MACRO-CODING

Instructional material for ITEC299

**Prepared By**

Cem Yağlı

Left as empty

INDEX

[MACRO-CODING 1](#_Toc19611551)

[INDEX 3](#_Toc19611552)

[01. What is Macro-Coding 6](#_Toc19611553)

[What is Macro? 6](#_Toc19611554)

[A Brief History of VBA 6](#_Toc19611555)

[There's a Big Difference between VB and VBA 7](#_Toc19611556)

[What Is a Spreadsheet Application? 7](#_Toc19611557)

[Steps for Application Development 8](#_Toc19611558)

[Determining User Needs 8](#_Toc19611559)

[Planning an Application That Meets User Needs 9](#_Toc19611560)

[Determining the Most Appropriate User Interface 11](#_Toc19611561)

[Concerning Yourself with the End User 14](#_Toc19611562)

[Creating a user Help system 18](#_Toc19611563)

[Documenting the development effort 18](#_Toc19611564)

[Distributing the application to the user 18](#_Toc19611565)

[Updating the application when necessary 18](#_Toc19611566)

[What VBA Can Do for You 19](#_Toc19611567)

[Automating a Recurring Task 19](#_Toc19611568)

[Automating a Repetitive Task 19](#_Toc19611569)

[Running a Macro Automatically if another Action Takes Place 19](#_Toc19611570)

[Creating Your Own Worksheet Functions 20](#_Toc19611571)

[Simplifying the Workbook's Look and Feel for Other Users 20](#_Toc19611572)

[Controlling Other Office Applications from Excel 21](#_Toc19611573)

[Liabilities of VBA 21](#_Toc19611574)

[ASSIGNMENT 1 22](#_Toc19611575)

[02. Quick look to the Office 23](#_Toc19611576)

[Where to store the macro? 23](#_Toc19611577)

[In Word 23](#_Toc19611578)

[In Excel 23](#_Toc19611579)

[Storing Your Macros 23](#_Toc19611580)

[Word 23](#_Toc19611581)

[Excel 24](#_Toc19611582)

[Understanding Word's Normal.dotm, Templates, and Documents 24](#_Toc19611583)

[Excel Short-Cut Keys 26](#_Toc19611584)

[Function keys in Microsoft Excel 26](#_Toc19611585)

[Entering data by using shortcut keys 27](#_Toc19611586)

[Work in cells or the formula bar by using shortcut keys 27](#_Toc19611587)

[Formating the data by using shortcut keys 28](#_Toc19611588)

[Editing data by using shortcut keys 29](#_Toc19611589)

[Inserting, deleting, and copying a selection by using shortcut keys 30](#_Toc19611590)

[Moving within a selection by using shortcut keys 30](#_Toc19611591)

[Select cells, columns, rows, or objects in worksheets and workbooks by using shortcut keys 31](#_Toc19611592)

[With SCROLL LOCK on 32](#_Toc19611593)

[Selecting cells with special characteristics by using shortcut keys 32](#_Toc19611594)

[Selecting the chart items by using shortcut keys 33](#_Toc19611595)

[Moving and scrolling on a worksheet or workbook by using shortcut keys 33](#_Toc19611596)

[In End mode 34](#_Toc19611597)

[With SCROLL LOCK turned on, 34](#_Toc19611598)

[Work in a data form by using shortcut keys 35](#_Toc19611599)

[Work with the AutoFilter feature by using shortcut keys 35](#_Toc19611600)

[Work with the PivotTable Wizard by using shortcut keys 36](#_Toc19611601)

[Work with page fields in a PivotTable by using shortcut keys 36](#_Toc19611602)

[Group and ungroup PivotTable items by using shortcut keys 36](#_Toc19611603)

[Keys for windows and dialog boxes 37](#_Toc19611604)

[In a dialog box, 37](#_Toc19611605)

[In a text box, 38](#_Toc19611606)

[Print and preview a document by using shortcut keys 39](#_Toc19611607)

[Work in print preview 39](#_Toc19611608)

[Outline data by using shortcut keys 39](#_Toc19611609)

[Keys for menus 39](#_Toc19611610)

[MS Word Shortcut Keys 40](#_Toc19611611)

[Keys used in editing 40](#_Toc19611612)

[Function Keys 42](#_Toc19611613)

[SHIFT+Function keys 42](#_Toc19611614)

[CTRL+Function Key 43](#_Toc19611615)

[CTRL+SHIFT+Function Key 43](#_Toc19611616)

[ALT+Function Key 44](#_Toc19611617)

[ALT+SHIFT+Function Key 44](#_Toc19611618)

[CTRL+ALT+Function Key 45](#_Toc19611619)

[ASSIGNMENT 2 45](#_Toc19611620)

[03. Let’s Start To VBA 46](#_Toc19611621)

[Knowing Your Tools: The Developer Tab 46](#_Toc19611622)

[Understanding Which File Types Allow Macros 47](#_Toc19611623)

[Macro Security 47](#_Toc19611624)

[Adding a Trusted Location 48](#_Toc19611625)

[Using Macro Settings to Enable Macros in Workbooks Outside Trusted Locations 48](#_Toc19611626)

[Using Disable All Macros with Notification 49](#_Toc19611627)

[Recording, Storing, and Running a Macro 49](#_Toc19611628)

[Filling Out the Record Macro Dialog 50](#_Toc19611629)

[Running a Macro 51](#_Toc19611630)

[Creating a Macro Button on the Ribbon 51](#_Toc19611631)

[Creating a Macro Button on the Quick Access Toolbar 52](#_Toc19611632)

[Assigning a Macro to a Form Control, Text Box, or Shape 52](#_Toc19611633)

[Understanding the VB Editor 54](#_Toc19611634)

[VB Editor Settings 54](#_Toc19611635)

[The Project Explorer 54](#_Toc19611636)

[The Properties Window 55](#_Toc19611637)

1. What is Macro-Coding

What is Macro?

The office works are mostly composed of very boring, time consuming, repeated tasks for every office staff, including generating customized letters, e-mails based on some specific criteria. Theses boring tasks also need additional resource consuming controls for ensuring the correctness of the final result. Hence, to produce the desired correctness and productivity in the office management, automated solutions are always preferred instead of manual ones. Although some enterprise scale complicated and expensive solutions are existing, many small firms are looking for cheaper and light weighted solutions which the one is developing it in the firm.

The tailor made solutions would be developed using a programming language like C#, C++, Python, Java (big brothers), data are able to be kept in any DBMS (at local or in cloud) and some add-ons and tools are possible to be used for obtaining a better user-friendliness, productivity and scalability; but these solutions will also be costly and will need skillful technical stuff (designer, developer, DBMS administrator, …) for development and managements of the solution.

Especially for small organizations having some tiny, fast and cheaper solutions would be essential for automating their office works. Hence, the macro-coding is becoming the most proper solution which needing only the office software (Word, Excel, Outlook, Access, etc.), a primitive or almost no coding experience (easy to be learned) to fulfil the automation needs of the office tasks.

A macro is a pre-recorded set of actions would be an automated input sequence that imitates keystrokes or mouse actions that you can use over and over again. They are super useful for automating any repetitive tasks you do every day. VBA (**V**isual **B**asic for **A**pplications) is the programming language derived from BASIC (**B**eginner’s **A**ll-purposed **S**ymbolic **I**nstruction **C**ode); which, you can use for coding in Excel, Word, Access, PowerPoint, Outlook like office tools to create macros. VBA is the language that manipulates Microsoft Office applications in Access, Word, PowerPoint, and Outlook. For the purposes here, VBA is the tool you use to develop macros and manipulate objects to control Excel and to control other Office applications from Excel.

You do not need to purchase anything more than the Office suite (or the individual application) to also own VBA. If you have Excel on your computer, you have VBA on your computer.

Even macro-coding can be produced for any of the office tools, it is heavily used with excel spreadsheets; hence the macro-coding solutions are also named as “Spreadsheet Applications” because of this reason.

A Brief History of VBA

VBA is a present-day dialect of the BASIC (Beginner's All-purpose Symbolic Instruction Code) programming language that was developed in the 1960s. BASIC became widely used in many software applications throughout the next two decades because it was easy to learn and understand.

Over the years, BASIC has evolved and improved in response to advancing technology and increased demands by its users for greater programming flexibility. In 1985, Microsoft released a much richer version of BASIC, named QuickBasic, which boasted the most up-to-date features found in programming languages of the day. In 1992, Microsoft released Visual Basic for Windows, designed to work within the burgeoning Windows environment.

Meanwhile, various software publishers were making their own enhancements to BASIC for their products' programming languages, resulting in a wide and confusing range of functionality and commands among software applications that were using BASIC.

Microsoft recognized the need for developing a standardized programming language for its software products, and created Visual Basic for Applications.

VBA was first released by Microsoft with Excel 5 in the Office 1995 suite. Since then, VBA has become the programming language for Microsoft's other popular Office applications, as well as for external software customers of Microsoft to whom VBA has been licensed for use.

There's a Big Difference between VB and VBA

With all the acronyms bandied about in the world of computing, it's easy to get some terms confused. VB stands for Visual Basic, and it is not the same as VBA. Though both VB and VBA are programming languages derived from BASIC and created by Microsoft, they are otherwise very different. VBA also has some cousins; VBScript – A scripting language that can be used for coding to do client-side controls on a WEB application, and ASP.NET framework for server side coding which are also derived from Basic.

VB is a language that enables you to create standalone executable applications that do not even require its users to have Office or Excel loaded onto their computers. VBA cannot create standalone applications, and it exists within a host application such as Excel and the workbook containing the VBA code. For a VBA macro to run, its host application workbook must be open. This book is about VBA and how it controls Excel.

What Is a Spreadsheet Application?

A spreadsheet application is a spreadsheet file (or group of related files) that is designed so that someone other than the developer can perform useful work without extensive training. According to this definition, most of the spreadsheet files that you’ve developed probably don’t qualify as spreadsheet applications. You may have dozens or hundreds of spreadsheet files on your hard drive, but it’s a safe bet that most of them aren’t designed for others to use.

A good spreadsheet application:

* Enables the end user to perform a task that he or she probably would not be able to do otherwise.
* Provides the appropriate solution to the problem. (A spreadsheet environment isn’t always the optimal approach.)
* Accomplishes what it is supposed to do. This prerequisite may be obvious, but it’s not at all uncommon for applications to fail this test.
* Produces accurate results and is free of bugs.
* Uses appropriate and efficient methods and algorithms to accomplish its job.
* Traps errors before the user is forced to deal with them.
* Does not allow the user to delete or modify important components accidentally (or intentionally).
* Has a clear and consistent user interface so that the user always knows how to be proceed.
* Has well-documented formulas, macros, and user interface elements that allow for subsequent changes, if necessary.
* Is designed so that it can be modified in simple ways without making major changes. A basic fact is that a user’s needs change over time.
* Has an easily accessible help system that provides useful information on at least the major procedures.
* Is designed to be portable and to run on any system that has the proper software (in this case, a copy of the appropriate version of Excel).

It should come as no surprise that it is possible to create spreadsheet applications for many different usage levels, ranging from a simple fill-in-the-blank template to an extremely complex application that uses a custom interface and may not even look like a spreadsheet.

Steps for Application Development

There is no simple, surefire recipe for developing an effective spreadsheet application. Everyone has his or her own style for creating such applications. In addition, every project is different and, therefore, requires its own approach. Finally, the demands and technical expertise of the people you work with (or for) also play a role in how the development process proceeds.

Spreadsheet developers typically perform the following activities:

* Determine the needs of the user(s)
* Plan an application that meets these needs
* Determine the most appropriate user interface
* Create the spreadsheet, formulas, macros, and user interface
* Test and debug the application
* Attempt to make the application bulletproof
* Make the application aesthetically appealing and intuitive
* Document the development effort
* Develop user documentation and Help systems
* Distribute the application to the user

Update the application when necessary not all these steps are required for each application, and the order in which these activities are performed varies from project to project.

Determining User Needs

When you undertake a new Excel project, one of your first steps is to identify exactly what the end users require. Failure to thoroughly assess the end users’ needs early on often results in additional work later when you have to adjust the application so that it does what it was supposed to do in the first place.

In some cases, you’ll be intimately familiar with the end users — you may even be an end user yourself. In other cases (for example, if you’re a consultant developing a project for a new client), you may know little or nothing about the users or their situations.

How do you determine the needs of the user? If you’ve been asked to develop a spreadsheet application, it’s a good idea to meet with the end users and ask specific questions. Better yet, get everything in writing, create flow diagrams, pay attention to minor details, and do anything else to ensure that the product you deliver is the product that is needed.

Here are some guidelines that may help make this phase easier:

* Don’t assume that you know what the user needs. Second-guessing at this stage almost always causes problems later.
* If possible, talk directly to the end users of the application, not just their supervisor or manager.
* Learn what, if anything, is currently being done to meet the users’ needs. You might be able to save some work by simply adapting an existing application. At the very least, looking at current solutions will familiarize you with the operation.
* Identify the resources available at the users’ site. For example, try to determine whether you must work around any hardware or software limitations.
* If possible, determine the specific hardware systems that will be used. If your application will be used on slow systems, you need to take that into account.
* Identify which versions of Excel are in use. Although Microsoft does everything in its power to urge users to upgrade to the latest version of the software, the majority of Excel users don’t.
* Understand the skill levels of the end users. This information will help you design the application appropriately.
* Determine how long the application will be used and whether any changes are anticipated during the lifetime of the project. Knowing this information may influence the amount of effort that you put into the project and help you plan for changes.

And finally, don’t be surprised if the project specifications change before you complete the application. This occurrence is common, and you’re in a better position if you expect changes rather than being surprised by them. Just make sure that your contract (if you have one) addresses the issue of changing specifications.

Planning an Application That Meets User Needs

After you determine the end users’ needs, it’s tempting to jump right in and start fiddling around in Excel. Take it from someone who suffers from this problem:

Try to restrain yourself. Builders don’t construct a house without a set of blueprints, and you shouldn’t build a spreadsheet application without some type of plan. The formality of your plan depends on the scope of the project and your general style of working, but you should spend at least some time thinking about what you’re going to do and coming up with a plan of action.

Before rolling up your sleeves and settling down at your keyboard, you’ll benefit by taking some time to consider the various ways you can approach the problem. This planning period is where a thorough knowledge of Excel pays off. Avoiding blind alleys rather than stumbling into them is always a good idea.

If you ask a dozen Excel experts to design an application based on precise specifications, chances are you’ll get a dozen different implementations of the project that meet those specifications. Of those solutions, some will be better than the others because Excel often provides several options to accomplish a task. If you know Excel inside and out, you’ll have a good idea of the potential methods at your disposal, and you can choose the one most appropriate for the project at hand. Often, a bit of creative thinking yields an unusual approach that’s vastly superior to other methods.

So at the beginning stage of this planning period, consider some general options, such as these:

* **File structure**: Think about whether you want to use one workbook with multiple sheets, several single-sheet workbooks, or a template file.
* **Data structure**: You should always consider how your data will be structured and also determine whether you will be using external database files or storing everything in worksheets.
* **Add-in or workbook file**: In some cases, an add-in may be the best choice for your final product. Or perhaps you might use an add-in with a standard workbook.
* **Version of Excel**: Will your Excel application be used with Excel 2016 only? With Excel 2010 and later? What about Excel 2003 and earlier versions? Will your application also be run on a Macintosh? These considerations are important because each new version of Excel adds features that aren’t available in previous versions. The Ribbon interface introduced in Excel 2007 makes it more challenging than ever to create an application that works with older versions.
* **Error handling**: Error handling is a major issue with applications. You need to determine how your application will detect and deal with errors. For example, if your application applies formatting to the active worksheet, you need to be able to handle a case in which a chart sheet is active.
* **Use of special features**: If your application needs to summarize a lot of data, you may want to consider using Excel’s pivot table feature. Or you may want to use Excel’s data validation feature as a check for valid data entry. Performance issues: The time to start thinking about increasing the speed and efficiency of your application is at the development stage, not when the application is completed and users are complaining.
* **Level of security**: As you may know, Excel provides several protection options to restrict access to particular elements of a workbook. For example, you can lock cells so that formulas cannot be changed, and you can assign a password to prevent unauthorized users from viewing or accessing specific files. Determining up front exactly what you need to protect — and what level of protection is necessary — will make your job easier.

Another design consideration is remembering to plan for change. You’ll do yourself a favor if you make your application ***as generic as possible***. For example, don’t write a procedure that works with only a specific range of cells. Rather, write a procedure that accepts any range as an argument. When the inevitable changes are requested, such a design makes it easier for you to carry out the revisions.

Also, you may find that the work that you do for one project is similar to the work that you do for another. Keep ***reusability*** in mind when you are planning a project. Avoid letting the end user completely guide your approach to a problem. For example, suppose that you meet with a manager who tells you that the department needs an application to write text files that will be imported into another application. Don’t confuse the user’s need with the solution. The user’s real need is to share data. Using an intermediate text file to do it is just one possible solution; better ways to approach the problem may exist. In other words, don’t let the users define their problem by stating it in terms of a solution approach. Determining the best approach is your job.

Determining the Most Appropriate User Interface

When you develop spreadsheets that others will use, you need to pay special attention to the user interface. By user interface, we mean the method by which the user interacts with the application and executes your VBA macros.

Since the introduction of Excel 2007, some of these user interface decisions are irrelevant. Custom menus and toolbars are, for all intents and purposes, obsolete. Consequently, developers must learn how to work with the Ribbon. Excel provides several features that are relevant to user interface design:

* Ribbon customization
* Shortcut menu customization
* Shortcut keys
* Custom dialog boxes (UserForms)
* Controls (such as a ListBox or a CommandButton) placed directly on a worksheet

Customizing the Ribbon

The Ribbon UI introduced in Excel 2007 is a dramatic shift in user interface design. Fortunately, the developer has a fair amount of control over the Ribbon. Although Excel allows the end user to modify the Ribbon, making UI changes via code isn’t a simple task.

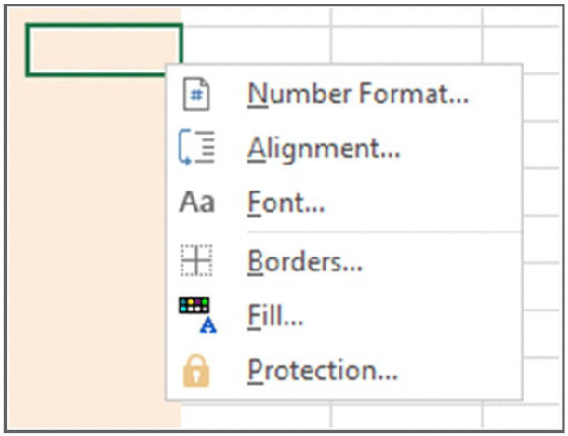
Customizing shortcut menus

Excel allows the VBA developer to customize the right-click shortcut menus. Right-click menus can offer users a way to easily trigger an action without having to move too far from the range in which they are working. Figure 1.1 illustrates a customized shortcut menu that appears when a cell is right-clicked.

Creating shortcut keys

Another user interface option at your disposal is a custom shortcut key. Excel lets you assign a Ctrl key (or Shift+Ctrl key) combination to a macro. When the user presses the key combination, the macro executes. Be aware, however, of two caveats. First, make it clear to the user which keys are active and what they do.

Second, do not assign a key combination that’s already used for something else. A key combination that you assign to a macro takes precedence over the built-in shortcut keys. For example, Ctrl+S is a built-in Excel shortcut key used to save the current file. If you assign this key combination to a macro, you lose the capability to save the file with Ctrl+S. Remember that shortcut keys are case sensitive, so you can use a combination such as Ctrl+Shift+S.



*Figure 1.1. A customised shortcut menu*

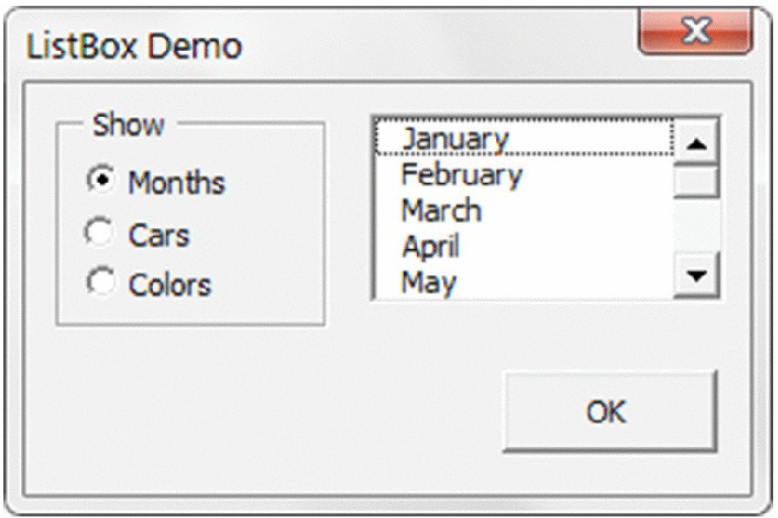
Creating custom dialog boxes

Anyone who has used a personal computer for any length of time is undoubtedly familiar with dialog boxes. Consequently, custom Excel dialog boxes can play a major role in the user interfaces that you design for your applications. Figure 1.2 shows an example of a custom dialog box.

A custom dialog box is known as a *UserForm*. A UserForm can solicit user input, get a user’s options or preferences, and direct the flow of your entire application. The elements that make up a UserForm (buttons, drop-down lists, check boxes, and so on) are called controls — more specifically, ActiveX controls. Excel provides a standard assortment of ActiveX controls, and you can also incorporate third-party controls.

After adding a control to a dialog box, you can link it to a worksheet cell so that it doesn’t require any macros (except a simple macro to display the dialog box).

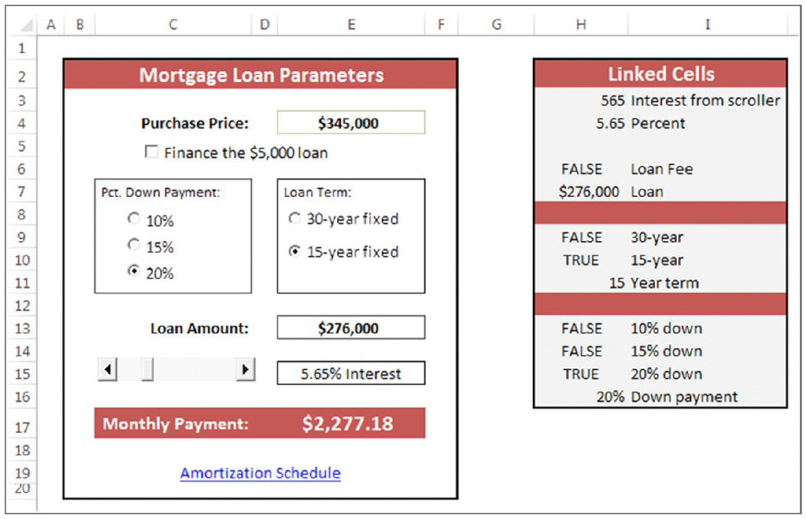
Linking a control to a cell is easy, but it’s not always the best way to get user input from a dialog box. Most of the time, you want to develop VBA macros that work with your custom dialog boxes.



*Figure 1.2 A dialog box created with Excel’s UserForm feature*.

Using ActiveX controls on a worksheet

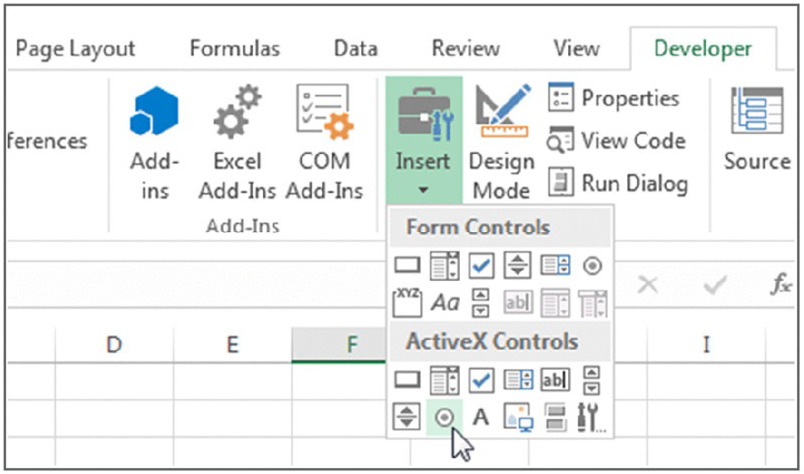
Excel also lets you add UserForm ActiveX controls to a worksheet’s drawing layer (an invisible layer on top of a sheet that holds pictures, charts, and other objects). Figure 1.3 shows a simple worksheet model with several UserForm controls inserted directly in the worksheet. This sheet contains the following ActiveX controls: a CheckBox, a ScrollBar, and two sets of OptionButtons. This workbook uses no macros. Rather, the controls are linked to worksheet cells.



*Figure 1.3 You can add UserForm controls to worksheets and link them to cells.*

Perhaps the most common control is a CommandButton. By itself, a CommandButton doesn’t do anything, so you need to attach a macro to each CommandButton. Using dialog box controls directly in a worksheet often eliminates the need for custom dialog boxes. You can often greatly simplify the operation of a spreadsheet by adding a few ActiveX controls (or Form controls) to a worksheet. These ActiveX controls let the user make choices by operating familiar controls rather than making entries in cells.

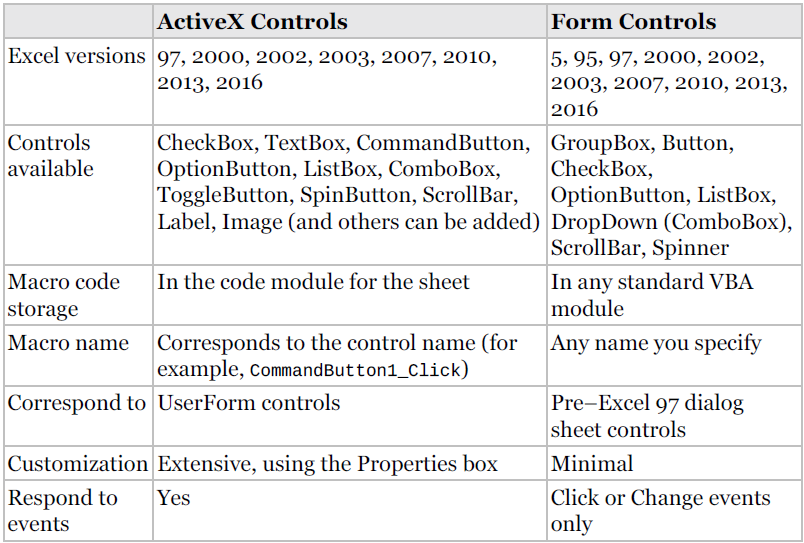
Access these controls by using the Developer ➜ Controls ➜ Insert command (see Figure 1.4). If the Developer tab isn’t on the Ribbon, add it by using the Customize Ribbon tab of the Excel Options dialog box.



*Figure 1.4 Using the Ribbon to add controls to a worksheet.*

**The controls come in two types**: Form controls and ActiveX controls. Both sets of controls have their advantages and disadvantages. Generally, Form controls are easier to use, but ActiveX controls are a bit more flexible. Table 1.1 summarizes these two classes of controls.

*Table 1.1 ActiveX Controls versus Form Controls*



Executing the development effort

After you identify user needs, determine the approach that you’ll take to meet those needs, and decide on the components that you’ll use for the user interface, it’s time to get down to the nitty-gritty and start creating the application. This step, of course, comprises a great deal of the total time that you spend on a particular project.

How you go about developing the application depends on your personal style and the nature of the application. Except for simple fill-in-the-blanks template workbooks, your application will probably use macros. Creating macros in Excel is easy, but creating good macros is difficult.

Concerning Yourself with the End User

In this section, we discuss the important development issues that surface as your application becomes more and more workable and as the time to package and distribute your work grows nearer.

Testing the application

How many times have you used a commercial software application, only to have it bomb out on you at a crucial moment? Most likely, the problem was caused by insufficient testing that didn’t catch all the bugs. All nontrivial software has bugs, but in the best software, the bugs are simply more obscure. As you’ll see, you sometimes must work around the bugs in Excel to get your application to perform properly.

After you create your application, you need to test it. Testing is one of the most crucial steps; it’s not uncommon to spend as much time testing and debugging an application as you did creating the application. Actually, you should be doing a great deal of testing during the development phase. After all, whether you’re writing a VBA routine or creating formulas in a worksheet, you want to make sure that the application is working the way it’s supposed to work.

Like standard compiled applications (Big brothers), spreadsheet applications that you develop are prone to bugs. A bug can be defined as (1) something that does happen but shouldn’t happen while a program (or application) is running, or (2) something that doesn’t happen when it should happen. Both species of bugs are equally nasty, and you should plan on devoting a good portion of your development time to testing the application under all reasonable conditions and fixing any problems that you find.

It’s important to thoroughly test any spreadsheet application that you develop for others. And depending on its eventual audience, you may want to make your application bulletproof. In other words, try to anticipate all the errors and screw-ups that could possibly occur and make concerted efforts to avoid them — or, at least, to handle them gracefully. This foresight not only helps the end user but also makes it easier on you and protects your reputation. Also consider using beta testing; your end users are likely candidates because they’re the ones who will be using your product. (See the upcoming sidebar “What about beta testing?”)

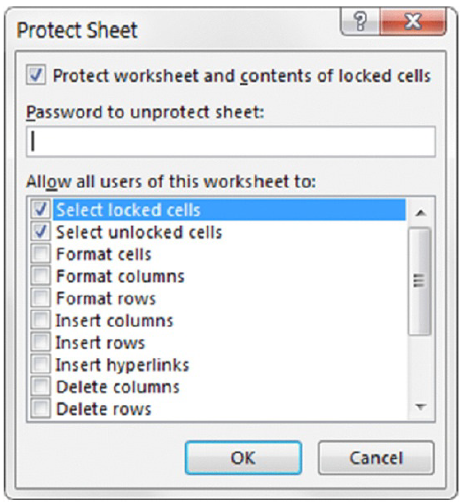
Although you can’t conceivably test for all possibilities, your macros should be able to handle common types of errors. For example, what if the user enters a text string instead of a numeric value? What if the user tries to run your macro when a workbook isn’t open? What if he cancels a dialog box without making any selections? What happens if the user presses Ctrl+F6 and jumps to the next window? When you gain experience, these types of issues become very familiar, and you account for them without even thinking.

Making the application bulletproof

If you think about it, destroying a spreadsheet is fairly easy. Erasing one critical formula or value can cause errors throughout the entire worksheet — and perhaps even other dependent worksheets. Even worse, if the damaged workbook is saved, it replaces the good copy on disk. Unless a backup procedure is in place, the user of your application may be in trouble, and you’ll probably be blamed for it.

Obviously, you can easily see why you need to add some protection when users — especially novices — will be using your worksheets. Excel provides several techniques for protecting worksheets and parts of worksheets:

* **Lock specific cells**: You can lock specific cells (by using the Protection tab in the Format Cells dialog box) so that users can’t change them. Locking takes effect only when the document is protected with the Review ➜ Changes ➜ Protect Sheet command. The Protect Sheet dialog box has options that allow you to specify which actions users can perform on a protected sheet (see Figure 1.5).
* **Hide the formulas in specific cells**: You can hide the formulas in specific cells (by using the Protection tab in the Format Cells dialog box) so that others can’t see them. Again, hiding takes effect only when the document is protected by choosing the Review ➜ Changes ➜ Protect Sheet command.
* **Protect an entire workbook**: You can protect an entire workbook — the structure of the workbook, the window position and size, or both. Use the Review ➜ Changes ➜ Protect Workbook command for this purpose.
* **Lock objects on the worksheet**: Use the Properties section in the task pane to lock objects (such as shapes) and prevent them from being moved or changed. To access this section of the task pane, right-click the object and choose Size and Properties. Locking objects takes effect only when the document is protected using the Review ➜ Changes ➜ Protect Sheet command. By default, all objects are locked.
* **Hide rows, columns, sheets, and documents**: You can hide rows, columns, sheets, and entire workbooks. Doing so helps prevent the worksheet from looking cluttered and also provides some modest protection against prying eyes.
* **Designate an Excel workbook as read-only recommended**: You can designate an Excel workbook as read-only recommended (and use a password) to ensure that the file can’t be overwritten with any changes. You make this designation in the General Options dialog box. Display this dialog box by choosing File ➜ Save As, clicking the Tools button, and then choosing General Options.
* **Assign a password**: You can assign a password to prevent unauthorized users from opening your file. Choose File ➜ Info ➜ Protect Workbook ➜ Encrypt with Password.
* **Use a password-protected add-in**: You can use a password-protected add-in, which doesn’t allow the user to change anything on its worksheets.



*Figure 1.5. Using the Protect Sheet dialog box to specify what users can and can’t do*.

***Be aware*** that Excel passwords can often be easily circumvented using commercially available password breaking programs. Excel 2007 and later versions seem to have stronger security than previous versions, but a determined user can still crack them. Bottom line? Don’t think of password protection as foolproof. Sure, it will be effective for the casual user. But if someone really wants to break your password, he or she probably can.

Making the application aesthetically appealing and intuitive

If you’ve used many different software packages, you’ve undoubtedly seen examples of poorly designed user interfaces, difficult-to-use programs, and just plain ugly screens. If you’re developing spreadsheets for other people, you should pay particular attention to how the application looks.

How a computer program looks can make all the difference in the world to users, and the same is true with the applications that you develop with Excel. Beauty, however, is in the eye of the beholder. If your skills lean more in the analytical direction, consider enlisting the assistance of someone with a more aesthetic sensibility to provide help with design.

The good news is that features found in Excel 2007 and later make creating better looking spreadsheets a relatively easy task. If you stick with predesigned cell styles, your work stands a good chance of looking good. And, with the click of a mouse, you can apply a new theme that transforms the look of the workbook — and still looks good.

End users appreciate a good-looking user interface, and your applications will have a much more polished and professional look if you devote additional time to design and aesthetic considerations. An application that looks good demonstrates that its developer cared enough about the product to invest extra time and effort.

Take the following suggestions into account:

* **Strive for consistency**. When designing dialog boxes, for example, try to emulate the look and feel of Excel’s dialog boxes whenever possible. Be consistent with formatting, fonts, text size, and colors.
* **Keep it simple**. A common mistake that developers make is trying to cram too much information into a single screen or dialog box. A good rule is to present only one or two chunks of information at a time.
* **Break down input screens**. If you use an input screen to solicit information from the user, consider breaking it up into several, less-crowded screens. If you use a complex dialog box, you may want to break it up by using a MultiPage control, which lets you create a familiar tabbed dialog box.
* **Don’t overdo color. Use color sparingly**. It’s very easy to overdo color and make the screen look gaudy.
* **Monitor typography and graphics**. Pay attention to numeric formats and use consistent typefaces, font sizes, and borders.
* **Evaluating aesthetic qualities is subjective**. When in doubt, strive for simplicity and clarity.

Creating a user Help system

With regard to user documentation, you basically have two options: paper-based documentation or electronic documentation. Providing electronic help is standard fare in Windows applications. Fortunately, your Excel applications can also provide help — even context-sensitive help. Developing help text takes quite a bit of additional effort, but for a large project, it may be worth it.

Another point to consider is support for your application. In other words, who gets the phone call if the user encounters a problem? If you aren’t prepared to handle routine questions, you need to identify someone who is. In some cases, you want to arrange it so that only highly technical or bug-related issues escalate to the developer.

Documenting the development effort

Putting a spreadsheet application together is one thing. Making it understandable for other people is another. As with traditional programming, it’s important that you thoroughly document your work. Such documentation helps you if you need to go back to it (and you will), and it helps anyone else whom you might pass it on to.

How do you document a workbook application? You can either store the information in a worksheet or use another file. You can even use a paper document, if you prefer. Perhaps the easiest way is to use a separate worksheet to store your comments and key information for the project. For VBA code, use comments liberally. (VBA text preceded with an apostrophe is ignored because that text is designated as a comment.) Although an elegant piece of VBA code can seem perfectly obvious to you today, when you come back to it in a few months, your reasoning may be completely obscured unless you use the VBA comment feature.

Distributing the application to the user

You’ve completed your project, and you’re ready to release it to the end users.

How do you go about distributing it? You can choose from many ways to distribute your application, and the method that you choose depends on many factors.

You could just hand over a CD-ROM or thumb drive, scribble a few instructions, and be on your way. Or you may want to install the application yourself — but this approach isn’t always feasible. Another option is to develop an official setup program that performs the task automatically. You can write such a program in a traditional programming language, purchase a generic setup program, or write your own in VBA.

Excel incorporates technology to enable developers to digitally sign their applications. This process is designed to help end users identify the author of an application, to ensure that the project has not been altered, and to help prevent the spread of macro viruses or other potentially destructive code. To digitally sign a project, you first apply for a digital certificate from a formal certificate authority (or you can self-sign your project by creating your own digital certificate). Refer to the Help system or the Microsoft website for additional information.

Updating the application when necessary

After you distribute your application, you’re finished with it, right? You can sit back, enjoy yourself, and try to forget about the problems that you encountered (and solved) during development. In rare cases, yes, you may be finished. More often, however, the users of your application won’t be completely satisfied. Sure, your application adheres to all the original specifications, but things change.

Seeing an application working often causes the user to think of other things that the application could be doing.

When you need to update or revise your application, you’ll appreciate that you designed it well in the first place and that you fully documented your efforts.

What VBA Can Do for You

Everyone reading this book uses Excel for their own needs, such as financial budgeting, forecasting, analyzing scientific data, creating invoices, or charting the progress of their favorite football team. One thing all readers have in common is the need to automate some kind of frequently encountered task that is either too time-consuming or too cumbersome to continue doing manually. That's where VBA comes in.

The good news is that utilizing VBA does not mandate that you first become a world-class professional programmer. Many VBA commands are at your disposal, and are relatively easy to implement and customize for your everyday purposes.

Anything you can do manually you can do with VBA, but VBA enables you to do it faster and with a minimized risk of human error. Many things that Excel does not allow you to do manually, you can do with VBA. The following sections describe a handful of examples of what VBA can do for you.

Automating a Recurring Task

If you find yourself needing to produce weekly or monthly sales and expense reports, a macro can create them in no time flat, in a style and format you (and more importantly, your boss) will be thrilled with. And if the source data changes later that day and you need to produce the updated report again, no problem—just run the macro again!

Automating a Repetitive Task

When faced with needing to perform the same task on every worksheet in your workbook, or in every workbook in a particular file folder, you can create a macro to “loop” through each object and do the deed. Figure 1.6 shows an example of worksheets that were sorted in alphabetical order by a macro that looped through each tab name, repositioning each sheet in the process.

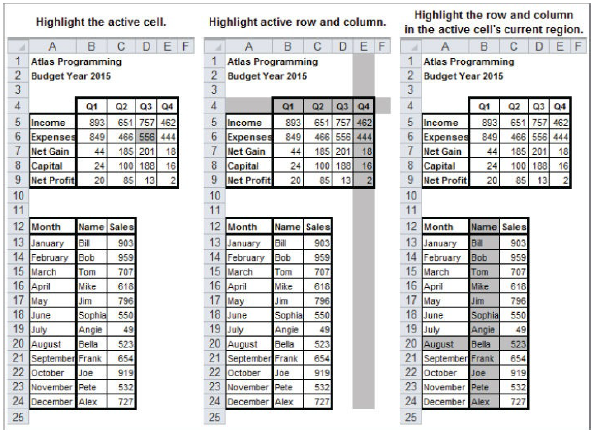
Running a Macro Automatically if another Action Takes Place

In some situations, you want a macro to run automatically so you don't have to worry about remembering to run it yourself. For example, to automatically refresh a pivot table the moment its source data changes, you can monitor those changes with VBA, ensuring that your pivot table always displays real-time results. This is called “event” programming, which is cool stuff, and is discussed in following chapters.

An event can also be triggered and programmed anytime a cell or range of cells is selected. A common request I've received from Excel users is to highlight the active cell, or the row and column belonging to the active cell, automatically when a cell is selected. Figure 1.6 shows three options to easily locate your active cell as you traverse your worksheet.

**

*Figure 1.5 Example of alphabetically sorted worksheets by a macro*



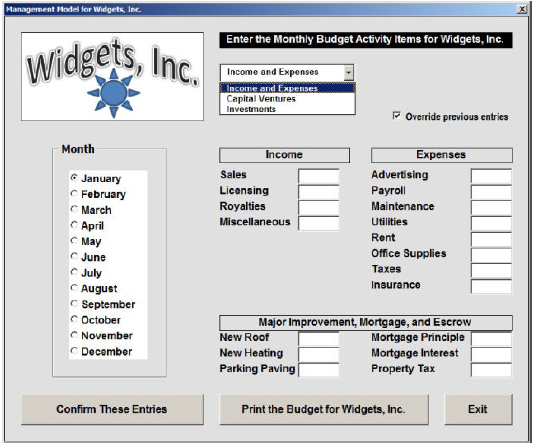
*Figure 1.6 Event programming in VBA*

Creating Your Own Worksheet Functions

You can create your own worksheet functions, known as user-defined functions, to handle custom calculations that Excel's built-in functions do not provide, or would be too complicated to use even if such native functions were available. For example, later in the book you see how to add up numbers in cells that are formatted a certain color. UDFs, as these custom functions are called “***User-Defined Functions***.”

Simplifying the Workbook's Look and Feel for Other Users

When you create a workbook for others to use, there will inevitably be users who know little to nothing about Excel, but who will still need to work in that file. You can build a customized interface with user-friendly menus and informational pop-up boxes to guide your novice users throughout their activities in the workbook. You might be surprised at how un-Excel-looking an Excel workbook can be, with VBA providing a visually comfortable and interactive experience for users unfamiliar with Excel, enabling them to get their work done. Figure 1.7 shows an example of accomplishing this with UserForms.



*Figure 1.7 UserForms example*

Controlling Other Office Applications from Excel

If you create narrative reports in Word that require an embedded list of data from Excel, or if you need to import a table from Access into an Excel worksheet, VBA can automate the process. VBA is the programming language for Microsoft's other Office applications, enabling you to write macros in Excel to perform tasks in those other applications, with the users being none the wiser that they ever left Excel while the macro was running.

As you might imagine, the list of advantages to using VBA could fill the capacity of your average flash drive. The point is, you are sure to have tasks in your everyday dealings with Excel that can be accomplished more quickly and efficiently with VBA.

Liabilities of VBA

Although VBA is a tremendously useful and versatile tool, it is not a 100 percent perfect programming language—but then, no programming language anywhere can truthfully claim infallibility. The pros of VBA far outweigh its cons, but learning and using VBA does come with a few objective caveats that you should be aware of:

* With each version release of Excel, Microsoft may add new VBA commands or stop supporting existing VBA commands, sometimes without advance warning. Surprises do happen, as was especially the case when Office 2007 was released with all its added features. Such is life in the world of Excel VBA. You will probably learn of coding errors from people who have upgraded to a newer version and are using the workbook you created in an earlier version.
* VBA does not run uniformly in all computer operating environments. Sometimes, no matter how extensively you test your code and how flawlessly the macros run on your computer as you develop a project, there will be users of your workbook who will eventually report an error in your code. It won't be your fault or VBA's fault, it's just the idiosyncrasies of how programming languages such as VBA mix with various operating systems, Office versions, and network configurations.
* Programming languages, including VBA, are not warmly received by all workplace IT departments. Many companies have set internal policies that forbid employees from downloading malicious software onto workplace computers. This is an understandable concern, but the corporate safety nets are sometimes cast far and wide to include Excel workbooks with VBA code. The tug of war in companies between the security interests of IT and the work efficiency needs of management can determine whether the VBA code you install will actually be allowed for use in some company venues.
* Finally, VBA is a large program. It has thousands of keywords and the language library is only getting larger. Actually, I see this as a good thing, because the more VBA you learn, the more productivity and control you will have with Excel. Just as with any language, be it spoken or programming, there is a level of rolling-up-your-shirt-sleeves commitment that'll be needed to learn VBA. Even the longest journey starts with a first step, and this book gets you on your way.

ASSIGNMENT 1

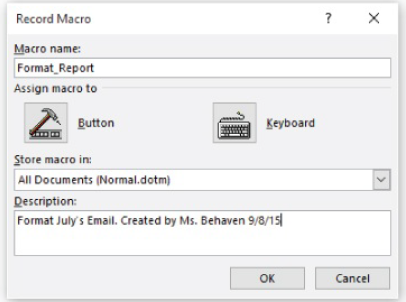
**Do a survey** about the security issues and the malicious use (ransomware, blackmailing) of macro coding, and ways of protection. Your document have to be around **2500-4000 words long**, and its **Turnitin similarity** have to be **less than 20%**. **Duration**: 4 weeks.

1. Quick look to the Office

Where to store the macro?

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 2.1. 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.



*Figure 2.1 Record Macro Dialog box in Word*

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. (It is stored in your %userprofile% \Application Data \Microsoft \Excel \XLSTART\ folder.) 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 ***modulenamed 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.

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.

Excel Short-Cut Keys

Function keys in Microsoft Excel

**Function Key SHIFT CTRL .**

F1 Display Help or What’s This?

the Office Assistant

F2 Edit the active cell Edit a cell comment

F3 Paste a name into Paste a function into

a formula a formula Define a name

F4 Repeat the last action Repeat the last Close the window

Find (Find Next)

F5 Go To Display the Find Restore the dialog

box window size

F6 Move to the next pane Move to the Move to the next

previous pane workbook window

F7 Spelling command Move the window

F8 Extend a selection Add to the selection Resize the window

F9 Calculate all sheets in all open Calculate the active Minimize the

Workbooks worksheet workbook

F10 Make the menu bar active Display a shortcut Maximize or restore

menu (right click) the workbook

window

F11 Create a chart Insert a new Insert a Microsoft

worksheet Excel 4.0 macro

sheet

F12 Save As command Save command Open command

Entering data by using shortcut keys

**To Press .**

Complete a cell entry ENTER

Cancel a cell entry ESC

Repeat the last action F4 or CTRL+Y

Start a new line in the same cell ALT+ENTER

Delete the character to the left of the insertion BACKSPACE

point, or delete the selection

Delete the character to the right of the insertion DELETE

point, or delete the selection

Delete text to the end of the line CTRL+DELETE

Move one character up, down, left, or right Arrow keys

Move to the beginning of the line HOME

Edit a cell comment SHIFT+F2

Create names from row and column labels CTRL+SHIFT+F3

Fill down CTRL+D

Fill to the right CTRL+R

Fill the selected cell range with the current entry CTRL+ENTER

Complete a cell entry and move down in the ENTER

selection

Complete a cell entry and move up in the SHIFT+ENTER

selection

Complete a cell entry and move to the right TAB

in the selection

Complete a cell entry and move to the left in SHIFT+TAB

The selection

Work in cells or the formula bar by using shortcut keys

**To Press .**

Start a formula = (EQUAL SIGN)

Cancel an entry in the cell or formula bar ESC

Edit the active cell F2

Edit the active cell and then clear it, or delete the BACKSPACE

preceding character in the active cell as you edit the

cell contents

Paste a name into a formula F3

Define a name CTRL+F3

Calculate all sheets in all open workbooks F9

Calculate the active worksheet SHIFT+F9

Insert the AutoSum formula ALT+= (EQUAL SIGN)

Enter the date CTRL+; (SEMICOLON)

Enter the time CTRL+SHIFT+:(COLON)

Insert a hyperlink CTRL+K

Complete a cell entry ENTER

Copy the value from the cell above the active cell into CTRL+SHIFT+”

the cell or the formula bar (QUOTATION MARK)

Alternate between displaying cell values and CTRL+` (SINGLE LEFT

displaying cell formulas QUOTATION MARK)

Copy a formula from the cell above the active cell into CTRL+’ (APOSTROPHE)

the cell or the formula bar

Enter a formula as an array formula CTRL+SHIFT+ENTER

Display the Formula Palette after you type a valid CTRL+A

function name in a formula

Insert the argument names and parentheses for a CTRL+SHIFT+A

function, after you type a valid function name in a

formula

Display the AutoComplete list ALT+DOWN ARROW

Formating the data by using shortcut keys

**To Press .**

Display the Style command (Format menu) ALT+’(APOSTROPHE)

Display the Cells command (Format menu) CTRL+1

Apply the General number format CTRL+SHIFT+~

Apply the Currency format with two decimal places CTRL+SHIFT+$

(Negative numbers appear in parentheses)

Apply the Percentage format with no decimal places CTRL+SHIFT+%

Apply the Exponential number format with two decimal CTRL+SHIFT+^

places

Apply the Date format with the day, month, and year CTRL+SHIFT+#

Apply the Time format with the hour and minute, and CTRL+SHIFT+@

Indicate A.M. or P.M.

Apply the Number format with two decimal places, CTRL+SHIFT+!

1000 separator, and – for negative values

Apply the outline border CTRL+SHIFT+&

Remove all borders CTRL+SHIFT+\_

Apply or remove bold formatting CTRL+B

Apply or remove italic formatting CTRL+I

Apply or remove an underline CTRL+U

Apply or remove strikethrough formatting CTRL+5

Hide rows CTRL+9

Unhide rows CTRL+SHIFT+(

Hide columns CTRL+0 (ZERO)

Unhide columns CTRL+SHIFT+)

Editing data by using shortcut keys

**To Press .**

Edit the active cell F2

Cancel an entry in the cell or formula bar ESC

Edit the active cell and then clear it, or delete the BACKSPACE

preceding character in the active cell as you edit the cell

contents

Paste a name into a formula F3

Complete a cell entry ENTER

Enter a formula as an array formula CTRL+SHIFT+ENTER

Display the Formula Palette after you type a valid CTRL+A

function name in a formula

Insert the argument names and parentheses for a CTRL+SHIFT+A

function, after you type a valid function name in a formula

Inserting, deleting, and copying a selection by using shortcut keys

**To Press .**

Copy the selection CTRL+C

Paste the selection CTRL+V

Cut the selection CTRL+X

Clear the contents of the selection DELETE

Insert blank cells CTRL+SHIFT+PLUS SIGN

Delete the selection CTRL+ –

Undo the last action CTRL+Z

Moving within a selection by using shortcut keys

**To Press .**

Move from top to bottom within the selection (down), or ENTER

In the direction that is selected on the Edit tab (Tools menu,

Options command)

Move from bottom to top within the selection (up), or SHIFT+ENTER

opposite to the direction that is selected on the Edit tab

(Tools menu, Options command)

Move from left to right within the selection, or TAB

move down one cell if only one column is selected

Move from right to left within the selection, or SHIFT+TAB

move up one cell if only one column is selected

Move clockwise to the next corner of the selection CTRL+PERIOD

Move to the right between nonadjacent selections CTRL+ALT+RIGHT

ARROW

Move to the left between nonadjacent selections CTRL+ALT+LEFT

ARROW

Select cells, columns, rows, or objects in worksheets and workbooks by using shortcut keys

**To Press .**

Select the current region around the active cell (the CTRL+SHIFT+\*

current region is an area enclosed by blank rows and (ASTERISK)

blank columns)

Extend the selection by one cell SHIFT+ arrow key

Extend the selection to the last nonblank cell in the CTRL+SHIFT+ arrow key

same column or row as the active cell

Extend the selection to the beginning of the row SHIFT+HOME

Extend the selection to the beginning of the CTRL+SHIFT+HOME

worksheet

Extend the selection to the last cell used on the CTRL+SHIFT+END

worksheet (lower-right corner)

Select the entire column CTRL+SPACEBAR

Select the entire row SHIFT+SPACEBAR

Select the entire worksheet CTRL+A

If multiple cells are selected, select only the active SHIFT+BACKSPACE

cell

Extend the selection down one screen SHIFT+PAGE DOWN

Extend the selection up one screen SHIFT+PAGE UP

With an object selected, select all objects on a sheet CTRL+SHIFT+SPACEBAR

Alternate between hiding objects, displaying objects, CTRL+6

and displaying placeholders for objects

Show or hide the Standard toolbar CTRL+7

In End mode, to Press

Turn End mode on or off END

Extend the selection to the last nonblank cell in the END, SHIFT+ arrow key

same column or row as the active cell

Extend the selection to the last cell used on the END, SHIFT+HOME

worksheet (lower-right corner)

Extend the selection to the last cell in the current END, SHIFT+ENTER

row; this keystroke is unavailable if you selected the

Transition navigation keys check box on the

Transition tab (Tools menu, Options command)

With SCROLL LOCK on

**To Press .**

Turn SCROLL LOCK on or off SCROLL LOCK

Scroll the screen up or down one row UP ARROW or DOWN

ARROW

Scroll the screen left or right one column LEFT ARROW or RIGHT

ARROW

Extend the selection to the cell in the upper-left SHIFT+HOME

corner of the window

Extend the selection to the cell in the lower-right SHIFT+END

corner of the window

**Tip:** When you use the scrolling keys (such as PAGE

UP and PAGE DOWN) with SCROLL LOCK

turned off, your selection moves the distance you

scroll. If you want to keep the same selection as you

scroll, turn on SCROLL LOCK first.

Selecting cells with special characteristics by using shortcut keys

**To Press .**

Select the current region around the active cell CTRL+SHIFT+\*

(the current region is an area enclosed by blank rows (ASTERISK)

and blank columns)

Select the current array, which is the array that the active CTRL+/

cell belongs to

Select all cells with comments CTRL+SHIFT+O

(the letter O)

Select cells whose contents are different from the CTRL+\

Comparison cell in each row (for each row, the

comparison cell is in the same column as the active cell)

Select cells whose contents are different from the CTRL+SHIFT+|

Comparison cell in each column (for each column, the

comparison cell is in the same row as the active cell)

Select only cells that are directly referred to by formulas CTRL+[

In the selection

Select all cells that are directly or indirectly referred CTRL+SHIFT+{

to by formulas in the selection

Select only cells with formulas that refer directly to CTRL+]

The active cell

Select all cells with formulas that refer directly or CTRL+SHIFT+}

Indirectly to the active cell

Select only visible cells in the current selection ALT+SEMICOLON

Selecting the chart items by using shortcut keys

**To Press .**

Select the previous group of items DOWN ARROW

Select the next group of items UP ARROW

Select the next item within the group RIGHT ARROW

Select the previous item within the group LEFT ARROW

Moving and scrolling on a worksheet or workbook by using shortcut keys

**To Press .**

Move one cell in a given direction Arrow key

Move to the edge of the current data region CTRL+ arrow key

Move between unlocked cells on a protected TAB

worksheet

Move to the beginning of the row HOME

Move to the beginning of the worksheet CTRL+HOME

Move to the last cell on the worksheet, which is the CTRL+END

cell at the intersection of the right-most used

column and the bottom-most used row (in the

lower-right corner); cell opposite the Home cell,

which is typically A1

Move down one screen PAGE DOWN

Move up one screen PAGE UP

Move one screen to the right ALT+PAGE DOWN

Move one screen to the left ALT+PAGE UP

Move to the next sheet in the workbook CTRL+PAGE DOWN

Move to the previous sheet in the workbook CTRL+PAGE UP

Move to the next workbook or window CTRL+F6 or CTRL+TAB

Move to the previous workbook or window CTRL+SHIFT+F6 or

CTRL+SHIFT+TAB

Move to the next pane F6

Move to the previous pane SHIFT+F6

Scroll to display the active cell CTRL+BACKSPACE

In End mode

**To Press .**

Turn End mode on or off END

Move by one block of data within a row or column END, arrow key

Move to the last cell on the worksheet, which is the cell at END, HOME

the intersection of the right-most used column and the

bottom-most used row (in the lower-right corner); cell

opposite the Home cell, which is typically A1

Move to the last cell to the right in the current row END, ENTER

that is not blank; unavailable if you have selected

the Transition navigation keys check box on the

Transition tab (Tools menu, Options command)

With SCROLL LOCK turned on,

**To Press .**

Turn SCROLL LOCK on or off SCROLL LOCK

Move to the cell in the upper-left corner of the window HOME

Move to the cell in the lower-right corner of the window END

Scroll one row up or down UP ARROW

or DOWN ARROW

Scroll one column left or right LEFT ARROW

or RIGHT ARROW

**Tip:** When you use the scrolling keys (such as PAGE UP and PAGE DOWN) with SCROLL LOCK turned off, your selection moves the distance you scroll. If you want to preserve your selection while you scroll through the worksheet, turn on SCROLL LOCK first.

Work in a data form by using shortcut keys

**To Press .**

Select a field or a command button ALT+ key, where key is the

underlined letter in the field or

command name

Move to the same field in the next record DOWN ARROW

Move to the same field in the previous record UP ARROW

Move to the next field you can edit in the record TAB

Move to the previous field you can edit in the SHIFT+TAB

record

Move to the first field in the next record ENTER

Move to the first field in the previous record SHIFT+ENTER

Move to the same field 10 records forward PAGE DOWN

Move to the same field 10 records back PAGE UP

Move to the new record CTRL+PAGE DOWN

Move to the first record CTRL+PAGE UP

Move to the beginning or end of a field HOME or END

Move one character left or right within a field LEFT ARROW or RIGHT ARROW

Extend a selection to the beginning of a field SHIFT+HOME

Extend a selection to the end of a field SHIFT+END

Select the character to the left SHIFT+LEFT ARROW

Select the character to the right SHIFT+RIGHT ARROW

Work with the AutoFilter feature by using shortcut keys

**To Press .**

Display the AutoFilter list for the current column Select the cell that contains the

column label, and then press

ALT+DOWN ARROW

Close the AutoFilter list for the current column ALT+UP ARROW

Select the next item in the AutoFilter list DOWN ARROW

Select the previous item in the AutoFilter list UP ARROW

Select the first item (All) in the AutoFilter list HOME

Select the last item in the AutoFilter list END

Filter the list by using the selected item in the ENTER

AutoFilter list

Work with the PivotTable Wizard by using shortcut keys

**In Step 3 of the PivotTable Wizard, To Press .**

Select the next or previous field button in the list UP ARROW

or DOWN ARROW

Select the field button to the right or left in a LEFT ARROW

multicolumn field button list or RIGHT ARROW

Move the selected field into the Page area ALT+P

Move the selected field into the Row area ALT+R

Move the selected field into the Column area ALT+C

Move the selected field into the Data area ALT+D

Display the PivotTable Field dialog box ALT+L

Work with page fields in a PivotTable by using shortcut keys

**To Press .**

Select the previous item in the list UP ARROW

Select the next item in the list DOWN ARROW

Select the first visible item in the list HOME

Select the last visible item in the list END

Display the selected item ENTER

Group and ungroup PivotTable items by using shortcut keys

**To Press .**

Group selected PivotTable items ALT+SHIFT+RIGHT

ARROW

Ungroup selected PivotTable items ALT+SHIFT+LEFT

ARROW

Keys for windows and dialog boxes

**In a window, to Press .**

Switch to the next program ALT+TAB

Switch to the previous program ALT+SHIFT+TAB

Show the Windows Start menu CTRL+ESC

Close the active workbook window CTRL+W

Restore the active workbook window CTRL+F5

Switch to the next workbook window CTRL+F6

Switch to the previous workbook window CTRL+SHIFT+F6

Carry out the Move command (workbook CTRL+F7

icon menu, menu bar)

Carry out the Size command (workbook CTRL+F8

icon menu, menu bar)

Minimize the workbook window to an icon CTRL+F9

Maximize or restore the workbook window CTRL+F10

Select a folder in the Open or Save As ALT+0 to select the folder list; arrow

dialog box (File menu) keys to select a folder

Choose a toolbar button in the Open or ALT+ number (1 is the leftmost

Save As dialog box (File menu) button, 2 is the next, and so on)

Update the files visible in the Open or Save F5

As dialog box (File menu)

In a dialog box,

**To Press .**

Switch to the next tab in a dialog box CTRL+TAB or CTRL+PAGE

DOWN

Switch to the previous tab in a dialog box CTRL+SHIFT+TAB or

CTRL+PAGE UP

Move to the next option or option group TAB

Move to the previous option or option SHIFT+TAB

group

Move between options in the active dropdown Arrow keys

list box or between some options in a group of

options

Perform the action assigned to the active SPACEBAR

button (the button with the dotted outline),

or select or clear the active check box

Move to an option in a drop-down list box Letter key for the first letter in the

option name you want (when a

dropdown list box is selected)

Select an option, or select or clear a check ALT+ letter, where letter is the key

Box for the underlined letter in the option

name

Open the selected drop-down list box ALT+DOWN ARROW

Close the selected drop-down list box ESC

Perform the action assigned to the default ENTER

command button in the dialog box (the

button with the bold outline ¾ often the OK

button)

Cancel the command and close the dialog box ESC

In a text box,

**To Press .**

Move to the beginning of the entry HOME

Move to the end of the entry END

Move one character to the left or right LEFT ARROW or RIGHT ARROW

Move one word to the left or right CTRL+LEFT ARROW or

CTRL+RIGHT ARROW

Select from the insertion point to the beginning SHIFT+HOME

of the entry

Select from the insertion point to the end of the SHIFT+END

entry

Select or unselect one character to the left SHIFT+LEFT ARROW

Select or unselect one character to the right SHIFT+RIGHT ARROW

Select or unselect one word to the left CTRL+SHIFT+LEFT ARROW

Select or unselect one word to the right CTRL+SHIFT+RIGHT ARROW

Print and preview a document by using shortcut keys

**To Press .**

Display the Print command (File menu) CTRL+P

Work in print preview

**To Press .**

Move around the page when zoomed in Arrow keys

Move by one page when zoomed out PAGE UP or PAGE DOWN

Move to the first page when zoomed out CTRL+UP ARROW or

CTRL+LEFT ARROW

Move to the last page when zoomed out CTRL+DOWN ARROW or

CTRL+RIGHT ARROW

Outline data by using shortcut keys

**To Press .**

Ungroup rows or columns ALT+SHIFT+LEFT ARROW

Group rows or columns ALT+SHIFT+RIGHT ARROW

Display or hide outline symbols CTRL+8

Hide selected rows CTRL+9

Unhide selected rows CTRL+SHIFT+(

Hide selected columns CTRL+0 (ZERO)

Unhide selected columns CTRL+SHIFT+)

Keys for menus

**To Press .**

Show a shortcut menu SHIFT+F10

Make the menu bar active F10 or ALT

Show the program icon menu (on the program ALT+SPACEBAR

title bar)

Select the next or previous command on the DOWN ARROW or UP

menu or submenu ARROW (with the menu or

submenu displayed)

Select the menu to the left or right, or, with a LEFT ARROW or RIGHT

submenu visible, switch between the main menu ARROW

and the submenu

Select the first or last command on the menu or HOME or END

submenu

Close the visible menu and submenu at the same ALT

time

Close the visible menu, or, with a submenu ESC

visible, close the submenu only

**Tip**: You can select any menu command on the menu bar or on a visible toolbar with the keyboard. Press ALT to select the menu bar. (To then select a toolbar, press CTRL+TAB; repeat until the toolbar you want is selected.) Press the letter that is underlined in the menu name that contains the command you want. In the menu that appears, press the letter underlined in the command name that you want.

MS Word Shortcut Keys

Keys used in editing

**Command Name Keys .**

All Caps Ctrl+Shift+A

Apply List Bullet Ctrl+Shift+L

Auto Format Alt+Ctrl+K

Auto Text F3

Bold Ctrl+B

Cancel ESC

Center Para Ctrl+E

Change Case Shift+F3

Clear Del

Close or Exit Alt+F4

Copy Ctrl+C

Create Auto Text Alt+F3

Cut Ctrl+X

Double Underline Ctrl+Shift+D

Find Ctrl+F

**Command Name Keys .**

Help F1

Hyperlink Ctrl+K

Indent Ctrl+M

Italic Ctrl+I

Justify Para Ctrl+J

Merge Field Alt+Shift+F

New Document Ctrl+N

Open Ctrl+O

Outline Alt+Ctrl+O

Overtype Insert

Page Alt+Ctrl+P

Page Break Ctrl+Return

Paste Ctrl+V

Paste Format Ctrl+Shift+V

Print Ctrl+P

Print Preview Ctrl+F2

Redo Alt+Shift+Backspace

Redo or Repeat Ctrl+Y

Save Ctrl+S

Select All Ctrl+A

Small Caps Ctrl+Shift+K

Style Ctrl+Shift+S

Subscript Ctrl+=

Superscript Ctrl+Shift+=

Task Pane Ctrl+F1

Time Field Alt+Shift+T

Underline Ctrl+U Undo Ctrl+Z

Update Fields F9

Word Count List Ctrl+Shift+G

Function Keys

**Key Meaning .**

F1 Get Help or visit Microsoft Office Online.

F2 Move text or graphics.

F3 Insert an AutoText (AutoText: A storage location for text

or graphics you want to use again, such as a standard

contract clause or a long distribution list. Each selection of

text or graphics is recorded as an AutoText entry and is

assigned a unique name.) entry (after Microsoft Word

displays the entry).

F4 Repeat the last action.

F5 Choose the Go To command (Edit menu).

F6 Go to the next pane or frame.

F7 Choose the Spelling command (Tools menu).

F8 Extend a selection.

F9 Update selected fields.

F10 Activate the menu bar.

F11 Go to the next field.

F12 Choose the Save As command (File menu).

SHIFT+Function keys

**Key Meaning .**

SHIFT+F1 Start context-sensitive Help or reveal formatting.

SHIFT+F2 Copy text.

SHIFT+F3 Change the case of letters.

SHIFT+F4 Repeat a Find or Go To action.

SHIFT+F5 Move to the last change.

SHIFT+F6 Go to the previous pane or frame.

SHIFT+F7 Choose the Thesaurus command (Tools menu, Language

submenu).

SHIFT+F8 Shrink a selection.

SHIFT+F9 Switch between a field code and its result.

**Key Meaning .**

SHIFT+F10 Display a shortcut menu.

SHIFT+F11 Go to the previous field.

SHIFT+F12 Choose the Save command (File menu).

CTRL+Function Key

**Key Meaning .**

CTRL+F2 Choose the Print Preview command (File menu).

CTRL+F3 Cut to the Spike (Spike: A special AutoText entry that stores

multiple deletions. Microsoft Word appends one item to

another until you paste the contents as a group in a new

location in your document. You can also use the Microsoft Office Clipboard to get the same result.)

CTRL+F4 Close the window.

CTRL+F5 Restore the document window size (for example, after

maximizing it).

CTRL+F6 Go to the next window.

CTRL+F7 Choose the Move command (title bar shortcut menu).

CTRL+F8 Choose the Size command (title bar shortcut menu).

CTRL+F9 Insert an empty field.

CTRL+F10 Maximize the document window.

CTRL+F11 Lock a field.

CTRL+F12 Choose the Open command (File menu).

CTRL+SHIFT+Function Key

**Key Meaning .**

CTRL+SHIFT+F3 Insert the contents of the Spike (Spike: A special AutoText

entry that stores multiple deletions. Microsoft Word appends

one item to another until you paste the contents as a group in

a new location in your document. You can also use the

Microsoft Office Clipboard to get the same result.)

CTRL+SHIFT+F5 Edit a bookmark.

CTRL+SHIFT+F6 Go to the previous window.

CTRL+SHIFT+F7 Update linked information in a Microsoft Word source

document.

CTRL+SHIFT+F8 Extend a selection or block (then press an arrow key).

CTRL+SHIFT+F9 Unlink a field.

CTRL+SHIFT+F11 Unlock a field.

CTRL+SHIFT+F12 Choose the Print command (File menu).

ALT+Function Key

**Key Meaning .**

ALT+F1 Go to the next field.

ALT+F3 Create an AutoText (AutoText: A storage location for text or

Graphics you want to use again, such as a standard contract

clause or a long distribution list. Each selection of text or

graphics is recorded as an AutoText entry and is assigned a

unique name.) entry.

ALT+F4 Quit Microsoft Word.

ALT+F5 Restore the program window size.

ALT+F6 Move from an open dialog box back to the document for

dialog boxes such as Find and Replace that support this

behavior.

ALT+F7 Find the next misspelling or grammatical error. The Check

spelling as you type check box must be selected (Tools

menu, Options dialog box, Spelling & Grammar tab).

ALT+F8 Run a macro.

ALT+F9 Switch between all field codes and their results.

ALT+F10 Maximize the program window.

ALT+F11 Display Microsoft Visual Basic code.

ALT+SHIFT+Function Key

**Key Meaning .**

ALT+SHIFT+F1 Go to the previous field.

ALT+SHIFT+F2 Choose the Save command (File menu).

ALT+SHIFT+F9 Run GOTOBUTTON or MACROBUTTON from the field

that displays the field results.

ALT+SHIFT+F10 Display the menu or message for a smart tag. If more than

one smart tag is present, switch to the next smart tag and

display its menu or message.

ALT+SHIFT+F11 Start the Microsoft Script Editor

CTRL+ALT+Function Key

**Key Meaning .**

CTRL+ALT+F1 Display Microsoft System Information.

CTRL+ALT+F2 Open command (File menu).

ASSIGNMENT 2

1. Let’s Start To VBA

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 to access the Developer tab:

1. Right-click the ribbon and choose Customize the Ribbon.

2. In the right list box, select the Developer check box, which is the eighth item.

3. Click OK to return to Excel.

Excel displays the Developer tab, as shown in Figure 2.1. The Developer tab provides an interface for running and recording macros.



*Figure 2.1 Developer Tab in Excel*

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

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

The ***Add-ins group*** provides icons for managing regular add-ins and COM add-ins. 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. 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.

The ***XML group*** of the Developer tab contains tools for importing and exporting XML documents.

The ***Modify group*** enables you to specify whether the Document Panel is always displayed for new documents. Users can enter keywords and a document description in the Document Panel.

Understanding Which File Types Allow Macros

Excel 2016 offers support for four file types. Macros are not allowed to be stored in the .xlsx file type, and this file type is 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 2016.

The available files types are as listed here:

* **Excel Workbook (.xlsx)—** Files are stored as a series of XML objects and then zipped into a single file. This creates 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 will not need to worry about malicious macros. However, if he sees an .xlsm file, he should be concerned that there might be macros attached.
* **Excel Binary Workbook (.xlsb)—** This is a binary format designed to handle the larger 1-million-row grid size introduced in Excel 2007. Legacy versions of Excel stored their files in a proprietary binary format. Although binary formats might load more quickly, they are more prone to corruption, and a few lost bits can destroy a 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 A1:IV65536. In addition, if someone opens the file in Excel 2003, she loses 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 Options.
2. In the Excel Options dialog, select the Save category from the left navigation pane.
3. Open the Save Files in This Format drop-down and select Excel Macro-Enabled Workbook (\*.xlsm). Click OK.

Macro Security

After a Word VBA macro was used as the delivery method for the ***Melissa virus***, Microsoft changed the default security settings to prevent macros from running. Therefore, before we can begin discussing the recording of a macro, it’s important to look at how to adjust the default settings.

In Excel 2016, you can either globally adjust the security settings or control macro settings for certain workbooks by saving the workbooks in a trusted location. Any workbook stored in a folder that is marked as a trusted location automatically has 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.

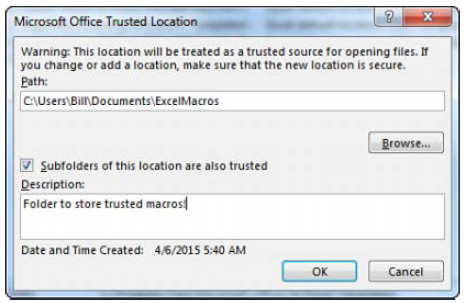
4. Click the Add New Location button. Excel displays the Microsoft Office Trusted Location dialog (see Figure 2.2).

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 Are Also Trusted.

8. Click OK to add the folder to the Trusted Locations list.



*Figure 2.2 Manage trusted folders in the Trusted Locations category of the Trust Center.*

Using Macro Settings to Enable Macros in Workbooks Outside 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 for 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— The operative words in this setting are “with Notification.” This means that you see a notification when you open a file with macros and you can choose to enable the content. If you ignore the notification, the macros remain disabled. This setting is similar to Medium security setting in Excel 2003 and is the recommended setting. In Excel 2016, a message is displayed in the Message Area indicating that macros have been disabled. You can choose to enable the content by clicking that option, as shown in Figure 2.3 .
* 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 it’s 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 the Low macro security setting in Excel 2003. Although it requires the least amount of hassle, it also opens your computer to attacks from malicious Melissa-like viruses. Microsoft suggests that you not use this setting.



*Figure 2.3 The Enable Content option appears when you use Disable All Macros with Notification.*

Using Disable All Macros with Notification

It is recommended that you set your macro settings to ***Disable All Macros with Notification***. If you use this setting and open a workbook that contains macros, you see a security warning in the area just above the formula bar. If you are 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.

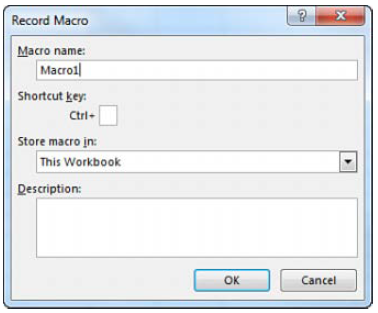
If you forget to enable the macros and attempt to run a macro, Excel indicates that you cannot run the macro because all macros have been disabled. If this occurs, close the workbook and reopen it to access the message bar again.

After you enable macros in a workbook stored on a local hard drive and then save the workbook, Excel remembers that you previously enabled macros in this workbook. The next time you open this workbook, macros are automatically enabled.

Recording, Storing, and Running a Macro

Recording a macro is useful when you do not have experience writing lines of code in a macro. As you gain more knowledge and experience, you will record macros 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 2.4 .



*Figure 2.4 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 that you will soon be creating many macros, use a meaningful name for the macro. A name such as FormatReport is more useful than one like Macro1.

The second field in the Record Macro dialog box is a shortcut key. If you type a lowercase j in this field and later press Ctrl+J, this macro runs. Be careful, however, because Ctrl+A through Ctrl+Z (except Ctrl+J) are all already assigned to other tasks in Excel. If you assign a macro to Ctrl+B, you won’t be able to use Ctrl+B for bold anymore. One alternative is to assign the macros to Ctrl+Shift+A through Ctrl+Shift+Z. To assign a macro to Ctrl+Shift+A, you type Shift+A in the shortcut key box.

You can reuse a shortcut key for a macro. For example, if you assign a macro to Ctrl+C, Excel runs 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, or This Workbook. It is recommended that you store macros related to a particular workbook in This Workbook.

The Personal Macro Workbook (Personal.xlsm) 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 opens 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.

It is not recommended that 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.

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. You can also access a Stop Recording icon in the lower-left corner of the Excel window. Look for a small white 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 usually has a Record Macro icon, which is a small 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. You can also assign macros to a button on the ribbon or the Quick Access Toolbar, form controls, or 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. Icons added to the ribbon are still enabled even when your macro workbook is not open. If you click the icon when the macro workbook is not open, Excel opens the workbook and runs the macro. Follow these steps to add a macro button to the ribbon:

1. Right-click the ribbon and choose Customize the Ribbon.

2. In the list box on the right, choose the tab name where you want to add an icon.

3. 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.

4. 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.

5. To rename the group, click the Rename button. Type a new name, such as Report Macros. Click OK. Excel shows the group in the list box as Report Macros (Custom). Note that the word Custom does not appear in the ribbon.

6. 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.

7. 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.

8. 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.

9. You can move the Report Macros group to a new location on the ribbon tab. Click Report Macros (Custom) and use the up and down arrow icons on the right of the dialog.

10. Click OK to close the Excel Options dialog. 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 a macro. If a 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. Right-click the Quick Access Toolbar and choose Customize Quick Access Toolbar.

2. 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.

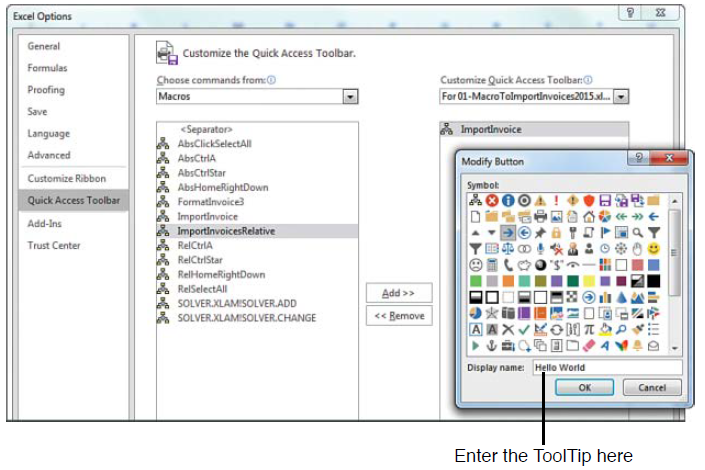
3. Open the upper-left drop-down and select Macros from the list. The Macros category is fourth in the list. Excel displays a list of available macros in the left list box.

4. 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.

5. 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 2.5). Choose an icon from the list. In the Display Name box, replace the macro name with a short name that appears in the tooltip for the icon.

6. Click OK to close the Modify Button dialog.

7. Click OK to close the Excel Options dialog. The new button appears on the Quick Access Toolbar.



*Figure 2.5 Attach a macro to a button 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, you can 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 in this one drop-down menu. The form controls are at the top, and the ActiveX controls are at the bottom. Most icons in the ActiveX section of the drop-down look identical to an icon in the form controls section of the drop-down. Click the Button Form Control icon at the upper-left corner of the Insert 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.

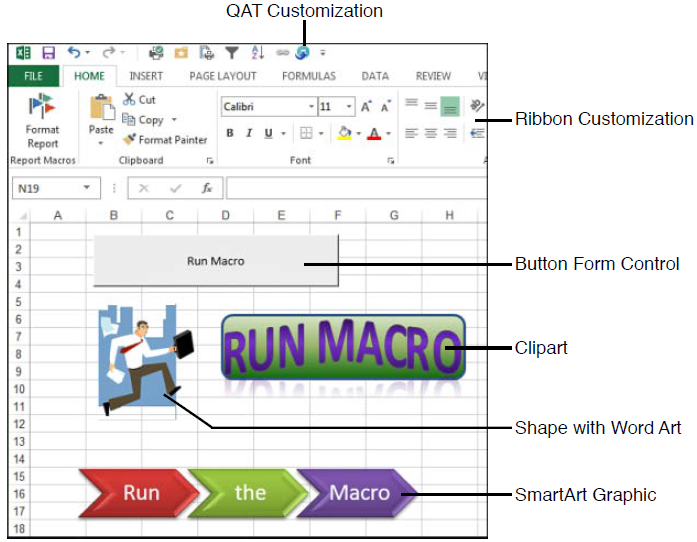
5. Type a new label for the button. Note that while you are typing, the selection border around the button changes from dots to diagonal lines to indicate that you are in Text Edit mode. You cannot change the button color while in Text Edit mode. To exit Text Edit mode, either click the diagonal lines 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, which has 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 a cell to deselect the button.

8. Click the new button to run the macro.

Macros can be assigned to any worksheet object, such as clip art, a shape, SmartArt graphics, or a text box. In Figure 2.6, 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.

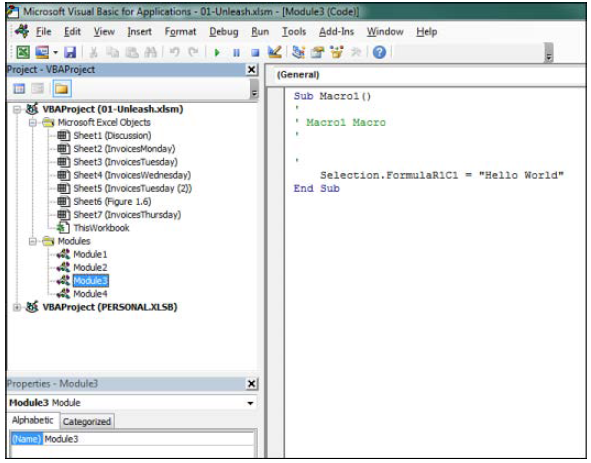


*Figure 2.6 Assigning a macro to a form control or an object is appropriate for macros stored in the same workbook as the control. You can assign a macro to any of these objects.*

Understanding the VB Editor

If you want to edit a recorded macro, you do it in the VB Editor. Press Alt+F11 or use the Visual Basic icon in the Developer tab.

Figure 2.7 shows an example of a typical VB Editor screen. You can see three windows: the 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 2.7 The VB Editor window.*

VB Editor Settings

Several settings in the VB Editor enable you to customize this editor and assist you in writing your macros. Under Tools, Options, Editor, you 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 selecting 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. Whether you turn this setting on or keep it off is a matter of your personal preference.

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 see that there is a folder containing 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 or press Ctrl+R or locate the bizarre Project Explorer icon just below the Tools menu, sandwiched between Design Mode and Properties Window.

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.
* Forms— Excel enables you to design your own forms to interact with the user.
* Modules— When you record a macro, Excel automatically creates a module in which to place the code. Most of your code resides 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’s needing to understand how it works.

The Properties Window

The Properties window enables you to edit the properties of various components such as sheets, workbooks, modules, and form controls. The properties 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.