrExcel Consulting, and explained her situation. In the course of about a week, I was able to produce a series of macros in Visual Basic that did all the mundane tasks. For example, it imported the data, backed out certain accounts, did five pivot tables, and applied the color formatting. From start to finish, the entire 40-hour manual process was reduced to two button clicks and about 4 minutes.

Right now, either you or someone in your company is probably stuck doing manual tasks in Excel that can be automated with VBA. I am confident that I can walk into any company with 20 or more Excel users and find a case equally amazing as Valerie's.

The Future of VBA and Windows Versions of Excel

Seven years ago, there were many rumblings that Microsoft might stop supporting VBA. There is now plenty of evidence to indicate that VBA will be around in Windows versions of Excel through 2025. When VBA was removed from the Mac version of Excel 2008, a huge outcry from customers led to it being included in the next Mac version of Excel.

Microsoft has stated that in Excel 15, which is the next version of Excel, it will stop providing support for XLM macros. These macros were replaced by VBA in 1993, and 17 years later, they are still supported. There is a chance that Microsoft will introduce a new programming language for the macro recorder in Excel 15. Assuming Microsoft continues to support VBA for 17 years after Excel 2012, you should be good through the mid-to-late 2020s.

However, you can see Microsoft's lack of commitment to VBA. Office 2003 offered a few features, such as the Research Pane and SmartTags, which could only be automated with Visual Basic.Net. The charting macro recorder, which was not finished in time to ship with Excel 2007, is included in Excel 2010.

The tools that you learn today will be good for the next 15 years. Even if Microsoft decides to scrap VBA in favor of another language, your coding skills will most likely transfer to the new platform.

Versions of Excel

This Third Edition of *VBA and Macros* is designed to work with Excel 2010. The previous editions of this book covered code for Excel 97 through Excel 2007. In 80 percent of the chapters, the code for Excel 2010 will be identical to code in previous versions. However, there are exceptions. For example, Microsoft offers new sorting logic, and charts have changed completely. In addition, the conditional formatting and data visualization tools in Chapter 15 are brand new. With Excel 2010, pivot tables offer new calculation options and slicers. The XML examples in Chapter 17 will work only with Excel 2003 or newer.

Differences for Mac Users

Although Excel for Windows and Excel for the Mac are similar in their user interface, there are a number of differences when you compare the VBA environment. Certainly, nothing in Chapter 24 that uses the Windows API will work on the Mac. The overall concepts discussed in the book apply to the Mac, but differences will exist. You can find a general list of differences as they apply to the Mac at http://www.mrexcel.com/macvba.html.

Special Elements and Typographical Conventions

The following typographical conventions are used in this book:

- *Italic*—Indicates new terms when they are defined, special emphasis, non-English words or phrases, and letters or words used as words
- `Monospace`—Indicates parts of VBA code such as object or method names, and filenames
- *`Italic monospace`*—Indicates placeholder text in code syntax
- **`Bold monospace`**—Indicates user input

In addition to these typographical conventions, there are several special elements. Each chapter has at least one case study that presents a real-world solution to common problems. The case study also demonstrates practical applications of topics discussed in the chapter.

In addition to the case studies, you will see New icons, Notes, Tips, and Cautions.

> **NOTE** Notes provide additional information outside the main thread of the chapter discussion that might be useful for you to know.

> **TIP** Tips provide quick workarounds and time-saving techniques to help you work more efficiently.

> **CAUTION**
>
> Cautions warn about potential pitfalls you might encounter. Pay attention to the Cautions; they alert you to problems that may otherwise cause you hours of frustration.

Code Files

As a thank you for buying this book, the authors have put together a set of 50 Excel workbooks that demonstrate the concepts included in this book. This set of files includes all the code from the book, sample data, additional notes from the authors, and 25 additional bonus macros. To download the code files, visit this book's web page at http://www.quepublishing.com or http://www.mrexcel.com/getcode2010.html.

Next Steps

Chapter 1 introduces the editing tools of the Visual Basic environment and shows why using the macro recorder is not an effective way to write VBA macro code.

Unleash the Power of Excel with VBA

1

IN THIS CHAPTER

The Power of Excel

Visual Basic for Applications (VBA) combined with Microsoft Excel is probably the most powerful tool available to you. VBA is sitting on the desktops of 500 million users of Microsoft Office, and most have never figured out how to harness the power of VBA in Excel. Using VBA, you can speed the production of any task in Excel. If you regularly use Excel to produce a series of monthly charts, you can have VBA do the same task for you in a matter of seconds.

Barriers to Entry

There are two barriers to learning successful VBA programming. First, Excel's macro recorder is flawed and does not produce workable code for you to use as a model. Second, for many who learned a programming language such as BASIC, the syntax of VBA is horribly frustrating.

The Macro Recorder Doesn't Work!

Microsoft began to dominate the spreadsheet market in the mid-1990s. Although it was wildly successful in building a powerful spreadsheet program to which any Lotus 1-2-3 user could easily transition, the macro language was just too different. Anyone proficient in recording Lotus 1-2-3 macros who tried recording a few macros in Excel most likely failed. Although the Microsoft VBA programming language is much more powerful than the Lotus 1-2-3 macro language, the fundamental flaw is that the macro recorder does not work.

With Lotus 1-2-3, you could record a macro today, play it back tomorrow, and it would faithfully work.

When you attempt the same feat in Microsoft Excel, the macro might work today but not tomorrow. In 1995, when I tried to record my first Excel macro, I was horribly frustrated by this.

Visual Basic Is Not Like BASIC

The code generated by the macro recorder was unlike anything I had ever seen. It said this was "Visual Basic" (VB). I had the pleasure of learning half a dozen programming languages at various times; this bizarre-looking language was horribly unintuitive and did not resemble the BASIC language I had learned in high school.

To make matters worse, even in 1995 I was the spreadsheet wizard in my office. My company had forced everyone to convert from Lotus 1-2-3 to Excel, which meant I was faced with a macro recorder that didn't work and a language that I couldn't understand. This was not a good combination of events.

My assumption in writing this book is that you are pretty talented with a spreadsheet. You probably know more than 90 percent of the people in your office. I also assume that even though you are not a programmer, you might have taken a class in BASIC at some point. However, knowing BASIC is not a requirement—it actually is a barrier to entry into the ranks of being a successful VBA programmer. There is a good chance that you have recorded a macro in Excel and a similar chance that you were not happy with the results.

Good News: Climbing the Learning Curve Is Easy

Even if you've been frustrated with the macro recorder, it is really just a small speed bump on your road to writing powerful programs in Excel. This book will not only teach you why the macro recorder fails, but also how to change the recorded code into something useful. For all the former BASIC programmers in the audience, I will decode VBA so that you can easily pick through recorded macro code and understand what is happening.

Great News: Excel with VBA Is Worth the Effort

Although you probably have been frustrated with Microsoft over the inability to record macros in Excel, the great news is that Excel VBA is powerful. Absolutely anything you can do in the Excel interface can be duplicated with stunning speed in Excel VBA. If you find yourself routinely creating the same reports manually day after day or week after week, Excel VBA will greatly streamline those tasks.

The authors of this book work for MrExcel Consulting. In this role, we have automated reports for hundreds of clients. The stories are often similar: The MIS department has a several-month backlog of requests. Someone in accounting or engineering discovers that he or she can import some data into Excel and get the reports necessary to run the business. This is a liberating event—you no longer need to wait months for the IT department to write a program. However, the problem is that after you import the data into Excel and win accolades from your manager for producing the report, you will likely be asked to produce the same report every month or every week. This becomes very tedious.

Again, the great news is that with a few hours of VBA programming, you can automate the reporting process and turn it into a few button clicks. The reward is great. So, hang with me as we cover a few of the basics.

This chapter exposes why the macro recorder does not work. It also walks through an example of recorded code and demonstrates why it will work today but fail tomorrow. I realize that the code you see in this chapter might not be familiar to you, but that's okay. The point of this chapter is to demonstrate the fundamental problem with the macro recorder. You also learn the fundamentals of the Visual Basic environment.

Knowing Your Tools: The Developer Tab

Let's start with a basic overview of the tools needed to use VBA. By default, Microsoft hides the VBA tools. You need to complete the following steps to change a setting in Excel options to access the Developer tab.

1. Open the File menu to get to the new Backstage view.
2. Along the left navigation bar, select Options under Excel.
3. In the Excel Options dialog, select Customize Ribbon from the left navigation.
4. In the Right list box, the Developer tab is third from the bottom. Select the check box next to this item.
5. Click OK to return to Excel.

Excel displays the Developer tab shown in Figure 1.1.

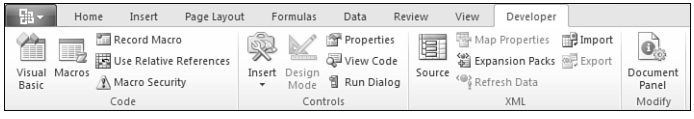


Figure 1.1
The Developer tab provides an interface for running and recording macros.

The Code group on the Developer tab contains the icons used for recording and playing back VBA macros, as listed here:

- **Visual Basic icon**—Opens the Visual Basic Editor.
- **Macros icon**—Displays the Macro dialog, where you can choose to run or edit a macro from the list of macros.
- **Record Macro icon**—Begins the process of recording a macro.
- **Use Relative Reference icon**—Toggles between using relative or absolute recording. With relative recording, Excel will record that you move down three cells. With absolute recording, Excel will record that you selected cell A4.
- **Macro Security icon**—Accesses the Trust Center, where you can choose to allow or disallow macros to run on this computer.

The Controls group of the Developer tab contains an Insert menu where you can access a variety of programming controls that can be placed on the worksheet. See "Assigning a Macro to a Form Control, Text Box, or Shape," later in this chapter. Other icons in this group enable you to work with the on-sheet controls. The Run Dialog button enables you to display a custom dialog box or userform that you designed in VBA. For more on userforms, see Chapter 10, "Userforms: An Introduction."

NOTE The XML group of the Developer ribbon contains tools for importing and exporting XML documents.

Macro Security

After VBA macros were used as the delivery method for some high-profile viruses, Microsoft changed the default security settings to prevent macros from running. Therefore, before we can begin discussing the recording of a macro, we need to show you how to adjust the default settings.

In Excel 2010, you can either globally adjust the security settings or control macro settings for certain workbooks by saving the workbooks in a trusted location. Any workbooks stored in a folder that is marked as a trusted location will automatically have its macros enabled.

You can find the macro security settings under the Macro Security icon on the Developer tab. When you click this icon, the Macro Settings category of the Trust Center is displayed. You can use the left navigation bar in the dialog to access the Trusted Locations list.

Adding a Trusted Location

You can choose to store your macro workbooks in a folder that is marked as a trusted location. Any workbook stored in a trusted folder will have its macros enabled. Microsoft suggests that a trusted location should be on your hard drive. The default setting is that you cannot trust a location on a network drive.

To specify a trusted location, follow these steps:

1. Click Macro Security in the Developer tab.
2. Click Trusted Locations in the left navigation pane of the Trust Center.
3. If you want to trust a location on a network drive, select Allow Trusted Locations on My Network.
4. Click the Add New Location button. Excel displays the Microsoft Office Trusted Locations dialog (see Figure 1.2).
5. Click the Browse button. Excel displays the Browse dialog.
6. Browse to the parent folder of the folder you want to be a trusted location. Click the trusted folder. Although the folder name does not appear in the Folder Name box, click OK. The correct folder name will appear in the Browse dialog.

7. If you want to trust subfolders of the selected folder, select Subfolders of This Location Will Be Trusted.
8. Click OK to add the folder to the Trusted Locations list.

> **CAUTION**
>
> Use care when selecting a trusted location. When you double-click an Excel attachment in an e-mail, Outlook stores the file in a temporary folder on your C: drive. You will not want to globally add C:\ and all subfolders to the Trusted Locations list.

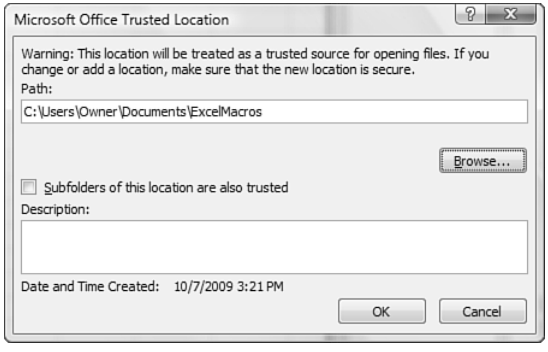


Figure 1.2
Manage trusted folders on the Trusted Locations category of the Trust Center.

Although trusted locations are not new in Excel 2010, Microsoft has made the process of adding trusted locations more discoverable in Excel 2010.

Using Macro Settings to Enable Macros in Workbooks Outside of Trusted Locations

For all macros not stored in a trusted location, Excel relies on the macro settings. The Low, Medium, High, and Very High settings that were familiar in Excel 2003 have been renamed.

To access the macro settings, click Macro Security in the Developer tab. Excel displays the Macro Settings category of the Trust Center dialog. Select the second option, Disable All Macros with Notification. A description of each option follows:

- **Disable All Macros Without Notification**—This setting prevents all macros from running. This setting is for people who never intend to run macros. Because you are currently holding a book that teaches you how to use macros, it is assumed that this setting is not you. This setting is roughly equivalent to the old Very High Security setting in Excel 2003. With this setting, only macros in the Trusted Locations folders can run.
- **Disable All Macros with Notification**—This setting is similar to Medium security in Excel 2003 and is the recommended setting. In Excel 2003, a Medium setting caused a box to be displayed when you opened a file containing macros. This box forced the

person to choose either Enable or Disable. Many novice Excel users randomly choose from this box. In Excel 2010, the message is displayed in the Message Area that macros have been disabled. You can choose to enable the content by clicking that option, as shown in Figure 1.3.

- **Disable All Macros Except Digitally Signed Macros**—This setting requires you to obtain a digital signing tool from VeriSign or another provider. This might be appropriate if you are going to be selling add-ins to others, but a bit of a hassle if you just want to write macros for your own use.
- **Enable All Macros (Not Recommended: Potentially Dangerous Code Can Run)**—This setting is similar to Low macro security in Excel 2003. Although it requires the least amount of hassle, it also opens your computer up to attacks from malicious Melissa-like viruses. Microsoft suggests that you do not use this setting.

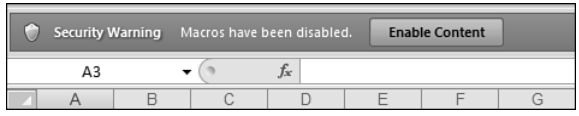


Figure 1.3
Open a macro workbook using the Disable All Macros with Notification setting to enable the macros.

Using Disable All Macros with Notification

It is recommended that you set your macro settings to Disable All Content with Notification. If you use this setting and open a workbook that contains macros, you will see a Security Warning in the area just above the formula bar. Assuming you were expecting macros in this workbook, click Enable Content.

If you do not want to enable macros for the current workbook, dismiss the Security Warning by clicking the *X* at the far right of the message bar.

If you forget to enable the macros and attempt to run a macro, Excel indicates that you cannot run the macro because all macros have been disabled. If this occurs, close the workbook and reopen it to access the message bar again.

> **CAUTION**
>
> After you enable macros in a workbook stored on a local hard drive and then save the workbook, Excel will remember that you previously enabled macros in this workbook. The next time you open this workbook, macros will be automatically enabled.

Overview of Recording, Storing, and Running a Macro

Recording a macro is useful when you do not have experience in writing lines of code in a macro. As you gain more knowledge and experience, you will begin to record lines of code less frequently.

To begin recording a macro, select Record Macro from the Developer tab. Before recording begins, Excel displays the Record Macro dialog box, as shown in Figure 1.4.

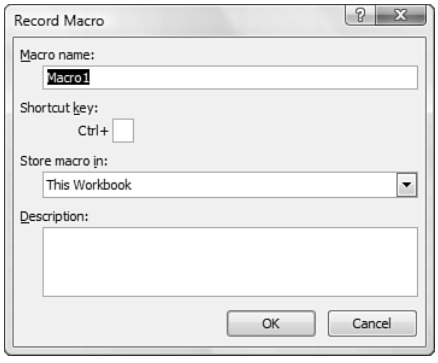


Figure 1.4
Use the Record Macro dialog box to assign a name and a shortcut key to the macro being recorded.

Filling Out the Record Macro Dialog

In the Macro Name field, type a name for the macro. Be sure to type continuous characters. For example, type `Macro1` without a space, not `Macro 1` with a space. Assuming you will soon be creating many macros, use a meaningful name for the macro. A name such as FormatReport is more useful than Macro1.

The second field in the Record Macro dialog box is a shortcut key. If you type `J` in this field, and then press Ctrl+J, this macro runs. Note that most of the lowercase shortcuts from Ctrl+a through Ctrl+z already have a use in Excel. Rather than being limited to the unassigned Ctrl+j, you can hold down the Shift key and type Shift+A through Shift+Z in the shortcut box. This will assign the macro to Ctrl+Shift+A.

> **CAUTION**
>
> You can reuse a shortcut key for a macro. If you assign a macro to Ctrl+c, Excel will run your macro instead of doing the normal action of copy.

In the Record Macro dialog box, choose where you want to save a macro when it is recorded: Personal Macro Workbook, New Workbook, This Workbook. It is recommended that you store macros related to a particular workbook in This Workbook.

The Personal Macro Workbook (`Personal.xlsb`) is not a visible workbook; it is created if you choose to save the recording in the Personal Macro Workbook. This workbook is used to save a macro in a workbook that will open automatically when you start Excel, thereby enabling you to use the macro. After Excel is started, the workbook is hidden. If you want to display it, select Unhide from the View tab.

> **TIP** It is not recommended you use the personal workbook for every macro you save. Save only those macros that assist you in general tasks—not in tasks that are performed in a specific sheet or workbook.

The fourth box in the Record Macro dialog is for a description. This description is added as a comment to the beginning of your macro. Note that legacy versions of Excel automatically noted the date and username of the person recording the macro. Excel 2010 no longer automatically inserts this information in the Description field.

After you select the location where you want to store the macro, click OK. Record your macro. When you are finished recording the macro, click the Stop Recording icon in the Developer tab.

> **TIP** You can also access a Stop Recording icon in the lower-left corner of the Excel window. Look for a small blue square to the right of the word *Ready* in the status bar. Using this Stop button might be more convenient than returning to the Developer tab. After you record your first macro, this area will usually have a Record Macro icon, which is a small red dot on an Excel worksheet.

Running a Macro

If you assigned a shortcut key to your macro, you can play it by pressing the key combination. Macros can also be assigned to toolbar buttons, forms controls, drawing objects, or you can run them from the Visual Basic toolbar.

Creating a Macro Button on the Ribbon

You can add an icon to a new group on the Ribbon to run your macro. This is appropriate for macros stored in the Personal Macro Workbook. Follow these steps to add a macro button to the Ribbon:

1. Click the File menu and select Excel Options to open the Excel Options dialog.
2. In the Excel Options dialog, select the Customize Ribbon category from the left-side navigation.

> **TIP** Note that a shortcut to replace steps 1 and 2 is to right-click the Ribbon and select Customize Ribbon.

3. In the list box on the right, choose the tab name where you want to add an icon.
4. Click the New Group button below the right list box. Excel adds a new entry called New Group (Custom) to the end of the groups in that ribbon tab.

5. To move the group to the left in the ribbon tab, click the up-arrow icon on the right side of the dialog several times.
6. To rename the group, click the Rename button. Type a new name, such as Report Macros. Click OK. Excel will show the group in the list box as Report Macros (Custom). Note that the word *Custom* will not appear in the Ribbon.
7. Open the upper-left drop-down and choose Macros from the list. The Macros category is fourth in the list. Excel displays a list of available macros in the left list box.
8. Choose a macro from the left list box. Click the Add button in the center of the dialog. Excel moves the macro to the right list box in the selected group. Excel uses a generic VBA icon for all macros. You can change the icon by following steps 9 and 10.
9. Click the macro in the right list box. Click the Rename button at the bottom of the right list box. Excel displays a list of 180 possible icons. Choose an icon. Alternatively, type a friendly label for the icon, such as Format Report.
10. Click OK to close Excel options. The new button appears on the selected Ribbon tab.

Creating a Macro Button on the Quick Access Toolbar

You can add an icon to the Quick Access toolbar to run your macro. If your macro is stored in the Personal Macro Workbook, you can have the button permanently displayed in the Quick Access toolbar. If the macro is stored in the current workbook, you can specify that the icon should appear only when the workbook is open. Follow these steps to add a macro button to the Quick Access toolbar:

1. Click the File menu and select Options to open the Excel Options dialog.
2. In the Excel Options dialog select the Quick Access Toolbar category from the left-side navigation.

> **TIP** Note that a shortcut to replace steps 1 and 2 is to right-click the Quick Access toolbar and select Customize Quick Access Toolbar.

3. If your macro should be available only when the current workbook is open, open the upper-right drop-down and change For All Documents (Default) to For <FileName.xlsm>. Any icons associated with the current workbook are displayed at the end of the Quick Access toolbar.
4. Open the upper-left drop-down and select Macros from the list. The Macros category is fourth in the list. Excel displays a list of available macros in the left list box.
5. Choose a macro from the left list box. Click the Add button in the center of the dialog. Excel moves the macro to the right list box. Excel uses a generic VBA icon for all macros. You can change the icon by following steps 6 through 8.
6. Click the macro in the right list box. Click the Modify button at the bottom of the right list box. Excel displays a list of 180 possible icons (see Figure 1.5).

> **NOTE** Considering Excel 2003 offered 4,096 possible icons and an icon editor, the list of 180 is a major disappointment.

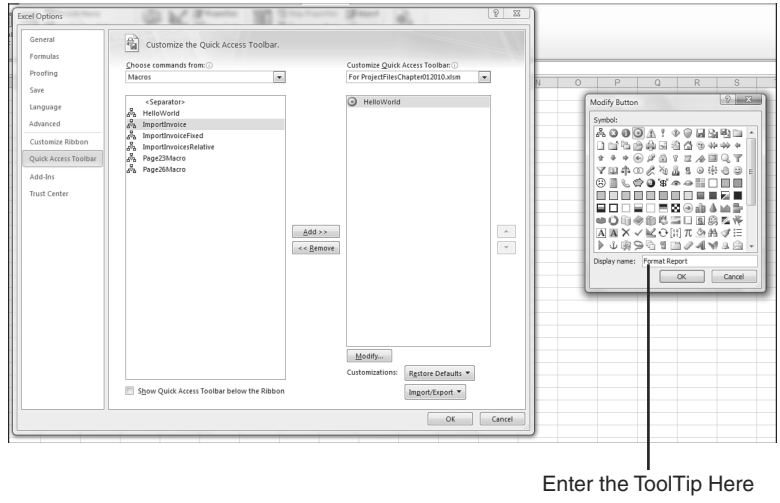


Figure 1.5
Attach a macro to a button on the Quick Access toolbar.

7. Choose an icon from the list. In the Display Name box, replace the macro name with a short name that will appear in the ToolTip for the icon.
8. Click OK to close the Modify Button dialog.
9. Click OK to close Excel options. The new button appears on the Quick Access toolbar.

Assigning a Macro to a Form Control, Text Box, or Shape

If you want to create a macro specific to a workbook, store the macro in the workbook and attach it to a form control or any object on the sheet.

Follow these steps to attach a macro to a form control on the sheet:

1. On the Developer tab, click the Insert button to open its drop-down list. Excel offers 12 form controls and 12 ActiveX controls. Many icons look similar in this drop-down. Click the Button Form Control icon at the upper-left icon in the drop-down.
2. Move your cursor over the worksheet; the cursor changes to a plus sign.
3. Draw a button on the sheet by clicking and holding the left mouse button while drawing a box shape. Release the button when you have finished.
4. Choose a macro from the Assign Macro dialog box and click OK. The button is created with generic text such as Button 1. To customize the text or the button appearance, follow steps 5 through 7.

5. Type a new label for the button. Note that while you are typing, the selection border around the button changes from dots to diagonal lines to indicate that you are in Text Edit mode. You cannot change the button color while in Text Edit mode. To exit Text Edit mode, either click the diagonal lines to change them to dots or Ctrl-click the button again. Note that if you accidentally click away from the button, you should Ctrl+click the button to select it. Then drag the cursor over the text on the button to select the text.
6. Right-click the dots surrounding the button and select Format Control. Excel displays the Format Control dialog with seven tabs across the top. If your Format Control dialog has only a Font tab, you failed to exit Text Edit mode. If this occurred, close the dialog, Ctrl-click the button, and repeat this step.
7. Use the settings in the Format Control dialog to change the font size, font color, margins, and similar settings for the control. Click OK to close the Format Control dialog when you have finished. Click on a cell to unselect the button.
8. Click the button to run the macro.

Macros can be assigned to any worksheet object such as clip art, a shape, SmartArt graphics, or text box. In Figure 1.6, the top button is a traditional button form control. The other images are clip art, a shape with WordArt, and a SmartArt graphic. To assign a macro to any object, right-click the object, and select Assign Macro.

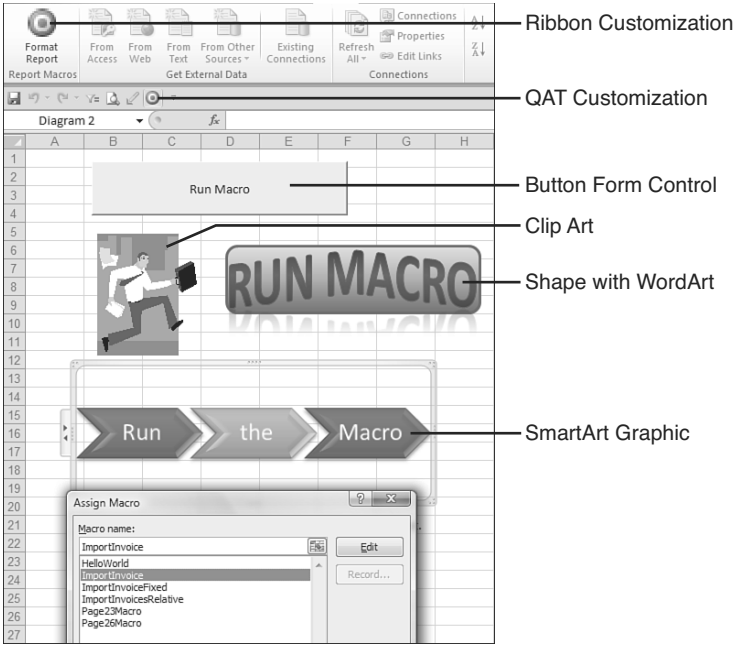


Figure 1.6
Assigning a macro to a form control or an object appropriate for macros stored in the same workbook as the control. You can assign a macro to any of these objects.

1

Using New File Types in Excel 2010

Excel 2010 offers support for four file types. Macros are not allowed to be stored in the default file type. You have to use the Save As setting for all of your macro workbooks, or you can change the default file type used by Excel 2010.

The available files types are as follows:

- **Excel Workbook (.xlsx)**—Files are stored as a series of XML objects and then zipped into a single file. This new file-saving paradigm in Excel 2010 allows for significantly smaller file sizes. It also allows other applications (even Notepad!) to edit or create Excel workbooks. Unfortunately, macros cannot be stored in files with an .xlsx extension.
- **Excel Macro-Enabled Workbook (.xlsm)**—This is similar to the default .xlsx format, except macros are allowed. The basic concept is that if someone has an .xlsx file, he or she will not need to worry about malicious macros. However, if they see an .xlsm file, they should be concerned that there might be macros attached.
- **Excel Binary Workbook (.xlsb)**—This is a binary format designed to handle the larger 1.1-million-row grid size in Excel 2010. Legacy versions of Excel stored their files in a proprietary binary format. Although binary formats might load quicker, they are more prone to corruption, and a few lost bits can destroy the whole file. Macros are allowed in this format.
- **Excel 97-2003 Workbook (.xls)**—This format produces files that can be read by anyone using legacy versions of Excel. Macros are allowed in this binary format; however, when you save in this format, you lose access to any cells outside of A1:IV65536. In addition, if someone opens the file in Excel 2003, he or she will lose access to anything that used features introduced in Excel 2007 or later.

To avoid having to choose a macro-enabled workbook in the Save As dialog, you can customize your copy of Excel to always save new files in the .xlsm format by following these steps:

1. Click the File menu and select Excel Options.
2. In the Excel Options dialog, select the Save category from the left navigation pane.
3. The first drop-down is Save Files in This Format. Open the drop-down and select Excel Macro-Enabled Workbook (*.xlsm). Click OK.

> **NOTE** Although you and I are not afraid to use macros, I have encountered some people who seem to freak out when they see the .xlsm file type. They actually seem angry that I sent them an .xlsm file that did not have any macros. Their reaction seemed reminiscent of King Arthur's "You got me all worked up!" line in *Monty Python and the Holy Grail*.

If you encounter someone who seems to have a fear of the .xlsm file type, remind them of these points:

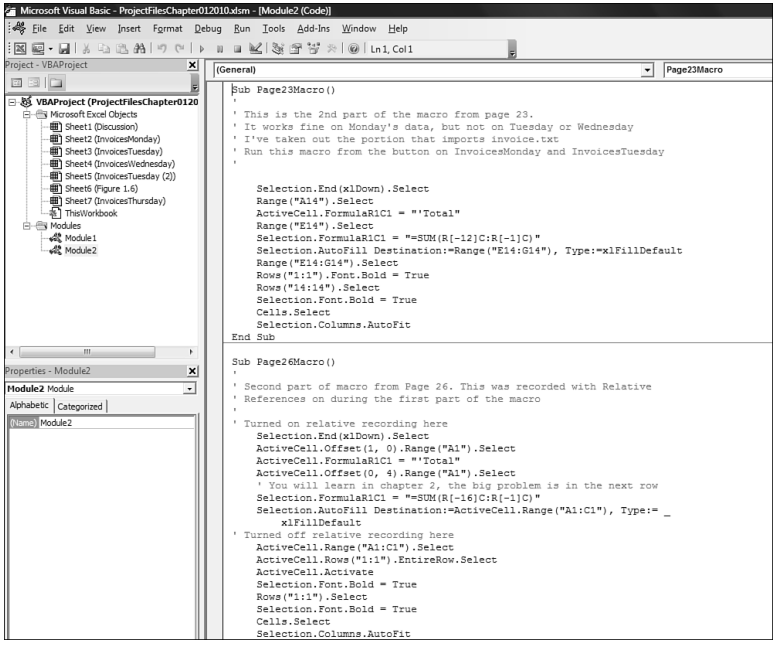
- Every workbook created in the past 20 years could have had macros, but in fact, most did not.
- If someone is trying to avoid macros, they should use the security settings to prevent macros from running anyway (refer to Figure 1.3). They can still open the .xlsm file to get the data in the spreadsheet.

With these arguments, I hope you can overcome any fears of the .xlsm file type so that it can be your default file type.

Understanding the VB Editor

Figure 1.7 shows an example of the typical VB Editor screen. You can see three windows: Project Explorer, the Properties window, and the Programming window. Don't worry if your window doesn't look exactly like this because you will see how to display the windows you need in this review of the editor.

Figure 1.7
The VB Editor window.



VB Editor Settings

Several settings in the VB Editor enable you to customize this editor. The following subsection covers the setting that will help with your programming.

Customizing VB Editor Options Settings

Under Tools, Options, Editor, you will find several useful settings. All settings except for one are set correctly by default. The remaining setting requires some consideration on your part. This setting is Require Variable Declaration. By default, Excel does not require you to declare variables. I prefer this setting because it can save time when you create a program. My coauthor prefers to change this setting to require variable declaration. This change forces the compiler to stop if it finds a variable that it does not recognize, which reduces misspelled variable names. It is a matter of your personal preference if you turn this setting on or keep it off.

The Project Explorer

The Project Explorer lists any open workbooks and add-ins that are loaded. If you click the + icon next to the VBA Project, you will see that there is a folder with Microsoft Excel objects. There can also be folders for forms, class modules, and standard modules. Each folder includes one or more individual components.

Right-clicking a component and selecting View Code or just double-clicking the components brings up any code in the Programming window. The exception is userforms, where double-clicking displays the userform in Design view.

To display the Project Explorer window, select View, Project Explorer from the menu, and then press Ctrl+R or click the Project Explorer icon on the toolbar.

Figure 1.8 shows the Project Explorer pane. This pane can show Microsoft Excel objects, userforms, modules, and class modules.

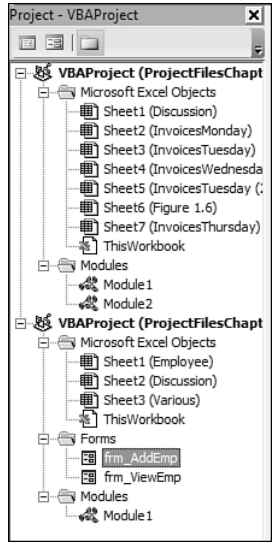


Figure 1.8
The Project Explorer window displays different types of modules.

To insert a module, right-click your project, select Insert, and then choose the type of module you want. The available modules are as follows:

- **Microsoft Excel objects—**By default, a project consists of sheet modules for each sheet in the workbook and a single ThisWorkbook module. Code specific to a sheet such as controls or sheet events is placed on the corresponding sheet. Workbook events are placed in the ThisWorkbook module. You learn more about events in Chapter 9, "Event Programming."
- **Forms—**Excel allows you to design your own forms to interact with the user. You learn more about these forms in Chapter 10.
- **Modules—**When you record a macro, Excel automatically creates a module in which to place the code. Most of your code will reside in these types of modules.
- **Class modules—**Class modules are Excel's way of letting you create your own objects. They also allow pieces of code to be shared among programmers without the programmer needing to understand how it works. You will learn more about class modules in Chapter 22, "Creating Classes, Records, and Collections."

The Properties Window

The Properties window enables you to edit the properties of various components such as sheets, workbooks, modules, and form controls. The Property list varies according to what component is selected. To display this window, select View, Properties Window from the menu, press F4, or click the Project Properties icon on the toolbar.

Understanding Shortcomings of the Macro Recorder

Suppose you work in an accounting department. Each day you receive a text file from the company system showing all the invoices produced the prior day. This text file has commas separating each field. The columns in the file are InvoiceDate, InvoiceNumber, SalesRepNumber, CustomerNumber, ProductRevenue, ServiceRevenue, and ProductCost (see Figure 1.9).

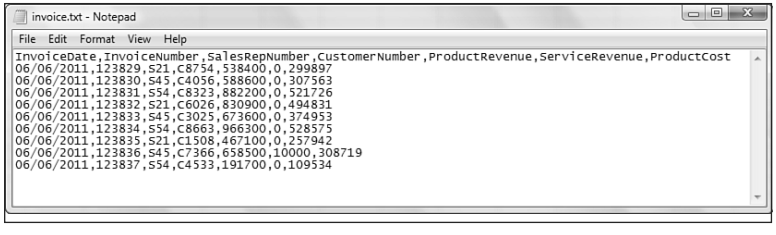


Figure 1.9
Invoice.txt file.

Each morning, you manually import this file into Excel. You add a total row to the data, bold the headings, and then print the report for distribution to a few managers.

This seems like a simple process that would be ideally suited to using the macro recorder. However, due to some problems with the macro recorder, your first few attempts might not be successful. The following case study explains how to overcome these problems.

1

CASE STUDY: PREPARING TO RECORD THE MACRO

The task mentioned in the previous section is perfect for a macro. However, before you record a macro, think about the steps you will use. In this case, the steps you will use are as follows:

1. Click the File menu and select Open.
2. Navigate to the folder where `Invoice.txt` is stored.
3. Select All Files(*.*) from the Files of Type drop-down list.
4. Select Invoice.txt.
5. Click Open.
6. In the Text Import Wizard—Step 1 of 3, select Delimited from the Original Data Type section.
7. Click Next.
8. In the Text Import Wizard—Step 2 of 3, clear the Tab key and select Comma in the Delimiters section.
9. Click Next.
10. In the Text Import Wizard—Step 3 of 3, select General in the Column Data Format section and change it to `Date: MDY`.
11. Click Finish to import the file.
12. Press the End key followed by the down arrow to move to the last row of data.
13. Press the down arrow one more time to move to the total row.
14. Type the word **`Total`**.
15. Press the right-arrow key four times to move to Column E of the total row.
16. Click the Autosum button and press Ctrl+Enter to add a total to the Product Revenue column while remaining in that cell.
17. Click the AutoFill handle and drag it from Column E to Column G to copy the total formula to Columns F and G.
18. Highlight Row 1 and click the Bold icon on the Home tab to set the headings in bold.
19. Highlight the Total row and click the Bold icon on the Home tab to set the totals in bold.
20. Press Ctrl+A to select all cells.
21. From the Home tab, select Format, AutoFit Column Width.

After you have rehearsed these steps in your head, you are ready to record your first macro. Open a blank workbook and save it with a name such as `MacroToImportInvoices.xlsm`. Click the Record Macro button on the Developer tab.

In the Record Macro dialog, the default macro name is Macro1. Change this to something descriptive like `ImportInvoice`. Make sure that the macros will be stored in This Workbook. You might want an easy way to run this macro later, so enter the letter `i` in the Shortcut Key field. In the Description field, add a little descriptive text to tell what the macro is doing (see Figure 1.10). Click OK when you are ready.

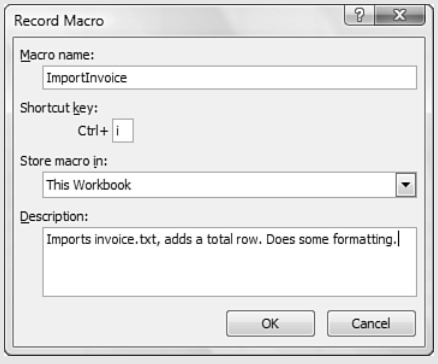


Figure 1.10
Before recording your macro, complete the Record Macro dialog box.

Recording the Macro

The macro recorder is now recording your every move, but don't be nervous. For this reason, perform your steps in exact order without extraneous actions. If you accidentally move to Column F, type a value, clear the value, and then move back to E to enter the first total, the recorded macro blindly makes that same mistake day after day after day. Recorded macros move fast, but there is nothing like watching the macro recorder play out your mistakes repeatedly.

Carefully, execute all the actions necessary to produce the report. After you have performed the final step, click the Stop button in the lower-left corner of the Excel window or click Stop Recording in the Developer tab.

Now it is time to look at your code. Switch to the VB Editor by selecting Visual Basic from the Developer tab or pressing Alt+F11.

Examining Code in the Programming Window

Let's look at the code you just recorded from the case study. Don't worry if it doesn't make sense yet.

To open the VB Editor, press Alt+F11. In your VBA Project (`MacroToImportInvoices.xls`), find the component Module1, right-click the module, and select View Code. Notice that some lines start with an apostrophe—these are comments and are ignored by the program. The macro recorder starts your macros with a few comments, using the description you entered in the Record Macro dialog. The comment for the Keyboard Shortcut is there to remind you of the shortcut.

NOTE

The comment does *not* assign the shortcut. If you change the comment to be Ctrl+J, it does not change the shortcut. You must change the setting in the Macro dialog box in Excel or run this line of code:

```
Application.MacroOptions Macro:="ImportInvoice", _
        Description:="", ShortcutKey:="j"
```

Recorded macro code is usually pretty neat (see Figure 1.11). Each noncomment line of code is indented four characters. If a line is longer than 100 characters, the recorder breaks it into multiple lines and indents the lines an additional four characters. To continue a line of code, type a space and an underscore at the end of the line.

> **NOTE** Note that the physical limitations of this book do not allow 100 characters on a single line. Therefore, the lines will be broken at 80 characters so that they fit on a page. For this reason, your recorded macro might look slightly different from the ones that appear in this book.

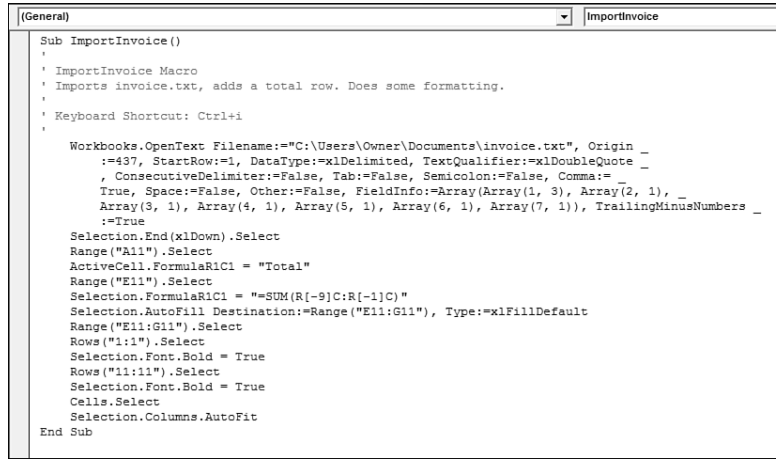


Figure 1.11
The recorded macro is neat looking and nicely indented.

Consider that the following seven lines of recorded code is actually only one line of code that has been broken into seven lines for readability:

```
Workbooks.OpenText Filename:= _
    "C:\invoice.txt", Origin:=437, StartRow:=1, DataType:=xlDelimited, _
    TextQualifier:=xlDoubleQuote, ConsecutiveDelimiter:=False,  _
    Tab:=True, Semicolon:=False, Comma:=True, Space:=False,  _
    Other:=False, FieldInfo:=Array(Array(1, 3), Array(2, 1), Array(3, 1), _
    Array(4, 1), Array(5, 1), Array(6, 1), Array(7, 1)), _
   TrailingMinusNumbers:=True
```

Counting this as one line, the macro recorder was able to record our 21-step process in 14 lines of code, which is pretty impressive.

> **NOTE** Each action you perform in the Excel user interface might equate to one or more lines of recorded code. Some actions might generate a dozen lines of code.

Test Each Macro

It is always a good idea to test macros. To test your new macro, return to the regular Excel interface by pressing Alt+F11. Close `Invoice.txt` without saving any changes. `MacroToImportInvoices.xls` is still open.

Press Ctrl+I to run the recorded macro. It should work beautifully if you completed the steps correctly. The data is imported, totals are added, bold formatting is applied, and the columns are made wider. This seems like a perfect solution (see Figure 1.12).

Figure 1.12
The macro formats the data in the sheet.

	A	B	C	D	E	F	G
1	InvoiceDate	InvoiceNumber	SalesRepNumber	CustomerNumber	ProductRevenue	ServiceRevenue	ProductCost
2	6/6/2011	123829	S21	C8754	538400	0	299897
3	6/6/2011	123830	S45	C4056	588600	0	307563
4	6/6/2011	123831	S54	C8323	882200	0	521726
5	6/6/2011	123832	S21	C6026	830900	0	494831
6	6/6/2011	123833	S45	C3025	673600	0	374953
7	6/6/2011	123834	S54	C8663	966300	0	528575
8	6/6/2011	123835	S21	C1508	467100	0	257942
9	6/6/2011	123836	S45	C7366	658500	10000	308719
10	6/6/2011	123837	S54	C4533	191700	0	109534
11	**Total**				**5797300**	**10000**	**3203740**
12							

Running the Macro on Another Day Produces Undesired Results

After testing the macro, be sure to save your macro file to use on another day. The next day, after receiving a new `Invoice.txt` file from the system, you open the macro, press Ctrl+I to run it, and disaster strikes. The data for June 6 happened to have 9 invoices, while the data for the June 7 has 17 invoices. However, the recorded macro blindly added the totals in Row 12 because this was where you put the totals when the macro was recorded (see Figure 1.13).

Figure 1.13
The intent of the recorded macro was to add a total at the end of the data, but the recorder made a macro that always adds totals at Row 11.

	A	B	C	D	E	F	G
1	InvoiceDate	InvoiceNumber	SalesRepNumber	CustomerNumber	ProductRevenue	ServiceRevenue	ProductCost
2	6/7/2011	123813	S82	C8754	716100	12000	423986
3	6/7/2011	123814		C4894	224200	0	131243
4	6/7/2011	123815	S43	C7278	277000	0	139208
5	6/7/2011	123816	S54	C6425	746100	15000	350683
6	6/7/2011	123817	S43	C6291	928300	0	488988
7	6/7/2011	123818	S43	C1000	723200	0	383069
8	6/7/2011	123819	S82	C6025	982600	0	544025
9	6/7/2011	123820	S17	C8026	490100	45000	243808
10	6/7/2011	123821	S43	C4244	615800	0	300579
11	**Total**	**123822**	**S45**	**C1007**	**5703400**	**72000**	**3005589**
12	6/7/2011	123823	S87	C1878	338100	0	165666
13	6/7/2011	123824	S43	C3068	567900	0	265775
14	6/7/2011	123825	S43	C7571	123456	0	55555
15	6/7/2011	123826	S55	C7181	37900	0	19811
16	6/7/2011	123827	S43	C7570	582700	0	292000
17	6/7/2011	123828	S87	C5302	495000	0	241504
18	6/7/2011	123828	S87	C5302	495000	0	241504
19							

This problem arises because the macro recorder is recording all your actions in absolute mode by default. Instead of using the default state of the macro recorder, the next section discusses relative recording and how this might get you closer to a final solution.

Possible Solution: Use Relative References When Recording

By default, the macro recorder records all actions as *absolute* actions. If you navigate to Row 11 when you record the macro on Monday, the macro will always go to Row 11 when the macro is run. This is rarely appropriate when dealing with variable numbers of rows of data. The better option is to use relative references when recording.

Macros recorded with absolute references note the actual address of the cell pointer, such as A11. Macros recorded with relative references note that the cell pointer should move a certain number of rows and columns from its current position. For example, if the cell pointer starts in cell A1, the code `ActiveCell.Offset(16, 1).Select` would move the cell pointer to B17, which is the cell 16 rows down and 1 column to the right.

Let's try the same case study again, this time using relative references. The solution will be much closer to working correctly.

CASE STUDY: RECORDING THE MACRO WITH RELATIVE REFERENCES

Let's try to record the macro again, but this time you will use relative references. Close `Invoice.txt` without saving changes. In the workbook `MacroToImportInvoices.xls`, record a new macro by selecting Record Macro from the Developer tab. Give the new macro a name of `ImportInvoicesRelative` and assign a different shortcut key such as Ctrl+Shift+J (see Figure 1.14).

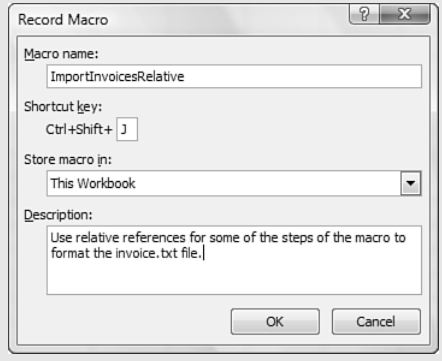


Figure 1.14
Getting ready to record a second try.

As you start to record the macro, go through the process of opening the `Invoice.txt` file. Before navigating to the last row of data by pressing the End key + then the down-arrow key, click the Use Relative Reference button on the Developer tab (refer to Figure 1.1).

Continue through the actions in the script from the case study:

1. Press the End key followed by the down-arrow key to move to the last row of data.
2. Press the down arrow one more time to move to the total row.
3. Type the word **`Total`**.

4. Press the right-arrow key four times to move to Column E of the Total row.
5. Click the Autosum button, and then press Ctrl+Enter to add a total to the Product Revenue column while remaining in that cell.
6. Click the AutoFill handle and drag from Column E to Column G to copy the total formula to Columns F and G.
7. Press Shift+spacebar to select the entire row. Type Ctrl+b to apply bold formatting to it. At this point, you need to move to Cell A1 to apply bold to the headings. You do not want the macro recorder to record the movement from Row 11 to Row 1 because it would record this as moving 10 rows up, which might not be correct tomorrow. Before moving to A1, toggle the Use Relative Recording button off, and then continue recording the rest of the macro.
8. Highlight Row 1 and click the Bold icon to set the headings in bold.
9. Press Ctrl+A to select all cells.
10. From the Home tab, select Format, AutoFit Column Width.
11. Select cell A1.
12. Stop recording.

Press Alt+F11 to go to the VB Editor to review your code. The new macro appears in Module1 below the previous macro.

If you close Excel between recording the first and second macro, Excel inserts a new module called Module2 for the newly recorded macro.

The following code has been edited with two comments that will help you remember where you turned the relative recording on and then off:

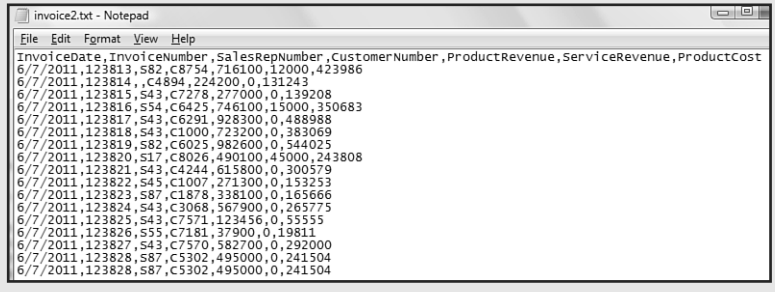
```
Sub ImportInvoicesRelative()
'
' ImportInvoicesRelative Macro
' Use relative references for some of the steps of the macro
' to format the invoice.txt file
'
    Workbooks.OpenText Filename:= _
        "C:\invoice.txt", Origin:=437, StartRow:=1, DataType:=xlDelimited, _
        TextQualifier:=xlDoubleQuote, ConsecutiveDelimiter:=False,  _
        Tab:=True, Semicolon:=False, Comma:=True, Space:=False,  _
        Other:=False, FieldInfo:=Array(Array(1, 3), Array(2, 1), Array(3, 1), _
        Array(4, 1), Array(5, 1), Array(6, 1), Array(7, 1)), _
       TrailingMinusNumbers:=True

' Turned on relative recording here
    Selection.End(xlDown).Select
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "'Total"
    ActiveCell.Offset(0, 4).Range("A1").Select
    Selection.FormulaR1C1 = "=SUM(R[-16]C:R[-1]C)"
    Selection.AutoFill Destination:=ActiveCell.Range("A1:C1"), Type:= _
        xlFillDefault
```

```
' Turned off relative recording here
    ActiveCell.Range("A1:C1").Select
    ActiveCell.Rows("1:1").EntireRow.Select
    ActiveCell.Activate
    Selection.Font.Bold = True
    Rows("1:1").Select
    Selection.Font.Bold = True
    Cells.Select
    Selection.Columns.AutoFit
    Range("A1").Select
End Sub
```

To test the macro, close `Invoice.txt` without saving, and then run the macro with Ctrl+J. Everything should look good, and you should get the same results.

The next test is to see whether the program works on the next day when you might have more rows. Figure 1.15 shows the data for June 7.

Figure 1.15
Will the macro with relative references work with this data?



Open `MacroToImportInvoices.xls` and run the new macro with Ctrl+J. This time, everything should look good with the totals in the correct places. Look at Figure 1.16—see anything out of the ordinary?

Figure 1.16
The result of running the Relative macro.

	A	B	C	D	E	F	G
1	InvoiceDate	InvoiceNumber	SalesRepNumber	CustomerNumber	ProductRevenue	ServiceRevenue	ProductCost
2	6/7/2011	123813	S82	C8754	716100	12000	423986
3	6/7/2011	123814		C4894	224200	0	131243
4	6/7/2011	123815	S43	C7278	277000	0	139208
5	6/7/2011	123816	S54	C6425	746100	15000	350683
6	6/7/2011	123817	S43	C6291	928300	0	488988
7	6/7/2011	123818	S43	C1000	723200	0	383069
8	6/7/2011	123819	S82	C6025	982600	0	544025
9	6/7/2011	123820	S17	C8026	490100	45000	243808
10	6/7/2011	123821	S43	C4244	615800	0	300579
11	6/7/2011	123822	S45	C1007	271300	0	153253
12	6/7/2011	123823	S87	C1878	338100	0	165666
13	6/7/2011	123824	S43	C3068	567900	0	265775
14	6/7/2011	123825	S43	C7571	123456	0	55555
15	6/7/2011	123826	S55	C7181	37900	0	19811
16	6/7/2011	123827	S43	C7570	582700	0	292000
17	6/7/2011	123828	S87	C5302	495000	0	241504
18	6/7/2011	123828	S87	C5302	495000	0	241504
19	Total				3527156	0	1735647

If you aren't careful, you might print these reports for your manager. If you did, you would be in trouble. When you look in cell E19, Excel has inserted a green triangle to tell you to look at the cell. If you happened to try this back in Excel 95 or Excel 97 before SmartTags, there would not have been an indicator that anything was wrong.

When you move the cell pointer to E19, an alert indicator pops up near the cell. This indicator tells you the formula fails to include adjacent cells. If you look in the formula bar, you will see that the macro totaled only from Row 10 to Row 18. Neither the relative recording nor the nonrelative recording is smart enough to replicate the logic of the AutoSum button.

At this point, some people would give up. However, imagine that you might have had fewer invoice records on this particular day. Excel would have rewarded you with the illogical formula of `=SUM(E6:E1048574)` and a circular reference, as shown in Figure 1.17.

Figure 1.17
The result of running the Relative macro with fewer invoice records.

E7 =SUM(E6:E1048574)

	A	B	C	D	E	F	G
1	InvoiceDate	InvoiceNumber	SalesRepNumber	CustomerNumber	ProductRevenue	ServiceRevenue	ProductCost
2	6/9/2011	123850		C1654	161000	0	90761
3	6/9/2011	123851		C6460	275500	10000	146341
4	6/9/2011	123852		C5143	925400	0	473515
5	6/9/2011	123853		C7868	148200	0	75700
6	6/9/2011	123854		C3310	890200	0	468333
7	Total				0	0	0

If you have tried using the macro recorder, most likely you would run into similar problems as the ones produced in the last two case studies. Although this is frustrating, you should be happy to know that the macro recorder actually gets you 95 percent of the way to a useful macro.

Your job is to recognize where the macro recorder is likely to fail and then to be able to dive into the VBA code to fix the one or two lines that require adjusting to have a perfect macro. With some added human intelligence, you can produce awesome macros to speed up your daily work.

If you are like me, you are cursing Microsoft about now. We have wasted a good deal of time over a couple of days, and neither macro works. What makes it worse is that this sort of procedure would have been handled perfectly by the old Lotus 1-2-3 macro recorder introduced in 1983. Mitch Kapor solved this problem 24 years ago, and Microsoft still can't get it right.

Did you know that up through Excel 97, Microsoft Excel secretly ran Lotus command-line macros? I found this out right after Microsoft quit supporting Excel 97. At that time, a number of companies upgraded to Excel XP, which no longer supported the Lotus 1-2-3 macros. Many of these companies hired us to convert the old Lotus 1-2-3 macros to Excel VBA. It is interesting that from Excel 5, Excel 95, and Excel 97, Microsoft offered an interpreter that could handle the Lotus macros that solved this problem correctly, yet their own macro recorder couldn't (and still can't!) solve the problem.

Never Use the AutoSum Button While Recording a Macro

There actually is a macro-recorder solution to the current problem. It is important to recognize that the macro recorder will never correctly record the intent of the AutoSum button.

If you are in cell E99 and click the AutoSum button, Excel starts scanning from cell E98 upward until it locates a text cell, a blank cell, or a formula. It then proposes a formula that sums everything between the current cell and the found cell.

However, the macro recorder records the particular result of that search on the day that the macro was recorded. Rather than record something along the lines of "do the normal AutoSum logic," the macro recorder inserts a single line of code to add up the previous 98 cells.

The somewhat bizarre workaround is to type a `SUM` function that uses a mix of relative and absolute row references. If you type `=SUM(E$2:E10)` while the macro recorder is running, Excel correctly adds code that will always sum from a fixed row two down to the relative reference that is just above the current cell.

Here is the resulting code with a few comments:

```
Sub FormatInvoice3()
'
' FormatInvoice2 Macro
' Third try. Use relative. Don't touch AutoSum
'
' Keyboard Shortcut: Ctrl+Shift+K
'
    Workbooks.OpenText Filename:="C:\Users\Owner\Documents\invoice.txt", Ori-
gin _
        :=437, StartRow:=1, DataType:=xlDelimited,
TextQualifier:=xlDoubleQuote _
        , ConsecutiveDelimiter:=False, Tab:=False, Semicolon:=False, Comma:= _
        True, Space:=False, Other:=False, FieldInfo:=Array(Array(1, 3), Ar-
ray(2, 1), _
        Array(3, 1), Array(4, 1), Array(5, 1), Array(6, 1), Array(7, 1)),
TrailingMinusNumbers _
        :=True
    ' Relative turned on here
    Selection.End(xlDown).Select
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "Total"
    ActiveCell.Offset(0, 4).Range("A1").Select
    ' Don't use AutoSum. Type this formula:
    Selection.FormulaR1C1 = "=SUM(R2C:R[-1]C)"
    Selection.AutoFill Destination:=ActiveCell.Range("A1:C1"), Type:= _
        xlFillDefault
    ActiveCell.Range("A1:C1").Select
    ' Relative turned off here
    ActiveCell.Rows("1:1").EntireRow.Select
    ActiveCell.Activate
```

```
    Selection.Font.Bold = True
    Cells.Select
    Selection.Columns.AutoFit
    Range("A1").Select
End Sub
```

This third macro will consistently work with any size dataset.

To see a demo of recording this macro, search for Excel VBA 1 at YouTube.

Three Tips When Using the Macro Recorder

You will rarely be able to record 100 percent of your macros and have them work. However, you will get much closer by using these three tips demonstrated in the following subsections.

Tip 1: Use Relative References Setting Usually Needs to Be On

Microsoft should have made this setting be the default. Unless you specifically need to move to Row 1 from the bottom of a dataset, you should usually leave the Use Relative References button in the Developer tab turned on.

Tip 2: Use Special Navigation Keys to Move to Bottom of a Dataset

If you are at the top of a dataset and need to move to the last cell with data, you can press Ctrl+down arrow or press the End key and then the down-arrow key.

Similarly, to move to the last column in the current row of the dataset, press Ctrl+right arrow or press End and then press the right-arrow key.

By using these navigation keys, you can jump to the end of the dataset, no matter how many rows or columns you have today.

Tip 3: Never Touch the AutoSum Icon While Recording a Macro

The macro recorder will not record the "essence" of the AutoSum button. Instead, it will hard-code the formula that resulted from pressing the AutoSum button. This formula does not work any time you have more or fewer records in the dataset.

Instead, type a formula with a single dollar sign, such as `=SUM(E$2:E10)`. When this is done, the macro recorder records the first `E$2` as a fixed reference and starts the `SUM` range directly below the Row 1 headings. Provided the active cell is E11, the macro recorder recognizes E10 as a relative reference pointing directly above the current cell.

Next Steps

Chapter 2, "This Sounds Like BASIC, So Why Doesn't It Look Familiar?" examines the three macros you recorded in this chapter to make more sense out of them. After you know how to decode the VBA code, it will feel natural to either correct the recorded code or simply write code from scratch. Hang on through one more chapter. You'll soon learn that VBA is the solution, and you'll be writing useful code that works consistently.

This page intentionally left blank

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X-Y-Z