Excel VBA and Macros with MrExcel

Bill Jelen

que®

800 East 96th Street

Indianapolis, Indiana 46240 USA

Excel VBA and Macros with MrExcel

ISBN-13: 978-0-7897-3938-4
ISBN-10: 0-7897-3938-0

Printed in the United States of America

Second Printing: July 2009

Associate Publisher
Greg Wiegand

Acquisitions Editor
Loretta Yates

Managing Editor
Patrick Kanouse

Project Editor
Mandie Frank

Copy Editor
Tonya Simpson

Video LiveLesson Post-Production
Lora White

Video Project Manager
John Herrin

Publishing Coordinator
Cindy Teeters

Designer
Gary Adair

Compositor
Mark Shirar

Table of Contents

About the Author

Bill Jelen is the host of MrExcel.com and the author of 24 books on Microsoft Excel, including *Special Edition Using Microsoft Office Excel 2007*, *Pivot Table Data Crunching for Microsoft Office Excel 2007*, *VBA and Macros for Microsoft Excel*, *Excel Gurus Gone Wild*, *Excel for Marketing Managers*, and *Guerilla Data Analysis Using Microsoft Excel*. He has made more than 60 guest appearances on TechTV with Leo Laporte and was voted guest of the year on the *Computer America* radio show. He has produced more than 900 episodes of his daily video podcast "Learn Excel from MrExcel." Bill will entertain you while showing you the powerful tricks in Excel. Before founding MrExcel.com in 1998, Jelen spent 12 years in the trenches as a financial analyst for the accounting, finance, marketing, and operations departments of a publicly held company. Since then, his company has automated Excel reports for hundreds of clients around the world. The website answers more than 30,000 questions a year—for free—for readers all over the world. Jelen joins us from Akron, Ohio.

Dedication

To Pam Gensel, for my first lessons in Excel VBA.

Acknowledgments

The videos in this edition are based on topics from my books on Excel VBA. I want to thank (again) everyone mentioned in the acknowledgements of *VBA and Macros for Microsoft Excel* and *VBA and Macros for Microsoft Office Excel 2007*. If you were in those acknowledgements, consider yourself part of these acknowledgements. (Some of you might consider it lazy of me to not copy and paste all those acknowledgements again into this book, but I can rationalize this away by saying that we are trying to save space in this smaller form factor.)

Also, thanks to Lora White and Cassie White. Lora does post-production on my daily podcast and did an over-the-top job of post-production on these lessons. Cassie is a second-year student in interactive media and lent her video editing skills as well. Thanks to Carl Palmer for assistance with the opening footage for each lesson.

We Want to Hear from You!

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

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

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

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

Email: feedback@quepublishing.com

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

Reader Services

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

Introduction

If you are holding this booklet, you love Excel. You use Excel every work day for a variety of tasks, and you want to take your Excel skills to the next level by adding some VBA macros to your skill set. This is an excellent move.

Excel VBA enables you to automate mundane processes. Whether you want some code to produce a report for every department from a data set, or simply want a quick routine to make an ugly 40-minute process disappear into a few button clicks, Excel VBA is up to the task.

Unfortunately, the Excel Macro recorder often fails to deliver usable code. To succeed, you need to learn five critical concepts. You can then combine these concepts with recorded code to produce stellar results.

The lessons on this DVD come directly from the current version of my Power Macros seminar. My assumption is that you already use Excel 20–40 hours per week. You might have taken a class in BASIC or COBOL somewhere along the way, although this is not a prerequisite. You've also probably turned on the macro recorder and had varying degrees of frustration with it. If that sounds like you, I can get you up to speed with writing your own macros.

—Bill Jelen

PART I

Recording Macros

Learn how to effectively record a macro. When you are starting a new VBA project, it is often helpful to use the macro recorder to get you 90% of the way toward a finished project. The lessons in Part I show you how to actually get the macro recorder to work more of the time.

There are seven lessons in this part:

lesson 1

Enable the Developer Tab

What You Will Learn

Microsoft moved all the VBA tools to a new ribbon tab called Developer. Unfortunately, they hide this ribbon tab. In Lesson 1, you learn how to unhide the Developer tab and change your default security settings.

To work along, use any workbook.

Notes from the Lesson

- Enable the Developer tab by selecting the Office Button, and then Excel Options.
- When adjusting macro security, choose Disable All Macros with Notification. This is the equivalent to Medium in the old versions of Excel.
- If you will always be writing macros, change the default file format on your computer to Excel Macro-Enabled Workbook. This setting can be found by selecting the Office Button, Excel Options, and then going to the Save category.

Differences in Excel 2003

- Access the Macro Security dialog by selecting Tools, Macro, Security. Set the setting to Medium and click OK.

lesson 2

Recording Your First Macro

What You Will Learn

Learn how to record your first macro. You will learn how to fill out the Record Macro dialog.

To work along, use Lesson02.xlsm.

Notes from the Lesson

- Start recording by clicking the Record Macro icon on the Developer tab.
- A macro name cannot have a space. Use a name such as FormatBlue instead of Format Blue.
- Although most shortcut keys are already in use, you can have 26 new letters by using Ctrl+Shift.
- If this is a one-time use macro, store it in this workbook. If it is a general-purpose macro, store it in the Personal Macro workbook.
- Although the description is optional, it provides comments that will help you remember what the macro does.

Tip

The shortcut keys Ctrl+j and Ctrl+k are the two keys that are not already assigned to other shortcuts.

Caution

As you work along, make sure you do not select another cell while recording the macro. If you record the process of selecting a cell before applying the format, the macro will always format that one particular selected cell.

Note

The process of joining words in the macro name and capitalizing each word is called *CamelCase*. The name comes from the uppercase "bumps" in the middle of the compound word, suggestive of the humps of a camel.

In Excel 2003

- To record a new macro, select Tools, Macros, Record New Macro.
- To stop recording a macro, click the Stop icon on the Stop Recording toolbar. If you cannot find this toolbar, select Tools, Macro, Stop Recording.

Code from the Lesson

Note

Don't worry if this code doesn't make sense to you at this time. You will learn to decode this code after finishing Part III.

```
Sub FormatBlue()
 '
 ' FormatBlue Macro
 ' This macro will format a cell to indicate that it is to be audited.
'  Bold dark blue font,
 '  light blue background, thick blue borders on the top and bottom.
 '
 ' Keyboard Shortcut: Ctrl+Shift+B
 '
    Selection.Font.Bold = True
    With Selection.Interior
         .Pattern = xlSolid
         .PatternColorIndex = xlAutomatic
         .ThemeColor = xlThemeColorLight2
         .TintAndShade = 0.599993896298105
         .PatternTintAndShade = 0
    End With
    With Selection.Font
         .ThemeColor = xlThemeColorLight2
         .TintAndShade = -0.249977111117893
    End With
    Selection.Borders(xlDiagonalDown).LineStyle = xlNone
    Selection.Borders(xlDiagonalUp).LineStyle = xlNone
    Selection.Borders(xlEdgeLeft).LineStyle = xlNone
    With Selection.Borders(xlEdgeTop)
         .LineStyle = xlContinuous
         .ThemeColor = 4
         .TintAndShade = -0.249946592608417
         .Weight = xlThick
    End With
    With Selection.Borders(xlEdgeBottom)
         .LineStyle = xlContinuous
         .ThemeColor = 4
         .TintAndShade = -0.249946592608417
         .Weight = xlThick
    End With
    Selection.Borders(xlEdgeRight).LineStyle = xlNone
    Selection.Borders(xlInsideVertical).LineStyle = xlNone
    Selection.Borders(xlInsideHorizontal).LineStyle = xlNone
End Sub
```

lesson 3

Running a Macro

What You Will Learn

Learn how to run the macro that you created in Lesson 2.

To work along, use Lesson03.xlsm.

Notes from the Lesson

- You can run a macro using the Macro dialog. There are three ways to access this dialog: pressing Alt+F8, selecting View, Macros, View Macros, or selecting Developer, Macros.
- In the Macro dialog, select the macro and click Run. If there are too many items in the list, use the Macros In drop-down to filter the list.
- You can run the macro by pressing Ctrl+shortcut key or Ctrl+Shift shortcut. Use the Options button in the Macro dialog to assign a shortcut key to the macro.
- You can run a macro from an icon on the Quick Access Toolbar. Right-click the Quick Access Toolbar and choose Customize. Choose Macros from the top-left drop-down. Click on your macro, and then click the Add>> button in the center of the screen. Click the macro name in the right list box and click Modify. Choose a meaningful icon and type a tooltip in the Display Name dialog.
- New in Excel 2007, you can have the toolbar icon appear only when a certain workbook is open. Using the top-right drop-down while customizing the Quick Access toolbar offers a new option that will limit the icon for a certain workbook. Select the workbook from the drop-down before clicking the Add>> button.
- You can run a macro using a Forms button. From the Developer tab, choose Insert, Forms, Button. Draw a rectangle on your worksheet to draw the button. Right-click the button and choose Assign Macro.
- You can use any shape or clipart to run the macro. Right-click the shape or clipart on your worksheet and choose Assign Macro.

Tip

Right-click the Quick Access toolbar and choose Show Quick Access Toolbar Below the Ribbon. This will give you more room for icons to stretch across the full width of the Excel screen.

Caution

If your Button is surrounded by diagonal lines, you are in text edit mode. Click on the diagonal lines to change them to dots before using Ctrl+1 to format the button.

In Excel 2003

- To add an icon to a toolbar, select Tools, Customize. Go to the Commands tab. In the left list box, choose Macros. In the right list box, drag the smiley face to any existing toolbar. Do not close the Customize dialog, but right-click the new icon in the toolbar. You can click Assign Macro, and then Change Button Image to choose from 48 icons. If none of those icons are appropriate, click Edit Button Image to draw any icon.
- To access the Forms Button icon, select View, Toolbars, Forms.

Code from the Lesson

The code in Lesson 3 is the same as in Lesson 2.

lesson 4

Keys to Successful Recording

What You Will Learn

The macro recorder almost always fails. In this lesson, you learn the one setting that you can change to have your macros work 95% of the time instead of 5% of the time.

To work along, use Lesson04.xlsm.

Notes from the Lesson

- In the default state, the macro recorder hard codes the cell addresses where the macro should work. This rarely is what you want.
- Turn on Use Relative Reference while recording your macros. This will allow the macro recorder to record relative actions, such as "move down one row from where you started."

In Excel 2003

- Relative Recording is controlled by the second icon on the Stop Recording toolbar. If you cannot see the Stop Recording toolbar, select View, Toolbars, Stop Recording.
- The Relative Reference button on the Stop Recording toolbar is a toggle that remembers whether you previously turned on relative recording during this session. Relative Recording is on when the icon is surrounded by a box and a gold or gray color. You might have to click the icon a few times to figure out whether the icon is on or off.

Code from the Lesson

```
Sub FixOneRecord()
 '
 ' FixOneRecord Macro
 '
 ' Keyboard Shortcut: Ctrl+s
 '
    ActiveCell.Offset(1, 0).Range("A1").Select
    Selection.Cut
    ActiveCell.Offset(-1, 1).Range("A1").Select
    ActiveSheet.Paste
    ActiveCell.Offset(2, -1).Range("A1").Select
    Selection.Cut
    ActiveCell.Offset(-2, 2).Range("A1").Select
    ActiveSheet.Paste
    ActiveCell.Offset(1, 0).Rows("1:3").EntireRow.Select
    Selection.Delete Shift:=xlUp
    ActiveCell.Select
End Sub
```

lesson 5

Variable Number of Rows

What You Will Learn

Learn how to write a macro that will deal with a variable number of rows. If you record a macro to format the daily invoice file, that macro needs to work whether there are 40 or 90 records in the file.

Related Lessons

- In Lesson 23, "Finding the Final Row," you will learn how to use this concept in combination with a variable.

To work along, use Lesson05.xlsm.

Notes from the Lesson

- Data sets frequently have various numbers of records. To get to the bottom of a data set, press the End key followed by the down arrow key.
- If there is a chance that there are some blank cells in column A, you could use the GoTo dialog (press F5) to move to the last row in the worksheet, and then type **`End`** followed by the up arrow.

Code from the Lesson

```
Sub AddTotal()
  '
  ' AddTotal Macro
  '
  ' Keyboard Shortcut: Ctrl+Shift+T
  '
    Selection.End(xlDown).Select
    ActiveCell.Offset(1, 0).Range("A1").Select
    Selection.FormulaR1C1 = "Total"
End Sub
```

lesson 6

Adding Totals

What You Will Learn

There are two major obstacles to successful use of the macro recorder: relative recording and the AutoSum button. You learned about relative recording in Lesson 4. In this lesson, you learn why you should never touch the AutoSum icon while recording a macro.

To work along, use Lesson06.xlsm.

Notes from the Lesson

- Never use the AutoSum icon while recording a macro. Even if relative recording is turned on, the macro recorder will not record the concept of clicking the AutoSum button.
- If you are in cell D99 and want to sum from D2 to D98, type the formula **`=SUM(D$2:D98)`**. Notice that there is only a single dollar sign in that formula. When you type this formula, the macro recorder will record the concept that you want to sum from row 2 down to the row just above the active cell.

Tip

A dollar sign in a reference indicates that the next letter or number is fixed. D$2 is a reference where only row 2 is frozen.

Code from the Lesson

Note that in the unsuccessful first macro, the formula produced by the AutoSum is hard-coded to grab the 12 rows above the active cell:

```
Selection.FormulaR1C1 = "=SUM(R[-12]C:R[-1]C)"
```

In the successful second macro, the formula produced will always start at row 2 and extend down to the cell directly above the current cell:

```
Selection.FormulaR1C1 = "=SUM(R2C:R[-1]C)"
```

Note

You learn more about R1C1 style formulas in Lesson 35, "R1C1-Style Formulas."

```
Sub AddTotalsTake1()
 '
 ' AddTotalsTake1 Macro
 '
 ' Keyboard Shortcut: Ctrl+Shift+T
 '
    Selection.End(xlDown).Select
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "Total"
    With ActiveCell.Characters(Start:=1, Length:=5).Font
         .Name = "Arial"
         .FontStyle = "Regular"
         .Size = 10
         .Strikethrough = False
         .Superscript = False
         .Subscript = False
         .OutlineFont = False
         .Shadow = False
         .Underline = xlUnderlineStyleNone
         .ColorIndex = xlAutomatic
         .TintAndShade = 0
         .ThemeFont = xlThemeFontNone
    End With
    ActiveCell.Offset(0, 4).Range("A1:C1").Select
    Selection.FormulaR1C1 = "=SUM(R[-12]C:R[-1]C)"
    Range("A1:G1").Select
    Selection.Font.Bold = True
    Cells.Select
    Cells.EntireColumn.AutoFit
End Sub

Sub AddTotalsTake2()
 '
 ' AddTotalsTake2 Macro
 '
 ' Keyboard Shortcut: Ctrl+Shift+R
 '
    Selection.End(xlDown).Select
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "Total"
    With ActiveCell.Characters(Start:=1, Length:=5).Font
         .Name = "Arial"
         .FontStyle = "Regular"
         .Size = 10
         .Strikethrough = False
         .Superscript = False
         .Subscript = False
```

```
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .ThemeFont = xlThemeFontNone
    End With
    ActiveCell.Offset(0, 4).Range("A1").Select
    Selection.FormulaR1C1 = "=SUM(R2C:R[-1]C)"
    Selection.AutoFill Destination:=ActiveCell.Range("A1:C1"),  _
        Type:=xlFillDefault
    ActiveCell.Range("A1:C1").Select
    Range("A1:G1").Select
    Selection.Font.Bold = True
    Cells.Select
    Cells.EntireColumn.AutoFit
    Range("A1").Select
End Sub
```

lesson 7

Opening a Workbook with Macros

What You Will Learn

Microsoft made a fundamental change in macro security in Excel 2007, and it is not a good change. In previous versions, before people could work on the spreadsheet, they had to choose whether they would enable macros. Now, when you open a workbook with macros in Excel 2007, the notification about macros appears in the message bar. This allows people to work with the workbook without answering the "Enable Macros" question. In this lesson, you learn about this frustrating shortcoming and find a couple of workarounds.

To work along, use any workbook that has macros.

Notes from the Lesson

- Macros are not automatically enabled, and the person is not forced to choose whether they should enable the macros. Instead, a notice appears in the message bar (see Figure 1). It is easy to ignore this notice and to not have access to the macros in the file.

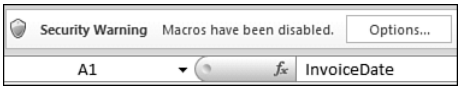


Figure 1 The old Enable Macros dialog has been replaced with an easy-to-ignore notification.

- One alternative is to store your macros in a trusted location. To set up a folder as trusted, go select the Office Button, Excel Options, Trust Center, Trust Center Settings, Trusted Locations, Add New Location. Workbooks stored in a trusted location will automatically have their macros enabled.
- Another alternative is to use Save As and save the file as an add-in. Macros in add-in files are automatically enabled.

In Excel 2003

- If you use Medium security, you must choose to enable or disable macros when you open the workbook.

PART II

Visual Basic Editor

Before you dive into Visual Basic coding, this part introduces your tools. The Visual Basic Editor is a robust programming environment, complete with many debugging tools.

There are seven lessons in this part:

8 Introducing the VBA Editor

9 VBA Help

10 Object Browser

11 Stepping Through Code

12 Immediate Window

13 The Watch Window

14 Breakpoints

Although all these tools are valuable, the Watch Window is the one that took me the longest to adopt and is a huge benefit. It is one of those things that you don't seem to find when you are self taught about macros.

lesson 8

Introducing the VBA Editor

What You Will Learn

Learn how to launch the Visual Basic Editor and how to configure it with the most commonly used panes.

To work along, use Lesson08.xlsm.

Notes from the Lesson

- To open the Visual Basic Editor, choose Developer, Visual Basic or press Alt+F11.
- The Project Explorer shows all open workbooks. Click the + sign next to a workbook to see all worksheets and modules. To display the Project Explorer select View, Project Explorer or press Ctrl+R.
- The Properties window is essential when you are creating userforms. Press F4 to display the Properties window.
- The code pane takes up most of the Visual Basic window and is where you actually write your Excel code. In the Project Explorer, right-click any module and choose View Code.

In Excel 2003

- Access the Visual Basic Editor using Tools, Macro, Visual Basic Editor.

Code from the Lesson

The code in this lesson is the same as in Lesson 6.

lesson 9

VBA Help

What You Will Learn

The VBA Help module is a must-have resource for writing VBA. This lesson shows you how to get the most from VBA Help.

To work along, use any workbook.

Notes from the Lesson

- By default, VBA Help is not installed. If you want to write macros, you need to have the VBA Help installed.
- To find out whether VBA Help is installed open Excel. Switch to the VBA Editor by pressing Alt+F11, and then select Insert, Module. In the code pane, type the word **`Offset`**. Click the word Offset and press F1. If you do not get a couple of Offset topics in Help, you need to find the original installation CDs and do a complete install.
- To access Help, click inside any word that is a VBA property, method, object, or collection. Then press F1 to bring up the topic for that word.
- When you press F1, you might have to choose between two related topics; otherwise, you will be taken directly to the proper topic.
- Many Help topics provide code snippets. You can copy the code by pressing Ctrl+C, and then paste to your VBA project using Ctrl+V.

lesson 10

Object Browser

What You Will Learn

Learn how to use the Object Browser to learn about every object, property, and method in Excel VBA.

To work along, use any workbook.

Notes from the Lesson

- The Object Browser is a virtual guide to every VBA object in Excel. To open the Object Browser, press the F2 key. You can also select View, Object Browser.
- Change the top-left drop-down from All Libraries to Excel.
- Type a word such as **`Worksheet`** in the search box, and click the binoculars. You will see a list of all properties and methods.
- Watch for the icons to the left of the words in the Members list. In Figure 2, the icon next to Calculate indicates a method—these are actions you can perform on a worksheet. That icon reminds me of a green flying box. The icon next to Cells is a hand pointing to a sheet. This is a property of the worksheet. The lightning bolt icon indicates an event. You can capture an event and write code to handle it. To learn more about Event macros, watch Lesson 42, "Event Handler Macros."

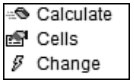


Figure 2 From the top, icons for methods, properties, and events.

lesson 11

Stepping Through Code

What You Will Learn

Learn how to run your macro one line at a time so you can see the results of the macro in slow motion.

Related Lessons

- R1C1 formulas are explained in Lesson 35, "R1C1-Style Formulas."

To work along, use Lesson11.xlsm.

Notes from the Lesson

- You can arrange your screen so that you can see both the VBA Editor and the Excel window at the same time. If you do this, you can actually watch the macro recorder write code in response to your actions in Excel.
- While recording a macro, if you perform an action in Excel, it might generate one, two, or more lines of code in the VBA window.
- To run your macro one line at a time, switch to VBA, click inside the macro code, and select Debug, Step Into. Alternatively, press F8.
- While stepping through code, the line in yellow is the line that is about to be executed in Excel. Press F8 to run that line of code and to move the yellow line to the next row.
- If you use `Selection.Offset(1, 0).Select`, you will move the cell pointer down one row; the numbers indicate the number of rows and the number of columns. If you specify `Offset(-2, 1)`, you will move up two rows and to the right by one column.
- Don't forget to keep pressing F8 until the macro is completely finished. If you leave the macro in break mode, you will not be able to run other macros.
- To stop stepping through the code, press the square Reset icon (this icon is below the word Run in the VBA menu). To allow the rest of the code to run without seeing each individual line, click the Continue icon (this is a green right-facing triangle underneath the word Debug).
- To rerun a few lines of code, select the yellow arrow to the left of the current line in the code window. Drag that arrow to another line (either up or down).

Tip

Figure 3 shows what I call the "VCR buttons." The first icon will run a macro or continue a macro when you are in break mode. The second icon will pause a macro. The third icon will stop a macro that is in break mode. When you encounter an error and click Debug, you will have to click the third button to exit break mode.

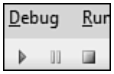


Figure 3 Run, Pause, and Stop are frequently used icons in the VBA Editor.

Code from the Lesson

```
Sub AddSummaryTable()
 '
 ' AddSummaryTable Macro
 '
 ' Keyboard Shortcut: Ctrl+Shift+T
 '
    Selection.End(xlDown).Select
    ActiveCell.Offset(2, 3).Range("A1").Select
    ActiveCell.FormulaR1C1 = "Central"
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "East"
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "West"
    ActiveCell.Offset(-2, 1).Range("A1").Select
    Selection.FormulaR1C1 = _
        "=SUMIF(R2C[-3]:R[-2]C[-3],RC[-1],R2C:R[-2]C)"
    Selection.AutoFill Destination:=ActiveCell.Range("A1:A3"), _
        Type:=xlFillDefault

    Range("A1").Select
End Sub
```

lesson 12

Immediate Window

What You Will Learn

Learn how to use the Immediate Window to learn the current state of anything. In this lesson, you will record some additional code to append to the end of the Lesson 11 code.

To work along, use Lesson12.xlsm.

Notes from the Lesson

- To open the Immediate Window while in the VBA Editor, press Ctrl+G or click View, Immediate Window.
- In the Immediate Window, type the word **`Print`** and then anything. For example, type **`Print ActiveSheet.Name`**.
- It is possible to type a single command in the Immediate Window and have that run. For example, **`ActiveCell.value = 17`**.
- In your macro code, you can use `Debug.Print Anything` to print something to the Immediate Window.

Tip

If you want to remove a line of code, consider commenting it out instead. To comment out a line of code, type an apostrophe before the first character in the code.

Code from the Lesson

```
Sub AddSummaryTable()
 '
 ' AddSummaryTable Macro
 '
 ' Keyboard Shortcut: Ctrl+Shift+T
 '
    Selection.End(xlDown).Select
    ActiveCell.Offset(2, 3).Range("A1").Select
    ActiveCell.FormulaR1C1 = "Central"
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "East"
```

```
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "West"
    ActiveCell.Offset(-2, 1).Range("A1").Select
    Selection.FormulaR1C1 = _
        "=SUMIF(R2C[-3]:R[-2]C[-3],RC[-1],R2C:R[-2]C)"
    Selection.AutoFill Destination:=ActiveCell.Range("A1:A3"), _
        Type:=xlFillDefault
    Selection.Offset(0, -1).Resize(3, 2).Select
    ' Range("A1").Select

    ' This is the second recorded macro. It should copy the
    ' table, paste as values, add a worksheet, copy the table
    ' there, and then move the new worksheet to a new workbook
    Selection.Copy
    Selection.PasteSpecial Paste:=xlPasteValues,  _
        Operation:=xlNone, SkipBlanks:=False, Transpose:=False
    Sheets.Add After:=Sheets(Sheets.Count)
    Sheets("Sheet1").Select
    Application.CutCopyMode = False
    Selection.Copy
    ' The code will not work from this point onwards
    Sheets("Sheet2").Select
    ActiveSheet.Paste
    Sheets("Sheet2").Select
    Application.CutCopyMode = False
    Sheets("Sheet2").Copy

End Sub
```

The lesson mentions that the final few lines of the macro would benefit from the skills learned in Lesson 17, "Using Object Variables." After you learn about variables, the previous code would be streamlined as follows:

```
Sub AddSummaryTableFixed()
Dim WSO as Worksheet
Dim WSN as Worksheet
 '
 ' AddSummaryTable Macro
 '
 ' Keyboard Shortcut: Ctrl+Shift+T
 '
    FinalRow = Selection.End(xlDown).Row
    Cells(FinalRow + 2, 3) = "Central"
    Cells(FinalRow + 3, 3) = "East"
    Cells(FinalRow + 4, 3) = "West"
    Cells(FinalRow + 2, 4).Resize(3, 1). _
        FormulaR1C1 = _
        "=SUMIF(R2C1:R" & FinalRow & "C1,RC[-1],R2C4:R" & _
```

```
        FinalRow & "C)"

    ' This is the second recorded macro. It should copy the
    ' table, paste as values, add a worksheet, copy the table
    ' there, and then move the new worksheet to a new workbook
    Cells(FinalRow + 2, 4).Resize(3, 1).Value = _
            Cells(FinalRow + 2, 4).Resize(3, 1)
    Set WSO = ActiveSheet
    Set WSN = Sheets.Add(After:=Sheets(Sheets.Count))
    WSO.Cells(FinalRow + 2, 4).Resize(3, 1).Copy _
        Destination:=WSN.Cells(1, 1)
    Application.CutCopyMode = False
    WSN.Copy

End Sub
```

lesson 13

The Watch Window

What You Will Learn

Learn how to use the Watch Window to watch the current value of a variable or a property as a macro is running.

To work along, use any workbook.

Notes from the Lesson

- To set up a watch, right-click on a variable or a property and choose Add Watch. Click OK in the Watch Expression dialog. As you step through a macro, you can see the value of the watched cell or property.
- If you set up a watch for an object variable, a plus sign will appear next to the variable in the Watch Window. Click the plus sign to see all the properties associated with that object.
- In the Add Watch dialog, select Break When Value Changes. You can then run the macro without stepping through. As soon as the variable changes, the macro will stop and you are back in break mode.

Code from the Lesson

```
Sub AddSummaryTable()
'
' AddSummaryTable Macro
'
' Keyboard Shortcut: Ctrl+Shift+T
    Dim WBO As Workbook
    Dim WBN As Workbook
    Dim WSO As Worksheet
    Dim WSN As Worksheet
    Set WBO = ThisWorkbook
    Set WSO = ActiveSheet

    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    For i = FinalRow To 2 Step -1
        If Cells(i, 3).Value = "S99" Then
```

```
            Cells(i, 3).EntireRow.Delete
        End If
    Next i

    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    TableRow = FinalRow + 2
    Cells(TableRow, 4).Value = "Central"
    Cells(TableRow + 1, 4).Value = "East"
    Cells(TableRow + 2, 4).Value = "West"

    Cells(TableRow, 5).Resize(3, 1).FormulaR1C1 = _
        "=SUMIF(R2C[-3]:R" & FinalRow & "C[-3],RC[-1],R2C:R" _
        & FinalRow & "C)"
    Cells(TableRow, 5).Resize(3, 1).Value = _
        Cells(TableRow, 5).Resize(3, 1).Value

    Set WBN = Workbooks.Add(xlWBATWorksheet)
    Set WSN = WBN.Worksheets(1)
    WSO.Cells(TableRow, 5).Resize(3, 1).Copy Destination:=WSN.Cells(1, 1)

End Sub
```

lesson 14

Breakpoints

What You Will Learn

Learn how to add breakpoints to your macro. This is useful when you need to debug something that occurs near the end of a macro.

To work along, use Lesson14.xlsm.

Notes from the Lesson

- Having to press F8 to get near the end of a macro can be tedious, particularly if you need to press F8 to get through a loop that loops through hundreds or thousands of rows.
- One debugging tool is called Run to Cursor. Click the cursor in any line late in the macro, and then click Debug, Run to Cursor. Excel will run all the lines of code up to but not including the line on which the cursor appears.
- Another method is to click in the gray bar to the left of a line of code. This will cause a large brown dot to appear in the gray area and the line is highlighted in brown. This line is now a breakpoint. When you run the code, Excel will enter break mode just before executing the line of code in the breakpoint.
- To clear a breakpoint, click the brown dot again.
- Using a breakpoint inside an `If` block will allow you to test whether the code eventually makes it into the `If` block.
- Before sending your code to others, always select Debug, Clear All Breakpoints to clear out any breakpoints you might have forgotten to clear.

Code from the Lesson

The code in this lesson is the same as in Lesson 13, “The Watch Window.”

PART III

VBA Syntax

The lessons in this part teach you about the various parts of VBA speech. You will learn two things the macro recorder will never use: variables and a better way to refer to cells.

There are five lessons in this part:

Using `CELLS()` instead of `RANGE()` as described in Lesson 18 is one of the powerful tools that is hard to learn when you are teaching yourself VBA.

lesson 15

VBA Parts of Speech

What You Will Learn

This lesson shows you the basics of VBA syntax.

To work along, use any workbook.

Notes from the Lesson

- VBA is an object-oriented language, which means the nouns come first. While you might say "Eat Dinner" in English, in VBA, you would say "Dinner.Eat." The typical construct is *Noun.Verb*. When you look at code and see a dot, the dot is inferring some action—the second word is going to act upon the first word. In the language of VBA, Noun.Verb is known as Object.Method.
- Be careful to watch for nouns with an "S" on the end. A plural noun is not referring to a single object, but to a collection of all similar objects. For example, Worksheets refers to a collection of all worksheets in the workbook.
- To refer to one item in a collection, use `Worksheets(1)` to refer to the first worksheet. However, this can cause a problem if someone drags a new worksheet into the number 1 position. A safer way is to use `Worksheets("Balance Sheet")`.
- Adverbs tell the verb how to act. You can tell it is an adverb when you see the colon-equals construct; for example: `ActiveSheet.Copy After:=Worksheets("Income")`. In the language of VBA, adverbs are called *parameters*.
- It is confusing, but if you know the exact order in which the parameters should appear, you can skip the name of the adverb. Although this will save you some typing, you should avoid this practice.
- The final part of speech is an adjective. You will see a dot separating the noun and the adjective, but then the adjective is followed by an equal sign. In the language of VBA, an adjective is a property. In the following line of code, `ActiveCell` is the object and `Formula` is the property: `ActiveCell.Formula = "=A2*2"`.

lesson 16

Using Regular Variables

What You Will Learn

Learn how to use variables in your macro code. The macro recorder will never record a variable for you. Variables are a key learning block as you move from recorded code to code that you write.

Related Lesson

- While this lesson talks about regular variables, Lesson 17, "Using Object Variables," introduces a super-variable called object variables.

To work along, use Lesson16.xlsm.

Notes from the Lesson

- A regular variable holds a single value.
- In old programming languages, variables were given names like x, y, j, k, but in VBA you can use meaningful variables names such as ServiceAmt, ThisRow, and so on.
- To assign a variable, use an equal sign: `FinalRow = ActiveCell.Row`.
- To add a comment, type an apostrophe followed by your comment. Comments will appear in green. Comments help you understand the purpose of a variable when you come back to look at the code months later.
- To assign a shortcut key to a macro that you've typed, select Developer, Macros, Options.
- When the second macro provides a wrong answer, this lesson walks through using the debugging tools from Part II to solve the problem.

In Excel 2003

- To assign a shortcut key to a macro select Tools, Macro, Macros, Options.

Code from the Lesson

```
Sub CommissionPlanA()
     ThisRow = ActiveCell.Row
     ThisAmt = Range("E" & ThisRow)
     CommissionAmt = ThisAmt * 0.015
     Range("F" & ThisRow) = CommissionAmt
End Sub

Sub CommissionPlanS()
     ThisRow = ActiveCell.Row
     ThisAmt = Range("E" & ThisRow)
     ServiceAmt = Range("E" & ThisRow - 1)
     ' Figure out the total amount less the service amount
     WhatsLeft = ThisAmt - ServiceAmt
     CommissionAmt = ServiceAmt * 0.25 + WhatsLeft * 0.015
     Range("F" & ThisRow) = CommissionAmt
End Sub
```

lesson 17

Using Object Variables

What You Will Learn

Learn how to use super-variables known as *object variables*.

To work along, use Lesson17.xlsm.

Notes from the Lesson

- Object variables are super-variables that hold more than one value. They represent all values associated with an object.
- To set up the object variable, use `Set VariableName = SomeObject`; for example, `Set WS = ActiveSheet`.
- Object variables behave better in the VBA Editor if you properly declare them before you use them. To declare a variable, use the `Dim` statement. The syntax is `Dim VariableName As ObjectType`. If you declare a variable, the AutoComplete drop-down will appear offering methods and properties associated with the object variable.

```
Dim MyRange as Range
```

- If you use `Set MyRange = Selection`, the variable is equal to the selection at the time that line of code runs. If something later in the macro changes the selection, the `MyRange` variable will continue to point to the original selection. This is a good way to preserve the original state of the selection.
- If you have several lines of code that all refer to the same object, you can surround that code with the `With...End With` construct. This way VBA has to figure out the object only once. For example, these lines of code are equivalent:

```
ThisCell.Value = 17
ThisCell.Font.Name = "Arial"
ThisCell.Interior.ColorIndex = 2
ThisCell.Font.ColorIndex = 4
```

and

```
With ThisCell
     .Value = 17
     .Font.Name = "Arial"
```

```
        .Interior.ColorIndex = 2
        .Font.ColorIndex = 3
    End With
```

- This lesson shows an example of copying a range of cells to different worksheets in the workbook.

Code from the Lesson

```
Sub Lesson17()
    Dim MyRange As Range

    Set MyRange = Selection
    With MyRange
        RowCount = .Rows.Count
        FirstRow = .Rows(1).Row
    End With

End Sub

Sub CreateReport()
    ' Copy the selected range to the Report worksheet
    Dim WSD As Worksheet
    Dim WSR As Worksheet

    Set WSD = Worksheets("Data")
    Set WSR = Worksheets("Report")

    ' Clear out the last report
    WSR.Cells.Clear

    ' Copy the selection to A4 on Report
    Selection.Copy Destination:=WSR.Range("A4")

    ' Copy the headings over
    WSD.Range("A1:E5").Copy Destination:=WSR.Range("A3")

    With WSR
        .Columns.AutoFit
        .Range("A1").Value = _
            "Custom Report from the Data Worksheet"
    End With

    WSR.Select

End Sub
```

lesson 18

Better Ways to Refer to Cells

What You Will Learn

Using `Range("C4")` is a difficult method for referring to cells when you are using variables. In this lesson, you will learn how to use Cells, Resize, and Offset.

To work along, use Lesson18.xlsm.

Notes from the Lesson

- `Range("A10")` is equivalent to `Cells(10, 1)`. Note that in the `Cells` collection, you must specify the row number first, followed by the column number.
- If `ThisRow` is 5 and `ThisColumn` is 4, it is very hard to use `Range` to refer to the intersection of `ThisRow` and `ThisColumn`. The lesson suggests that you might try using `Mid` and the alphabet to convert column 4 to a D; however, this gets to be tedious when you have 16,384 columns in Excel 2007. Instead, use `Cells(ThisRow, ThisColumn)` to solve the problem.
- `Range` can refer to a rectangular block of cells; for example, `Range("A10:D14")`. To replicate this using `Cells`, point to the top-left corner cell, and then use `.Resize`. Again, the parameters for `.Resize` are the number of rows and the number of columns. The equivalent reference to `Range("A10:D14")` would be `Cells(10, 1).Resize(5, 4)`. This says that you want to start in row 10, column 1 and include five rows by four columns.
- You can use `.Offset` to move down a certain number of rows and right a certain number of columns from a starting point. If you have a variable such as `Set ThisCell = Selection` and need to refer to the cell three cells to the right, you could use `ThisCell.Offset(0, 3)`.
- Negative numbers in `Offset` refer to rows above or columns to the left. `Range("J10").Offset(-3,-1)` would point to cell I7 because it is three rows above and one column to the left.

Code from the Lesson

```
Sub ColorAnotherCellOriginal()
'
' ColorAnotherCell Macro
'
' Keyboard Shortcut: Ctrl+k
'
```

```
    ThisRow = ActiveCell.Row
    ColorRow = ThisRow + 3
    ThisColumn = ActiveCell.Column
    ColorColumn = ThisColumn + 3

    With Cells(ColorRow, ColorColumn).Interior
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
        .Color = 65535
        .TintAndShade = 0
        .PatternTintAndShade = 0
    End With
End Sub

Sub ColorAnotherCellFinal()
 '
 ' ColorAnotherCell Macro
 '
 ' Keyboard Shortcut: Ctrl+k
 '

    With Selection.Offset(3, 3).Interior
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
        .Color = 65535
        .TintAndShade = 0
        .PatternTintAndShade = 0
    End With
End Sub
```

lesson 19

To Declare or Not To Declare?

What You Will Learn

Some people say you should always declare your variables. This is optional in VBA. This lesson shows you how to declare variables and discusses the pros and cons of declaring.

This lesson also introduces the concept of scoping your variables. This will allow the value of the variable to be remembered after the code finishes.

To work along, use Lesson19.xlsm.

Notes from the Lesson

Here are reasons why you might want to declare your variables:

- Save a tiny bit of space in memory. If you declare a variable as an integer instead of as a variant, you will save 5 bytes in memory. With today's computers having 1, 2, or 4GB of memory, this hardly seems like a good reason any more.
- Avoid spelling errors. If you declare a variable as `FinalRow` and later type `finalrwo`, you will notice that the variable did not change to uppercase when you go to a new line. If you don't declare the variable, you will always have to capitalize the F and R in `FinalRow` or every occurrence of the variable in the macro will switch to `finalrow`.
- AutoComplete will work for your object variables if you declare them as the proper object type.
- Requiring variable declaration means that you must figure out which variables you are going to use. It encourages you to plan ahead. If you ever had a college professor who made you flowchart your programs before writing code, that professor would suggest you use variable declaration.

Here is one reason to not declare variables:

- If you don't declare your variables, you can create new variables on the fly as you write your code. This is how I usually write my programs.

Tip

If you are a college professor and you want to force your students to declare their variables, have them follow these steps: In the VBA Editor, choose Tools, Options. Choose the setting for Require Variable Declaration. Close Excel and reopen it. Every new module will now start with the line `Option Explicit`.

Variable declaration usually happens as the first lines of code in a program. You might have a variable declaration block like this:

```
Sub MyMacro()
Dim FinalRow as Long
Dim WSO as Worksheet
Dim WBO as Workbook
```

This lesson also introduces the concept of the scope of a variable. Most variables are in scope only while that macro is running. If you want a variable to be available to other macros in the same module, move the `Dim` statement to before the first macro in the module:

```
Dim ThisRow as Long
Sub MyMacro()
```

If you want your variable to be available to every module in the workbook, change the word `Dim` to `Public`:

```
Public ThisRow as Long
Sub MyMacro()
```

Code from the Lesson

Code in ModDim:

```
Option Explicit

Sub TestMacro()
    Dim ThisRow As Long
    MsgBox ThisRow
    ThisRow = ActiveCell.Row
    ThisRow = 5
    ThisRow = ActiveCell.Row
End Sub
```

Code in ModScope:

```
Public ThisRow As Long

Sub MacroA()
    ThisRow = ActiveCell.Row
    TestMacro
End Sub

Sub MacroB()
    MsgBox "You are in row " & ThisRow
End Sub
```

PART IV

Making Decisions

Through the first three parts, the macros have been designed to always carry out a particular task. In this part, you learn how to make decisions while the macro is running. You can do one task for some records and another task for other records.

There are three lessons in this part:

20 `If…Then`

21 `If…Then…Else`

22 `Select Case`

Although I use `If` frequently, there are times when the `Select Case` logic in Lesson 22 will simplify what would be an otherwise complicated macro.

lesson 20

If…Then

What You Will Learn

Learn how to use an `If…Then` statement to run a line of code only if a certain condition is true.

To work along, use Lesson20.xlsm.

Notes from the Lesson

- If you have only one line of code to run as the change, you can use `If LogicalTest then LineOfCode`. This statement would all appear on a single line.
- It is much more common to have several lines of code that must be run if the logical test is true. In that case, you would put the `If` on line 1, have many lines of code, and then finish up with a line called `End If`:

```
If LogicalTest Then
    LineOfCode
    LineOfCode
    LineOfCode
End If
```

Code from the Lesson

```
Sub CommissionPlanS()
    ThisRow = ActiveCell.Row
    ThisAmt = Range("E" & ThisRow)
    If Cells(ThisRow - 1, 2).Value = "Z" Then
        ServiceAmt = Range("E" & ThisRow - 1)
        WhatsLeft = ThisAmt - ServiceAmt
    End If
    ' Figure out the total amount less the service amount
    CommissionAmt = ServiceAmt * 0.25 + WhatsLeft * 0.015
    Range("F" & ThisRow) = CommissionAmt
End Sub
```

lesson 21

If…Then…Else

What You Will Learn

Learn how to use the `If` concept introduced in Lesson 20 and extend it with `ElseIf` or `Else` blocks of code.

To work along, use Lesson21.xlsm.

Notes from the Lesson

- If something is true you want to do one thing; otherwise, you want to do something else. In this situation, you can add an `Else` block or multiple `ElseIf` blocks to your code.
- You will have only one `End If` statement for the `If…Then`, `ElseIf`, `ElseIf`, …, `ElseIf`, `Else` construct.

Code from the Lesson

```
Sub CategorizeRevenue()
    ThisRow = ActiveCell.Row
    ThisAmt = Range("E" & ThisRow)
    ThisProd = Cells(ThisRow, 2)

    If ThisProd = "A" Then
        Cells(ThisRow, 6).Value = ThisAmt
    ElseIf ThisProd = "B" Then
        Cells(ThisRow, 6).Value = ThisAmt
    ElseIf ThisProd = "C" Then
        Cells(ThisRow, 7).Value = ThisAmt
    ElseIf ThisProd = "D" Then
        Cells(ThisRow, 7).Value = ThisAmt
    Else
        Cells(ThisRow, 8).Value = ThisAmt
    End If

End Sub
```

lesson 22

Select Case

What You Will Learn

Learn how to replace multiple `ElseIf` statements with a new construct called `Select Case`.

To work along, use Lesson22.xlsm.

Notes from the Lesson

- Use the `Select` block to replace multiple `ElseIf` constructs.
- Start the block with `Select Case` *`VariableName`*. End the block with `End Select`. In between, you will have one or more subblocks that follow this syntax:

```
Case value[, value]
     Code to run if VariableName is value.
     Code to run if VariableName is value.
     Code to run if VariableName is value.
```

- To handle the situation in which *`VariableName`* is something that you did not count on, you can have one final subblock with `Case Else` and the code to run in that case.

Code from the Lesson

```
Sub CategorizeRevenue()
    ThisAmt = Range("E" & Selection.Row)

    Select Case Cells(Selection.Row, 2)
        Case "Z"
            Cells(Selection.Row, 8).Value = ThisAmt
        Case "A", "B"
            Cells(Selection.Row, 6).Value = ThisAmt
        Case "C", "D"
            Cells(Selection.Row, 7).Value = ThisAmt
    End Select

End Sub

Sub OldMacro()
```

```
    ThisRow = ActiveCell.Row
    ThisAmt = Range("E" & ThisRow)
    ThisProd = Cells(ThisRow, 2)

    If ThisProd = "A" Then
        Cells(ThisRow, 6).Value = ThisAmt
    ElseIf ThisProd = "B" Then
        Cells(ThisRow, 6).Value = ThisAmt
    ElseIf ThisProd = "C" Then
        Cells(ThisRow, 7).Value = ThisAmt
    ElseIf ThisProd = "D" Then
        Cells(ThisRow, 7).Value = ThisAmt
    Else
        Cells(ThisRow, 8).Value = ThisAmt
    End If

End Sub
```

PART V

Looping

Loops will never be recorded by the macro recorder, yet they are a critical component of macros in Excel VBA. If you've had a class in BASIC before, you might be familiar with the `For...Next` loop. VBA offers this loop plus many more.

There are seven lessons in this part:

23 Finding the Final Row

24 Basic Looping

25 Loop Example

26 Deleting While Looping

27 Every Other Row Loops

28 Other Legacy Loops

29 `For Each` Loops

My favorite loop in this part is the `For Each` loop in Lesson 29.

lesson 23

Finding the Final Row

What You Will Learn

Learn how to locate the last row in a data set. This will allow you to deal with situations in which you might have differing numbers of data each day.

To work along, use Lesson23.xlsm.

Notes from the Lesson

- Finding the last row of data is critical for processing data sets that have a different number of rows each day. Conceptually, this is equivalent to going to cell A1 and pressing the End key followed by the down arrow key. In VBA this is `Range("A1").End(xlDown).Row`.
- However, if there was a blank cell in the middle of column A, the previous command would not find the true last row of data. Even if you don't expect your data to have blanks, there will come a day in the next five years where someone manages to get a blank in the data. The solution is to start at the bottom of the worksheet, press the End key, and then press the up arrow.
- Starting in the last row is complicated because the last row in Excel 97 through 2003 is 65,536. The last row in Excel 2007 is 1,048,576, unless you are in compatibility mode—then it is 65,536. Luckily, if you ask Excel for `Rows.Count`, it will tell you the number of rows in the current workbook. Thus, rather than hard-coding `Range("A65536").End(xlUp).Row`, you can use `Cells(Rows.Count, 1).End(xlUp).Row`.
- The first line of code in most macros that I write is
  ```
  FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
  ```
- The equivalent code to find the last column in a data set would be
  ```
  FinalCol = Cells(1, Columns.Count).End(xlToLeft).Column
  ```

Code from the Lesson

```
Sub FindLastRow()
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    FinalCol = Cells(1, Columns.Count).End(xlToLeft).Column

    MsgBox "The last row today is " & FinalRow
End Sub
```

lesson 24

Basic Looping

What You Will Learn

Learn how to repeat a section of code several times. This is useful when you want to perform some action on each record in a data set.

To work along, use Lesson24.xlsm.

Notes from the Lesson

- Suppose you want to repeat some code 100 times. You must choose a control variable. In the following code, the variable `x` is the control variable:

```
For x = 1 to 100
' Lines of code to repeat here
' Lines of code to repeat here
' Lines of code to repeat here
Next x
```

- Here is the cool part: While VBA is running code inside the loop, you can use the control variable in your code. So, if you want to look at a different row each time, you could refer to `Cells(x, 3)` to look at column C of each row from 1 to 100.

Caution

If you refer to the control variable after the loop, you might find that it is incremented to one more than the last value in the loop. After the loop above, `x` will have the value of 101.

- If you need to loop through all the records, you might set up the loop to run from 2 to `FinalRow`:

```
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
For x = 2 to FinalRow
' Lines of code to repeat here
Next x
```

- Technically, the `Next` line doesn't need to have the control variable, but it is common to put `Next x` instead of just `Next`. This is handy when you have a loop within a loop as shown in the Loop2 macro next.

Code from the Lesson

```
Sub FirstLoop()
    FinalRow = Cells(Rows.Count, 2).End(xlUp).Row
    For i = 1 To FinalRow
        Select Case Cells(i, 2)
            Case "Apple", "Cherry"
                Cells(i, 3) = "Red"
            Case "Orange"
                Cells(i, 3) = "Orange"
            Case "Banana"
                Cells(i, 3) = "Yellow"
            Case "Eggplant"
                Cells(i, 3) = "Purple"

        End Select

    Next i

End Sub

Sub Loop2()
    For i = 5 To 15
        For j = 5 To 10
            Cells(i, j).Value = i * j
        Next j
    Next i
End Sub
```

lesson 25

Loop Example

What You Will Learn

In this lesson, you will see how to use the loop concept from Lesson 24 in a real-life example. This lesson combined concepts from Part IV and Part V to process a report.

To work along, use Lesson25.xlsm.

Notes from the Lesson

- The macro starts out by finding the FinalRow in today's data set.
- Plan out your macro by writing comments before you start coding.
- If you need to fill a large range with zeroes, you can do it in a single line of code. Set the `.Value` property of the entire range to 0.
- The lesson shows three equivalent ways to enter the totals. Initially, the code had one line per total column:

```
Cells(TotalRow, 5).Formula = "=SUM(E2:E" & FinalRow & ")"
Cells(TotalRow, 6).Formula = "=SUM(F2:F" & FinalRow & ")"
Cells(TotalRow, 7).Formula = "=SUM(G2:G" & FinalRow & ")"
Cells(TotalRow, 8).Formula = "=SUM(H2:H" & FinalRow & ")"
```

  Next, the lesson suggested entering one formula and then copying:

```
Cells(TotalRow, 5).Formula = "=SUM(E2:E" & FinalRow & ")"
Cells(TotalRow, 5).Copy Destination:=Cells(TotalRow, 6).Resize(1, 3)
```

  Finally, the lesson switched over to an R1C1 style formula where you can enter the formula in one line:

```
Cells(TotalRow, 5).Resize(1, 4).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
```

  For more about R1C1 formulas, watch Lesson 35, "R1C1-Style Formulas."
- Use flow control to perform a different action depending on the values in that row

Code from the Lesson

```
Sub FormatReport()
    ' Figure out where the last row is
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row

    ' Fill in F G & H with zeroes
    Cells(1, 6).Resize(FinalRow, 3).Value = 0
    Cells(1, 6) = "A/B"
    Cells(1, 7) = "C/D"
    Cells(1, 8) = "Z"

    ' loop through all rows from 2 to Final Row
    For i = 2 To FinalRow
        Select Case Cells(i, 2)
        ' check to see which product we have & copy revenue
            Case "A", "B"
                Cells(i, 6) = Cells(i, 5)
            Case "C", "D"
                Cells(i, 7) = Cells(i, 5)
            Case "Z"
                Cells(i, 8) = Cells(i, 5)
        End Select
    Next i

    ' Add some totals
    TotalRow = FinalRow + 1
    Cells(TotalRow, 1) = "Total"

    Cells(TotalRow, 5).Resize(1, 4).FormulaR1C1 = _
        "=SUM(R2C:R[-1]C)"

    ' Add a title in row 1
    Range("A1:A2").EntireRow.Insert
    Range("A1") = "Product Report for Today"

End Sub
```

lesson 26

Deleting While Looping

What You Will Learn

Learn about a problem that occurs when you need to delete records within your loop. This lesson will show you a simple workaround.

To work along, use Lesson26.xlsm.

Notes from the Lesson

- Here is the problem. Suppose you are looking at each row from 2 to FinalRow. On one pass through the loop, you are looking at row 4. You decide to delete row 4 in the macro. The data that used to be in row 5 moves up to row 4. However, in the next pass through the code the loop will automatically move ahead and start looking at row 5. Thus, the data that moved from row 5 to row 4 will never be checked.
- The solution is to run your loop from the bottom to the top. You can do this by adding a new clause to the end of the first line of your loop: `Step -1`.

  ```
  For x = FinalRow to 2 Step -1
  ```

- While most loops don't include a `Step` clause, the Excel default says that if you don't specify a `Step`, it will automatically use a positive 1.

Code from the Lesson

```
Sub DeleteZProducts()
    ' Figure out where the last row is
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row

    ' loop through all rows from 2 to Final Row
    For i = FinalRow To 2 Step -1
        If Cells(i, 2) = "Z" Then
            Cells(i, 1).EntireRow.Delete
        End If
    Next i

End Sub
```

lesson 27

Every Other Row Loops

What You Will Learn

Learn how to use the `Step` clause from Lesson 26 to solve other problems.

To work along, use Lesson27.xlsm.

Notes from the Lesson

- Use `Step` when moving forward through a data set to skip rows in a pattern.
- To color every other row, use `Step 2`.
- To select every twentieth record, use `Step 20`.
- A common problem happens during the course of this lesson: You need to figure out how to color a row red. My solution is very common; turn on the macro recorder and record a short macro that shows you how to change a row color to red.

Code from the Lesson

Code in Module1:

```
Sub ColorEveryOtherRowRed()
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row

    For i = 3 To FinalRow Step 3
        ' color this row red
        With Cells(i, 1).Resize(1, 5).Interior
            .Pattern = xlSolid
            .PatternColorIndex = xlAutomatic
            .Color = 255
            .TintAndShade = 0
            .PatternTintAndShade = 0
        End With

    Next i

End Sub
```

```
Sub ChooseAuditRows()
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row

    For i = 2 To FinalRow Step 7
        Cells(i, 6).Value = "Audit!"
    Next i

End Sub

Sub FixUglyData()
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    NextRow = 2

    For i = 1 To FinalRow Step 5
        ' rearrange the data starting in G
        Cells(NextRow, 7) = Cells(i, 2)
        Cells(NextRow, 8) = Cells(i + 1, 3)
        Cells(NextRow, 9) = Cells(i + 2, 3)
        Cells(NextRow, 10) = Cells(i + 2, 5)
        Cells(NextRow, 11) = Cells(i + 3, 3)
        NextRow = NextRow + 1
    Next i

End Sub
```

Code in Module2:

```
Sub HowTheHeckDoIDoRed()
 '
 ' HowTheHeckDoIDoRed Macro
 '
    With Selection.Interior
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
        .Color = 255
        .TintAndShade = 0
        .PatternTintAndShade = 0
    End With
End Sub
```

lesson 28

Other Legacy Loops

What You Will Learn

Learn about other loops such as `Do Until`, `Do While`, and `While`.

To work along, use Lesson28.xlsm.

Notes from the Lesson

- VBA supports other loops that you might have encountered in old programming languages such as Pascal.
- **`Do…Loop` with `Exit Do`:** Start with `Do` and end with `Loop`. The code between these lines will be repeated infinitely. Unless you are planning on breaking this loop using Ctrl+Break, you should have an `IF` statement somewhere inside the loop that ends with `Then Exit Do`.
- **`Do While…Loop`:** Start with `Do While` *`LogicalTest`* and end with `Loop`. Because the logical test appears on the first line, it is possible that the code in the loop will never be executed.
- **`Do…Loop Until`:** Start with `Do` and end with `Loop Until` *`LogicalTest`*. Because the test condition doesn't appear until the end of this loop, the code in the loop will always be executed at least once. This is a subtle difference from the `Do While` loop.
- **`While…Wend`:** Start with `While` *`LogicalTest`* and end with `Wend`. This loop is like the `Do While` loop and is included for backward compatibility. If you see a `While` loop in existing code, you will understand how it works. You should not use `While…Wend` in new macros.

Code from the Lesson

```
Sub FixOne()
    ThisRow = Selection.Row
    If Cells(ThisRow, 5).Value > 500 Then
        Cells(ThisRow, 6) = "Bell Ringer!"
        Cells(ThisRow, 5).Font.Bold = True
        Beep
        Cells(ThisRow, 7) = Cells(ThisRow, 5)
    ElseIf Cells(ThisRow, 5).Value < 20 Then
        Cells(ThisRow, 6) = "Accessory"
        Cells(ThisRow, 1).Resize(1, 7).Font.Italic = True
    End If
```

```
    Cells(ThisRow + 1, 1).Select
End Sub

Sub DoLoop()
    Do
        FixOne
    Loop

End Sub

Sub DoLoopWithExit()
    Do
        FixOne
        If Selection.Value < " " Then Exit Do
    Loop

End Sub

Sub DoWhile()
    Do While Selection.Value > " "
        FixOne
    Loop

End Sub

Sub LoopUntil()
    Do
        FixOne
    Loop Until Selection.Value < " "

End Sub

Sub WhileWend()
    While Selection.Value > " "
        FixOne
    Wend

End Sub
```

lesson 29

For Each Loops

What You Will Learn

Learn how to use the cool loop that is new in object-oriented languages. This loop will loop through all items in a collection.

To work along, use Lesson29.xlsm.

Notes from the Lesson

- The `For Each` loop is not a carryover from BASIC or Pascal; it is a new loop for object-oriented languages.
- Start with the words `For Each`, and then insert a variable name, then the word `In` followed by a collection of objects. It is common to use a variable name that is indicative of the object type. The following are some examples:

```
For Each cell in Selection

For Each WS in ActiveWorkbook.Worksheets

For Each hl in ActiveSheet.Hyperlinks

For Each pt in ActiveSheet.PivotTables
```

- End the loop with either `Next` or `Next` followed by the variable name.
- The loop will run once for each item in the collection. For example, if you loop through each worksheet in the active workbook and there are five sheets, the loop will run five times. Each time through the loop, the object variable `WS` will refer to a different worksheet in the workbook.
- The loop control variable is an object variable with all the properties and values associated with the object.
- One downside to this loop is that no counter is inherent in the loop. If you wanted to log the totals from each worksheet in a new row on a summary workbook, you might have to set up your own counter variable. In the following code, `NextRow` is the counter:

```
    NextRow = 2
    For each WS in ActiveWorkbook.Worksheets
        ThisTotal = WS.Cells(20, 6)
        ThisSheet = WS.Name
        ' Log the totals to WST:
        WST.Cells(NextRow, 1).Value = ThisSheet
        WST.Cells(NextRow, 2).Value = ThisTotal
        NextRow = NextRow + 1
    Next WS
```

Code from the Lesson

```
Sub ProperEachCell()
    For Each cell In Selection
        cell.Value = _
            Application.WorksheetFunction.Proper(cell.Value)
    Next cell
End Sub

Sub TotalEachWorksheet()
    GrandTotal = 0
    Msg = ""

    For Each ws In ActiveWorkbook.Worksheets
        ThisTotal = ws.Cells(Rows.Count, 2).End(xlUp).Value
        ThisSheet = ws.Name
        GrandTotal = GrandTotal + ThisTotal
        Msg = Msg & ThisSheet & "'s total is $" _
            & ThisTotal & vbCr
    Next ws

    Msg = Msg & "Grand total is $" & GrandTotal
    MsgBox Msg

End Sub

Sub DecodeEachHyperlink()
    Dim hl As Hyperlink
    For Each hl In ActiveSheet.Hyperlinks
        ThisURL = hl.Address
        hl.Parent.Offset(0, 1).Value = ThisURL
    Next hl

End Sub
```

PART VI

Worksheets and Workbooks

The lessons up to this point have focused on working within a single worksheet. In this part, you will learn how to add new worksheets or workbooks on the fly. This is useful when you need to take one large dataset and break it out for distribution to multiple departments. The first two lessons focus on breaking out data. The last two lessons focus on taking those multiple reports and bringing them back into a single workbook.

There are four lessons in this part:

30 Creating Worksheets

31 Creating Workbooks

32 Listing Files in a Folder

33 Combining Workbooks

I regularly use the macro from Lesson 32 on a weekly basis. There are so many times that I would like to take the results of a Windows Explorer window and copy it into an Excel worksheet for sorting or further analysis.

lesson 30

Creating Worksheets

What You Will Learn

Learn how to break data out from one worksheet to multiple worksheets. You frequently will use object variables to keep track of the original and new worksheets.

Related Lesson

- Object variables are introduced in Lesson 17, "Using Object Variables."

To work along, use Lesson30.xlsm.

Notes from the Lesson

- You are going to make heavy use of object variables in this code to keep track of the original and other worksheets. For example, I use `WSO` to refer to the original worksheet and `WSN` to refer to the new worksheet. Alternatively, I use `WSD` for the data worksheet and `WSR` for the report worksheet.
- You can copy data from one worksheet to another worksheet without ever selecting the various worksheets. You often won't care which sheet is the active worksheet. Because your code explicitly refers to `WSD.Cells` or `WSR.Cells`, the macro will be able to access cells without activating the worksheet. This makes the macro run faster.
- To add a new worksheet, use `Worksheets.Add`. By default, the new sheet is inserted to the left of the active worksheet.
- To control where the new sheet is inserted, use either the `After:=` or `Before:=` clause. Suppose you want the worksheet added after the last worksheet. If you knew for sure that there were three worksheets already, you could use

  ```
  Worksheets.Add After:=Worksheets(3)
  ```

  However, you often won't know how many worksheets are already in the workbook. You can replace the `3` above with `Worksheets.Count` to show how many worksheets are already in the book. The following line of code adds a new worksheet after the last worksheet:

  ```
  Worksheets.Add After:=Worksheets(Worksheets.Count)
  ```

- After adding a worksheet, the new worksheet becomes the active sheet.

- You can assign the new worksheet to an object variable using two lines of code or one. When you shorten the two lines of code to one line, you must add parentheses around the arguments of the `.Add` method. The following sets up the object variable with two lines of code:

```
Worksheets.Add After:=Worksheets(Worksheets.Count)
Set WSR = ActiveSheet
```

  The following sets up the object variable with one line of code:

```
Set WSR = Worksheets.Add(After:Worksheets(Worksheets.Count))
```

- Near the end of the lesson you use the CreateManyWorksheets macro to add many new worksheets to a workbook, one for each value in a selection. The bonus code at the end of the code listing shows how to make many copies of a template worksheet based on a similar list.

Tip

When a line of code ends with a space and an underscore, this is a continuation mark. The next line of code is part of the current line of code. You can break very long lines of code to improve readability and ensure all of the code is visible in the width of the code window.

Code from the Lesson

```
Sub ServiceReport()
    ' Copy the selected range to the Report worksheet
    Dim WSD As Worksheet ' Data worksheet
    Dim WSR As Worksheet ' Report worksheet

    Set WSD = Worksheets("Data")

    ' Add a new worksheet to this workbook
    Set WSR = Worksheets.Add(after:=Worksheets("Data"))

    ' Rename the new worksheet & set up titles
    WSR.Name = "Service"
    WSR.Cells(1, 1) = "Service Report"
    WSR.Cells(1, 1).Font.Size = 14

    WSD.Range("A1:E1").Copy Destination:=WSR.Cells(3, 1)
    NextRow = 4

    ' Loop through all records on WSD
    FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
    For i = 2 To FinalRow
```

```
        If WSD.Cells(i, 2) = "Z" Then
            ' Copy this record to the next row on WSR
            WSD.Cells(i, 1).Resize(1, 5).Copy _
                Destination:=WSR.Cells(NextRow, 1)
            NextRow = NextRow + 1
        End If
    Next i

    ' Make sure WSR is the active sheet
    WSR.Select

    ' Report that the macro is done
    MsgBox Prompt:=NextRow - 4 & _
        " service records copied to the service report.", _
        Title:="MrExcel.com"
End Sub

Sub CreateManyWorksheets()
    For Each cell In Selection
        ThisWS = cell.Value
        Worksheets.Add After:=Worksheets(Worksheets.Count)
        ActiveSheet.Name = ThisWS
    Next cell
End Sub
```

Bonus Code

This code is an adaption of the last macro in the lesson. Rather than creating several new blank worksheets, it copies a template worksheet several times, renaming each sheet with the value from a cell in the selection.

```
Sub CreateManyWorksheetsFromTemplate()
    Dim WST as Worksheet
    Set WST = Worksheets("Template"
    For Each cell In Selection
        ThisWS = cell.Value
        WST.Copy After:=Worksheets(Worksheets.Count)
        ActiveSheet.Name = ThisWS
    Next cell
End Sub
```

lesson 31

Creating Workbooks

What You Will Learn

Learn how to split a data set into multiple workbooks, one for each department. You will make heavy use of workbook object variables to track the original and new workbook.

To work along, use Lesson31.xlsm.

Notes from the Lesson

- This macro uses a few built-in properties: `ActiveWorkbook` refers to the currently active workbook, `ThisWorkbook` refers to the workbook in which the code is saved, and `ActiveSheet` refers to the currently active worksheet in the active workbook.
- You are going to make heavy use of object variables in this code to keep track of the original and other workbooks. For example, I use `WBO` to refer to the original workbook and `WBN` to refer to the new workbook. As in Lesson 30, I use `WSO` to refer to the original worksheet and `WSN` to refer to the worksheet in the new workbook.
- If you use `Workbooks.Add`, you will get a workbook that inherits the settings from the person's computer. Some people might have Excel set up to start with three worksheets and some people start new workbooks with one. Some people use `Book.xltx` to override settings for their new workbooks. So using only `Workbooks.Add` will give you an unpredictable number of worksheets. Instead, specify a template called `xlWBATWorksheet` to create a new workbook with a single worksheet:

  ```
  Workbooks.Add Template:=xlWBATWorksheet
  ```

- After adding a workbook, the new workbook becomes the active workbook.
- You can assign the new workbook to an object variable using two lines of code or one. When you shorten the two lines of code to one line, you must add parentheses around the arguments of the `.Add` method. The following sets up the object variable with two lines of code:

  ```
  Workbooks.Add Template:=xlWBATWorksheet
  Set WBN = ActiveWorkbook
  ```

  The following sets up the object variable with one line of code:

  ```
  Set WBN = Workbooks.Add(Template:=xlWBATWorksheet)
  ```

- Usually you will need to set up a worksheet object variable to refer to the single worksheet in the new workbook. Use `Set WSN = WBN.Worksheets(1)` after setting up the new workbook.
- If you use the workbook object variable and the worksheet object variable, you can copy data from one to the other without ever activating the other workbook:
  `WBO.WSO.Range("A1:F1").Copy Destination:=WBN.WSN.Range("A1")`

- The loop in this lesson uses what is known as Control Break logic. It loops through each record to find out whether the department on this line is different than the department on the previous line. This approach requires the data to be sorted by department. It also requires you to do something to handle the very last department. In this lesson, the loop makes sure to go to one row past the end of the data set to force the `If` statement that checks to see whether the department changed to recognize the last department.
- After creating the new workbook, use the `.SaveAs` method to save the workbook. You will have to supply a path and filename. For the path, I often use
  `ThisWorkbook.Path & Application.PathSeparator`

- After saving the new workbook, close it with

  ```
  WBN.Close SaveChanges:=False
  ```

Code from the Lesson

```
Sub CreateWorkbooks()
    Dim WBO As Workbook ' original workbook
    Dim WBN As Workbook ' new workbook
    Dim WSO As Worksheet ' original worksheet
    Dim WSN As Worksheet ' new worksheet

    Set WBO = ActiveWorkbook
    Set WSO = ActiveSheet

    FinalRow = WSO.Cells(Rows.Count, 1).End(xlUp).Row + 1

    LastDept = Cells(2, 1)
    StartRow = 2

    For i = 2 To FinalRow
        ThisDept = WSO.Cells(i, 1)
        If ThisDept = LastDept Then
            ' do nothing
        Else
            ' We have a new department starting
            ' Copy all of the previous rows to a new workbook
            LastRow = i - 1
            RowCount = LastRow - StartRow + 1
```

```
            ' Create a new workbook.
            Set WBN = Workbooks.Add(Template:=xlWBATWorksheet)
            Set WSN = WBN.Worksheets(1)

            ' Set up the headings for the report
            WSN.Cells(1, 1).Value = "Budget Report"
            WSN.Cells(1, 1).Font.Size = 14

            WSN.Cells(2, 1).Value = LastDept & _
                 " - " & WSO.Cells(StartRow, 2)
            WSO.Range("C1:Q1").Copy Destination:=WSN.Cells(4, 1)

            ' copy all of the records for this department
            WSO.Range(WSO.Cells(StartRow, 3), _
                 WSO.Cells(LastRow, 17)).Copy _
                 Destination:=WSN.Cells(5, 1)

            FN = LastDept & ".xlsx"
            FP = WBO.Path & Application.PathSeparator

            WBN.SaveAs Filename:=FP & FN
            WBN.Close SaveChanges:=False

            LastDept = ThisDept
            StartRow = i
        End If
    Next i
End Sub
```

lesson 32

Listing Files in a Folder

What You Will Learn

Learn how to list all Excel files in a folder. This utility enables you to bring the detail view from Windows Explorer and put it in a sortable worksheet.

To work along, use Lesson32.xlsm.

Notes from the Lesson

- Microsoft removed support for `Application.FileSearch` in Excel 2007. So although the ListFiles2003 macro will work in all previous versions of Excel, you must use ListFiles2007 in Excel 2007. It is possible that this feature will return in Excel 2010.
- To customize the 2003 macro, change these three lines of code:

```
.LookIn = "C:\Users\Public\Documents\LLVBA\Post\Budget\"
.SearchSubFolders = True
.Filename = "*.xlsx"
```

- To customize the 2007 macro, change these lines of code:

```
' Enter the folder name here
Const strDir As String = "C:\aaa\"
Let strName = Dir$(strDir & "*.xlsx")
```

- To use the macro to list all files, use `"*.*"` instead of `"*.xlsx"`.
- The bonus code listing is the actual macro that I use to generate a list of files. Column A contains the full path and filename. Column B contains the path. Column C contains the filename. Column D contains the last modified date, and column E contains the file size. This code is found in Module3 of the sample file for this lesson.

Code from the Lesson

Code in Module1:

```
Sub ListFiles2003()
    ' This macro lists all files in a folder
    ' Clear out all cells in the sheet
```

```
    Cells.Clear
    ' Add Headings
    Range("A1:C1").Value = Array("FileName", "Path", "FileName")
    NextRow = 2
    ' Use the FileSearch Object
    With Application.FileSearch
        .NewSearch
        ' Customize the following line
        .LookIn = "C:\Users\Public\Documents\LLVBA\Post\Budget"
        .SearchSubFolders = True
        .Filename = "*.xlsx"
        .Execute
        FilesToProcess = .FoundFiles.Count
        ' loop through each workbook in the directory
        For i = 1 To .FoundFiles.Count
            ThisEntry = .FoundFiles(i)
            Cells(NextRow, 1).Value = ThisEntry
            ' Find to path portion. Start looking from the end
            ' of This Entry for a backslash
            For j = Len(ThisEntry) To 1 Step -1
                If Mid(ThisEntry, j, 1) = _
                    Application.PathSeparator Then
                    Cells(NextRow, 2) = Left(ThisEntry, j)
                    Cells(NextRow, 3) = Mid(ThisEntry, j + 1)
                    Exit For
                End If
            Next j
            NextRow = NextRow + 1
        Next i
    End With
End Sub
```

Code in Module2:

```
Sub FindFiles2007()
    ' New method for Excel 2007
    ' You need this macro, plus the following macro

    Dim fso As Object
    Dim strName As String
    Dim strArr(1 To 1048576, 1 To 1) As String, i As Long

    ' Enter the folder name here
    Const strDir As String = "C:\aaa\"

    Let strName = Dir$(strDir & "*.xlsx")
    Do While strName <> vbNullString
        Let i = i + 1
```

```
        Let strArr(i, 1) = strDir & strName
        Let strName = Dir$()
    Loop
    Set fso = CreateObject("Scripting.FileSystemObject")
    Call recurseSubFolders(fso.GetFolder(strDir), strArr(), i)
    Set fso = Nothing
    If i > 0 Then
        Range("A1").Resize(i).Value = strArr
    End If

    ' Next, loop through all found files
    ' and break into path and filename
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    For i = 1 To FinalRow
        ThisEntry = Cells(i, 1)
        For j = Len(ThisEntry) To 1 Step -1
            If Mid(ThisEntry, j, 1) = _
                Application.PathSeparator Then
                Cells(i, 2) = Left(ThisEntry, j)
                Cells(i, 3) = Mid(ThisEntry, j + 1)
                Exit For
            End If
        Next j
    Next i

End Sub

Private Sub recurseSubFolders(ByRef Folder As Object, _
    ByRef strArr() As String, _
    ByRef i As Long)
Dim SubFolder As Object
Dim strName As String
For Each SubFolder In Folder.SubFolders
    Let strName = Dir$(SubFolder.Path & "*.jpg")
    Do While strName <> vbNullString
        Let i = i + 1
        Let strArr(i, 1) = SubFolder.Path & strName
        Let strName = Dir$()
    Loop
    Call recurseSubFolders(SubFolder, strArr(), i)
Next
End Sub

Sub CopyToNewFolder()
    ' Page 115 Case Study
    ' After running the above macro to find images
    ' Type a new path name in column D
```

```
    FinalRow = Range("A65536").End(xlUp).Row
    For Each Cell In Range("A2:A" & FinalRow)
        OrigFile = Cell.Value
        If Cell.Offset(0, 4).Value > "" Then
            NewFile = Cell.Offset(0, 3) & _
                Application.PathSeparator & _
                Cell.Offset(0, 4)
            FileCopy OrigFile, NewFile
        End If
    Next Cell
End Sub
```

Bonus Code

```
Sub ListAllFiles()
    ' This is bonus code. It is the Excel 2003
    ' version of the code that I use to list all files
    ' in any folder and the subfolders underneath it.

    ' The results are written starting in
    ' row 5 of the FileList worksheet
    ' Enter the desired path in cell of FileList.
    ' This macro will populate cells A6 & down with a file list
    ' Column A = Path & File. Column B = Path
    ' Column C = File
    ' Column D = Date modified
    ' Column E = File size

    ' Set up an output range
    Worksheets("FileList").Select
    Range("A5").Value = "Test"
    Range("A5").CurrentRegion.Clear
    Range("A5:E5").Value = _
        Array("Path & File", "Path", "File", "Date", "Size")
    NextWBRow = 5

    UsePath = Range("CustPath").Value
    If Not Right(UsePath, 1) = "\" Then
        UsePath = UsePath & "\"
    End If

    With Application.FileSearch
        .NewSearch
        .LookIn = UsePath
        .SearchSubFolders = True
        .Filename = "*.*"
        .Execute
```

```
        FilesToProcess = .FoundFiles.Count
        ' loop through each workbook in the directory
        Application.EnableEvents = False
        For i = 1 To .FoundFiles.Count
            Application.StatusBar = .FoundFiles(i)
            ' log this file in the file log
            NextWBRow = NextWBRow + 1
            Cells(NextWBRow, 1).Value = .FoundFiles(i)
            ThisFile = .FoundFiles(i)
            For j = Len(ThisFile) To 1 Step -1
                If Mid(ThisFile, j, 1) = _
                    Application.PathSeparator Then
                    OldPath = Left(ThisFile, j)
                    OldFile = Mid(ThisFile, j + 1)
                    Cells(NextWBRow, 2).Value = OldPath
                    Cells(NextWBRow, 3).Value = OldFile
                    Exit For
                End If
            Next j
            Cells(NextWBRow, 4).Value = _
                FileDateTime(Cells(NextWBRow, 1).Value)
            Cells(NextWBRow, 5).Value = _
                FileLen(Cells(NextWBRow, 1).Value)
        Next i
    End With

End Sub
```

lesson 33

Combining Workbooks

What You Will Learn

Learn how to combine data from multiple workbooks into a single workbook. This is a common task—you collect data from multiple people and then need to combine all those disparate files into a single consolidated workbook.

Related Lesson

- This lesson starts out with a list of files. The easy way to generate that list of files is to use the macro from Lesson 32.

To work along, use Lesson33.xlsm.

Notes from the Lesson

- Set up a workbook with two worksheets. One worksheet has a list of the files that you want to open, and the second worksheet is a blank worksheet where you will collect the data. Perhaps you will start with headings in this worksheet.
- `WBO` is a workbook object variable for the original workbook. `WBN` is a workbook object variable that will be reused and will refer to each individual workbook one at a time. In a similar fashion, `WSN` is a worksheet object variable that will be reused to point to the first worksheet in each workbook.
- `ThisWorkbook` and `ActiveWorkbook` are two built-in keywords. `ActiveWorkbook` refers to the currently active workbook, and `ThisWorkbook` will always refer to the workbook in which the code is stored.
- Although `FinalRow` is my favorite variable name for storing the last row with data, this macro requires storing two last rows. One of them is the last row of the list of files. The macro uses `FinalRow` to hold that row number. The other instance is the last row with data in each individual workbook. In that case, the macro uses a variable name of `LastRow`.
- The macro will copy from one workbook and paste to another workbook in a single line of code. This is achieved by using the `Destination` argument. It is pretty wild that you can specify one worksheet as the object in the `Copy` command and a different worksheet in the `Destination` argument, as in this example: `WSN.Cells().Copy Destination:=WSD.Cells()`. Note that this line of code does not change the active workbook to another workbook. This is

better than the method used by the macro recorder. The macro recorder does the copy in one line, switches to another workbook, and then uses `ActiveSheet.Paste`. The macro recorder method always causes you to switch between the workbooks. The line of code used in this lesson does the copy without switching back and forth between workbooks.

- In this macro, I stepped through the code to test the first pass through the loop. After I saw that the first pass worked OK, I used the Run icon to run the rest of the passes through the loop.

Tip

Late in this lesson I select several lines of code, and you see those lines all get indented by four characters. You can indent all the selected lines by pressing Tab. Pressing Shift+Tab will move the columns back to the left by four characters. This makes it easy to take a block of code and nicely indent it inside a loop.

Code from the Lesson

Code in Module1 and Module2:

Module1 and Module2 contains the ListFiles macro from Lesson 32.

Code in Module3:

```
Sub CombineFiles()
    Dim WBO As Workbook ' original workbook
    Dim WBN As Workbook ' individual data workbooks
    Dim WSL As Worksheet ' List of files worksheet
    Dim WSD As Worksheet ' data collection worksheet
    Dim WSN As Worksheet

    ' Define object variables
    Set WBO = ThisWorkbook
    Set WSL = WBO.Worksheets("List")
    Set WSD = WBO.Worksheets("Data")

    ' Clear out any previous data on WSD, but leave the headings
    WSD.Cells(2, 1).Resize(Rows.Count - 1, Columns.Count).Clear

    NextRow = 2

    ' Loop through all the files on WSL
    FinalRow = WSL.Cells(Rows.Count, 1).End(xlUp).Row
    For i = 1 To FinalRow
        ThisFile = WSL.Cells(i, 1)
        ' Open a file
        Set WBN = Workbooks.Open(Filename:=ThisFile)

        For Each WSN In WBN.Worksheets
```

```
            ' Last row in this file?  RowCount is 4 less
            LastRow = WSN.Cells(Rows.Count, 1).End(xlUp).Row
            RowCount = LastRow - 4

            ' Copy from 5 to last row over to column B
            WSN.Cells(5, 1).Resize(RowCount, 15).Copy _
                Destination:=WSD.Cells(NextRow, 2)
            ' Replicate the department from A2 to column A in WSD
            WSD.Cells(NextRow, 1).Resize(RowCount, 1).Value _
                = WSN.Range("A2")

            ' Set up the new NextRow
            NextRow = NextRow + RowCount
        Next WSN

        ' Close WSN, don't save
        WBN.Close SaveChanges:=False

    Next i

End Sub
```

PART VII

Formulas

VBA is more focused on formulas in Excel than in any other application. People writing macros in Word or PowerPoint don't care about formulas, but they are key to Excel functionality. In this part, you learn two different ways of entering formulas into Excel using VBA. You learn how to take advantage of Excel's built-in functions in your code and how to build additional functions that can be available for people to use in Excel worksheets.

There are four lessons in this part:

34 A1-Style Formulas

35 R1C1-Style Formulas

36 Using Excel Functions in Code

37 Adding Functions to Excel

My favorite lesson in this part is also the lesson that you think you can skip over. If you take the 15 minutes to watch and understand R1C1 formulas in Lesson 35, you will be able to write code quicker every day.

lesson 34

A1-Style Formulas

What You Will Learn

Using A1-style formulas is the most intuitive way to enter a formula, but it requires the most wrangling in VBA. You must concatenate the current row number into the formula in many cases.

Related Lesson

- Lesson 35, "R1C1-Style Formulas," shows a faster way of entering formulas with less concatenation.

To work along, use Lesson34.xlsm.

Notes from the Lesson

- Writing a macro to enter formulas can simplify adding difficult formulas to a workbook. Perhaps you must use `VLOOKUP` functions every day to add fields to a data set. If you have trouble entering `VLOOKUP` formulas, write a macro and you won't have to build new formulas every day.
- Most people enter formulas using a syntax that is known as A1 style references. In the system, the first cell in the worksheet is known as cell A1. Unfortunately, the macro recorder uses something called R1C1 style references. In this system, the first cell is called R1C1.
- So although usually you can record a few steps with the macro recorder, you will never get an A1 style formula as the result of recording a macro. The alternative is to enter the formula in the worksheet without the macro recorder running. Then, copy the formula from the formula bar and paste that formula to VBA.
- You might want to enter the formula as a comment, then build macro lines to enter the formula in the first cell, and then copy the formula down to a suitable number of rows.
- A formula entered in cell F2 might refer to other cells in row 2; for example, `=IF(E2=0,0,B2/E2)`. A formula that performs a lookup might have to refer to a variable number of rows. Often you will find that you are building the formula by concatenating bits of the formula with variables:

```
MyFormula = "=IF(E" & ThisRow & "=0,0,B" & ThisRow & "/E" & ThisRow & ")"
```

- After entering the first formula, copy it to the other rows in the data set using the `.Copy` or `.AutoFill` method.
- If your formula in Excel contains a quotation mark, be careful when building this formula in VBA. You must put two quotation marks where you want a quotation mark to appear. If your formula in Excel is `=IF(A2>20000, "Bonus","No Bonus")`, the VBA will look like this:

```
MyForm = "=IF(A2>20000,""Bonus"","""No Bonus"")"
```

This can get tedious, particularly when you already have double quotes in the formula. For example, this formula in Excel: `=IF(A2>20000,"","No ")&"Bonus"` becomes

```
MyForm = "=IF(A2>20000,""""","""No """)&""Bonus"""
```

The two quotation marks in the value if true section of the `If` function turns into four quotation marks in VBA. The final quotation mark in the formula turns into two quotation marks, immediately followed by the quotation mark to end the formula.

- To remove a module from a project, right-click the module in the Project Explorer and choose Remove *ModuleName*.

Caution

Just as every formula in Excel must start with an equal sign, don't forget to put the opening equal sign inside the quotation marks for your formula. It seems confusing because you have an equal sign, then the starting quotation mark, and then you must type the equal sign to start the formula. If you fail to put the starting equal sign, Excel will enter the text from your formula into the cell as text.

Note

Many people see that a formula changes as it gets copied from row 2 to rows 3 through 99. This is why you frequently see the formula being entered in the first row and then copied down to other rows. If you are careful to write the formula as it would appear in the first cell of the range, you can actually enter the formula for the entire dataset:

```
Cells(1, 6).Resize(FinalRow, 1).Formula = _
     "=IF(A1>20000,0.02*A1,0)"
```

Code from the Lesson

```
Sub AddDescriptions()

    '   =VLOOKUP(A2,'Lookup Table'!$A$2:$B$29,2,FALSE)

     FinalRow = Worksheets("Lookup Table").Cells(Rows.Count, 1). _
         End(xlUp).Row
     Range("D2").Formula = _
          "=VLOOKUP(A2,'Lookup Table'!$A$2:$B$" & FinalRow & ",2,FALSE)"

     FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
     Range("D2").AutoFill Destination:=Range("D2:D" & FinalRow)

     ' Add a formula to E in this style:
     ' =NOT(ISERROR(FIND("Earrings",D2)))
     Range("E2").Formula = "=NOT(ISERROR(FIND(""Earrings"",D2)))"
     Range("E2").AutoFill Destination:=Range("E2:E" & FinalRow)

End Sub
```

lesson 35

R1C1-Style Formulas

What You Will Learn

Learn how to understand and write formulas in the R1C1 reference style. Although few people use this style, it is a better way to write formulas in VBA.

To work along, use Lesson35.xlsm.

Notes from the Lesson

- It's easier to enter formulas using the R1C1 style because the formula is identical as you copy a formula down to all the rows of your data set.
- To refer to a cell in the R1C1 style, you need to have the letters R and C. If you simply enter `=RC`, you are referring to the cell in which the formula is stored. Because this would create a circular reference, you will hardly ever see this. Instead, most references will modify either the R or the C by following it with a number in square brackets:

  `=RC[-3]` refers to this row, three columns to the left.

  `=R[-1]C` refers to one row above, this column.

  `=RC[1]` is the cell to the right of the current cell.

  `=R[10]C[2]` is 10 rows down and 2 columns to the right.

  Figure 4 illustrates the R1C1 name for several cells around cell D7.

	A	B	C	D	E	F
1						
2						
3						
4		=R[-3]C[-2]				
5			=R[-2]C[-1]			
6				=R[-1]C		
7			=RC[-1]	=RC	=RC[1]	
8				=R[1]C		
9					=R[2]C[1]	
10						=R[3]C[2]

Figure 4 The text in each cell shows how you would refer to that cell in a formula entered in cell D7.

- In A1 style, you might use `=$J$1` to set up an absolute reference to cell J1. The equivalent reference in R1C1 style omits the square brackets: `=R1C10`.

- In A1 style, in a formula in row 2, you might use `=$A2` to go back to column A of the current row. In R1C1 style, you would use `=RC1`. This says, "Use the current row and always go to column 1."
- The macro recorder always records formulas as R1C1 formulas, even if you write them as A1 style formulas. Using the information above, you can decode what this formula means.
- You don't need to be able to write R1C1 formulas from scratch—you can enter a formula in a worksheet in A1 style. Select the cell that has that formula and switch to VBA. Switch to the Immediate Window by pressing Ctrl+G, and type **`Print ActiveCell.FormulaR1C1`**. You can then copy and paste this R1C1 formula into your macro code.
- This lesson also shows you how to enter several headings across a row. To fill in headings in A1 through E1 you must specify the headings inside an `Array` function:

```
Range("A1:E1") = Array("Sales","COGS","Profit","","Tax")
```

Tip

To quickly enter Show Formulas mode, press Ctrl+`. To toggle out of Show Formulas mode press Ctrl+` again. (The ` is the grave accent and is often found on a key just under the Esc key.)

Code from the Lesson

```
Sub AddFormulas()
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    DataCount = FinalRow - 1

    Range("G1:N1").Value = Array("Revenue", "COGS", _
        "Profit", "GP%", "Sales Tax", "", "Tax Rate", 0.065)

    Range("G2").Resize(DataCount, 1).FormulaR1C1 = "=RC[-2]*RC[-3]"
    Range("H2").Resize(DataCount, 1).FormulaR1C1 = "=RC[-2]*RC[-4]"
    Range("I2").Resize(DataCount, 1).FormulaR1C1 = "=RC[-2]-RC[-1]"
    Range("J2").Resize(DataCount, 1).FormulaR1C1 = "=RC[-1]/RC[-3]"
    Range("K2").Resize(DataCount, 1).FormulaR1C1 = "=RC[-4]*R1C14"

End Sub
```

lesson 36

Using Excel Functions in Code

What You Will Learn

Learn how to use Excel functions in your VBA code.

To work along, use Lesson36.xlsm.

Notes from the Lesson

- A few Excel functions are also used in VBA. Some examples are `MIN`, `MAX`, and `LEFT`.
- Some Excel functions have equivalent functionality in VBA but have a different name; for example, the Excel functions `=UPPER()` and `=LOWER()` appear in VBA as `=UCASE()` and `=LCASE()`.
- For most other functions, you must precede the function name with `Application.WorksheetFunction` in order for them to work in VBA.
- Some Excel functions have no equivalent in VBA. In these cases, you could use VBA to enter the function in a cell in Excel and then find the `.Value` of that cell.
- Table 36.1 provides an alphabetical list of Excel functions. The columns indicate whether the function is natively supported or whether you must use `Application.WorksheetFunction`.
- If the 2003 column indicates ATP*, you must follow these steps before you can use them in Excel 2003:
    1. In the Excel interface, select Tools, Add-Ins.
    2. Place a checkmark next to Analysis ToolPak – VBA and click OK.
    3. Switch to VBA by pressing Alt+F11.
    4. Select your workbook in the Project Explorer window.
    5. Select Tools, References. Scroll through the list until you find atpvbaen.xls. Place a checkmark next to this box.
    6. You can now use the function as if it were a native function: `DEC2HEX(60)`. Alternatively, you can fully qualify the function like this: `[atpvbaen.xls].DEC2HEX(60)`.

- If you will be using many functions, you can shorten the `Application.Worksheet` function by setting up an object variable to refer to it:

```
Dim wf As WorksheetFunction
Set wf = Application.WorksheetFunction
X = wf.PMT(0.05/12,60,-25000)
```

Code from the Lesson

```
Sub Test()
    Dim wf As WorksheetFunction
    Set wf = Application.WorksheetFunction

    ThisItem = ActiveCell.Value

    ' Find the description
    On Error Resume Next
    Desc = wf.VLookup(ThisItem, Range("LookupTable"), 2, False)
    x = wf.
    On Error GoTo 0

    MsgBox Desc

End Sub
```

Table 36.1 Alphabetical List of All Excel Functions and Their Support in VBA

Function	Native to VBA	2003	2007
ABS	ABS()		
ACCRINT		ATP*	X
ACCRINTM		ATP*	X
ACOS		X	X
ACOSH		X	X
ADDRESS	ActiveCell.Address		
AMORDEGRC		ATP*	X
AMORLINC		ATP*	X
AND		X	X
AREAS	Selection.Areas.Count		
ASC		X	X
ASIN		X	X
ASINH		X	X
ATAN	ATN()		
ATAN2		X	X
ATANH		X	X
AVEDEV		X	X
AVERAGE		X	X
AVERAGEA			
AVERAGEIF		N/A	X

Table 36.1 Alphabetical List of All Excel Functions and Their Support in VBA continued

Function	Native to VBA	2003	2007
AVERAGEIFS		N/A	X
BAHTTEXT		X	X
BESSELI		ATP*	X
BESSELJ		ATP*	X
BESSELK		ATP*	X
BESSELY		ATP*	X
BETADIST		X	X
BETAINV		X	X
BIN2DEC		ATP*	X
BIN2HEX		ATP*	X
BIN2OCT		ATP*	X
BINOMDIST		X	X
CEILING		X	X
CELL			
CHAR	CHR()		
CHIDIST		X	X
CHIINV		X	X
CHITEST		X	X
CHOOSE		X	X
CLEAN		X	X
CODE	ASC()		
COLUMN	ActiveCell.Column		
COLUMNS	Selection.Columns.Count		
COMBIN		X	X
COMPLEX		ATP*	X
CONCATENATE	"B" & FinalRow		
CONFIDENCE		X	X
CONVERT		ATP*	X
CORREL		X	X
COS	COS()		
COSH		X	X
COUNT		X	X
COUNTA		X	X

Table 36.1 Alphabetical List of All Excel Functions and Their Support in VBA continued

Function	Native to VBA	2003	2007
COUNTBLANK		X	X
COUNTIF		X	X
COUNTIFS		N/A	X
COUPDAYBS		ATP*	X
COUPDAYS		ATP*	X
COUPDAYSNC		ATP*	X
COUPNCD		ATP*	X
COUPNUM		ATP*	X
COUPPCD		ATP*	X
COVAR		X	X
CRITBINOM		X	X
CUMIPMT		ATP*	X
CUMPRINC		ATP*	X
DATE			
DATEDIF			
DATESTRING			
DATEVALUE	DATEVALUE()		
DAVERAGE		X	X
DAY	DAY()		
DAYS360		X	X
DB		X	X
DBSC		X	X
DCOUNT		X	X
DCOUNTA		X	X
DDB		X	X
DEC2BIN		ATP*	X
DEC2HEX		ATP*	X
DEC2OCT		ATP*	X
DEGREES		X	X
DELTA		ATP*	X
DEVSQ		X	X
DGET		X	X
DISC		ATP*	X

Table 36.1 Alphabetical List of All Excel Functions and Their Support in VBA continued

Function	Native to VBA	2003	2007
DMAX		X	X
DMIN		X	X
DOLLAR		X	X
DOLLARDE		ATP*	X
DOLLARFR		ATP*	X
DPRODUCT		X	X
DSTDEV		X	X
DSTDEVP		X	X
DSUM		X	X
DURATION		ATP*	X
DVAR		X	X
DVARP		X	X
EDATE		ATP*	X
EFFECT		ATP*	X
EOMONTH		ATP*	X
ERF		ATP*	X
ERFC		ATP*	X
ERROR.TYPE			
EVEN		X	X
EXACT			
EXP	EXP()		
EXPONDIST		X	X
FACT		X	X
FACTDOUBLE		ATP*	X
FALSE	FALSE		
FDIST		X	X
FIND		X	X
FINDB		X	X
FINV		X	X
FISHER		X	X
FISHERINV		X	X
FIXED		X	X
FLOOR		X	X

Table 36.1 Alphabetical List of All Excel Functions and Their Support in VBA continued

Function	Native to VBA	2003	2007
FORECAST		X	X
FREQUENCY		X	X
FTEST		X	X
FV		X	X
FVSCHEDULE		ATP*	X
GAMMADIST		X	X
GAMMAINV		X	X
GAMMALN		X	X
GCD		ATP*	X
GEOMEAN		X	X
GESTEP		ATP*	X
GETPIVOTDATA			
GROWTH		X	X
HARMEAN		X	X
HEX2BIN		ATP*	X
HEX2DEC		ATP*	X
HEX2OCT		ATP*	X
HLOOKUP		X	X
HOUR	HOUR()		
HYPERLINK			
HYPGEOMDIST		X	X
HYPGEOMVERT			
IF			
IFERROR		N/A	X
IMABS		ATP*	X
IMAGINARY		ATP*	X
IMARGUMENT		ATP*	X
IMCONJUGATE		ATP*	X
IMCOS		ATP*	X
IMDIV		ATP*	X
IMEXP		ATP*	X
IMLN		ATP*	X
IMLOG10		ATP*	X

Table 36.1 Alphabetical List of All Excel Functions and Their Support in VBA continued

Function	Native to VBA	2003	2007
IMLOG2		ATP*	X
IMPOWER		ATP*	X
IMREAL		ATP*	X
IMSIN		ATP*	X
IMSQRT		ATP*	X
IMSUB		ATP*	X
IMSUM			X
INDEX		X	X
INDIRECT			
INFO			
INT	INT()		
INTERCEPT		X	X
INTRATE		ATP*	X
IPMT		X	X
IRR		X	X
ISBLANK	= ""		
ISERR		X	X
ISERROR		X	X
ISEVEN		ATP*	X
ISLOGICAL		X	X
ISNA		X	X
ISNONTEXT		X	X
ISNUMBER		X	X
ISODD		ATP*	X
ISPMT		X	X
ISREF			
ISTEXT		X	X
KURT		X	X
LARGE		X	X
LCM		ATP*	X
LEFT	LEFT()		
LEFTB	LEFTB()		
LEN	LEN()		
LENB			

Table 36.1 Alphabetical List of All Excel Functions and Their Support in VBA continued

Function	Native to VBA	2003	2007
LINEST		X	X
LN		X	X
LOG		X	X
LOG10		X	X
LOGEST		X	X
LOGINV		X	X
LOGNORMDIST		X	X
LOOKUP		X	X
LOWER	LCASE()		
MATCH		X	X
MAX		X	X
MAXA			
MDETERM		X	X
MDURATION		ATP*	X
MEDIAN		X	X
MID	MID()		
MIDB	MIDB()		
MIN		X	X
MINA			
MINUTE	MINUTE()		
MINVERSE		X	X
MIRR		X	X
MMULT		X	X
MNORMSINV			
MOD	7 Mod 2		
MODE		X	X
MONTH	MONTH()		
MROUND		ATP*	X
MULTINOMIAL		ATP*	X
N			
NA			
NEGBINOMDIST		X	X
NETWORKDAYS		ATP*	X
NOMINAL		ATP*	X
NORMDIST		X	X

Table 36.1 Alphabetical List of All Excel Functions and Their Support in VBA continued

Function	Native to VBA	2003	2007
NORMINV		X	X
NORMSDIST		X	X
NORMSINV		X	X
NOT	NOT()		
NOW	Now		
NPER		X	X
NPV		X	X
NUMBERSTRING			
OCT2BIN		ATP*	X
OCT2DEC		ATP*	X
OCT2HEX		ATP*	X
ODD		X	X
ODDFPRICE		ATP*	X
ODDFYIELD		ATP*	X
ODDLPRICE		ATP*	X
ODDLYIELD		ATP*	X
OFFSET	.OFFSET		
OR		X	X
PEARSON		X	X
PERCENTILE		X	X
PERCENTRANK		X	X
PERMUT		X	X
PHONETIC		X	X
PI		X	X
PMT		X	X
POISSON		X	X
POWER		X	X
PPMT		X	X
PRICE		ATP*	X
PRICEDISC			X
PRICEMAT		ATP*	X
PROB		X	X
PRODUCT		X	X
PROPER		X	X

Table 36.1 Alphabetical List of All Excel Functions and Their Support in VBA continued

Function	Native to VBA	2003	2007
PV		X	X
QUARTILE		X	X
QUOTIENT		ATP*	X
RADIANS		X	X
RAND	RND()		
RANDBETWEEN		ATP*	X
RANK		X	X
RATE		X	X
RECEIVED		ATP*	X
REPLACE		X	X
REPLACEB		X	X
REPT		X	X
RIGHT	RIGHT()		
RIGHTB	RIGHTB()		
ROMAN		X	X
ROUND		X	X
ROUNDDOWN		X	X
ROUNDUP		X	X
ROW	ActiveCell.Row		
ROWS	Selection.Rows.Count		
RSQ		X	X
RTD		X	X
SEARCH		X	X
SEARCHB		X	X
SECOND	SECOND()		
SERIESSUM		ATP*	X
SIGN	—		
SIN	SIN()		
SINH		X	X
SKEW		X	X
SLN		X	X
SLOPE		X	X
SMALL		X	X
SQRT	^ 0.5		

Table 36.1 Alphabetical List of All Excel Functions and Their Support in VBA continued

Function	Native to VBA	2003	2007
SQRTPI		ATP*	X
STANDARDIZE		X	X
STDEV		X	X
STDEVA			
STDEVP		X	X
STDEVPA			
STEYX		X	X
SUBSTITUTE		X	X
SUBTOTAL		X	X
SUM		X	X
SUMIF		X	X
SUMIFS		N/A	X
SUMPRODUCT		X	X
SUMSQ		X	X
SUMX2MY2		X	X
SUMX2PY2		X	X
SUMXMY2		X	X
SYD		X	X
T			
TAN	TAN()		
TANH		X	X
TBILLEQ		ATP*	X
TBILLPRICE		ATP*	X
TBILLYIELD		ATP*	X
TDIST		X	X
TEXT		X	X
TIME	Time		
TIMEVALUE	TIMEVALUE()		
TINV		X	X
TODAY	Date		
TRANSPOSE		X	X
TREND		X	X
TRIM		X	X
TRIMMEAN		X	X

Table 36.1 Alphabetical List of All Excel Functions and Their Support in VBA continued

Function	Native to VBA	2003	2007
TRUE	TRUE		
TRUNC			
TTEST		X	X
TYPE			
UPPER	UCASE()		
USDOLLAR		X	X
VALUE			
VAR		X	X
VARA			
VARP		X	X
VARPA			
VDB		X	X
VLOOKUP		X	X
WEEKDAY		X	X
WEEKNUM		ATP*	X
WEIBULL		X	X
WORKDAY		ATP*	X
XIRR		ATP*	X
XNPV		ATP*	X
YEAR	YEAR()		
YEARFRAC		ATP*	X
YIELD			
YIELDDISC		ATP*	X
YIELDMAT		ATP*	X
ZTEST		X	X

lesson 37

Adding Functions to Excel

What You Will Learn

Learn how to write functions in VBA that can be used in formulas in your Excel worksheets.

To work along, use Lesson37.xlsm.

Notes from the Lesson

- You can add new functions to Excel. Your industry might have a calculation that Excel does not support, and you might design a new function to perform this calculation.
- To start a function in VBA, type the word **`Function`**, then the name of the function, and then an opening and closing parenthesis.
- If your function requires arguments, list them in the parentheses. Each function needs a variable name and a type.
- The code in the worksheet function is not allowed to change anything in a worksheet.
- During the course of the macro code, you must assign the answer to the function to a variable that has the same name as the name of the function.
- Excel must decide whether cells containing your function should be included in the calculation chain. If your calculation might change even when no cell values have changed, you need to declare your function as volatile. Include this line of code in your function:
  `Application.Volatile`
- If the function is stored in Workbook1, the function is available only to worksheets in Workbook1. If you move the function to your personal macro workbook, you will have to include the name of the workbook in the formula; for example, `=Personal.xlsb!Sumcolor (B11,B2:B23)`.
- Functions stored in add-in files are available to all open workbooks while the add-in is installed. You must be sure that anyone who will open your workbook has the same add-in installed.

Code from the Lesson

Code in Module1:

```
Function MyPath()
    MyPath = ThisWorkbook.Path
End Function
```

Code in Module2:

```
Function SumColor(CellColor As Range, SumRange As Range)
    Application.Volatile
    ThisColor = CellColor.Interior.ColorIndex
    SumColor = 0
    For Each cell In SumRange
        If cell.Interior.ColorIndex = ThisColor _
            Then SumColor = SumColor + cell
    Next cell
End Function
```

PART VIII

Power Tools

Lessons in this part show you how to capitalize on Excel's power tools in Excel VBA. You learn how to use filters, pivot tables, and charts. The final lesson in this part shows off a special type of macro that will run in response to many events.

There are five lessons in this part:

38 Advanced Filter

39 Filter Instead of Looping

40 Pivot Tables

41 Charts

42 Event Handler Macros

Each of these lessons includes a favorite technique. Lesson 42 has to win as the coolest lesson in this part because of the speed with which the event handler macro can run.

lesson 38

Advanced Filter

What You Will Learn

Learn how to take advantage of the powerful Advanced Filter tool. Although this tool is difficult to use in the Excel interface, it is easy in VBA.

To work along, use Lesson38.xlsm.

Notes from the Lesson

- The Advanced Filter command in Excel is a tool that can do eight different things. The plethora of options is why it you rarely see people using Advanced Filter in the regular Excel interface. You learn how to use two of the eight options in this lesson.
- You can use Advanced Filter to get a unique list of values in a column. To do this in the Excel interface, you would specify a list range: Specify a single-cell output range that contains the heading from the column from which you want unique values. Then specify that you want Unique Records Only. Figure 5 shows the Advanced Filter dialog for getting a unique list of values.

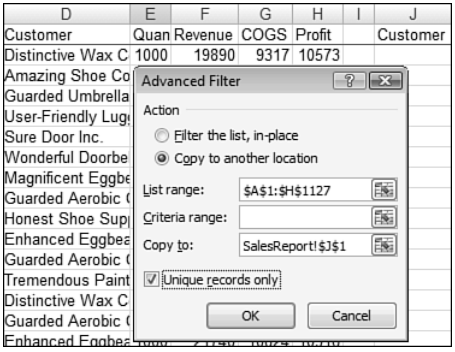


Figure 5 Use these settings in the Excel interface to get a unique list of customers in the data set.

The following is the equivalent code in VBA:

```
' Set up output range. Copy heading from D1 there
Range("D1").Copy Destination:=Range("J1")
```

```
' Do the Advanced Filter to get unique list of customers
Range("A1:H1127").AdvancedFilter Action:=xlFilterCopy, _
        CriteriaRange:="", CopyToRange:=Range("J1"), Unique:=True
```

- You can use Advanced Filter to copy all records that match a certain criteria. A simple criteria could consist of a field heading in cell J1 and the value to match in J2. You can then specify all or a subset of field headings in an output range starting in cell L1. The dialog box in Figure 6 shows how to get all records for the Amazing Shoe Company. Only columns for Date, Product, and Quantity will be copied to the output range.

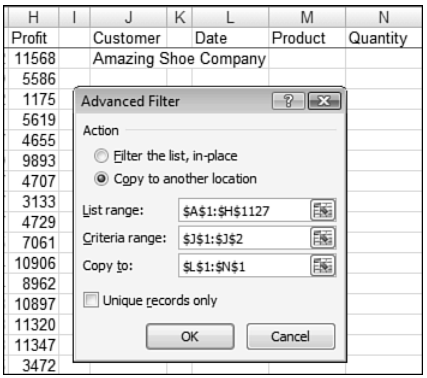


Figure 6 Use these settings in the Excel interface to extract all records for the customer shown in J2.

The following is the equivalent code in VBA:

```
' Set up the Criteria Range with one customer
Range("J1") = Range("D1")
Range("J2") = "Amazing Shoe Company"
' Set up output range.
' We want Date, Product, Quantity
' These columns are in C, B, and E
Range("L1:N1").Value = _
    Array(Cells(1, 3), Cells(1, 2), Cells(1, 5))
' Do the Advanced Filter to get unique list of cust & prod
Range("A1:H1127").AdvancedFilter Action:=xlFilterCopy, _
    CriteriaRange:=Range("J1:J2"), _
    CopyToRange:=Range("L1:N1")
```

- A common routine is to follow these steps:
    1. Use the Advanced Filter to get a unique list of customers to the right of the data set.
    2. Set up a criteria range and an output range.
    3. Loop through each customer in the result for step 1.
    4. For each customer, copy the customer to row 2 of the criteria range and then do the second type of advanced filter. Copy those results to a new workbook or a new worksheet.

Code from the Lesson

```
Sub RunReportForEachCustomer()
    Dim IRange As Range ' Input Range
    Dim ORange As Range ' Output Range
    Dim CRange As Range ' Criteria Range
    Dim WBN As Workbook ' New Workbook
    Dim WSN As Worksheet ' New Worksheet
    Dim WSO As Worksheet ' Original Worksheet

    ' Since this is called from a button on Menu,
    ' first select the sample data sheet
    Worksheets("SalesReport").Select
    ' Clear out results of previous macros
    Range("J1:AZ1").EntireColumn.Delete

    Set WSO = ActiveSheet
    ' Find the size of today's dataset
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    NextCol = Cells(1, Columns.Count).End(xlToLeft).Column + 2

    ' First - get a unique list of customers in J
    ' Set up output range. Copy heading from D1 there
    Range("D1").Copy Destination:=Cells(1, NextCol)
    Set ORange = Cells(1, NextCol)

    ' Define the Input Range
    Set IRange = Range("A1").Resize(FinalRow, NextCol - 2)

    ' Do the Advanced Filter to get unique list of customers
    IRange.AdvancedFilter Action:=xlFilterCopy, _
        CriteriaRange:="", CopyToRange:=ORange, Unique:=True

    FinalCust = Cells(Rows.Count, NextCol).End(xlUp).Row

    MyPath = ActiveWorkbook.Path & Application.PathSeparator

    ' Loop through each customer
    For Each cell In Cells(2, NextCol).Resize(FinalCust - 1, 1)
        ThisCust = cell.Value

        ' Set up the Criteria Range with one customer
        Cells(1, NextCol + 2).Value = Range("D1").Value
        Cells(2, NextCol + 2).Value = ThisCust
        Set CRange = Cells(1, NextCol + 2).Resize(2, 1)

        ' Set up output range.
        ' We want Date, Quantity, Product, Revenue
```

```
        ' These columns are in C, E, B, and F
        Cells(1, NextCol + 4).Resize(1, 4).Value = _
            Array(Cells(1, 3), Cells(1, 5), Cells(1, 2), Cells(1, 6))
        Set ORange = Cells(1, NextCol + 4).Resize(1, 4)

        ' Do the Advanced Filter to get unique list of cust & prod
        IRange.AdvancedFilter Action:=xlFilterCopy, _
            CriteriaRange:=CRange, CopyToRange:=ORange

        ' Create a new workbook with 1 sheet to hold the output

        Set WBN = Workbooks.Add(xlWBATWorksheet)
        Set WSN = WBN.Worksheets(1)

        ' Set up a title on WSN
        WSN.Cells(1, 1).Value = "Report of Sales to " & ThisCust

        ' Copy data from WSO to WSN
        WSO.Cells(1, NextCol + 4).CurrentRegion.Copy _
            Destination:=WSN.Cells(3, 1)
        TotalRow = WSN.Cells(65536, 1).End(xlUp).Row + 1
        WSN.Cells(TotalRow, 1).Value = "Total"
        WSN.Cells(TotalRow, 2).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
        WSN.Cells(TotalRow, 4).FormulaR1C1 = "=SUM(R2C:R[-1]C)"

        ' Format the new report with bold
        WSN.Cells(3, 1).Resize(1, 4).Font.Bold = True
        WSN.Cells(TotalRow, 1).Resize(1, 4).Font.Bold = True
        WSN.Cells(1, 1).Font.Size = 18

        WBN.SaveAs MyPath & ThisCust & ".xls"
        WBN.Close SaveChanges:=False

        WSO.Select
        Set WSN = Nothing
        Set WBN = Nothing

        ' clear the output range, etc.
        Cells(1, NextCol + 2).Resize(1, 10).EntireColumn.Clear
    Next cell

    Cells(1, NextCol).EntireColumn.Clear
    MsgBox FinalCust - 1 & " Reports have been created!"
End Sub
```

lesson 39

Filter Instead of Looping

What You Will Learn

Learn a new approach to Lesson 26, "Deleting While Looping." In this lesson, you learn how to delete or copy certain records without looping through each record in the data set.

Related Lesson

- The original problem appeared in Lesson 26.

To work along, use Lesson39.xlsm.

Notes from the Lesson

- You can use the `AutoFilter` method to find all the records that match a certain criteria. In VBA, apply the `.AutoFilter` method to any cell in your data set. Excel will expand that one cell to the current region around it. The following is the VBA code to turn on the filter drop-downs and filter column D:

```
Cells(1, 1).AutoFilter field:=4, Criteria1:="S29"
```

- The obscure Go To Special dialog in Excel enables you to select only the visible cells that are the result of the filter. In VBA, use `.SpecialCells(xlCellTypeVisible)` to refer to the visible cells.
- Note that you apply the `.SpecialCells` property to a range starting in row 2. If you included the headings in the range, you would delete the headings as well. This is not what you want to do.
- To turn off the AutoFilter drop-downs, reapply the `.AutoFilter` method without any parameters.

Code from the Lesson

```
Sub LoopWay()
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    For i = FinalRow To 2 Step -1
        If Cells(i, 4) = "S29" Then
            Cells(i, 1).EntireRow.Delete
        End If
    Next i
End Sub

Sub FasterWay()
    Dim rng As Range
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row

    ' While the headings are in row 1, this range start in row 2
    Set rng = Cells(2, 1).Resize(FinalRow - 1, 1)

    ' Apply a filter to the dataset
    Cells(1, 1).AutoFilter field:=4, Criteria1:="S29"

    ' Delete the visibile cells starting in row 2
    rng.SpecialCells(xlCellTypeVisible).EntireRow.Delete

    ' Turn of the filter
    Cells(1, 1).AutoFilter
End Sub
```

lesson 40

Pivot Tables

What You Will Learn

Learn how to create a pivot table in Excel VBA. Pivot tables are Excel's most powerful feature. You can use a pivot table to summarize thousands of rows down to a one-page summary very quickly.

To work along, use Lesson40.xlsm.

Notes from the Lesson

- The Excel interface's most powerful feature works great in VBA.
- To start, you will define a pivot table cache. The cache is an area in memory that holds all the data from the original worksheet. In the code listing for this lesson, look for the line with `Set PTCache`. This line uses the `PivotCaches.Add` method to set up the cache.
- After you've defined the pivot table cache, use the `.CreatePivotTable` method to turn the cache into a pivot table. This will create an empty pivot table report.
- Before you start adding fields to the pivot table, set the `.ManualUpdate` property to True. This enables you to run several lines of code, adding fields. After you've added all the fields, change the `.ManualUpdate` property to False to actually draw the finished pivot table.
- You can add row fields, column fields, and page fields in one line of code using the `.AddFields` method. Note that in the Excel 2007 interface, the Page Field area is now called the Report Filter area. In VBA, it is still called a Page Field.
- You will have several lines of code for each field in the data area of the pivot table. You define the field as an `xlDataField` and specify the caption, the function, the number format, and so on.
- If you add two or more data fields to your report, include a virtual field called "Data" in either the `RowField` or `ColumnField` parameter of the `.AddFields` method.
- There are a variety of advanced tools that you can use with your pivot table. The following macro shows how to group daily dates up to months, quarters, and years. The sample workbook includes another module with other pivot table examples for using `.AutoSort`, the Top 10 filter, and more.
- After you create the pivot table, you might want to convert it to values. Use the `.TableRange2` property to refer to the entire pivot table, including the page area. Use the `.TableRange1` property to ignore the page fields and select from the column headings down. Because you cannot convert part of a pivot table to values, use the `.TableRange2` property in the following code:

```
     PT.TableRange2.Copy
     PT.TableRange2.PasteSpecial xlPasteValues
```

Code from the Lesson

```
Sub CreatePivotLesson40()
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long

    Set WSD = Worksheets("PivotTable")
    Dim WSR As Worksheet

    ' Delete any prior pivot tables
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("J1:Z1").EntireColumn.Clear

    ' Define input area and set up a Pivot Cache
    FinalRow = WSD.Cells(Application.Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Application.Columns.Count). _
        End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:= _
        xlDatabase, SourceData:=PRange.Address)

    ' Create the Pivot Table from the Pivot Cache
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD. _
        Cells(2, FinalCol + 2), TableName:="PivotTable1")

    ' Turn off updating while building the table
    PT.ManualUpdate = True

    ' Set up the row fields
    PT.AddFields RowFields:="Date", ColumnFields:="Data", _
        PageFields:="Customer"

    ' Set up the data fields
    With PT.PivotFields("Revenue")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
        .NumberFormat = "#,##0,K"
        .Name = "Total Revenue"
    End With
```

```
With PT.PivotFields("Revenue")
     .Orientation = xlDataField
     .Function = xlCount
     .Position = 2
     .NumberFormat = "#,##0"
     .Name = "Number Orders"
End With

With PT.PivotFields("Revenue")
     .Orientation = xlDataField
     .Function = xlAverage
     .Position = 3
     .NumberFormat = "#,##0"
     .Name = "Average Revenue"
End With

With PT.PivotFields("Revenue")
     .Orientation = xlDataField
     .Function = xlMin
     .Position = 4
     .NumberFormat = "#,##0,K"
     .Name = "Smallest Order"
End With

With PT.PivotFields("Revenue")
     .Orientation = xlDataField
     .Function = xlMax
     .Position = 5
     .NumberFormat = "#,##0"
     .Name = "Largest Order"
End With

' Set up a percentage of total
With PT.PivotFields("Revenue")
     .Orientation = xlDataField
     .Caption = "PctOfTotal"
     .Function = xlSum
     .Position = 6
     .NumberFormat = "#0.0%"
     .Calculation = xlPercentOfTotal
End With

' Set up Running Total
With PT.PivotFields("Revenue")
     .Orientation = xlDataField
     .Function = xlSum
```

```
        .Caption = "YTD Total"
        .Calculation = xlRunningTotal
        .Position = 7
        .NumberFormat = "#,##0,K"
        .BaseField = "Date"
    End With

    ' Ensure that we get zeros instead of blanks in the data area
    PT.NullString = "0"

    ' Calc the pivot table to allow the date label to be drawn
    PT.ManualUpdate = False
    PT.ManualUpdate = True

    ' Group ShipDate by Month, Quarter, Year
    PT.PivotFields("Date").LabelRange.Group Start:=True, _
        End:=True, Periods:= _
        Array(False, False, False, False, True, True, True)

    ' Calc the pivot table
    PT.ManualUpdate = False
    PT.ManualUpdate = True
    WSD.Activate
    Range("J2").Select

    ' Selecting the PivotTable
    PT.TableRange2.Select

    PT.TableRange1.Select

End Sub
```

lesson 41

Charts

What You Will Learn

Learn how to create charts using Excel VBA.

To work along, use Lesson41.xlsm.

Notes from the Lesson

- The charting engine was rewritten for Excel 2007. The macro recorder for charting was not finished in time for the product to ship.
- The easiest solution is to record macros in Excel 2003. If you do not have access to Excel 2003, you might have to use the Watch Window to discover properties to use.
- Use the `.AddChart` method to add a new chart. Specify one of the 73 chart types as the type. Use the `.SetSourceData` method to specify the data for the chart.
- The `.AddChart` method includes `.Left` and `.Top` properties to specify the location of the chart. These properties use a strange measurement—the number of points from the left edge of the Excel worksheet and the number of points from the top of the worksheet. One cool trick is to assign this property a value such as `Range("E4").Left`.
- Without a macro recorder to rely on, you might need to investigate properties for a chart. Select a chart and run the ExploreChartElementsLL macro. Add a watch for `cht`, `chtg`, and `ser`. Expand the Watch Window to find current settings for the properties in the chart. If you do this, then change something in the chart, and then repeat these steps, you might notice a change in the VBA properties.
- The following code is stored in Module2 of the sample files included with the DVD. Explore additional code in Module1 for examples of other charting tasks.

Code from the Lesson

```
Sub CreateChartLesson41()
    Dim WS As Worksheet
    Dim upb As UpBars
    Dim cht As Chart

    Set WS = Worksheets("Lesson41")

    WS.Shapes.AddChart(xlLine, _
        Left:=WS.Range("E2").Left, _
        Top:=WS.Range("E2").Top, _
        Width:=WS.Range("E2:L14").Width, _
        Height:=WS.Range("E2:L14").Height _
        ).Select
    ActiveChart.SetSourceData Source:=WS.Range("A3:C15")

    With ActiveChart.ChartGroups(1)
        .HasUpDownBars = True
 '       .DownBars.Interior.ColorIndex = 3
 '       .UpBars.Interior.ColorIndex = 5
        .DownBars.Format.Fill.ForeColor.RGB = RGB(255, 0, 0)
        .UpBars.Format.Fill.ForeColor.RGB = RGB(0, 255, 0)
    End With
End Sub

Sub ExploreChartElementsLL()
    ' Select a chart and then run this macro
    ' The macro defines 3 objects and stops in debug mode.
    ' You can then use the Debug - Add Watch to explore the
    ' objects and their properties.
    Dim cht As Chart
    Dim chtg As ChartGroup
    Dim ser As Series
    Set cht = ActiveChart
    Set chtg = cht.ChartGroups(1)
    Set ser = cht.SeriesCollection(1)
    Stop
End Sub
```

lesson 42

Event Handler Macros

What You Will Learn

Learn how to have a macro automatically run in response to an event. Some of the many events that can trigger a macro include selecting a new cell in a worksheet, changing a cell in a worksheet, opening a workbook, closing a workbook, printing a workbook, and so on.

To work along, use Lesson42.xlsm.

Notes from the Lesson

- The first macro in the lesson is designed to allow you to enter a time without entering the colon. You type a three- or four-digit number and the macro automatically inserts the colon.
- Event handlers are not entered in a module; they must be entered in the code pane associated with the worksheet or the workbook. To get to this code pane, right-click the worksheet in the Project Explorer and choose View Code.
- Two drop-downs appear at the top of the code window. The left drop-down includes only one selection, either Worksheet or Workbook. Choose the one item in the drop-down, and the right drop-down then will show a list of the available events. When you choose an item from the right drop-down, VBA automatically inserts the first and last line of the macro. Depending on the item chosen, various variables might be available to the macro. VBA will type in these variable names as the first line of the macro.
- Note that as soon as you choose Worksheet from the left drop-down, the right drop-down defaults to SelectionChange, and the first few lines of the Worksheet_SelectionChange macro are added to the code pane. Feel free to delete these lines if you don't want to have code run for this event.
- In the lesson, the first event is designed to run every time a cell is changed. If the macro writes back to the worksheet, this will be considered another change and the macro will run again, which sets up a recursive loop. To prevent the Worksheet_Change macro from running again, use `Application.EnableEvents = False` before you write to the worksheet. You must be sure you set the `.EnableEvents` property back to True before the macro ends.
- Other macros in this lesson include highlighting the current cell in yellow with a worksheet SelectionChange macro. The lesson also talks about code that is run every time a workbook opens.

Tip

If you have event handler macros in a workbook and want to open the workbook for debugging, you can turn off the event handlers before opening the workbook. Open Excel and switch to VBA, and then press Ctrl+G to open the immediate pane. Type **`Application.EnableEvents = False`** and press Enter. Switch back to the Excel window and open your workbook. None of the event handlers will be running.

Code from the Lesson

Code in Sheet1:

```
Private Sub Worksheet_Change(ByVal Target As Range)
    Select Case Target.Column
        Case 2, 3
            ThisTime = Target.Value
            Select Case Len(ThisTime)
                Case 3
                    NewTime = Left(ThisTime, 1) & ":" & _
                        Right(ThisTime, 2)
                Case 4
                    NewTime = Left(ThisTime, 2) & ":" & _
                        Right(ThisTime, 2)
                Case Else
                    NewTime = ThisTime
            End Select

            Application.EnableEvents = False
            Target.Value = NewTime
            Application.EnableEvents = True
    End Select

End Sub
```

Code in Sheet11:

```
 ' Page 165
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Dim iColor As Integer

    On Error Resume Next
    iColor = Target.Interior.ColorIndex
    If iColor < 0 Then
        iColor = 36
    Else
        iColor = iColor + 1
    End If
    If iColor = Target.Font.ColorIndex Then iColor = iColor + 1
    Cells.FormatConditions.Delete
```

```
    With Range("A" & Target.Row, Target.Address)
         .FormatConditions.Add Type:=2, Formula1:="TRUE"
         .FormatConditions(1).Interior.ColorIndex = iColor
    End With
    With Range(Target.Offset(1 - Target.Row, 0).Address _
         & ":" & Target.Offset(-1, 0).Address)
         .FormatConditions.Add Type:=2, Formula1:="TRUE"
         .FormatConditions(1).Interior.ColorIndex = iColor
    End With
End Sub
```

Code in ThisWorkbook:

```
Private Sub Workbook_Open()
    Worksheets("TimeLog").Select
    MsgBox "Read the instructions before proceeding."

End Sub
```

PART IX

Interacting

When you first start writing Excel macros, you probably will be writing utilities to make your life easier. Eventually, though, you will be writing macros that other people will use. Because these people might not be familiar with the macro or the steps happening behind the scenes, you often need to report what is happening and occasionally ask them a question. The lessons in this part show you how to interact with the person operating the computer.

There are six lessons in this part:

Lesson 48 will get you the most bang for your buck—cutting run times by 50% or more. Frankly, Lesson 45 is the most important, but I usually try to hang on to the techniques in Lesson 43 before I ever open a new userform, simply because designing userforms can be a tedious process.

lesson 43

Interacting

What You Will Learn

Learn how to report information to the person running the macro. Learn how to ask simple questions.

To work along, use Lesson43.xlsm.

Notes from the Lesson

- Use Msgbox "The Report is Done" as a simple way to report a status. The message appears with a single OK button. The second macro in the lesson modifies the message box with a title and one of four built-in icons.
- To report a status without requiring the person to click OK repeatedly, you can display a message in the status bar at the bottom of the screen. Use `Application.StatusBar = "Message"` throughout your code. This way, the person can watch the status bar to see that your macro continues to make progress. At the end of the macro, use `Application.StatusBar = False` to allow the status bar to go back to the usual messages of Ready, Enter, and Edit.
- To ask a question, use the `InputBox` method as shown in the AskSomething macro. Put your question as the `Prompt` argument. You also can specify a title and default value. There is no error checking in the `InputBox`, so always be sure your code makes sure the person typed a suitable answer. If you are expecting a number and they type a word, you will have problems when you use the answer in a loop.
- You can ask questions with the `MsgBox`! `Use the Buttons` parameter to specify whether the dialog should include OK and Cancel; Yes, No, Cancel; Retry, Cancel; and so on. If you use multiple buttons, the `MsgBox` will return an answer such as `vbYes`, `vbNo`, `vbCancel`, and so on.
- The `Buttons` parameter offers many options that do different things. Suppose you want OK and Cancel buttons, you want the icon to be the question mark, and you want the second button to be the default. You would add all those settings together as in this example:

```
Buttons:=vbOKCancel + vbDefaultButton2 + vbQuestion
```

Code from the Lesson

```
Sub SaySomething()
    MsgBox "The Report is Done"
End Sub

Sub SaySomethingWithTitle()
    MsgBox Prompt:="The Report is Done", Title:="MrExcel.com"
End Sub

Sub SaySomethingWithTitleAndIcon()
    MsgBox Prompt:="The Report is Done", _
        Title:="MrExcel.com", Buttons:=vbCritical
End Sub

Sub StatusBarProcess()
    For i = 1 To 1000
        Cells(i, 20).Value = i
        Application.StatusBar = i & " out of 1000"
    Next i

    For i = 1000 To 5000
        Cells(i, 20).Value = i * i
        Application.StatusBar = i & " out of 5000"
    Next i

    Application.StatusBar = False

End Sub

Sub AskSomething()
    x = InputBox(Prompt:="How many months should be included?", _
        Title:="Enter Months", Default:=3)
    ActiveCell.Value = x
End Sub

Sub AskSomethingWithMsgBox()
    ans = MsgBox(Prompt:="Color this cell red?", _
        Buttons:=vbYesNo + vbQuestion + vbDefaultButton2, _
        Title:="MrExcel.com")
    If ans = vbYes Then
        ActiveCell.Interior.ColorIndex = 3
    End If
End Sub
```

lesson 44

Getting File Names

What You Will Learn

Learn how to ask for a path and file name. This is useful when you are trying to figure out which file to open or when you want the person to specify where to save the new report that your macro just created.

To work along, use Lesson44.xlsm.

Notes from the Lesson

- Suppose you want to design a dialog that enables a person to browse to a folder and specify a file name. It would take you hours to re-create all the functionality in the Excel Open dialog. Luckily, VBA provides a way to display either the Open or Save As dialogs; however, it isn't the real Open or Save As dialog. When the person clicks Save or Open, the file is neither saved or opened. Instead, VBA returns a variable that gives you the path and file name.
- Use `Application.GetSaveAsFileName` to display something similar to the Save As dialog.
- Use `Application.GetOpenFileName` to display something similar to the Open dialog.

Code from the Lesson

```
Sub CreateReport()
    ' Lots of code here

    ' Save the file
    Filename = Application.GetSaveAsFilename( _
        fileFilter:="Excel Files (*.xlsx), *.xlsx")
    MsgBox "You selected " & Filename
    Stop
    If Filename = False Then

    Else
        ActiveWorkbook.SaveAs Filename:=Filename
    End If
End Sub
```

```
Sub WhichFileToOpen()
    FN = Application.GetOpenFilename()
    MsgBox "You selected " & FN

    Stop
    Workbooks.Open Filename:=FN
End Sub
```

lesson 45

Userforms

What You Will Learn

Learn how to create your own custom dialog boxes using the UserForm tools in Excel VBA.

Related Lessons

- If you need to ask only simple questions, consider using the InputBox tool in Lesson 43.

To work along, use Lesson45.xlsm.

Notes from the Lesson

- A userform enables you to build your own custom dialog. Figure 7 shows the dialog built during this lesson. Note that code in the userform ensures that the Shift question appears only when the employee is in the Manufacturing department.
- The userform in Figure 7 contains five labels, two text boxes, one list box, one frame, two option buttons, one spin button, and two command buttons. Note that although you are asking only five questions, 21 separate controls had to be added to the userform.

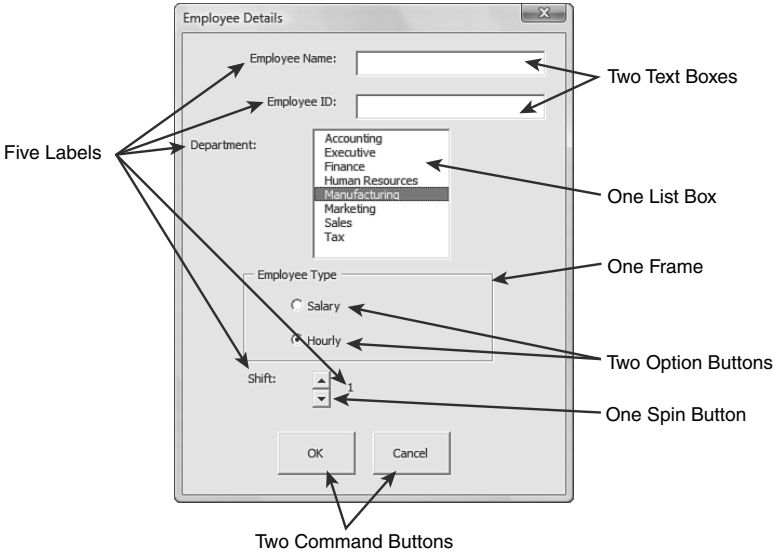


Figure 7 A custom dialog box to collect five fields contains 21 separate controls.

- To start, switch to VBA and select Insert UserForm. You will get a generic userform as shown in Figure 8. Use the handle in the lower-right corner to resize it.
- While the userform is selected, click inside the properties pane. Change the Name property to something easier to remember, such as frmEmployee. Change the Caption property to reflect the words that should appear in the title bar of the dialog box.

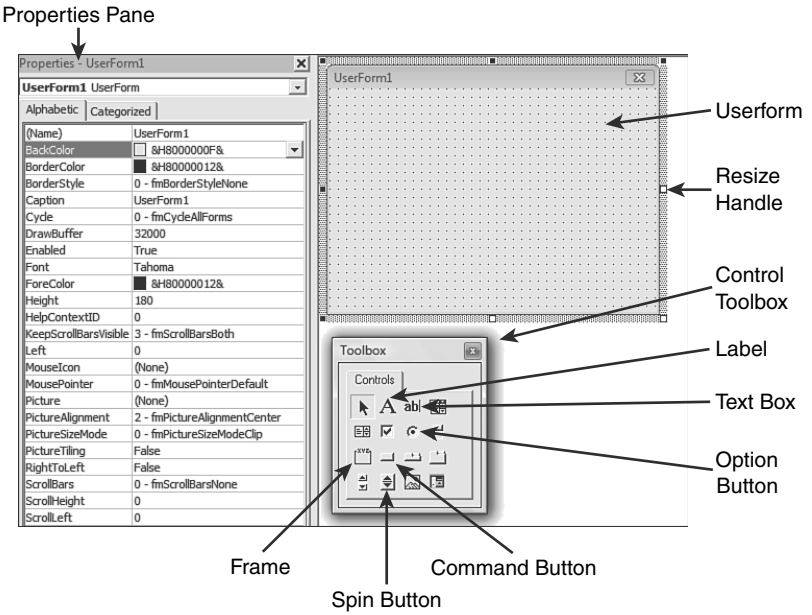


Figure 8 The initial userform starts out at this size. Use the lower-right resize handle to resize.

- Click the TextBox icon in the Control Toolbox, and then drag a rectangular area on the userform. This will be the area where someone can type a value. While the text box is selected, change the name of the control in the properties pane from TextBox1 to tbEmployeeName. The "tb" prefix indicates that this is a textbox control. EmployeeName is an easier to remember name. You will refer to these names later when you are writing the code behind the form.
- Click the Label icon in the Control Toolbox, and then drag a rectangular label area to the left of the textbox that you just added. Change the Caption property to Employee Name:. Change the TextAlign property to 3 – frmTextAlignRight. You probably never will refer to this label programmatically, so it is not necessary to change its name.
- After you've designed a label and textbox, select them both by clicking one and Ctrl+clicking the other. Replicate these controls by Ctrl+Dragging to a new location. This will ensure that your fields all have similar size and shape.
- A list box presents items from a list. A combo box enables you to choose from a list but also type a new value that is not in the list.
- To fill a list box, use the `RowSource` property. If you hide the valid list on a worksheet, you can specify that range in the `RowSource` property; for example, `Lists!A1:A9`.
- You can select only one option button at a time. If you need to have multiple sets of option

buttons on the same form, enclose each set in its own frame. Click the Frame icon in the Control Toolbox and draw a box on the userform. Change the Caption property to display a title for the box. Then, click the OptionButton icon and draw multiple option buttons inside the frame. Use the Caption property to change the label next to the option buttons.

- A spin button is perfect for numeric entries. You can specify a lower limit (Min), upper limit (Max), and the amount to jump (SmallChange) every time a person adjusts the spin button up or down. The problem with a spin button is that there is no caption property to display the value of the spin button, so you must draw in your own label next to the spin button. Note the name of the label. Right-click the spin button and choose View Code. The default macro is the `_Change` event. The code in this macro will write the current value of the spin button to the label: `Me.LabShift = Me.SbShift.Value`. You will also add this line of code to the Userform_Initialize macro, later.
- For the command button, use the Caption property to change the words on the button.
- Try to add fields in the proper order as someone would flow through the form. If you add a field in the center of the form at the end, the TabIndex property is likely to be out of sequence. Check that the TabIndex property for your controls flows from lower to higher.
- Right-click the Cancel button and choose View Code. The one line of code is `Unload Me`, which hides the dialog box. Note that `Me` is a built-in object variable that refers to the current userform.
- The code behind the OK button must write the values from the userform out to your worksheet. See the CommandButton1_Click macro in the code for this lesson.
- In this lesson, the shift information is to appear only when the department is Manufacturing. To do this, the Userform_Initialize macro, which is run as the form is being displayed, made all those controls hidden. You can add code here to hide items, add items on the fly to list boxes, and so on. After you've hidden the shift information you need to redisplay it if the department is Manufacturing. Right-click the list box and choose View Code.
- When you right-click a control and choose View Code, you will be taken to a default event for that control. For a list box, the default macro is the Click event. Although this is often the correct macro, there are many other events for which you can have code. The control name will appear in the left drop-down at the top of the code window. Explore the various events in the right drop-down at the top of the code window. For the department list box, choose Change from the top-right drop-down, and then you can add code that will check to see whether the new department is Manufacturing. If it is, the three controls related to shift have their `.Visible` property set back to True. If the person chooses something other than Manufacturing, change the `.Visible` property back to False.

Note

It is tempting to code only for the Manufacturing example because `Userform_Initialize` originally hid the shift controls. However, imagine what happens if someone accidentally chooses Manufacturing and then changes to Marketing. The original mischoice of Manufacturing will make the shift controls appear. The code must make those hidden when the person chooses Marketing again.

- You need a one-line macro in a regular module to display the form. The code is *`frmName`*`.Show`.

Code from the Lesson

Code in frmEmployee:

```
Private Sub CommandButton1_Click()
    ' Find the next row on the data sheet
    NextRow = Worksheets("Data"). _
        .Cells(Rows.Count, 1).End(xlUp).Row + 1

    ' Write the values from this form to that row
    With Worksheets("Data")
        .Cells(NextRow, 1).Value = Me.tbEmpName
        .Cells(NextRow, 2).Value = Me.tbID
        .Cells(NextRow, 3).Value = Me.lbDept.Value
        If Me.OptHourly = True Then
            .Cells(NextRow, 4) = "Hourly"
        Else
            .Cells(NextRow, 4) = "Salary"
        End If
        .Cells(NextRow, 5).Value = Me.SBShift.Value
    End With

    ' Unload the form
    Unload Me
End Sub

Private Sub CommandButton2_Click()
    Unload Me
End Sub

Private Sub lbDept_Change()
    If Me.lbDept.Value = "Manufacturing" Then
        Me.SBShift.Visible = True
        Me.LabShift.Visible = True
        Me.Label4.Visible = True
    Else
        Me.SBShift.Visible = False
        Me.LabShift.Visible = False
        Me.Label4.Visible = False
    End If

End Sub

Private Sub SBShift_Change()
    Me.LabShift = Me.SBShift.Value
End Sub

Private Sub UserForm_Initialize()
    Me.SBShift.Value = 1
    Me.LabShift = Me.SBShift.Value
    Me.SBShift.Visible = False
    Me.LabShift.Visible = False
    Me.Label4.Visible = False

End Sub
```

Code in Module1:

```
Sub ShowEmployeeForm()
    frmEmployee.Show
End Sub
```

lesson 46

Error Handling

What You Will Learn

Learn how to make your application more robust. If you can anticipate the places where things could go wrong in your macro, you can add an error handler that will provide a friendlier message to the person running the macro.

To work along, use Lesson46.xlsm.

Notes from the Lesson

- Typically, an error encountered while running a macro will cause an error message to appear. The person then can choose Continue, End, Debug, or Help. If that person is a programmer, they will often click Debug, but if you are writing the macro for someone else, they will often call you.
- Suppose you are counting on the I.T. department to put a file out on a network drive every night. Your macro loads this file and produces some reports. If there are delays in the nightly processing, this file might not get created. You can predict that the `Workbook.Open` code might generate an error. This would be an ideal line to protect with a custom error handler.
- Before the line of code that might cause an error, add a new line with `On Error GoTo` *`HandlerName`*. Add new lines of code at the bottom of the macro, just before `End Sub`:

```
Exit Sub
HandlerName:
Msgbox "It looks like this error happened. Call xyz and do xyz"
```

- The `Exit Sub` line prevents the macro from running the error handler lines at the end of the macro. `HandlerName` followed by a colon provides a labeled section of code. The `GoTo` *`HandlerName`* directive will send code execution to that section of the macro. In the error handler, you can display a friendlier message, provide detailed instructions, and so on. At the end of the error handler, you can either use `Resume Next` or `Exit Sub`.
- Your custom error handler is designed to handle an error with the `Workbooks.Open` line of code. If you get through that line of code without an error, you should turn off the custom error handler. The somewhat cryptic way to do this is with `On Error GoTo 0`. Students in my classes frequently ask, "Where is zero?" There is no zero—this is just the way to tell the error handler to go back to the default behavior of displaying Continue, End, Debug, and Help.

- Some errors can be ignored. If you try to delete a file and it is not there, Excel will halt the macro execution with an error. Think about this: You are trying to delete the file, and it actually is a good thing if it's not there. If you want to prevent an error message for this situation, precede the line of code with `On Error Resume Next`. After deleting the file, go back to `On Error GoTo 0`. Don't be tempted to simply use `On Error Resume Next` throughout your code! Some errors cannot allow code execution to continue, in which case Excel will stop the current macro but go back to the macro that called this one and continue executing. This is always a disaster.
- When an error is encountered, a special object variable called `Err` is available. After every line of code, you can check `Err.Number` and `Err.Description`. If the line of code ran successfully, `Err.Number` will be zero. You can use this variable to check for the existence of a worksheet. In the IsItThere macro, the code loops through every worksheet in a workbook, looking for a particular worksheet. In the faster IsItThere2 macro, the code simply refers to the worksheet and then checks the value of `Err.Value`. If it is zero, the worksheet exists. If it is nonzero, the worksheet is missing. Be sure to use `On Error Resume Next` before this code to allow execution to continue to the `If` statement that checks `Err.Value`.
- You can shorten code by directly referring to an object that might or might not exist. In essence, you are accepting that your code would cause an error. By using `On Error Resume Next` and then checking the value of the `Err` object, you can determine if the object exists.

Code from the Lesson

```
Sub IsItThere()
    FridayFound = False
    For Each ws In ActiveWorkbook.Worksheets
        If ws.Name = "Friday" Then
            FridayFound = True
            Exit For
        End If
    Next ws

    If Not FridayFound Then
        MsgBox "The required Friday worksheet is missing"
        Exit Sub
    End If
End Sub

Sub IsItThere2()

    Err.Clear
    On Error Resume Next
    x = Worksheets("Friday").Cells(1, 1)

    If Err.Number = 0 Then
        ' Friday is there. Everything is cool
    Else
```

```
        MsgBox "The required Friday worksheet is missing"
        Exit Sub
    End If
    On Error GoTo 0
End Sub

Sub RunReportForEachCustomer()
    ' Page 272
    Dim IRange As Range ' Input Range
    Dim ORange As Range ' Output Range
    Dim CRange As Range ' Criteria Range
    Dim WBN As Workbook ' New Workbook
    Dim WSN As Worksheet ' New Worksheet
    Dim WSO As Worksheet ' Original Worksheet

    ' Open the workbook from the I.T. department
    On Error GoTo FileNotThere
    Workbooks.Open ("C:\DailyOrders.xlsb")
    On Error GoTo 0

    ' first select the data sheet
    Worksheets("SalesReport").Select
    ' Clear out results of previous macros
    Range("J1:AZ1").EntireColumn.Delete

    Set WSO = ActiveSheet
    ' Find the size of today's dataset
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    NextCol = Cells(1, Columns.Count).End(xlToLeft).Column + 2

    ' First - get a unique list of customers in J
    ' Set up output range. Copy heading from D1 there
    Range("D1").Copy Destination:=Cells(1, NextCol)
    Set ORange = Cells(1, NextCol)

    ' Define the Input Range
    Set IRange = Range("A1").Resize(FinalRow, NextCol - 2)

    ' Do the Advanced Filter to get unique list of customers
    IRange.AdvancedFilter Action:=xlFilterCopy, _
        CriteriaRange:="", CopyToRange:=ORange, Unique:=True

    FinalCust = Cells(Rows.Count, NextCol).End(xlUp).Row

    MyPath = ActiveWorkbook.Path & Application.PathSeparator

    ' Loop through each customer
```

```
For Each cell In Cells(2, NextCol).Resize(FinalCust - 1, 1)
    ThisCust = cell.Value

    ' Set up the Criteria Range with one customer
    Cells(1, NextCol + 2).Value = Range("D1").Value
    Cells(2, NextCol + 2).Value = ThisCust
    Set CRange = Cells(1, NextCol + 2).Resize(2, 1)

    ' Set up output range.
    ' We want Date, Quantity, Product, Revenue
    ' These columns are in C, E, B, and F
    Cells(1, NextCol + 4).Resize(1, 4).Value = Array( _
        Cells(1, 3), Cells(1, 5), Cells(1, 2), Cells(1, 6))
    Set ORange = Cells(1, NextCol + 4).Resize(1, 4)

    ' Do an Advanced Filter to get unique list of cust & prod
    IRange.AdvancedFilter Action:=xlFilterCopy, _
        CriteriaRange:=CRange, CopyToRange:=ORange

    ' Create a new workbook with 1 sheet to hold the output
    Set WBN = Workbooks.Add(xlWBATWorksheet)
    Set WSN = WBN.Worksheets(1)

    ' Set up a title on WSN
    WSN.Cells(1, 1).Value = "Report of Sales to " & ThisCust

    ' Copy data from WSO to WSN
    WSO.Cells(1, NextCol + 4).CurrentRegion.Copy _
        Destination:=WSN.Cells(3, 1)
    TotalRow = WSN.Cells(65536, 1).End(xlUp).Row + 1
    WSN.Cells(TotalRow, 1).Value = "Total"
    WSN.Cells(TotalRow, 2).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
    WSN.Cells(TotalRow, 4).FormulaR1C1 = "=SUM(R2C:R[-1]C)"

    ' Format the new report with bold
    WSN.Cells(3, 1).Resize(1, 4).Font.Bold = True
    WSN.Cells(TotalRow, 1).Resize(1, 4).Font.Bold = True
    WSN.Cells(1, 1).Font.Size = 18

    ' Delete any previous copy of this file
    On Error Resume Next
    Kill (MyPath & ThisCust & ".xls")
    On Error GoTo 0

    WBN.SaveAs MyPath & ThisCust & ".xls"
    WBN.Close SaveChanges:=False
```

```
        WSO.Select
        Set WSN = Nothing
        Set WBN = Nothing

        ' clear the output range, etc.
        Cells(1, NextCol + 2).Resize(1, 10).EntireColumn.Clear
    Next cell

    Cells(1, NextCol).EntireColumn.Clear
    MsgBox FinalCust - 1 & " Reports have been created!"
    Exit Sub

FileNotThere:
    MsgBox "It appears the DailyOrders Excel file has not " & _
        "been created. Please call George in the I.T. depart" & _
        "ment at extension 1234 to see if the nightly " & _
        "processing is finished."
End Sub
```

lesson 47

Suppressing Alerts

What You Will Learn

Learn how to prevent Excel from interrupting code execution to display alerts.

To work along, use Lesson47.xlsm.

Notes from the Lesson

- When you try to delete a worksheet, Excel asks whether you are sure. This will make no sense to someone watching the macro run. It will slow down the macro because someone will have to stay at the computer to dismiss this message instead of being free to work on something else.
- You can suppress these alerts with this line of code:

```
Application.DisplayAlerts = False
```

- If you later want the alerts to appear again, use this line of code:

```
Application.DisplayAlerts = True
```

Code from the Lesson

```
Sub RunReportForEachCustomer()
    ' Non-relevant code removed
    Application.DisplayAlerts = False
    Worksheets("Data").Delete
    Application.DisplayAlerts = True

    MsgBox FinalCust - 1 & " Reports have been created!"
End Sub
```

lesson 48

Faster Code

What You Will Learn

Learn how to make your code run in half the time with one simple statement.

To work along, use Lesson48.xlsm.

Notes from the Lesson

- Excel redraws the screen after every line of the macro, dramatically slowing down execution of the program.
- Suppress the screen redrawing with this line of code at the beginning of your macro:
  `Application.ScreenUpdating = False`
- When the macro finishes, Excel usually will turn the `ScreenUpdating` back to True. However, I have seen a situation in which the `ScreenUpdating` was not reset to True when Excel was called from a DOS utility. This left Excel in an unusable state because the screen would not redraw in response to a person moving around the worksheet. So make sure you turn `ScreenUpdating` back to True before you exit the macro:
  `Application.ScreenUpdating = True`
- If your macro runs for several minutes, you will want to show some progress to the person running it. Use the `.StatusBar` property to display progress (see Lesson 43, "Interacting"), or show progress by occasionally turning `ScreenUpdating` back to True and then False:
  `Application.ScreenUpdating = True`
  `Application.ScreenUpdating = False`

Code from the Lesson

```
Sub RunReportForEachCustomer()
    ' Non-relevant code removed
    StartTime = Time
    Application.ScreenUpdating = False

    ' Non-relevant code removed

    EndTime = Time
    Debug.Print StartTime, EndTime
    Application.ScreenUpdating = True
    MsgBox FinalCust - 1 & " Reports have been created!"
End Sub
```

PART X

Best Practices

If you are planning to build robust Excel VBA macros, you should consider adopting the two-workbook solution described in Lesson 49. Keeping your data and code in separate workbooks can help you roll out solutions to many coworkers with the least amount of hassle. Lesson 50 provides a recap of how to transform recorded code into fast, efficient code.

There are two lessons in this part:

lesson 49

Separating Code and Data

What You Will Learn

Learn how to use a two-workbook system. In this system, you keep your code workbook separate from the data workbook.

To work along, use any workbook.

Notes from the Lesson

- If you store macro code and data in the same workbook, and then send it out to all the people in a department, you will have a problem when you want to update the programs on all those computers.
- You will design a two-workbook system. The first workbook is the Data workbook and has very little code. It has a Workbook_Open procedure that goes out and opens the Code workbook. When the Code workbook is opened, the Data workbook runs a macro called CustFileOpen in the Code workbook.
- The second workbook is the Code workbook, which contains all your program code. The code in this workbook needs to be written to work on data in another workbook—you should refer to ActiveWorkbook instead of ThisWorkbook.
- Use the CustFileOpen macro to display any special menus or to do any initialization. This macro can also be used to deliver changes to the Data workbook. For example, if you discover that you need a new worksheet added to the Data workbook, you can write a program in CustFileOpen to add those worksheets.
- It is best if the Code workbook is hidden. Save the workbook in Excel. In the Excel interface, choose View, Hide to hide it. You can't see the workbook in the Excel interface anymore, but this means you cannot save it in the Excel interface, either. Switch to VBA and click the Code.xlsm workbook in the Project Explorer. Click the Save icon in the VBA Editor.

Code from the Lesson

```
Private Sub Workbook_Open()
    ' If the code file has been opened?
    On Error Resume Next
    x = Workbooks("Code.xlsm").Name
    ErrHolder = Err.Number
```

```
    On Error GoTo 0
    If ErrHolder <> 0 Then
        ' Try to open code file
        CodeFile = ThisWorkbook.Path & _
            Application.PathSeparator & "Code.xlsm"
        On Error Resume Next
        Workbooks.Open Filename:=CodeFile
        ErrHolder = Err.Number
        On Error GoTo 0
    End If

    ' Run Macro in code file
    If ErrHolder = 0 Then
        Application.Run "'Code.xlsm'!CustFileOpen"
    End If
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
    ' Note, here, we will ask user to decide if he
    ' really wants to save/unsave/cancel the changes
    ' If we closed the code file, and then the person
    ' cancelled to close the workbook,
    ' the program will have to run without macros.
    If ThisWorkbook.Saved = False Then
        Msg = "Workbook has been changed, " & _
            "would you like to save the changes?"
        sTitle = "Close Workbook"
        Ans = MsgBox(Prompt:=Msg, Buttons:=vbYesNoCancel + _
            vbExclamation, Title:=sTitle)
        Select Case Ans
        Case vbYes
            ThisWorkbook.Save
        Case vbNo
            ThisWorkbook.Saved = True
        Case vbCancel
            Cancel = True
            Exit Sub
        End Select
    End If

    ' If the code file has been opened?
    On Error Resume Next
    x = Workbooks("Code.xlsm").Name
    ErrHolder = Err.Number
    On Error GoTo 0
    ' Run Macro in code file
    If ErrHolder = 0 Then
        Application.Run "'Code.xlsm'!CustFileClose"
        Workbooks(x).Close SaveChanges:=False
    End If
End Sub
```

lesson 50

Cleaning Recorded Code

What You Will Learn

This lesson is a recap of best practices for converting recorded macro code into efficient, streamlined code.

To work along, use Lesson50.xlsm.

Notes from the Lesson

- You rarely need to use `.Select`. While recording a macro you usually select a cell and then do something to it. This results in two lines of macro code:

  ```
  Range("E14").Select
  Selection.FormulaR1C1 = "=SUM(R2C:R[-1]C)"
  ```

  Using your mouse, highlight everything from the first `Select` through the `Selection.` on the next line (see Figure 9). Press the Delete key to shorten this line to one line:

  ```
  Range("E14").FormulaR1C1 = "=SUM(R2C:R[-1]C)"
  ```

```
    Selection.FormulaR1C1 = "'Total"
    Range("E14").Select
    Selection.FormulaR1C1 = "=SUM(R[-
```

Figure 9 Delete the selected characters to make your macro more efficient.

- Although the macro recorder will hard-code the cells that you act upon, you should replace those cell addresses with variables. Instead of using `Range("E14")`, use `Cells(TotalRow, 5)`.
- If you copy and paste, the macro recorder will create four lines of code. It will select the original range, do the copy, select the destination range, and paste. Here is some recorded code:

  ```
  Range("E14").Select
  Selection.Copy
  Range("F14:G14").Select
  ActiveSheet.Paste
  ```

  These four lines of code can be simplified to one line of code; a `.Copy` with the `Destination` specified as an argument:

  ```
  Range("E14").Copy Destination:=Range("F14:G14")
  ```

You can combine this concept using a variable instead of hard-coding the addresses. Here is the complete code snippet for adding the totals:

```
TotalRow = Cells(Rows.Count, 1).End(xlUp).Row + 1
Cells(TotalRow, 5).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
Cells(TotalRow, 5).Copy Destination:=Cells(TotalRow, 6).Resize(1, 2)
```

In fact, because you are using R1C1 formulas, you could simply enter this formula in all three cells at once:

```
TotalRow = Cells(Rows.Count, 1).End(xlUp).Row + 1
Cells(TotalRow, 5).Resize(1, 3).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
```

- If you change several properties of the same object, you can refer to the object once in a `With` statement. This saves processing time because VBA has to set up the reference to the object only once. For example, consider this code with four lines adjusting the font of the same range:

```
Range("A1:F1").Font.Bold = True
Range("A1:F1").Font.Size = 12
Range("A1:F1").Font.ColorIndex = 4
Range("A1:F1").Font.Name = "Arial"
```

You can specify `Range().Font` once in an opening `With` statement. Until the macro encounters an `End With`, any line that starts with a dot is referring to `Range().Font`:

```
With Range("A1:F1").Font
  .Bold = True
  .Size = 12
  .ColorIndex = 4
  .Name = "Arial"
End With
```

Code from the Lesson

```
Sub ImportInvoiceFixed()
 '
 ' ImportInvoice Macro
 ' This macro will import invoice.txt and add totals.
 '
 ' Keyboard Shortcut: Ctrl+i
 '
        Workbooks.OpenText Filename:="C:\invoice.txt", _
            Origin:=437, StartRow:=1, DataType:=xlDelimited, _
            TextQualifier:=xlDoubleQuote, _
            ConsecutiveDelimiter:=False, Tab:=True, _
            Semicolon:=False, Comma:=True, Space:=False, _
            Other:=False, FieldInfo:=Array(Array(1, 3), _
            Array(2, 1), Array(3, 1), Array(4, 1), Array(5, 1), _
            Array(6, 1), Array(7, 1)), TrailingMinusNumbers:=True
```

```
    TotalRow = Cells(Rows.Count, 1).End(xlUp).Row + 1
    Cells(TotalRow, 1) = "Total"
    Cells(TotalRow, 5).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
    Cells(TotalRow, 5).Copy _
        Destination:=Cells(TotalRow, 6).Resize(1, 2)
    With Rows("1:1").Font
        .Bold = True
        .Size = 12
    End With
    Cells(TotalRow, 1).EntireRow.Font.Bold = True
    Cells.Columns.AutoFit
End Sub

Sub ImportInvoiceRecorded()
 '
 ' ImportInvoice Macro
 ' This macro will import invoice.txt and add totals.
 '
 ' Keyboard Shortcut: Ctrl+i
 '
        Workbooks.OpenText Filename:="C:\invoice.txt", _
            Origin:=437, StartRow:=1, DataType:=xlDelimited, _
            TextQualifier:=xlDoubleQuote, _
            ConsecutiveDelimiter:=False, Tab:=True, _
            Semicolon:=False, Comma:=True, Space:=False, _
            Other:=False, FieldInfo:=Array(Array(1, 3), _
            Array(2, 1), Array(3, 1), Array(4, 1), Array(5, 1), _
            Array(6, 1), Array(7, 1)), TrailingMinusNumbers:=True

    Selection.End(xlDown).Select
    Range("A14").Select
    Selection.FormulaR1C1 = "Total"
    Range("E14").Select
    Selection.FormulaR1C1 = "=SUM(R2C:R[-1]C)"
    Selection.Copy
    Range("F14:G14").Select
    ActiveSheet.Paste
    Application.CutCopyMode = False
    Rows("1:1").Select
    Selection.Font.Bold = True
    Selection.Font.Size = 12
    Rows("14:14").Select
    Selection.Font.Bold = True
    Cells.Select
    Selection.Columns.AutoFit
End Sub
```

Next Steps

The macro recorder can handle 80% of your coding needs. My goal in this product is to teach you to know when you must add variables, loops, and `If` logic to solve all your VBA coding opportunities.

If you like this book, check out the book that I co-wrote with Tracy Syrstad, *VBA and Macros for Microsoft Office Excel 2007*. It's published by Que Publishing and is available at booksellers everywhere.

If you run into coding problems, I invite you to sign up to join the community at MrExcel.com. Look for the Message Board link in the left navigation bar of the home page. The community there is filled with people who love Excel VBA. Post your coding problems, and fellow members of the community will volunteer to help you with the code. The message board is free.

livelessons

video instruction from technology experts

Trying to keep your skills up to date with the latest technology and advance your career?

Tired of expensive off-site training programs?

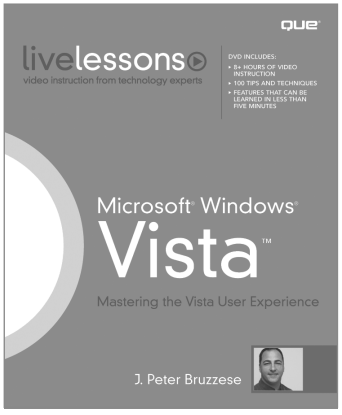
Wish you could learn from the best technologists in the industry?

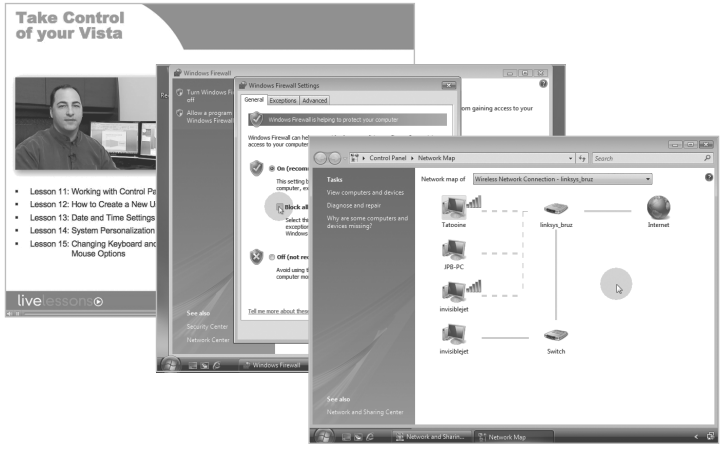
LiveLessons: self-paced, personal video instruction from the world's leading technology experts

- INSTRUCTORS YOU TRUST
- CUTTING EDGE TOPICS
- CUSTOMIZED, SELF-PACED LEARNING
- LEARN BY DOING

PACKAGE INCLUDES:

- 1 DVD featuring 3-4 hours of instructor-led classroom sessions divided into 15-20 minute step-by-step hands-on labs
- Sample code and printed study guide





The power of the world's leading experts at your fingertips!

To learn more about **LiveLessons:** visit **www.mylivelessons.com**

Addison Wesley · PRENTICE HALL · SAMS · que®