Part 1

Recording Macros and Getting Started with VBA

Chapter 1

Recording and Running Macros in the Office Applications

In this chapter, you'll learn the easiest way to get started with Visual Basic for Applications (VBA): recording simple *macros* using the Macro Recorder that's built into the Office applications. Then you'll see how to run your macros to perform useful tasks.

I'll define the term *macro* in a moment. For now, just note that by recording macros, you can automate straightforward but tediously repetitive tasks and speed up your regular work. You can also use the Macro Recorder to create VBA code that performs the actions you need and then edit the code to customize it—adding flexibility and power. In fact, VBA is a real power-house if you know how to use it. This book shows you how to tap into that power.

IN THIS CHAPTER, YOU WILL LEARN TO DO THE FOLLOWING:

◆ Record a macro

◆ Assign a macro to a button or keyboard shortcut

◆ Run a macro

◆ Delete a macro

What Is VBA and What Can You Do with It?

Visual Basic for Applications is a programming language created by Microsoft that is built into applications. You use VBA to automate operations in all the main Office applications—Word, Excel, Outlook, Access, and PowerPoint.

Please don't be put off by the notion that you'll be *programming*. As you'll see shortly, working with VBA is nearly always quite easy. In fact, quite often you need not actually write any VBA yourself; you can merely *record* it—letting the Office application write all the VBA "code."

The phrase *automate operations in applications* is perhaps a bit abstract. So here are a few examples of how to use VBA to streamline tasks, avoid burdensome repetition, customize the applications' interfaces, and in general improve your efficiency.

◆ You can record a macro that automatically carries out a series of actions that you frequently perform. Let's say that you often edit Word documents written by a coworker, but she sets the zoom level to 100. You prefer to zoom to 150. All you need to automatically change the zoom level is this VBA code:

```
ActiveWindow.ActivePane.View.Zoom.Percentage = 150
```

◆ Don't worry! You do not even need to know the programming terms `ActiveWindow` or `View.Zoom`. Just turn on the Macro Recorder, manually click View, then click Zoom, and then set to 150 percent. The recorder will watch the steps you take and then write the necessary VBA code that can reproduce those steps. You write no code at all!

◆ You can write code that performs actions a certain number of times and makes decisions depending on the situation in which it is running. For example, you could write code that takes a series of actions on every presentation that's open in PowerPoint.

◆ You can use VBA to modify the look or behavior of the user interface. VBA can, for example, interact with the user by displaying *forms*, or custom dialog boxes, that enable the user to make choices and specify settings. You might display a set of formatting options—showing captioned controls such as check boxes and option buttons—that the user can select. Then when the user closes the dialog box, your macro takes appropriate actions based on the user's input.

◆ You can take actions via VBA that you can't do easily, or at all, when directly manipulating the user interface by hand. For example, when you're working interactively in most applications, you're limited to working with the active file—the active document in Word, the active workbook in Excel, and so on. By using VBA, you can access and manage files that aren't active.

◆ You can make one application control another application. For example, you can make Word place a table from a Word document into an Excel worksheet.

The Difference Between Visual Basic and Visual Basic for Applications

VBA is based on Visual Basic, a programming language derived from BASIC. *BASIC* stands for Beginner's All-Purpose Symbolic Instruction Code. BASIC is designed to be user-friendly because it employs recognizable English words (or variations on them) rather than the abstruse and incomprehensible programming terms found in languages like COBOL. In addition to its English-like diction, BASIC's designers endeavored to keep its punctuation and syntax as simple and familiar as possible as well.

Visual Basic is *visual* in that it offers efficient shortcuts such as drag-and-drop programming techniques and many graphical elements.

Visual Basic for Applications is a version of Visual Basic tailored to manage the Microsoft Office applications. The set of *objects* (features and behaviors) available in each application differs somewhat because no two applications share the exact same set of features and commands.

For example, some VBA objects available in Word are not available in Excel (and vice versa) because some of Word's features, like the Table of Contents generator, are not appropriate in Excel.

However, the large set of primary commands, fundamental structure, and core programming techniques of VBA in Word and VBA in Excel are the same. So you'll find that it's often quite easy to translate your knowledge of VBA in Word to VBA in Excel (or indeed in any VBA-enabled application).

For example, you'd use the `Save` method (a *method* is essentially an action that can be carried out) to save a file in Excel VBA, Word VBA, or PowerPoint VBA. What differs is the object involved. In Excel VBA, the command would be `ActiveWorkbook.Save`, whereas in Word VBA it would be `ActiveDocument.Save` and in PowerPoint it would be `ActivePresentation.Save`.

VBA always works within a host application (such as Access or Word). With the exception of some few stand-alone programs that are usually best created with Visual Basic .NET, a host application always needs to be open for VBA to run. This means that you can't build stand-alone applications with VBA the way you can with Visual Basic. If you like, you can *hide* the host application from the user so that all they see is the interface (typically user forms) that you give to your VBA procedures. By doing this, you can create the illusion of a stand-alone application—but VBA is rarely used for this purpose. If you want to write self-sufficient programs, investigate Visual Basic Express.

WHAT ARE VISUAL BASIC .NET AND VISUAL BASIC EXPRESS?

Visual Basic .NET (VB .NET) is just one version of Microsoft's long history of BASIC language implementations. BASIC contains a vast set of libraries of prewritten code that allow you to do pretty much anything that Windows is capable of doing. Although VB .NET is generally employed to write stand-alone applications, you can tap into its libraries from within a VBA macro should you need to. Just remember, each Office application has its own object library, but the .NET libraries themselves contain many additional capabilities (often to manipulate the Windows operating system). So, if you need a capability that you can't find within VBA or an Office application's object library, the resources of the entire .NET library are also available to you. Visual Basic Express is included in a free version of VB .NET. After you've worked with VBA in this book, you might want to download and explore Visual Studio Express for Desktop at:

`https://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx`

Visual Studio Express includes several languages, and VB is one of them.

Understanding Macro Basics

A *macro* is a sequence of commands you or a user can repeat at will. That's also exactly the definition of a *computer program*. Macros, however, are generally short programs—dedicated to a single task. Think of it like this: A normal computer program, such as Photoshop or Internet Explorer (IE), has many capabilities. It can prevent pop-up ads, block websites, display full-screen when you press F11, and so on. A macro is smaller, dedicated to accomplishing just one of these tasks, such as displaying full-screen. So a macro would likely add one new feature to the huge collection of features already built into an Office application.

In some applications, you can set a macro to run itself automatically. For instance, you might create a macro in Word to automate basic formatting tasks on a type of document you regularly receive incorrectly formatted. As you'll see in Chapter 6, "Working with Variables, Constants, and Enumerations," in a discussion of the AutoExec feature, you can specify that a macro run automatically upon opening a document of that type.

A macro is a type of *subroutine* (sometimes also called a *subprocedure*). Generally, people tend to use the shorter, more informal terms *sub, procedure,* and *routine*. In the Visual Basic Editor, each of your macros starts with the word Sub. Note that a macro is a single procedure, whereas a computer program like IE is a collection of many procedures.

In an Office application that supports the VBA Macro Recorder (such as Word or Excel), you can create macros in two ways:

◆ Turn on the Macro Recorder and just perform by hand the sequence of actions you want the macro to perform. Clicks, typing, dragging, dropping—whatever you do is recorded.

◆ Open the Visual Basic Editor and type the VBA commands into it to write a macro without first recording it.

There's also a useful hybrid approach that combines recording with editing. First record the sequence of actions, and then later, in the Visual Basic Editor, you can view and edit your macro. You could delete any unneeded commands or type in new commands. You could also use the Editor's Toolbox feature to drag and drop user-interface elements (such as message boxes and dialog boxes) into your macro so users can make decisions and choose options for how to run it. Macros are marvelously flexible, and the VBA Editor is famously powerful yet easy to use. This Editor is to programming what Word is to writing—a very mature, efficient, and well-designed toolbox.

Once you've created a macro, you specify how you want the user to trigger it. In most applications, you can assign a macro to the Ribbon, to the Quick Access Toolbar, or to a shortcut key combination. This makes it very easy to run the macro by merely clicking an icon or pressing a shortcut key (such as Alt+R). You can also optionally assign your macro to a Quick Access Toolbar button or keyboard shortcut when you first record the macro, via a dialog box that automatically appears when you begin a recording. You'll see how all this works shortly. It's simple. (To assign a macro to the Ribbon, first record it, and then right-click the Ribbon and choose Customize The Ribbon. Click the Choose Commands From drop-down box, and then click the Macros entry to display all your macros.)

Recording a Macro

The easiest way to create VBA code is to record a macro using the Macro Recorder. Only Word and Excel include a Macro Recorder.

You switch on the Macro Recorder, optionally assign a trigger that will later run the macro (a toolbar button or a shortcut key combination), perform the actions you want in the macro, and then switch off the Macro Recorder. As you perform the actions, the Macro Recorder translates them into commands—*code*—in the VBA programming language.

Once you finish recording the macro, you can view the code in the Visual Basic Editor and modify it if you want. If the code works perfectly as you recorded it, you never have to look at it—you can just run the macro at any time by clicking the toolbar button or key combination you assigned to the macro.

Displaying the Developer Tab on the Ribbon

Before going any further, ensure that the Developer (programmer) tab is visible in your Ribbon. This tab is your gateway to macros, VBA, and the VBA Editor. By default, Outlook doesn't display this tab. Word, Excel, and PowerPoint do. (Access doesn't even *have* this tab. Word, Excel, PowerPoint and Outlook do.) To add the Developer tab to your Ribbon, click the File tab and then click Options. Click Customize Ribbon. In the list box on the right, click Developer to select it. Click the OK button to close the Options dialog box. You'll now see a new Developer tab to the right of the default tabs.

In the following sections, you'll look at the stages involved in recording a macro. The process is easy, but you need to be familiar with some background if you haven't recorded macros previously. After the general explanations, you'll record example macros in Word and Excel. (Later in the book you'll examine and modify those macros, after you learn how to use the Visual Basic Editor. So don't delete them.)

Planning the Macro

Before you even start the Macro Recorder, it's sometimes a good idea to do a little planning. Think about what you will do in the macro. In most cases, you can just record a macro and not worry about the context. You can just record it with a document open. But in some situations you need to ensure that a special context is set up before you start the recording. For example, you might want to create a macro in Word that does some kind of editing, such as italicizing and underlining a word. To do this, you'll want to first have the blinking "insertion" cursor on a word *that's not italicized or underlined*. You don't want to record the actions of moving the insertion cursor to a particular word. That would make your macro specific to this document and this word in this document. You usually want a macro to work well with more than just one particular document.

Your macro is intended to just italicize and underline whatever word is currently under the blinking cursor in any document. Nevertheless, most simple macros can be recorded without any special planning. Just record whatever you want the macro to do.

PAUSING A MACRO

Word (but not Excel) lets you pause the Macro Recorder if you need to stop while recording to do something that you do not want to record. This capability allows you to deal with problems you hadn't anticipated when planning the macro—for example, having to open a document that should have been open before you started recording the macro.

Some macros should perform any necessary setup themselves. The setup will be part of the macro. In these cases, you should make sure the application is in the state that the macro expects before you start recording the macro. For example, if, to do its job, a macro needs a blank active

workbook in Excel, the macro itself should create that blank workbook rather than using which-ever workbook happens to be active at the time. This saves the user a step when the macro runs. To do this, start recording before launching a blank active workbook.

A WARNING ABOUT SECURITY

Macros are computer programs, albeit usually small. You can even tap into all the features in the Windows operating system itself from within a macro. The result is that viruses and other harm-ful code can be contained within macros (and such code can execute automatically merely by the user opening an infected document via the AutoExec feature discussed in Chapter 6 and via other techniques, such as employing the application's Startup folder). For example, a virus embedded in a macro could delete files on the hard drive if the user opened an infected Word document. This is obviously dangerous.

Office 2016 applications, not to mention the Windows operating systems, contain multiple layers of security to protect against such viruses and harmful code. Specific to macros is a macro "trust" technology that's built into Office applications. To see or modify these trust settings, open the Trust Center dialog box by clicking the Developer tab on the Ribbon, and then click the Macro Security icon (in the Code section of the Ribbon) in Word, Excel, Outlook, or PowerPoint. (Access, as is often the case, does things a bit differently than the other Office applications. Access has no Developer tab. To manage macro security in Access you click the File tab, click the Options link on the left side, click Trust Center, click the Trust Center Settings button, and then click Macro Settings.)

The main point here is that you might have to make some adjustments if you can't run macros or if you get mysterious error messages such as "The Macro Could Not Be Created" or "Access Is denied." If this happens, your first step should be to look at the Trust Center and choose Disable All Macros With Notification. This setting asks the user for permission to run macros. While you're working with macros in this book, you might want to just select Enable All Macros in the Trust Center. Then deselect this option before closing a document that you worked on in this book. The idea is that you can trust your own macros but you don't want to trust *all* macros from *all* documents you might get from outside sources.

If you are working on a document that you created and it contains macros that you wrote, you can trust that document and agree to activate the macros. However, if you open a document from someone else, you have to be careful.

Additional security issues can be solved by managing the various strata of security that now, out of necessity, are embedded within operating systems and applications. One way to deal with security issues is to explore security topics in Windows 7, 8, or 10 applications' Help features. You can also sometimes get good answers by posting questions in online user groups or searching expert websites such as *Wikipedia*. Also, you can find a good overview of Office 2016 security here:

```
http://technet.microsoft.com/en-us/library/ee857085(v=office.15).aspx
```

Chapter 19, "Securing Your Code with VBA's Security Features," covers Office 2016 security issues in depth.
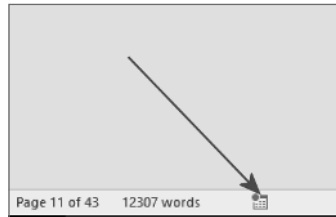
Starting the Macro Recorder

Start the Macro Recorder by clicking the Developer tab on the Ribbon and then clicking the Record Macro button. You can also click the Macro Record button on the left side of the status

bar at the bottom of the application. Note that if you don't see this Macro Record button, right-click the status bar and select Macro Recording from the context menu.

By clicking the Macro Record button, you don't have to open the Developer tab. The button looks like the one shown in Figure 1.1.

FIGURE 1.1
Find this Record
Macro button on
the status bar.

Page 11 of 43   12307 words

As soon as you start the Macro Recorder, the Record Macro dialog box opens. You see that this new macro has been given a default macro name (Macro1, Macro2, and so on). You can accept that default name or change it. There's also an optional description to fill in if you like.

To stop the Macro Recorder, you can click the Stop Recording button in the Developer tab. You can alternatively stop the recording by clicking the square button that appears during record-ing on the status bar, down on the bottom left of the application's window. Once the Recorder is stopped, the square button is replaced with the icon that you can click to start recording a new macro. In Word for the Mac, click the REC indicator rather than double-clicking it.

The appearance of the Record Macro dialog box varies somewhat in Word and Excel because the dialog box must offer suitable options to accommodate the varying capabilities particular to each application. In each case, you get to name the macro and add a description of it. In most cases, you can also specify where to save the macro—for example, Word offers two options. For *global* use (making the macro available to all Word documents), store it in the file named `Normal.dotm`. If it is merely to be used in the currently active document, choose to store it in a file with the document's name and the `.dotm` filename extension. An ordinary Word template has a `.dotx` filename extension, but macros are stored in a file with the file-name extension `.dotm`.

Excel allows you three options: to store macros in the current workbook, in a new workbook, or for use with *all* Excel workbooks, in the Personal Macro Workbook. That's the equivalent of Word's `Normal.dotm` file, and Excel's Personal Macro workbook is saved in a file named `Personal.xlsb`.

WHERE TO STORE MACROS IN POWERPOINT

You can't record macros in the 2016 version of PowerPoint, but you can create them by writing programming code using the Visual Basic Editor. Then you can store macros in the currently active presentation or in any other open presentation or template. PowerPoint also provides a global macro storage container (similar to Word's `Normal.dotm` file). In PowerPoint, choose the All Open Presentations option in the Macro list box, which is found by clicking the Macros icon in the Code section of the Ribbon's Developer tab.

The Record Macro dialog box also lets you specify how you want the macro triggered. Word displays buttons you can click to either open a dialog for entering a shortcut key combination

or open the Word Options dialog where you can create a button for this macro that will appear on the Quick Access Toolbar. Excel limits you to Ctrl+ shortcut key combinations as a way of launching macros, so there is no button to display a full keyboard shortcut dialog like the one in Word. Excel has only a small text box where you can enter the key that will be paired with Ctrl as the shortcut.

Most of the Microsoft applications that host VBA have the Developer tab from which you control macro recording, launch the Visual Basic Editor, and otherwise manage macros. Access, however, groups several of its macro-related tools in a Database Tools tab (which is visible by default) and also has a Macro option on its Create tab.

Figure 1.2 shows the Record Macro dialog box for Word with a custom name and description entered. Figure 1.3 shows Word's version of the Developer tab on the Ribbon.

FIGURE 1.2
In the Record Macro dialog box, enter a name for the macro you're about to record. You can type a concise description in the Description box. This is the Record Macro dialog box for Word.
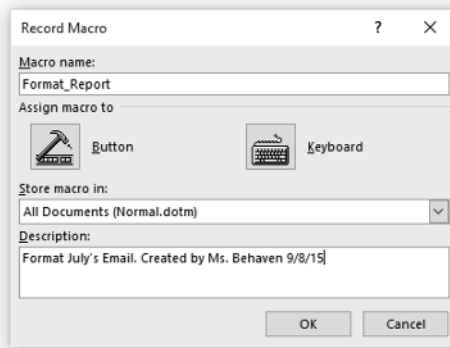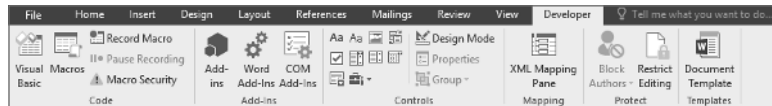

FIGURE 1.3
You can use the Developer tab on the Ribbon to work with macros.

Here's what the primary Visual Basic features on the Ribbon's Developer tab (or Access's Database Tools tab) do:

Run Macro button　Only Access has this Ribbon button. It displays a Run Macro dialog box, in which you can choose the macro to execute (run). Many aspects of VBA in Access are unique only to Access, and Chapter 28, "Understanding the Access Object Model and Key Objects," covers them in depth.

Record Macro button　Displays the Record Macro dialog box in Word or Excel.

Macro Security button　Displays the Trust Center Macro Settings dialog. You'll examine this feature in detail in Chapter 19. This button allows you to specify whether and how you want macros enabled.

Visual Basic button　Switches to the Visual Basic Editor. You'll begin working in the Visual Basic Editor in Chapter 2, "Getting Started with the Visual Basic Editor" (and you'll spend most of the rest of the book employing it).

Macros button   Opens the classic Macros dialog from which you can run, step into (start the Visual Basic Editor in Break mode; more about this in Chapter 3, "Editing Recorded Macros"), edit, create, delete, or open the macro project organizer dialog. (Not all of these options are available in all applications. For example, PowerPoint has no organizer.) Word and Excel have a similar Macros button in the Ribbon's View tab. This button has the ability to open the Macros dialog but can also start recording a macro. Note that Break mode is also referred to as Step mode.

Add-Ins   This is where you can access templates, styles, and specialized code libraries.

Controls   A set of control buttons that, when clicked, insert user-interface components—such as a drop-down list box—into an open document. Similar components can also be added to macros that you create in the VBA Editor. Chapter 14, "Creating Simple Custom Dialog Boxes," and Chapter 15, "Creating Complex Forms," explore this user-interface topic.

Design Mode button   Toggles between Design mode and Regular mode. When in Design mode, you can add or edit embedded controls in documents. In Regular mode, you can interact normally with controls (controls can accept information from the user via typing or mouse clicks).

Properties button   This button is enabled only if you're in Design mode. It allows you to edit the properties of the document (such as removing personal information).

XML button   This section of the Developer tab is explored in Chapters 21 through 24.

Restrict Editing button   Allows you to specify what formatting or editing others are allowed to perform.

Document Template button   Here you can see or modify the current template, or manage add-ins or the global template.

THE EMERGENCE OF XML

XML has become an industry standard for storing and transmitting data. With Office 2007, the Office applications' documents began to employ XML extensively. This switch to XML is the primary reason that documents created in versions of Office 2007, 2010, 2013, and 2016 are not compatible with earlier versions of Office, such as Office 2003 documents. Therefore, you must *convert* old Office documents to the newer Office formats. People still using older versions of Office must install the Microsoft Office Compatibility Pack for Word, Excel, and PowerPoint File Formats. Note that starting with Word 2010, document files are saved with a `.docx` filename extension, the `x` reflecting the underlying XML format on which Office now rests.

Naming the Macro

Next, enter a name for the new macro in the Macro Name text box in the Record Macro dialog box. The name must comply with the following conventions:

◆   It must start with a letter; after that, it can contain both letters and numbers.

◆   It can be up to 80 characters long.

◆ It can contain underscores, which are useful for separating words, such as File_Save.

◆ It cannot contain spaces, punctuation, or special characters, such as ! or *.

NAME AND DESCRIBE YOUR MACROS

Some people insist that to properly manage your set of macros, you must follow some clerical procedures that involve giving your macros descriptive names and also typing in a narrative description of each macro's purpose. They claim that if you create many macros, you should organize them carefully. Recording macros is so easy; you can create code so quickly that you can end up with a *pile* of macros—as Southerners say—making it easy to get confused about which macro does what.

You may be tempted not to assign a macro description when you're in a hurry or when you're playing with different ways to approach a problem and you're not sure which (if any) of your test macros you'll keep.

For simple, obvious code, perhaps using the default names (such as Macro12, Macro13) and omitting a description isn't a problem. Also, if you find it easy to read and understand VBA code, you can usually just look at a macro and see what it does.

Even so, for more complex macros, and for people who find code hard to read—go ahead and enter a few notes for each macro that you record. Otherwise, you can end up with that pile of recorded macros that have the cryptic default names and no descriptions. To figure out what each macro does and which ones you no longer use and can safely delete, you'll have to plow through the code—and a recorded macro's code can be surprisingly long, even if the macro does nothing more than adjust a few options in a couple of dialog boxes.

You might also want to employ a macro-naming convention to indicate which macros are simply tests that you can delete without remorse. Start the name with a word like Temp, and then add numeric values sequentially to keep track of the versions—for example, Scratch (Scratch01, Scratch02, and so on) and Temp (Temp01, Temp02, and so on).

Each new macro you record is by default placed at the bottom of the set of macros in the VBA Editor. You can, however, always open the Visual Basic Editor and rename or add a description any time you want because macros are fully editable.

Personally, I like to put a little descriptive note inside more complicated macros' code, right at the top, under the Sub line. It looks like this:

```
Sub AltH()
' Applies Heading 1 style
    Selection.Style = ActiveDocument.Styles("Title1")
End Sub
```

Any text following a single-quote symbol ' on a line of code is ignored by VBA. The single quote indicates that what follows is a *comment* to assist the programmer in understanding the code rather than actual code that VBA should try to execute. (VBA would not know what to make of the words Applies Heading 1 style. They are not part of VBA's vocabulary.)

Note that if you type a description in the Description field of the Record Macro dialog when you first start recording, that comment is automatically inserted into your code—complete with the single-quote symbol.

Also, my preferred way to name any macros that are triggered by keyboard shortcuts is to use the name of the keyboard shortcut itself. Therefore, Sub AltH tells me that this macro is triggered by the Alt+H keyboard shortcut.

But whatever system you adopt, it's generally better to err on the side of greater description or commenting within the code rather than too little. It only takes a moment to provide an expressive, meaningful name and a clear description of the purpose of the macro.

Invalid Macro Names

Word and Excel raise objections to invalid macro names. If you enter a prohibited macro name in the Record Macro dialog box, these applications let you know—in their own way—as soon as you click the OK button. Word displays a brief, rather cursory message, while Excel gives more helpful info. Figure 1.4 shows how these applications respond to an invalid macro name once it's entered.

Figure 1.4
The dialog boxes supplied by Word and Excel show invalid macro names.
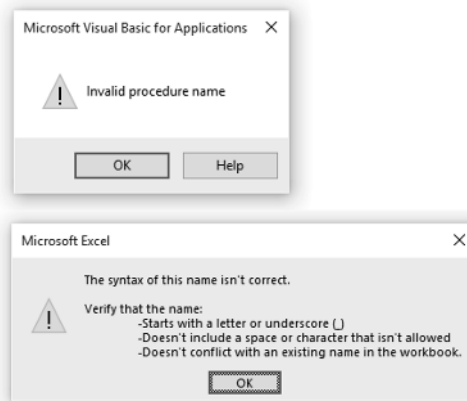
Describing Your Macros

Type a description for the macro in the Description text box. Recall that this description is to help you (and anyone you share the macro with) identify the macro and understand when to use it. If the macro runs successfully only under particular conditions, you can note them briefly in the Description text box. For example, if the user must make a selection in the document before running the macro in Word, mention that.

You now need to choose where to store the macro. Your choices with Word and Excel are as follows:

Word   Recall that in Word, if you want to restrict availability of the macro to just the current template (`.dotm` file) or document (`.docm` file), choose that template or document from the Store Macro In drop-down list in the Record Macro dialog box shown in Figure 1.2. If you want the macro to be available no matter which template you're working in, make sure the default setting—All Documents (`Normal.dotm`)—appears in the Store Macro In combo box. (If you're not clear on what Word's templates are and what they do, see the sidebar "Understanding Word's `Normal.dotm`, Templates, and Documents" later in this chapter).

Excel   In Excel, you can choose to store the macro in This Workbook (the active workbook), a new workbook, or Personal Macro Workbook. The Personal Macro Workbook is a special workbook named `Personal.xlsb`. Excel creates this Personal Macro Workbook the first time you choose to store a macro in the Personal Macro Workbook. By keeping your macros and other customizations in the Personal Macro Workbook, you can make them available to any of your procedures. Recall that the Personal Macro Workbook is similar to Word's global macros storage file named `Normal.dotm`. If you choose New Workbook, Excel creates a new workbook for you and creates the macro in it.

STORING YOUR MACROS

Word and Excel automatically store recorded macros in a default location in the specified document, template, workbook, or presentation:

Word   Word stores each recorded macro in a *module* named `NewMacros` in the selected template or document, so you'll always know where to find a macro after you've recorded it. This can be a bit confusing because there can be multiple NewMacros folders visible in the Project Explorer pane in the Visual Basic Editor. (This happens because there can be more than one project open—such as several documents open simultaneously, each with its own NewMacros folder holding the macros embedded within each document.) Think of NewMacros as merely a holding area for macros—until you move them to another module with a more descriptive name. (Of course, if you create only a handful of macros, you don't need to go to the trouble of creating various special modules to subdivide them into categories. You can just leave everything in a `NewMacros` module. As always, how clerical you need to be depends on how organized your mind and memory are—and also on the size of the collection with which you're dealing.)

If a `NewMacros` module doesn't yet exist, the Macro Recorder creates it. Because it receives each macro recorded into its document or template, a `NewMacros` module can soon grow large if you record many macros. The `NewMacros` module in the default global template, `Normal.dotm`, is especially likely to grow bloated, because it receives each macro you record unless you specify another document or template prior to recording. Some people like to clear out the `NewMacros` module from time to time, putting recorded macros they want to keep into other modules and disposing of any useless or temp recorded macros. I don't have *that* many macros, so I find no problem simply leaving them within the `NewMacros` module.

Excel   Excel stores each recorded macro for any given session in a new module named `Module` *n*, where *n* is the lowest unused number in ascending sequence (`Module1`, `Module2`,

and so on). Any macros you create in the next session go into a new module with the next available number. If you record macros frequently with Excel, you'll most likely need to consolidate (copy and paste) the macros you want to keep so that they're not scattered across many modules.

Understanding Word's *Normal.dotm*, Templates, and Documents

Word since version 2007 stores data differently than earlier versions of Word. For one thing, in Word 2003 you could create custom menus and toolbars that you stored in templates. Later versions of Word do not permit menus, nor do they permit any toolbars other than the Quick Access Toolbar. What's more, customizing that toolbar has a *global* impact. In other words, any modifications you make to the Quick Access Toolbar will be visible in all Word documents, no matter which template(s) is currently active.

The versions of Word since 2007 feature three kinds of templates:

◆ Legacy templates from Word 2003 and earlier versions. These have a .dot filename extension. If you are working with one of these templates, the phrase (Compatibility Mode) appears on the Word title bar.

◆ Templates that contain no macros use a .dotx filename extension. You can save macros in a document that employs a .dotx template, but the macro will not be saved within the template. This type of template made its first appearance with Word 2010.

◆ Templates with a .dotm filename extension contain macros. Recall that because macros written by malicious people can do damage just like a virus can, Word segregates macros into this special kind of template with a .dotm filename extension. A .dotm template can do anything that a .dotx template can do, but the .dotm template features the additional capability of hosting macros.

Word has a four-layer architecture. Starting from the bottom, these layers are the application itself, the global template (Normal.dotm), the active document's template, and finally, the active document itself (the text and formatting). Each of the four layers can affect how Word appears and how it behaves, but all four layers are not necessarily active at any given time.

The bottom layer, which is always active, is the Word application itself. This layer contains all the Word objects and built-in commands, such as Open. Also always active are objects such as Word's Quick Access Toolbar, the Ribbon, and so on. This layer is the most difficult to picture because usually you don't see it directly. Normal.dotm, the global template, forms the second layer and is also always active.

When you start Word, it loads Normal.dotm automatically, and Normal.dotm stays loaded until you exit Word. (There's a special switch you can use—winword /n—to prevent the macros in Normal.dotm from being active if you need to troubleshoot it. Press the Start key [the Windows key] in Windows 8 and 10 [the Start key in earlier versions of Windows], and then type **Run** to launch Word in this special way.)

Normal.dotm contains styles (such as the default paragraph style), AutoText entries, formatted AutoCorrect entries, and customizations. These customizations also show up in the other layers unless specifically excluded.

Continues

Continued

Default new blank documents (such as the document that Word normally creates when you start it and any document you create by clicking Ctrl+N or by clicking the Ribbon's File tab and then choosing New and Blank Document) are based on `Normal.dotm`. When you're working in a default blank document, you see the Word interface as it is specified in `Normal.dotm`.

The currently active template sits on top of the Word application and `Normal.dotm`. This template can contain styles, macro modules (if it is a macro-enabled `.dotm` file type), and settings for the template, along with any boilerplate text needed for this particular type of document. This is the third layer, but it is used only if the current document (or *active document*) is attached to a template other than `Normal.dotm`.

On top of the current template sits the current document, which contains the text and graphics in the document, its formatting, and its layout. Documents can also contain macro modules specific to it, along with custom keyboard shortcuts, so the document itself can act as a fourth layer. This layer is always present when a document is open, but it has no effect on Word's interface or behavior unless the document contains its own, local customizations.

Because these layers might contain conflicting information (such as two different font styles with the same name), there has to be an order of precedence to specify which layer "wins" in any such conflict. Customized settings work from the top layer downward, so customized settings in the active document take precedence over those in the active template. Likewise, any settings in the active template take precedence over any global templates (templates that automatically apply to all Word documents) or add-ins other than `Normal.dotm`. Customized settings in those global templates or add-ins take precedence over those in `Normal.dotm`.

As another example, say you have the key combination Ctrl+Shift+K assigned to different actions in `Normal.dotm`, in a loaded global template, in a document's template, and in the document itself. When you press that key combination, only the procedure assigned in the document runs because that is the topmost layer. If you remove the key-combination assignment from the document, the template then becomes the topmost layer containing a definition of this key combination, so the procedure assigned in the template runs. If you remove the key combination from the template as well, the procedure in the loaded global template runs. Finally, if you remove that template's key combination too, the procedure in `Normal.dotm` runs. It is the lowest layer.

Choosing How to Run a New Macro

Continuing our exploration of the Record Macro dialog box shown in Figure 1.2, at this point, after you've named the macro, typed a description, and chosen where to store it, it's time to choose how to trigger the macro. In other words, how do you want the user to *run* the macro: via a shortcut key or via a Quick Access Toolbar button? Good typists generally prefer shortcut keys (they don't have to take their hand off the keyboard to reach for the mouse), but buttons provide at least a visual hint of the macro's purpose, plus hovering your mouse on the button also displays the name of the macro.

Shortcut keys and buttons are handy for people who record a moderate number of macros and don't organize them in complex ways—moving them from one module to another. If you create a great number of macros and feel the need to move them into other modules, assigning a shortcut key or button prior to recording becomes less useful. This is because moving a macro from one module to another disconnects any way you've assigned for running the macro.

This limitation means that it makes sense to assign a way of running a macro—prior to recording—only if you're planning to use the macro in its recorded form (as opposed to, say, using part of it to create another macro) *and* from its module location. If you plan to move the macro or rename it, don't assign a way of running it now. Instead, wait until the macro is in its final form and location, and then assign the means of running it. See "Specifying How to Trigger an Existing Macro," later in this chapter, for details on how to do this.

Personally, I don't have more than a few dozen macros that I use all the time, so I avoid the complications described in the previous paragraph and this chapter's sidebar titled "Manage Your Macros with Modules." Instead, I just add shortcut keys when I first create the macro and leave them all in a single version of Normal.dotm template. However, if you face more complicated situations—such as managing a big set of macros for a company—you might want to manage your macros with modules.

Manage Your Macros with Modules

By moving your recorded macros into different modules, you can group related macros thereby making it easier to compare the code, edit them, and distribute them.

To assign a way to run the macro, follow the instructions in the next sections.

Remember that you don't have to assign a button or keyboard shortcut prior to recording a macro. You can do it later, or at any time. In Word and Excel, you can use File ➢ Options ➢ Quick Access Toolbar dialog box to assign a button on the Quick Access Toolbar to a macro. You can also use File ➢ Options ➢ Customize Ribbon dialog box to assign a shortcut key to a macro (click the Customize button at the bottom of this dialog box).

Excel limits you to Ctrl+ or Ctrl+Shift key combinations. PowerPoint and Access do not permit you to assign keyboard shortcuts to macros.

Running a Macro from the Ribbon

Although it's not available in the Record Macro dialog box, you can add a macro to the Ribbon, like this:

1. Right-click anywhere on the Ribbon.

2. Click Customize The Ribbon on the menu. The Word Options dialog box appears.

3. In the Choose Commands From drop-down list, select Macros.

4. Click a macro's name to select it in the list.

5. Click an existing tab in the list of tabs in the right dialog box where you want to locate your macro.

6. Then click the New Group button and specify the name of your custom group.

7. Click the Rename button to give your new group a name.

8. Click OK to close the Rename dialog box.

9. Click the Add button to add your macro.

10. Click the Rename button to give your macro an easily understood name and optionally an icon.

11. Click OK to close the Rename dialog box.

12. Click OK to close the Word Options dialog box.

Running a Macro from the Quick Access Toolbar

Here's how to use the Word Options dialog box to assign a macro to a button on the Quick Access Toolbar:

1. Right-click anywhere on the Quick Access Toolbar (it's the set of icons in the upper-left corner, normally above the Ribbon), and a menu will appear. (This toolbar will be just below the Ribbon if you've previously selected the Show Quick Access Toolbar Below The Ribbon option from this menu.)

2. Click Customize Quick Access Toolbar on the menu. The Word Options dialog box appears.

3. In the Choose Commands From drop-down list, select Macros.

4. Click a macro's name to select it in the list, as shown in Figure 1.5.

FIGURE 1.5
Choose a way to run the macro in Word's Options dialog box.
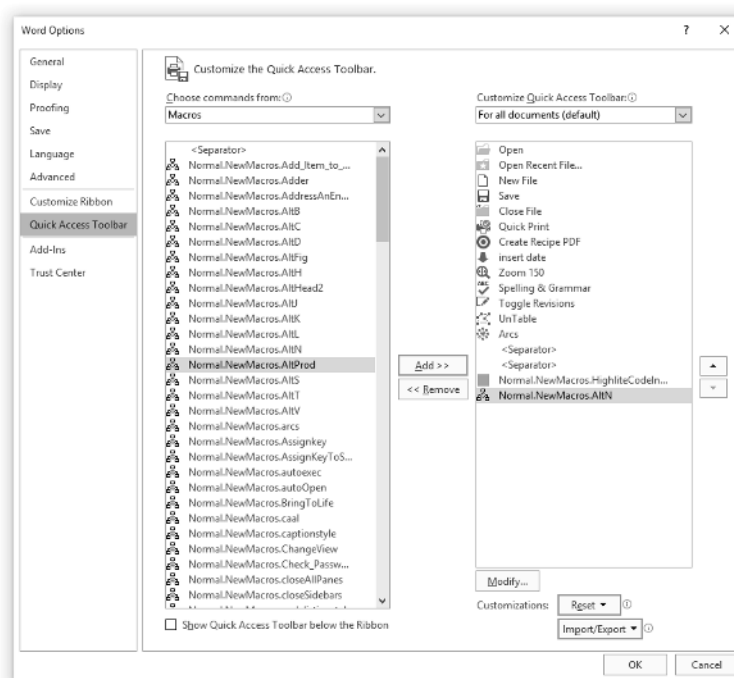
5. Click the Add button to insert this macro's name in the Customize Quick Access Toolbar list, as shown in Figure 1.5.

6. Word adds a button to the toolbar for the macro, giving it the macro's fully qualified name (its location plus its name), such as `Normal.NewMacros.CreateDailyReport`. This name consists of the name of the template or document in which the macro is stored, the name of the module that contains the macro, and the macro's name, respectively. You don't need all this information displayed when you hover your mouse pointer over the button.

7. So rename the button or menu item: Click the Modify button at the bottom of the Customize Quick Access Toolbar list (see Figure 1.5). Whatever macro is highlighted (currently selected) in the list of toolbar items will be the one you're modifying.

MACRO BUTTON LABELS NEED NOT MATCH THEIR MACROS' OFFICIAL NAMES

Notice that a macro's button name (displayed as its tooltip caption when you hover your mouse over it) doesn't have to bear any relation to the macro's actual name as it appears in the Visual Basic Editor or the Macro dialog.

---

8. While you're modifying the macro's name, you might also want to choose a button icon that visually cues you about the macro's purpose. To do that, just double-click whatever icon you want to use, and then click OK (see Figure 1.6).

FIGURE 1.6
Word gives the menu item or toolbar button the full name of the macro. Use this Modify Button dialog to change the name to something shorter and better.
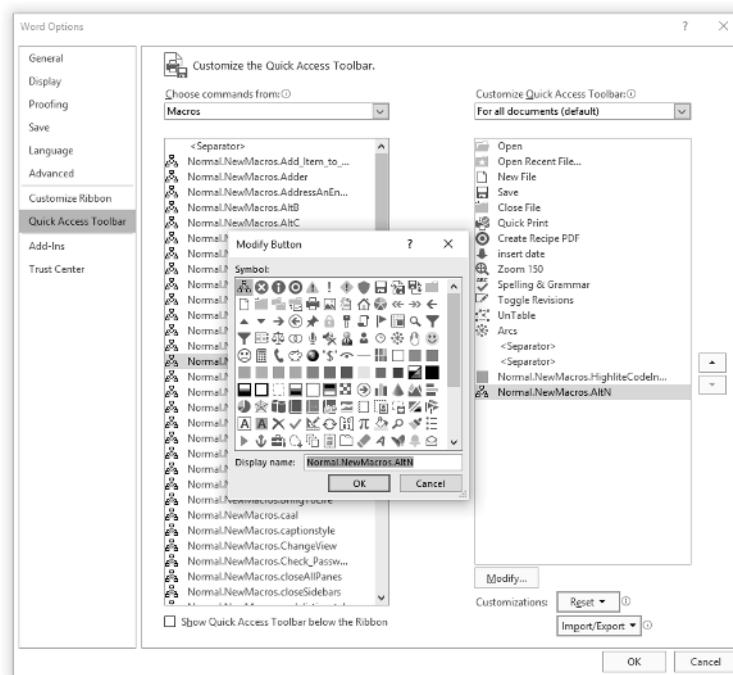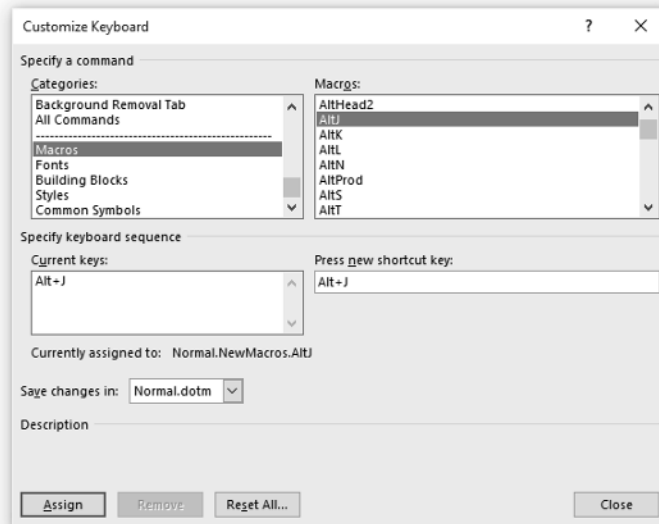
RUNNING A MACRO VIA A SHORTCUT KEY COMBINATION

To assign the macro to a key combination, follow these steps:

1. Right-click the Ribbon and choose Customize The Ribbon from the menu that appears. This opens the Word Options dialog.

2. Click the Customize button next to Keyboard Shortcuts in the bottom left of the Word Options dialog box.

3. Scroll down the Categories list box until you see Macros, and then click Macros to select it.

4. Click to select the name of the macro to which you want to assign a shortcut key combination.

5. Check the Current Keys list box to see if a key combination is already assigned. If it is, you can press the Backspace key to clear the key combination if you want, or you can employ multiple key combinations to launch the macro.

6. In the Press New Shortcut Key field, type the key combination you want to use to trigger the macro (see Figure 1.7).

7. Check to see if this key combination is already used for another purpose. If so, you can reassign it, or you can choose a different combination by pressing the Backspace key in the Press New Shortcut Key field.

8. Be sure to click the Assign button when you're finished. Just closing this dialog does *not* assign the key combination.

FIGURE 1.7
Set a shortcut key combination for the macro in the Customize Keyboard dialog box.

You Can Postpone Assigning a Shortcut Key Combination

Remember that, as with the other ways of running a macro, you can assign a key combination to run a macro either at the time you record the macro or at any point after you finish recording it. If you intend to move the macro from the `NewMacros` module to another module, remember that you need not assign the key combination until the macro has reached its ultimate destination.

A key combination in Word can be any of the following:

◆ Alt plus either a function key or a regular key not used as a menu-access key.

◆ Ctrl plus a function key or a regular key.

◆ Shift plus a function key.

◆ Ctrl+Alt, Ctrl+Shift, Alt+Shift, or even Ctrl+Alt+Shift plus a regular key or function key. Pressing Ctrl+Alt+Shift and another key tends to be too awkward for practical use.

Specify Two-Step Key Combinations

You can set up shortcut keys that have two steps—for example, Ctrl+Alt+F, 1 and Ctrl+Alt+F, 2—by pressing the second key (in this case, the 1 or the 2) after pressing the key combination. However, these shortcuts are probably more trouble than they're worth.

Running a Macro the Old-Fashioned Way

A clumsy, rarely used way to run a macro is to click the Developer tab in the Ribbon. To see how this works, follow these steps:

1. Click the Macros icon.

2. Click the name of the macro in a displayed list.

3. Finally, click the Run button.

By the way, you can also run a macro from within the Visual Basic Editor by pressing F5. This is how you test macros while you're editing them. The macro in which the insertion cursor is located (the blinking vertical line) is the one that will execute when you press F5 in the Editor.

Assigning a Way to Run a Macro in Excel

When you're recording a macro, Excel allows you to assign only a Ctrl shortcut key, not a button, to run it. If you want to assign a Quick Access Toolbar button to the macro, you need to do so *after* recording the macro (using the Customize feature as described shortly).

To assign a Ctrl shortcut key to run the macro you're recording, follow these steps:

1. Start recording the macro to display the Record Macro dialog box, and then click the Shortcut Key Ctrl+text box to display the blinking insertion cursor. Press the shortcut key you want to use. (Press the Shift key at the same time if you want to include Shift in the shortcut.)

2. In the Store Macro In drop-down list, specify where you want the Macro Recorder to store the macro. Your choices are as follows:

◆ *This Workbook* stores the macro in the active workbook. This option is useful for macros that belong to a particular workbook and do not need to be used elsewhere.

◆ *New Workbook* causes Excel to create a new workbook for you and store the macro in it. This option is useful for experimental macros that you'll need to edit before unleashing them on actual work.

◆ *Personal Macro Workbook* stores the macro in the Personal Macro Workbook, a special workbook named `PERSONAL.XLSB`. By keeping your macros and other customizations in the Personal Macro Workbook, you can make them available to any of your procedures—in that way, the Personal Macro Workbook is similar to Word's `Normal .dotm`. If the Personal Macro Workbook does not exist yet, the Macro Recorder creates it automatically.

3. Click the OK button to start recording the macro.

Assigning a Way to Run a Macro in PowerPoint

PowerPoint does not let you record macros, but you can assign a way to run macros written in the Visual Basic Editor, as discussed in the section "Specifying How to Trigger an Existing Macro" later in this chapter.

Assigning a Way to Run a Macro in Outlook

Outlook doesn't let you record macros, and by default macros are disabled. To enable macros in Outlook, click the Developer tab on the Ribbon, and then click the Macro Security icon (it's on the left in the Code section of the Ribbon). The Trust Center dialog box opens. Click the Notification For All Macros option or the Enable All Macros option. To see how to assign a way to run macros, see the section "Specifying How to Trigger an Existing Macro" later in this chapter.

Recording the Actions in a Macro

When you close the Record Macro dialog box in Word or Excel, the Macro Recorder begins recording the macro. The Macro Recorder displays the Stop Recording icon (a dark gray square) in the status bar at the bottom left of the screen (and a Stop Recording button in the Developer tab on the Ribbon). In addition, Word displays a small symbol of a cassette tape with the mouse pointer (these tapes were used in the old days, prior to the invention of the CD).

Now you perform the sequence of actions you want to record. What exactly you can do varies from application to application, but in general, you can use the mouse to select items, make choices in dialog boxes, and select defined items in documents (such as cells in spread-sheets). You'll find a number of things that you can't do with the mouse, such as select items within a document window in Word. To select items in a Word document window, you have to use the keyboard (Shift+arrow keys, for example).

THE MACRO RECORDER RECORDS EVERYTHING—THE COMPLETE CURRENT STATUS

When you make choices in a dialog box and click the OK button, the Macro Recorder records the current settings for all the options on that page of the dialog box. So, for example, when you change the left indentation of a paragraph in the Paragraph dialog box in Word, the Macro Recorder records *all the other settings* on the Indents And Spacing page as well (Alignment, Before and After spacing, and so forth).

In Word, if you need to perform any actions that you don't want recorded, pause the Macro Recorder by clicking the Pause Recording button on the Ribbon. The button changes to Resume Recording. Click the button again to start recording again.

To stop recording, click either the Stop Recording button on the Ribbon or the other one on the status bar.

The Macro Recorder has now recorded your macro and optionally assigned it to a key combination or button.

Running a Macro

To run a macro you've recorded, you can use four methods to run it within the application:

◆ If you assigned a Quick Access Toolbar button, use that.

◆ If you added your macro to the Ribbon, you can use that.

◆ If you specified a shortcut-key-combination macro, use it.

◆ A less convenient approach is to press Alt+F8 to display the Macros dialog box, select the macro, and then click the Run button. (Alternatively, you could double-click the macro name in the list box.)

RUNNING IN THE EDITOR

You can also run a macro from the Visual Basic Editor, which is useful when you're working in the Editor. Just press F5.

The macro runs, performing the actions in the sequence in which you recorded them. For example, suppose you create a macro in Excel that selects cell A2 in the current worksheet, bold-faces that cell, enters the text **Yearly Sales**, selects cell B2, and enters the number **100000** in it. The Macro Recorder recognizes and saves those five actions. VBA then performs all five actions, step-by-step, each time you run the macro—albeit quite rapidly.

How to Stop an Executing Macro

To stop a running macro, press Ctrl+Break (Break is usually the unshifted Pause key on the keyboard). VBA stops running the code and displays a dialog box telling you that code execution has been interrupted. Click the End button to dismiss this dialog box.

Some applications (such as Word) let you undo most actions executed via VBA after the macro stops running (by pressing Ctrl+Z or clicking the Undo button on the Quick Access Toolbar, undoing one command at a time); other applications do not.

Macro Errors Are Often Caused by Incorrect Contexts

If running the macro results in an error, often this means that the macro is trying to do something to a file or an object that isn't available. For example, if you record a macro in Excel that works on the active workbook, the macro causes an error if you run it when no workbook is open (therefore, there is no such thing as an active workbook). Likewise, if you write a macro in PowerPoint that works with the third shape on the active slide, that macro fails if you run it on a slide that has no third shape. To get the macro to run properly, re-create the conditions it needs and then try it again.

Recording a Sample Word Macro

In this section, you'll record a sample macro in Word. This macro selects the current word, cuts it, moves the insertion point one word to the right, and pastes the word back in. This is a straightforward sequence of actions that you'll work with later in the book, and view and edit in the Visual Basic Editor.

Follow these steps to record the macro:

1. Create a new document by pressing Ctrl+N.

2. Start the Macro Recorder by clicking the Developer tab on the Ribbon, and then clicking the Record Macro button. Alternatively you could click the Macro Record button on the status bar at the bottom of the application. (With this approach, you don't have to open the Developer tab. Just click the button on the status bar.)

3. In the Macro Name text box, enter `Transpose_Word_Right`.

4. In the Store Macro In drop-down list, make sure All Documents (`Normal.dotm`) is selected, unless you want to assign the macro to a different template. (This and future examples in this book assume this macro is located in `Normal.dotm`, so do store it there.)

5. In the Description box, enter a description for the macro (see Figure 1.8). Be fairly explicit and enter a description such as **Transposes the current word with the word to its right. Created 9/14/15 by Toro Selest-Gomes**.

FIGURE 1.8
Creating the sample macro in Word
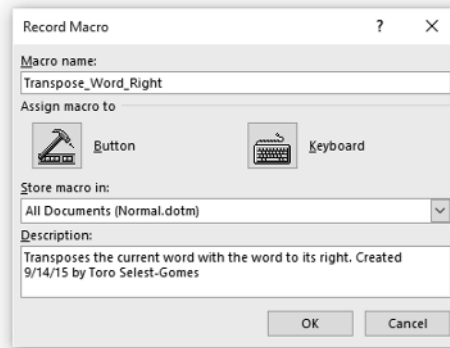
6. Assign a method of running the macro, as described in the previous section, if you like. Create a toolbar button or assign a keyboard shortcut. (The method or methods you choose is strictly a matter of personal preference.) If you'll need to move the macro to a different module (or a different template or document) later, don't assign a method of running the macro at this point.

7. Click the OK button to dismiss the Word Options dialog box or the Customize Keyboard dialog box (or just click the OK button to dismiss the Record Macro dialog box if you chose not to assign a way of running the macro). Now you're ready to record the macro. The Stop Recording option appears on the Ribbon and on the status bar, and the mouse pointer has a cassette-tape icon attached to it.
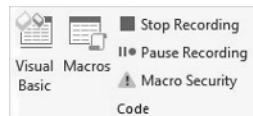
8. As a quick demonstration of how you can pause recording, click the Pause Recording button on the Ribbon. The cassette-tape icon disappears from the mouse pointer, and the

Pause Recording button changes into a Resume Recording button. Enter a line of text in the document: **The quick brown fox jumped over the lazy dog.** Position the insertion point anywhere in the word *quick*, and then click the Resume Recording button on the Ribbon to reactivate the macro recorder.

9. Record the actions for the macro as follows:

   a. Use Word's extend selection feature to select the word *quick* by pressing the F8 key twice.

   b. Press the Esc key to cancel Extend mode.

   c. Press Shift+Delete to cut the selected word to the Clipboard.

   d. The insertion point is now at the beginning of the word *brown*. Press Ctrl+right arrow to move the insertion point right by one word so that it's at the beginning of the word *dog*.

   e. Press Shift+Insert or Ctrl+V to paste in the cut word from the Clipboard.

   f. Press Ctrl+left arrow to move the insertion point one word to the left. This restores the cursor to its original position.

10. Click the Stop Recording button on the Ribbon or status bar. Your sentence now reads, "The brown quick fox jumped over the lazy dog."

FINDING BUILT-IN KEYBOARD SHORTCUTS

You can find a complete list of the built-in keyboard shortcuts (such as Ctrl+left arrow) by searching an application's Help system for "Keyboard Shortcuts." (In Office 2016 you can just use the Tell Me What You Want To Do feature on the Ribbon.)

You can now run this macro by using the toolbar button or keyboard shortcut that you assigned (if you chose to assign one). Alternatively, click the Macros button in the Developer tab and run the macro from the Macros dialog box.

At this point, Word has stored the macro in `Normal.dot`. If you don't save macros until you exit Word (or until an automated backup takes place), Word doesn't, by default, prompt you to save them then. It just saves them automatically. But it's best to click the Save button in the File tab to store Normal now. That way, if Word or Windows crashes, you will avoid losing the macro.

YOU CAN FORCE WORD TO PROMPT YOU TO SAVE THE NORMAL TEMPLATE

Word, by default, automatically saves new macros added to the Normal template. But if you prefer to have Word prompt you to save any changes to the Normal template, choose Options on the File tab, click the Advanced button, and then scroll down until you see the section of Save options. Select the Prompt Before Saving Normal Template check box, and then click the OK button.

Recording a Sample Excel Macro

In the following sections, you'll record a sample Excel macro. This macro creates a new workbook, enters a sequence of months into it, and then saves it. You'll work with this macro again in Chapter 3, so don't delete it.

Creating a Personal Macro Workbook

If you don't already have a Personal Macro Workbook in Excel, you'll need to create one before creating this procedure. (If you do have a Personal Macro Workbook, skip to the next section.) Follow these steps:

1. Click the Developer tab in the Ribbon, and then click the Record Macro button on the Ribbon (or just click the Record Macro button on the status bar) to display the Record Macro dialog box.

2. Accept the default name for the macro because you'll be deleting it momentarily.

3. In the Store Macro In drop-down list, choose Personal Macro Workbook.

4. Click the OK button to close the Record Macro dialog box and start recording the macro.

5. Type a single character in whichever cell is active, and press the Enter key.

6. Click the Stop Recording button on the Ribbon or status bar to stop recording the macro.

7. Click the Unhide button on the View tab to display the Unhide dialog box. Select `PERSONAL.XLSB` and click the OK button.

8. Click the Developer tab in the Ribbon, and then click the Macros button on the Ribbon to display the Macros dialog box.

9. Select the macro you recorded and click the Delete button to delete it. Click the Yes button in the confirmation message box.

You now have caused Excel to generate a Personal Macro Workbook that you can use from now on to hold your global macros.
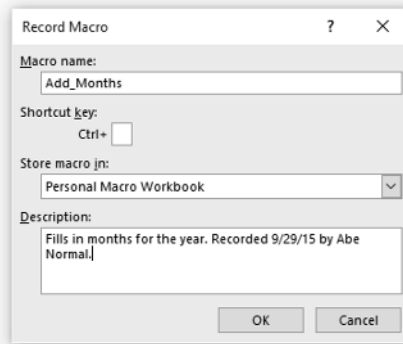
Recording the Macro

To record this macro, start Excel and follow these steps:

1. Create a new workbook by choosing File ➢ New ➢ Blank Workbook.

2. Click the Developer tab in the Ribbon, and then click the Record Macro button on the Ribbon (or just click the Record Macro button on the status bar). This displays the Record Macro dialog box, shown in Figure 1.9, with information entered.

3. Enter the name for the macro in the Macro Name text box: **Add_Months**.

4. In the Shortcut Key text box, enter a shortcut key if you like. (Remember that you can always change the shortcut key later, so you're not forced to enter one at this point.)

5. In the Store Macro In drop-down list, choose whether to store the macro in your Personal Macro Workbook, in a new workbook, or in this active workbook. As discussed a little earlier in this chapter, storing the macro in the Personal Macro Workbook gives you the most flexibility because it is Excel's global macro container. For this example, don't store the macro in the active workbook. Instead, store it in your Personal Macro Workbook. Remember, we'll use this macro in future examples.

FIGURE 1.9
Display the Record
Macro dialog box
for Excel and make
your choices in it.

6. Type a description for the macro in the Description text box.

7. Click the OK button to dismiss the Record Macro dialog box and start recording the macro. When Excel starts running, it sometimes randomly doesn't open the Personal Macro Workbook. So if you see an error message telling you that your "Personal Macro Workbook in the startup folder must stay open for recording," you'll need to open it by hand. Close the Record Macro dialog box. Choose File ➢ Options ➢ Trust Center. Click the Trust Center Settings button. Click Trusted Locations in the left pane. Click to select Excel Default Location: User Startup, and then look down to where it says Path and you'll find the location you want on the hard drive. It should resemble this: `C:\Users\Yourname\AppData\Roaming\Microsoft\Excel\XLSTART`. Use File Explorer to find this path, and then double-click the `PERSONAL.XLSB` file to open it.

8. Click cell A1 to select it. (It may already be selected; click it anyway because you need to record this click instruction.)

9. Enter **January 2016** and press the right-arrow key to select cell B1. Excel automatically changes the date to your default date format. That's fine.

10. Enter **February 2016** and press the left-arrow key to select cell A1 again.

11. Drag from cell A1 to cell B1 so that the two cells are selected.

12. Drag the fill handle from cell B1 to cell L1 so that Excel's AutoFill feature enters the months March 2016 through December 2016 in the cells. (The fill handle is the small black dot in the lower-right corner of the selection frame. You'll know you're on it when the cursor changes from a white to a black cross.)

13. Click the Stop Recording button on the Ribbon or status bar to stop recording the macro.

Now test the new macro. Create a new workbook by choosing File ➤ New ➤ Blank Workbook. On the Developer tab, click the Macros button to open the Macro dialog box. Double-click `PERSONAL.XLSB!Add_Months`. You should see the months filled into the new workbook. You can delete the two workbooks you created in this session. They're no longer needed. Just click the X in the upper-right corner to close them, and then choose Don't Save. But *do* save the `PERSONAL.XLST` workbook.

Specifying How to Trigger an Existing Macro

If you didn't assign a way of running the macro when you recorded it, you can do that now as described here.

Assigning a Macro to a Quick Access Toolbar Button in Word

To assign a macro to the Quick Access Toolbar, follow these steps:

1. Right-click anywhere on the Quick Access Toolbar (it's the set of icons in the upper-left corner, above the Ribbon). A menu appears.

2. Click Customize Quick Access Toolbar on the menu. The Word Options dialog box appears.

3. In the Choose Commands From drop-down list, select Macros.

4. Click the name of the macro to which you want to assign a button.

5. Click the Add button to copy the macro name into the list of buttons on the right.

6. Click the Modify button if you want to assign a different icon or modify the button's name.

7. Click OK to close the dialog.

Assigning a Macro to a Shortcut Key Combination

The section "Running a Macro via a Shortcut Key Combination," earlier in this chapter, explained how to do this in Word. PowerPoint and Access do not let you assign a macro to a key combination. Excel uses a slightly different approach than Word, limiting you to Ctrl and Shift combinations, as described earlier in this chapter in the section "Assigning a Way to Run a Macro in Excel."

Deleting a Macro

To delete a macro you no longer need, follow these steps:

1. Press Alt+F8 to display the Macros dialog box.

2. Choose the macro in the Macro Name list box.

3. Click the Delete button.

4. In the warning message box that appears, click the Yes button.

5. Click the Close button or the Cancel button to close the Macros dialog box.

🌐 Real World Scenario

ORGANIZING MACROS IN WORD WITH THE ORGANIZER DIALOG BOX

Most VBA-enabled applications require you to use the Visual Basic Editor (which is discussed in the next chapter) to move code modules, user forms, and other code items from one file to another file. (A *code module* is a virtual container used for storing macros. A *user form* is a custom dialog box displayed to the user for input.) Word provides a useful tool called the Organizer dialog box that you can use to copy, move, rename, and delete code modules, user forms, and other code items directly in the Word interface without opening the Visual Basic Editor.

To use the Organizer dialog box, follow these steps:

1. In Word, press Alt+F8.

2. Click the Organizer button to display the Organizer dialog box, and click the Macro Project Items tab if the Macro Project Items page (shown here) isn't automatically displayed.
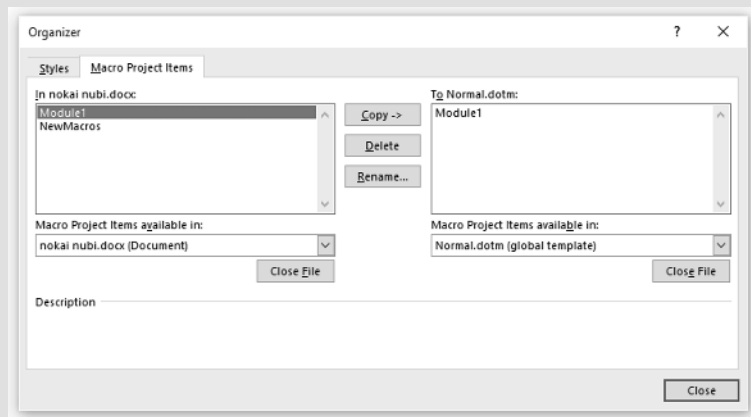
3. Look at the two documents or templates listed in the readouts above the two list boxes. Usually, the left list box shows the active document, and the right one shows Normal.dotm. Change these so that one list box shows the document or template that contains the code you want to copy or move and the other list box shows the destination document or template. (If you want only to delete or rename code items, you need only make the Organizer dialog box list the document or template that contains the items.) To change the document or template listed, click the Close File button underneath the list box on the corresponding side. The Close File button changes to an Open File button. Click this button to display the Open dialog box, navigate to and select the document or template you want, and then click the Open button. The Open dialog will automatically default to displaying the Templates folder.

4. You can then delete, rename, copy, and move macro project items. The following list details how to do this:

- To delete one or more macro project items from a template, choose the item or items from either panel of the Organizer dialog box and click the Delete button. Click the Yes button in the confirmation message box. Any copies of the items in other templates are unaffected.

- To rename a macro project item, select it from either panel and click the Rename button to open the Rename dialog box. Enter the new name and click the OK button. Any copies of the same item in other templates are unaffected.

- To copy one or more macro project items from one template to another, open the templates in the Organizer dialog box. Select the item or items to copy in either panel of the dialog box (the arrows on the Copy button change direction to point to the other panel). Then click the Copy button. If the recipient template contains a macro project item of the same name as one you're copying, Word displays a warning message box telling you that it can't copy the item. If you still want to copy the item, rename either the item you're copying or the item with the same name in the destination template, and then perform the copy operation.

- To move a macro project item from one template to another, copy it as described in the previous paragraph, and then delete the macro project item from the source template.

5. Once you've deleted, renamed, copied, or moved macro project items, click the Close button to close the Organizer dialog box. If Word prompts you to save any changes to affected documents or templates that aren't open in your Word session, click the Yes button.

The Bottom Line

Record a macro.   The easiest way to create a macro is to simply record it. Whatever you type or click—all your behaviors—are translated into VBA automatically and saved as a macro.

Master It   Turn on the macro recorder in Word and create a macro that moves the insertion cursor up three lines. Then turn off the macro recorder and test the new macro.

Assign a macro to a button or keyboard shortcut.   You can trigger a macro using three convenient methods: clicking an entry on the Ribbon, clicking a button in the Quick Access Toolbar, or using a keyboard shortcut. You are responsible for assigning a macro to any or all of these methods.

Master It   Assign an existing macro to a new Quick Access Toolbar button.

Run a macro.   Macros are most efficiently triggered via a Ribbon entry, by clicking a button on the Quick Access Toolbar, or by pressing a shortcut key combination such as Alt+N

or Ctrl+Alt+F. When you begin recording a macro, the Record Macro dialog has buttons that allow you to assign the new macro to a shortcut key or toolbar button. However, if you are using the Visual Basic Editor, you can run a macro by simply pressing F5.

Master It   Execute a macro from within the Visual Basic Editor.

Delete a macro.   It's useful to keep your collection of macros current and manageable. If you no longer need a macro, remove it. Macros can be directly deleted from the Visual Basic Editor or by clicking the Delete button in the Macros dialog (opened by pressing Alt+F8).

Master It   Temporarily remove a macro, and then restore it using the Visual Basic Editor.