



VBA AND MACROS

Microsoft® Excel® 2013

AUTOMATE REPORTS
BUILD FUNCTIONS
VISUALIZE DATA
WRITE FAST, RELIABLE SCRIPTS

que®

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Contents at a Glance



Excel® 2013 VBA and Macros

Bill Jelen

Tracy Syrstad

Introduction	1
1 Unleash the Power of Excel with VBA	7
2 This Sounds Like BASIC, So Why Doesn't It Look Familiar?	33
3 Referring to Ranges	65
4 Looping and Flow Control	79
5 R1C1-Style Formulas	99
6 Create and Manipulate Names in VBA	111
7 Event Programming	123
8 Arrays	149
9 Creating Classes, Records, and Collections	159
10 Userforms: An Introduction	175
11 Data Mining with Advanced Filter	197
12 Using VBA to Create Pivot Tables	231
13 Excel Power	271
14 Sample User-Defined Functions	305
15 Creating Charts	331
16 Data Visualizations and Conditional Formatting	377
17 Dashboarding with Sparklines in Excel 2013	399
18 Reading from and Writing to the Web	419
19 Text File Processing	439
20 Automating Word	451
21 Using Access as a Back End to Enhance Multiuser Access to Data	469
22 Advanced Userform Techniques	485
23 Windows API	509
24 Handling Errors	519
25 Customizing the Ribbon to Run Macros	533
26 Creating Add-Ins	555
27 An Introduction to Creating Apps for Office	563
28 What Is New in Excel 2013 and What Has Changed	583
Index	591

que®

800 East 96th Street,
Indianapolis, Indiana 46240 USA

Excel® 2013 VBA and Macros

Copyright © 2013 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-7897-4861-4

ISBN-10: 0-7897-4861-4

Library of Congress Cataloging-in-Publication Data is on file.

Printed in the United States of America

First Printing: February 2013

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Que Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Bulk Sales

Que Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales

1-800-382-3419

corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales

international@pearsoned.com

Editor-in-Chief

Greg Wiegand

Executive Editor

Loretta Yates

Development Editor

Charlotte Kughen

Managing Editor

Sandra Schroeder

Project Editor

Mandie Frank

Copy Editor

Cheri Clark

Indexer

Tim Wright

Proofreader

Paula Lowell

Technical Editor

Bob Umlas

Editorial Assistant

Cindy Teeters

Designer

Anne Jones

Compositor

Jake McFarland

Table of Contents

Introduction.....	1
Getting Results with VBA.....	1
What Is in This Book?.....	1
Reduce the Learning Curve.....	1
Excel VBA Power.....	2
Techie Stuff Needed to Produce Applications.....	2
Does This Book Teach Excel?.....	2
The Future of VBA and Windows Versions of Excel.....	4
Versions of Excel.....	4
Special Elements and Typographical Conventions.....	4
Code Files.....	5
Next Steps.....	5
1 Unleash the Power of Excel with VBA	7
The Power of Excel.....	7
Barriers to Entry.....	7
The Macro Recorder Doesn't Work!.....	7
Visual Basic Is Not Like BASIC.....	8
Good News: Climbing the Learning Curve Is Easy.....	8
Great News: Excel with VBA Is Worth the Effort.....	8
Knowing Your Tools: The Developer Tab.....	9
Understanding Which File Types Allow Macros.....	10
Macro Security.....	11
Adding a Trusted Location.....	12
Using Macro Settings to Enable Macros in Workbooks Outside of Trusted Locations.....	13
Using Disable All Macros with Notification.....	13
Overview of Recording, Storing, and Running a Macro.....	14
Filling Out the Record Macro Dialog.....	14
Running a Macro.....	16
Creating a Macro Button on the Ribbon.....	16
Creating a Macro Button on the Quick Access Toolbar.....	17
Assigning a Macro to a Form Control, Text Box, or Shape.....	18
Understanding the VB Editor.....	19
VB Editor Settings.....	20
The Project Explorer.....	20
The Properties Window.....	21
Understanding Shortcomings of the Macro Recorder.....	21
Examining Code in the Programming Window.....	23
Running the Macro on Another Day Produces Undesired Results.....	25
Possible Solution: Use Relative References When Recording.....	26
Never Use the AutoSum or Quick Analysis While Recording a Macro.....	30
Three Tips When Using the Macro Recorder.....	31
Next Steps.....	31

2 This Sounds Like BASIC, So Why Doesn't It Look Familiar?	33
I Can't Understand This Code	33
Understanding the Parts of VBA "Speech"	34
VBA Is Not Really Hard	37
VBA Help Files: Using F1 to Find Anything.....	37
Using Help Topics	38
Examining Recorded Macro Code: Using the VB Editor and Help	39
Optional Parameters.....	40
Defined Constants	41
Properties Can Return Objects	45
Using Debugging Tools to Figure Out Recorded Code	46
Stepping Through Code	46
More Debugging Options: Breakpoints.....	49
Backing Up or Moving Forward in Code.....	49
Not Stepping Through Each Line of Code.....	50
Querying Anything While Stepping Through Code	50
Using a Watch to Set a Breakpoint	53
Using a Watch on an Object.....	54
Object Browser: The Ultimate Reference	55
Seven Tips for Cleaning Up Recorded Code	56
Tip 1: Don't Select Anything	56
Tip 2: Cells(2,5) Is More Convenient Than Range("E2")	57
Tip 3: Use More Reliable Ways to Find the Last Row	58
Tip 4: Use Variables to Avoid Hard-Coding Rows and Formulas.....	59
Tip 5: R1C1 Formulas That Make Your Life Easier	59
Tip 6: Learn to Copy and Paste in a Single Statement.....	59
Tip 7: Use With...End With to Perform Multiple Actions	60
Next Steps.....	63
3 Referring to Ranges	65
The Range Object.....	65
Syntax to Specify a Range.....	66
Named Ranges.....	66
Shortcut for Referencing Ranges	66
Referencing Ranges in Other Sheets.....	67
Referencing a Range Relative to Another Range	67
Use the Cells Property to Select a Range.....	68
Use the Offset Property to Refer to a Range.....	69
Use the Resize Property to Change the Size of a Range	71
Use the Columns and Rows Properties to Specify a Range.....	72
Use the Union Method to Join Multiple Ranges.....	72
Use the Intersect Method to Create a New Range from Overlapping Ranges.....	72

Use the ISEMPY Function to Check Whether a Cell Is Empty	73
Use the CurrentRegion Property to Select a Data Range	73
Use the Areas Collection to Return a Noncontiguous Range	76
Referencing Tables	77
Next Steps.....	77
4 Looping and Flow Control.....	79
For...Next Loops.....	79
Using Variables in the For Statement	82
Variations on the For...Next Loop	82
Exiting a Loop Early After a Condition Is Met.....	83
Nesting One Loop Inside Another Loop	84
Do Loops.....	85
Using the While or Until Clause in Do Loops.....	87
While...Wend Loops	88
The VBA Loop: For Each	89
Object Variables.....	89
Flow Control: Using If...Then...Else and Select Case	92
Basic Flow Control: If...Then...Else	92
Conditions	92
If...Then...End If.....	93
Either/Or Decisions: If...Then...Else..End If	93
Using If...Elseif...End If for Multiple Conditions.....	93
Using Select Case...End Select for Multiple Conditions	94
Complex Expressions in Case Statements	95
Nesting If Statements.....	95
Next Steps.....	97
5 R1C1-Style Formulas	99
Referring to Cells: A1 Versus R1C1 References	99
Toggling to R1C1-Style References	100
The Miracle of Excel Formulas.....	101
Enter a Formula Once and Copy 1,000 Times.....	101
The Secret: It's Not That Amazing.....	102
Explanation of R1C1 Reference Style	103
Using R1C1 with Relative References.....	104
Using R1C1 with Absolute References	104
Using R1C1 with Mixed References.....	105
Referring to Entire Columns or Rows with R1C1 Style	105
Replacing Many A1 Formulas with a Single R1C1 Formula.....	106
Remembering Column Numbers Associated with Column Letters.....	107
Array Formulas Require R1C1 Formulas	108
Next Steps.....	109

6 Create and Manipulate Names in VBA	111
Excel Names.....	111
Global Versus Local Names	111
Adding Names	112
Deleting Names	113
Adding Comments.....	114
Types of Names.....	114
Formulas.....	114
Strings	115
Numbers.....	116
Tables.....	117
Using Arrays in Names.....	117
Reserved Names.....	118
Hiding Names	119
Checking for the Existence of a Name	119
Next Steps.....	121
7 Event Programming.....	123
Levels of Events	123
Using Events	124
Event Parameters	124
Enabling Events	125
Workbook Events.....	125
Workbook Level Sheet and Chart Events	129
Worksheet Events.....	132
Chart Sheet Events.....	137
Embedded Charts	137
Application-Level Events	140
Next Steps.....	148
8 Arrays	149
Declare an Array	149
Declare a Multidimensional Array.....	150
Fill an Array.....	151
Retrieve Data from an Array	152
Use Arrays to Speed Up Code.....	153
Use Dynamic Arrays	155
Passing an Array	156
Next Steps.....	157

9 Creating Classes, Records, and Collections	159
Inserting a Class Module	159
Trapping Application and Embedded Chart Events.....	159
Application Events.....	160
Embedded Chart Events.....	161
Creating a Custom Object	163
Using a Custom Object.....	163
Using Property Let and Property Get to Control How Users Utilize Custom Objects	165
Using Collections to Hold Multiple Records.....	167
Creating a Collection in a Standard Module.....	167
Creating a Collection in a Class Module	168
Using User-Defined Types to Create Custom Properties.....	172
Next Steps.....	174
10 Userforms: An Introduction	175
User Interaction Methods	175
Input Boxes.....	175
Message Boxes	176
Creating a Userform.....	176
Calling and Hiding a Userform	177
Programming the Userform.....	178
Userform Events	178
Programming Controls	180
Using Basic Form Controls.....	181
Using Labels, Text Boxes, and Command Buttons	181
Deciding Whether to Use List Boxes or Combo Boxes in Forms	183
Adding Option Buttons to a Userform	186
Adding Graphics to a Userform.....	187
Using a Spin Button on a Userform.....	188
Using the MultiPage Control to Combine Forms.....	190
Verifying Field Entry	192
Illegal Window Closing	192
Getting a Filename	193
Next Steps.....	195
11 Data Mining with Advanced Filter	197
Replacing a Loop with AutoFilter.....	197
Using New AutoFilter Techniques.....	200
Selecting Visible Cells Only	203
Advanced Filter Is Easier in VBA Than in Excel	204
Using the Excel Interface to Build an Advanced Filter.....	205

Using Advanced Filter to Extract a Unique List of Values	206
Extracting a Unique List of Values with the User Interface	206
Extracting a Unique List of Values with VBA Code	207
Getting Unique Combinations of Two or More Fields	211
Using Advanced Filter with Criteria Ranges	212
Joining Multiple Criteria with a Logical OR	213
Joining Two Criteria with a Logical AND	214
Other Slightly Complex Criteria Ranges	214
The Most Complex Criteria: Replacing the List of Values with a Condition Created as the Result of a Formula	214
Using Filter in Place in Advanced Filter	221
Catching No Records When Using Filter in Place	222
Showing All Records After Filter in Place	222
The Real Workhorse: xlFilterCopy with All Records Rather Than Unique Records Only	222
Copying All Columns	223
Copying a Subset of Columns and Reordering	224
Excel in Practice: Turning Off a Few Drop-Downs in the AutoFilter	229
Next Steps	230
12 Using VBA to Create Pivot Tables	231
Introducing Pivot Tables	231
Understanding Versions	231
Building a Pivot Table in Excel VBA	232
Defining the Pivot Cache	232
Creating and Configuring the Pivot Table	233
Adding Fields to the Data Area	234
Learning Why You Cannot Move or Change Part of a Pivot Report	237
Determining the Size of a Finished Pivot Table to Convert the Pivot Table to Values	238
Using Advanced Pivot Table Features	240
Using Multiple Value Fields	240
Grouping Daily Dates to Months, Quarters, or Years	241
Changing the Calculation to Show Percentages	243
Eliminating Blank Cells in the Values Area	246
Controlling the Sort Order with AutoSort	246
Replicating the Report for Every Product	246
Filtering a Dataset	249
Manually Filtering Two or More Items in a Pivot Field	249
Using the Conceptual Filters	250
Using the Search Filter	254
Setting Up Slicers to Filter a Pivot Table	257
Setting Up a Timeline to Filter an Excel 2013 Pivot Table	259
Using the Data Model in Excel 2013	262
Adding Both Tables to the Data Model	262
Creating a Relationship Between the Two Tables	263
Defining the PivotCache and Building the Pivot Table	263

Adding Model Fields to the Pivot Table	264
Adding Numeric Fields to the Values Area	264
Putting It All Together	265
Using Other Pivot Table Features	267
Calculated Data Fields	267
Calculated Items	268
Using ShowDetail to Filter a Recordset	268
Changing the Layout from the Design Tab	268
Settings for the Report Layout	269
Suppressing Subtotals for Multiple Row Fields	269
Next Steps	270
13 Excel Power	271
File Operations	271
List Files in a Directory	271
Import CSV	273
Read Entire TXT to Memory and Parse	274
Combining and Separating Workbooks	275
Separate Worksheets into Workbooks	275
Combine Workbooks	276
Filter and Copy Data to Separate Worksheets	277
Export Data to Word	278
Working with Cell Comments	279
List Comments	279
Resize Comments	281
Place a Chart in a Comment	282
Utilities to Wow Your Clients	283
Using Conditional Formatting to Highlight Selected Cell	283
Highlight Selected Cell Without Using Conditional Formatting	285
Custom Transpose Data	286
Select/Deselect Noncontiguous Cells	288
Techniques for VBA Pros	290
Excel State Class Module	290
Pivot Table Drill-Down	292
Custom Sort Order	293
Cell Progress Indicator	294
Protected Password Box	295
Change Case	297
Selecting with SpecialCells	298
ActiveX Right-Click Menu	299
Cool Applications	300
Historical Stock/Fund Quotes	301
Using VBA Extensibility to Add Code to New Workbooks	302
Next Steps	303

14 Sample User-Defined Functions	305
Creating User-Defined Functions	305
Sharing UDFs	307
Useful Custom Excel Functions.....	308
Set the Current Workbook's Name in a Cell	308
Set the Current Workbook's Name and File Path in a Cell.....	308
Check Whether a Workbook Is Open.....	309
Check Whether a Sheet in an Open Workbook Exists.....	309
Count the Number of Workbooks in a Directory	310
Retrieve USERID.....	311
Retrieve Date and Time of Last Save.....	312
Retrieve Permanent Date and Time.....	312
Validate an Email Address	313
Sum Cells Based on Interior Color	315
Count Unique Values	316
Remove Duplicates from a Range	316
Find the First Nonzero-Length Cell in a Range.....	318
Substitute Multiple Characters	318
Retrieve Numbers from Mixed Text	320
Convert Week Number into Date	320
Separate Delimited String	321
Sort and Concatenate	321
Sort Numeric and Alpha Characters	323
Search for a String Within Text.....	324
Reverse the Contents of a Cell	325
Multiple Max	326
Return Hyperlink Address	326
Return the Column Letter of a Cell Address	327
Static Random	327
Using Select Case on a Worksheet	328
Next Steps.....	329
15 Creating Charts	331
Charting in Excel 2013	331
Considering Backward Compatibility.....	332
Referencing the Chart Container	332
Understanding the Global Settings.....	333
Specifying a Built-in Chart Type	333
Specifying Location and Size of the Chart.....	336
Referring to a Specific Chart	337
Creating a Chart in Various Excel Versions	338
Using .AddChart2 Method in Excel 2013.....	338
Creating Charts in Excel 2007–2013	340
Creating Charts in Excel 2003–2013	341

Customizing a Chart.....	342
Specifying a Chart Title.....	342
Quickly Formatting a Chart Using New Excel 2013 Features.....	343
Using SetElement to Emulate Changes from the Plus Icon.....	350
Using the Format Method to Micromanage Formatting Options.....	355
Creating a Combo Chart.....	359
Creating Advanced Charts.....	363
Creating True Open-High-Low-Close Stock Charts.....	364
Creating Bins for a Frequency Chart.....	365
Creating a Stacked Area Chart.....	368
Exporting a Chart as a Graphic.....	372
Creating Pivot Charts.....	373
Next Steps.....	375
16 Data Visualizations and Conditional Formatting.....	377
Introduction to Data Visualizations.....	377
VBA Methods and Properties for Data Visualizations.....	378
Adding Data Bars to a Range.....	380
Adding Color Scales to a Range.....	384
Adding Icon Sets to a Range.....	385
Specifying an Icon Set.....	386
Specifying Ranges for Each Icon.....	388
Using Visualization Tricks.....	388
Creating an Icon Set for a Subset of a Range.....	388
Using Two Colors of Data Bars in a Range.....	390
Using Other Conditional Formatting Methods.....	392
Formatting Cells That Are Above or Below Average.....	392
Formatting Cells in the Top 10 or Bottom 5.....	393
Formatting Unique or Duplicate Cells.....	393
Formatting Cells Based on Their Value.....	395
Formatting Cells That Contain Text.....	395
Formatting Cells That Contain Dates.....	396
Formatting Cells That Contain Blanks or Errors.....	396
Using a Formula to Determine Which Cells to Format.....	396
Using the New NumberFormat Property.....	398
Next Steps.....	398
17 Dashboarding with Sparklines in Excel 2013.....	399
Creating Sparklines.....	399
Scaling Sparklines.....	401
Formatting Sparklines.....	405
Using Theme Colors.....	405
Using RGB Colors.....	408

Formatting Sparkline Elements	410
Formatting Win/Loss Charts	412
Creating a Dashboard	413
Observations About Sparklines	414
Creating Hundreds of Individual Sparklines in a Dashboard	414
Next Steps.....	418
18 Reading from and Writing to the Web.....	419
Getting Data from the Web	419
Manually Creating a Web Query and Refreshing with VBA.....	420
Using VBA to Update an Existing Web Query.....	423
Building Many Web Queries with VBA.....	424
Using Application.OnTime to Periodically Analyze Data	427
Scheduled Procedures Require Ready Mode.....	428
Specifying a Window of Time for an Update	428
Canceling a Previously Scheduled Macro.....	429
Closing Excel Cancels All Pending Scheduled Macros.....	429
Scheduling a Macro to Run x Minutes in the Future.....	429
Scheduling a Verbal Reminder	430
Scheduling a Macro to Run Every Two Minutes	431
Publishing Data to a Web Page	432
Using VBA to Create Custom Web Pages	434
Using Excel as a Content Management System	434
Bonus: FTP from Excel.....	437
Next Steps.....	438
19 Text File Processing.....	439
Importing from Text Files	439
Importing Text Files with Fewer Than 1,048,576 Rows.....	439
Reading Text Files One Row at a Time	445
Writing Text Files.....	449
Next Steps.....	449
20 Automating Word	451
Using Early Binding to Reference the Word Object	451
Using Late Binding to Reference the Word Object.....	453
Using the New Keyword to Reference the Word Application	454
Using the CreateObject Function to Create a New Instance of an Object	454
Using the GetObject Function to Reference an Existing Instance of Word	455
Using Constant Values	456
Using the Watch Window to Retrieve the Real Value of a Constant	456
Using the Object Browser to Retrieve the Real Value of a Constant.....	457
Understanding Word's Objects	458
Document Object.....	458
Selection Object.....	460

Range Object	461
Bookmarks.....	464
Controlling Form Fields in Word	465
Next Steps.....	467
21 Using Access as a Back End to Enhance Multiuser Access to Data.....	469
ADO Versus DAO.....	470
The Tools of ADO.....	472
Adding a Record to the Database.....	473
Retrieving Records from the Database.....	475
Updating an Existing Record.....	476
Deleting Records via ADO.....	478
Summarizing Records via ADO.....	479
Other Utilities via ADO	480
Checking for the Existence of Tables.....	480
Checking for the Existence of a Field	481
Adding a Table On the Fly.....	482
Adding a Field On the Fly.....	482
SQL Server Examples.....	483
Next Steps.....	484
22 Advanced Userform Techniques	485
Using the UserForm Toolbar in the Design of Controls on Userforms.....	485
More Userform Controls.....	485
Check Boxes.....	485
Tab Strips.....	487
RefEdit.....	489
Toggle Buttons	491
Using a Scrollbar As a Slider to Select Values.....	491
Controls and Collections.....	493
Modeless Userforms.....	495
Using Hyperlinks in Userforms.....	495
Adding Controls at Runtime	496
Resizing the Userform On the Fly.....	498
Adding a Control On the Fly.....	498
Sizing On the Fly.....	498
Adding Other Controls.....	499
Adding an Image On the Fly.....	499
Putting It All Together.....	500
Adding Help to the Userform	502
Showing Accelerator Keys	502
Adding Control Tip Text.....	503
Creating the Tab Order	503

Coloring the Active Control.....	503
Creating Transparent Forms	506
Next Steps.....	507
23 Windows API	509
What Is the Windows API?.....	509
Understanding an API Declaration	509
Using an API Declaration	510
Making 32-Bit and 64-Bit Compatible API Declarations	511
API Examples	512
Retrieving the Computer Name	512
Checking Whether an Excel File Is Open on a Network	513
Retrieving Display-Resolution Information.....	513
Customizing the About Dialog.....	514
Disabling the X for Closing a Userform	515
Running Timer.....	516
Playing Sounds	517
Next Steps.....	517
24 Handling Errors.....	519
What Happens When an Error Occurs?.....	519
Debug Error Inside Userform Code Is Misleading	520
Basic Error Handling with the On Error GoTo Syntax	522
Generic Error Handlers	524
Handling Errors by Choosing to Ignore Them	524
Suppressing Excel Warnings	526
Encountering Errors on Purpose	526
Train Your Clients.....	526
Errors While Developing Versus Errors Months Later	527
Runtime Error 9: Subscript Out of Range	527
Runtime Error 1004: Method Range of Object Global Failed.....	528
The Ills of Protecting Code	529
More Problems with Passwords.....	530
Errors Caused by Different Versions	530
Next Steps.....	531
25 Customizing the Ribbon to Run Macros	533
Out with the Old, In with the New.....	533
Where to Add Your Code: customui Folder and File.....	534
Creating the Tab and Group.....	535
Adding a Control to Your Ribbon	536
Accessing the File Structure	542
Understanding the RELS File.....	542

Renaming the Excel File and Opening the Workbook	543
Using Images on Buttons	543
Using Microsoft Office Icons on Your Ribbon	544
Adding Custom Icon Images to Your Ribbon	545
Troubleshooting Error Messages.....	548
The Attribute “ <i>Attribute Name</i> ” on the Element “ <i>customui Ribbon</i> ” Is Not Defined in the DTD/Schema	548
Illegal Qualified Name Character	548
Element “ <i>customui Tag Name</i> ” Is Unexpected According to Content Model of Parent Element “ <i>customui Tag Name</i> ”	549
Excel Found a Problem with Some Content	549
Wrong Number of Arguments or Invalid Property Assignment	550
Invalid File Format or File Extension	550
Nothing Happens.....	551
Other Ways to Run a Macro	551
Using a Keyboard Shortcut to Run a Macro.....	551
Attaching a Macro to a Command Button	552
Attaching a Macro to a Shape.....	552
Attaching a Macro to an ActiveX Control	553
Running a Macro from a Hyperlink.....	554
Next Steps.....	554
26 Creating Add-Ins	555
Characteristics of Standard Add-Ins	555
Converting an Excel Workbook to an Add-In	556
Using Save As to Convert a File to an Add-In	557
Using the VB Editor to Convert a File to an Add-In	558
Having Your Client Install the Add-In	558
Closing Add-Ins.....	560
Removing Add-Ins.....	560
Using a Hidden Workbook as an Alternative to an Add-In.....	561
Next Steps.....	562
27 An Introduction to Creating Apps for Office	563
Creating Your First App—Hello World	563
Adding Interactivity to Your App	568
A Basic Introduction to HTML.....	570
Tags.....	570
Buttons.....	570
CSS.....	571
Using XML to Define Your App.....	571
Using JavaScript to Add Interactivity to Your App.....	572
The Structure of a Function	572
Variables.....	573
Strings	574

Arrays	574
JS for Loops	575
How to Do an if Statement in JS	576
How to Do a Select..Case Statement in JS	576
How to Do a For each..next Statement in JS	577
Mathematical, Logical, and Assignment Operators	578
Math Functions in JS	579
Writing to the Content or Task Pane.....	581
JavaScript Changes for Working in the Office App	581
Napa Office 365 Development Tools	582
Next Steps.....	582
28 What Is New in Excel 2013 and What Has Changed	583
If It Has Changed in the Front End, It Has Changed in VBA	583
The Ribbon	583
Single Document Interface (SDI).....	583
Quick Analysis Tool	585
Charts	585
PivotTables	585
Slicers	586
SmartArt.....	586
Learning the New Objects and Methods	587
Compatibility Mode	587
Version	587
Excel8CompatibilityMode	588
Next Steps.....	588
Index	591

About the Author

Bill Jelen, Excel MVP and the host of MrExcel.com, has been using spreadsheets since 1985, and he launched the MrExcel.com website in 1998. Bill was a regular guest on *Call for Help* with Leo Laporte and has produced more than 1,500 episodes of his daily video podcast, *Learn Excel from MrExcel*. He is the author of 39 books about Microsoft Excel and writes the monthly Excel column for *Strategic Finance* magazine. His Excel tips appear regularly in CFO Excel Pro Newsletter and *CFO Magazine*. Before founding MrExcel.com, Bill Jelen spent 12 years in the trenches—working as a financial analyst for finance, marketing, accounting, and operations departments of a \$500 million public company. He lives near Akron, Ohio, with his wife, Mary Ellen.

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

Dedication

Bill Jelen

For Mary Ellen Jelen

Tracy Syrstad

To Nate P. Oliver, who shared his love of Excel with the world.

Acknowledgments

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

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

Along the way, I've learned a lot about VBA programming from the awesome community at the MrExcel.com message board. VoG, Richard Schollar, and Jon von der Heyden all stand out as having contributed posts that led to ideas in this book. Thanks to Pam Gensel for Excel macro lesson #1. Mala Singh taught me about creating charts in VBA, and Oliver Holloway brought me up to speed with accessing SQL Server. Scott Ruble and Robin Wakefield at Microsoft helped with the charting chapter. And I give a tip of the cap to JWalk for that HWND trick.

At MrExcel.com, thanks to Barb Jelen, Wei Jiang, Tracy Syrstad, Tyler Nash, and Scott Pierson.

My family was incredibly supportive during this time. Thanks to Zeke Jelen, Dom Grossi, and Mary Ellen Jelen.

—Bill

Juan Pablo Gonzalez Ruiz is a great programmer, and I really appreciate his time and patience showing me new ways to write better programs.

Thank you to all the moderators at the MrExcel forum who keep the board organized, despite the best efforts of the spammers.

Programming is a constant learning experience, and I really appreciate the clients who have encouraged me to program outside my comfort zone so that my skills and knowledge have expanded.

And last, but not least, thanks to Bill Jelen. His site, MrExcel.com, is a place where thousands come for help. It's also a place where I, and others like me, have an opportunity to learn from and assist others.

—Tracy

We Want to Hear from You!

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

We welcome your comments. You can email or write to let us 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 we cannot help you with technical problems related to the topic of this book.

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

Email: feedback@quepublishing.com

Mail: Que Publishing
 ATTN: Reader Feedback
 800 East 96th Street
 Indianapolis, IN 46240 USA

Reader Services

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

Introduction

Getting Results with VBA

As corporate IT departments have found themselves with long backlogs of requests, Excel users have discovered that they can produce the reports needed to run their business themselves using the macro language *Visual Basic for Applications* (VBA). VBA enables you to achieve tremendous efficiencies in your day-to-day use of Excel. Without your waiting for resources from IT, VBA helps you figure out how to import data and produce reports in Excel.

What Is in This Book?

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

Reduce the Learning Curve

This Introduction provides a case study of the power of macros. Chapter 1 introduces the tools and confirms what you probably already know: The macro recorder does not work reliably. Chapter 2 helps you understand the crazy syntax of VBA. Chapter 3 breaks the code on how to work efficiently with ranges and cells.

Chapter 4 covers the power of looping using VBA. The case study in this chapter creates a program to produce a department report, and then wraps that report routine in a loop to produce 46 reports.

Chapter 5 covers R1C1-style formulas. Chapter 6 covers names. Chapter 7 includes some great tricks that use event programming. Chapters 8 and 9 cover arrays, classes, records, and collections. Chapter 10 introduces custom dialog boxes that you can use to collect information from the human using Excel.

IN THIS INTRODUCTION

Getting Results with VBA	1
What Is in This Book?.....	1
The Future of VBA and Windows Versions of Excel.....	4
Special Elements and Typographical Conventions.....	4
Code Files	5
Next Steps	5

Excel VBA Power

Chapters 11 and 12 provide an in-depth look at Filter, Advanced Filter, and pivot tables. Any report automation tool will rely heavily on these concepts. Chapters 13 and 14 include 25 code samples designed to exhibit the power of Excel VBA and custom functions.

Chapters 15 through 20 handle charting, data visualizations, web queries, sparklines, and automating another Office program such as Word.

Techie Stuff Needed to Produce Applications

Chapter 21 handles reading and writing to Access databases and SQL Server. The techniques for using Access databases enable you to build an application with the multiuser features of Access while keeping the friendly front end of Excel.

Chapter 22 discusses advanced userform topics. Chapter 23 teaches some tricky ways to achieve tasks using the Windows application programming interface. Chapters 24 through 26 deal with error handling, custom menus, and add-ins. Chapter 27 is a brief introduction to building your own Java application within Excel. Chapter 28 summarizes the changes in Excel 2013.

Does This Book Teach Excel?

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

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

The point? You and I both know a lot about Excel. However, I assume that in any given chapter, maybe 58 percent of the people have not used pivot tables before and maybe even fewer have used the “Top 10 Filter” feature of pivot tables. With this in mind, before I show how to automate something in VBA, I briefly cover how to do the same task in the Excel interface. This book does not teach you how to do pivot tables, but it does alert you when you might need to explore a topic and learn more about it elsewhere.

CASE STUDY: MONTHLY ACCOUNTING REPORTS

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

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

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

Becoming the Hero

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

More Cheers

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

Cheers Turn to Dread

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

VBA to the Rescue

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

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

The Future of VBA and Windows Versions of Excel

Several years ago, there were many rumblings that Microsoft might stop supporting VBA. There is now plenty of evidence to indicate that VBA will be around in Windows versions of Excel through 2030. When VBA was removed from the Mac version of Excel 2008, a huge outcry from customers led to its being included in the next Mac version of Excel.

Microsoft has hinted that in Excel 16, which is the next version of Excel, it will stop providing support for XLM macros. These macros were replaced by VBA in 1993, and 20 years later, they are still supported. Some would say that Microsoft introduced a new programming language for Excel with the JavaScript applications that are discussed in Chapter 28. Assuming that Microsoft continues to support VBA for 22 years after Excel 2013, you should be good through the mid-2030s.

Versions of Excel

This fourth edition of *VBA and Macros* is designed to work with Excel 2013. The previous editions of this book covered code for Excel 97 through Excel 2010. In 80 percent of the chapters, the code for Excel 2013 is identical to the code in previous versions. However, there are exceptions. For example, Microsoft offers new pivot table models and timelines that will add some new methods to the pivot table chapter.

Differences for Mac Users

Although Excel for Windows and Excel for the Mac are similar in their user interface, there are a number of differences when you compare the VBA environment. Certainly, nothing in Chapter 23 that uses the Windows API will work on the Mac. The overall concepts discussed in the book apply to the Mac, but differences exist. You can find a general list of differences as they apply to the Mac at <http://www.mrexcel.com/macvba.html>.

Special Elements and Typographical Conventions

The following typographical conventions are used in this book:

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

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

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

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

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

CAUTION

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

Code Files

As a thank-you for buying this book, the authors have put together a set of 50 Excel workbooks that demonstrate the concepts included in this book. This set of files includes all the code from the book, sample data, additional notes from the authors, and 25 bonus macros. To download the code files, visit this book's web page at <http://www.quepublishing.com> or <http://www.mrexcel.com/getcode2013.html>.

Next Steps

Chapter 1, “Unleash the Power of Excel with VBA,” introduces the editing tools of the Visual Basic environment and shows why using the macro recorder is not an effective way to write VBA macro code.

This page intentionally left blank

Referring to Ranges

3

A *range* can be a cell, a row, a column, or a grouping of any of these. The `RANGE` object is probably the most frequently used object in Excel VBA—after all, you are manipulating data on a sheet. Although a range can refer to any grouping of cells on a sheet, it can refer to only one sheet at a time. If you want to refer to ranges on multiple sheets, you must refer to each sheet separately.

This chapter shows you different ways of referring to ranges such as specifying a row or column. You also find out how to manipulate cells based on the active cell and how to create a new range from overlapping ranges.

The Range Object

The following is the Excel object hierarchy:

Application > Workbook > Worksheet > Range

The `Range` object is a property of the `Worksheet` object. This means it requires that a sheet be active or it must reference a worksheet. Both of the following lines mean the same thing if `Worksheets(1)` is the active sheet:

```
Range("A1")  
Worksheets(1).Range("A1")
```

There are several ways to refer to a `Range` object. `Range("A1")` is the most identifiable because that is how the macro recorder refers to it. However, each of the following is equivalent when referring to a range:

```
Range("D5")  
[D5]  
Range("B3").Range("C3")  
Cells(5,4)  
Range("A1").Offset(4,3)  
Range("MyRange") 'assuming that D5 has a Name  
'of MyRange
```

Which format you use depends on your needs. Keep reading—it will all make sense soon!

IN THIS CHAPTER

The Range Object	65
Syntax to Specify a Range	66
Named Ranges.....	66
Shortcut for Referencing Ranges	66
Referencing Ranges in Other Sheets	67
Referencing a Range Relative to Another Range.....	67
Use the <code>Cells</code> Property to Select a Range	68
Use the <code>Offset</code> Property to Refer to a Range	69
Use the <code>Resize</code> Property to Change the Size of a Range	71
Use the <code>Columns</code> and <code>Rows</code> Properties to Specify a Range.....	72
Use the <code>Union</code> Method to Join Multiple Ranges.....	72
Use the <code>Intersect</code> Method to Create a New Range from Overlapping Ranges	72
Use the <code>IsEmpty</code> Function to Check Whether a Cell Is Empty	73
Use the <code>CurrentRegion</code> Property to Select a Data Range	73
Use the <code>Areas</code> Collection to Return a Noncontiguous Range	76
Referencing Tables	77
Next Steps	77

Syntax to Specify a Range

The Range property has two acceptable syntaxes. To specify a rectangular range in the first syntax, specify the complete range reference just as you would in a formula in Excel:

```
Range("A1:B5")
```

In the alternative syntax, specify the upper-left corner and lower-right corner of the desired rectangular range. In this syntax, the equivalent statement might be this:

```
Range("A1", "B5")
```

For either corner, you can substitute a named range, the Cells property, or the ActiveCell property. The following line of code selects the rectangular range from A1 to the active cell:

```
Range("A1", ActiveCell).Select
```

The following statement selects from the active cell to five rows below the active cell and two columns to the right:

```
Range(ActiveCell, ActiveCell.Offset(5, 2)).Select
```

Named Ranges

You probably have already used named ranges on your worksheets and in formulas. You can also use them in VBA.

Use the following code to refer to the range "MyRange" in Sheet1:

```
Worksheets("Sheet1").Range("MyRange")
```

Notice that the name of the range is in quotes—unlike the use of named ranges in formulas on the sheet itself. If you forget to put the name in quotes, Excel thinks you are referring to a variable in the program. One exception is if you use the shortcut syntax discussed in the next section. In this case, quotes are not used.

Shortcut for Referencing Ranges

A shortcut is available when referencing ranges. The shortcut uses square brackets, as shown in Table 3.1.

Table 3.1 Shortcuts for Referencing Ranges

Standard Method	Shortcut
Range("D5")	[D5]
Range("A1:D5")	[A1:D5]
Range("A1:D5, G6:I17")	[A1:D5, G6:I17]
Range("MyRange")	[MyRange]

Referencing Ranges in Other Sheets

Switching between sheets by activating the needed sheet can dramatically slow down your code. To avoid this slowdown, you can refer to a sheet that is not active by first referencing the `Worksheet` object:

```
Worksheets("Sheet1").Range("A1")
```

This line of code references `Sheet1` of the active workbook even if `Sheet2` is the active sheet.

If you need to reference a range in another workbook, include the `Workbook` object, the `Worksheet` object, and then the `Range` object:

```
Workbooks("InvoiceData.xlsx").Worksheets("Sheet1").Range("A1")
```

Be careful if you use the `Range` property as an argument within another `Range` property. You must identify the range fully each time. For example, suppose that `Sheet1` is your active sheet and you need to total data from `Sheet2`:

```
WorksheetFunction.Sum(Worksheets("Sheet2").Range(Range("A1"), Range("A7")))
```

This line does not work. Why not? Because `Range("A1"), Range("A7")` is meant to refer to the sheet at the beginning of the code line. However, Excel does not assume that you want to carry the `Worksheet` object reference over to these other `Range` objects and assumes they refer to `Sheet1`. So what do you do? Well, you could write this:

```
WorksheetFunction.Sum(Worksheets("Sheet2").Range(Worksheets("Sheet2"). _  
    Range("A1"), Worksheets("Sheet2").Range("A7")))
```

But this not only is a long line of code but also is difficult to read! Thankfully, there is a simpler way, using `With...End With`:

```
With Worksheets("Sheet2")  
    WorksheetFunction.Sum(.Range(.Range("A1"), .Range("A7")))  
End With
```

Notice now that there is a `.Range` in your code, but without the preceding object reference. That's because `With Worksheets("Sheet2")` implies that the object of the range is the worksheet. Whenever Excel sees a period without an object reference directly to the left of it, it looks up the code for the closest `With` statement and uses that as the object reference.

Referencing a Range Relative to Another Range

Typically, the `RANGE` object is a property of a worksheet. It is also possible to have `RANGE` be the property of another range. In this case, the `Range` property is relative to the original range, which makes for unintuitive code. Consider this example:

```
Range("B5").Range("C3").Select
```

This code actually selects cell `D7`. Think about cell `C3`, which is located two rows below and two columns to the right of cell `A1`. The preceding line of code starts at cell `B5`. If we assume that `B5` is in the `A1` position, VBA finds the cell that would be in the `C3` position

relative to B5. In other words, VBA finds the cell that is two rows below and two columns to the right of B5, which is D7.

Again, I consider this coding style to be very unintuitive. This line of code mentions two addresses, and the actual cell selected is neither of these addresses! It seems misleading when you are trying to read this code.

You might consider using this syntax to refer to a cell relative to the active cell. For example, the following line of code activates the cell three rows down and four columns to the right of the currently active cell:

```
Selection.Range("E4").Select
```

This syntax is mentioned only because the macro recorder uses it. Recall that when you recorded a macro in Chapter 1, “Unleash the Power of Excel with VBA,” with Relative References on, the following line was recorded:

```
ActiveCell.Offset(0, 4).Range("A2").Select
```

This line found the cell four columns to the right of the active cell, and from there it selected the cell that would correspond to A2. This is not the easiest way to write code, but that is the way the macro recorder does it.

Although a worksheet is usually the object of the Range property, occasionally, such as during recording, a range may be the property of a range.

Use the Cells Property to Select a Range

The Cells property refers to all the cells of the specified range object, which can be a worksheet or a range of cells. For example, this line selects all the cells of the active sheet:

```
Cells.Select
```

Using the Cells property with the Range object might seem redundant:

```
Range("A1:D5").Cells
```

This line refers to the original Range object. However, the Cells property has an Item property that makes the Cells property very useful. The Item property enables you to refer to a specific cell relative to the Range object.

The syntax for using the Item property with the Cells property is as follows:

```
Cells.Item(Row, Column)
```

You must use a numeric value for Row, but you may use the numeric value or string value for Column. Both of the following lines refer to cell C5:

```
Cells.Item(5, "C")  
Cells.Item(5, 3)
```

Because the Item property is the default property of the RANGE object, you can shorten these lines as follows:

```
Cells(5, "C")  
Cells(5, 3)
```

The ability to use numeric values for parameters is particularly useful if you need to loop through rows or columns. The macro recorder usually uses something like `Range("A1").Select` for a single cell and `Range("A1:C5").Select` for a range of cells. If you are learning to code only from the recorder, you might be tempted to write code like this:

```
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
For i = 1 to FinalRow
    Range("A" & i & ":E" & i).Font.Bold = True
Next i
```

This little piece of code, which loops through rows and bolds the cells in Columns A through E, is awkward to read and write. But, how else can you do it?

```
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
For i = 1 to FinalRow
    Cells(i, "A").Resize(, 5).Font.Bold = True
Next i
```

Instead of trying to type the range address, the new code uses the `Cells` and `Resize` properties to find the required cell, based on the active cell. See the “Use the Resize Property to Change the Size of a Range” section later in this chapter for more information on the `Resize` property.

`Cells` properties can be used as parameters in the `Range` property. The following refers to the range A1:E5:

```
Range(Cells(1,1), Cells(5,5))
```

This is particularly useful when you need to specify your variables with a parameter, as in the previous looping example.

Use the Offset Property to Refer to a Range

You have already seen a reference to `Offset` when the macro recorder used it when you recorded a relative reference. `Offset` enables you to manipulate a cell based off the location of the active cell. In this way, you do not need to know the address of a cell.

The syntax for the `Offset` property is as follows:

```
Range.Offset(RowOffset, ColumnOffset)
```

The syntax to affect cell F5 from cell A1 is

```
Range("A1").Offset(RowOffset:=4, ColumnOffset:=5)
```

Or, shorter yet, write this:

```
Range("A1").Offset(4, 5)
```

The count of the rows and columns starts at A1 but does not include A1.

But what if you need to go over only a row or a column, but not both? You don't have to enter both the row and the column parameter. If you need to refer to a cell one column over, use one of these lines:

```
Range("A1").Offset(ColumnOffset:=1)
Range("A1").Offset(, 1)
```


Both lines mean the same, so the choice is yours. Referring to a cell one row up is similar:

```
Range("B2").Offset(RowOffset:=-1)
Range("B2").Offset(-1)
```

Once again, you can choose which one to use. It is a matter of readability of the code.

Suppose you have a list of produce in column A with totals next to them in column B. If you want to find any total equal to zero and place LOW in the cell next to it, do this:

```
Set Rng = Range("B1:B16").Find(What:="0", LookAt:=xlWhole, LookIn:=xlValues)
Rng.Offset(, 1).Value = "LOW"
```

Used in a sub and looping through a table, it would look like this:

```
Sub FindLow()
With Range("B1:B16")
    Set Rng = .Find(What:="0", LookAt:=xlWhole, LookIn:=xlValues)
    If Not Rng Is Nothing Then
        firstAddress = Rng.Address
        Do
            Rng.Offset(, 1).Value = "LOW"
            Set Rng = .FindNext(Rng)
        Loop While Not Rng Is Nothing And Rng.Address <> firstAddress
    End If
End With
End Sub
```

The LOW totals are noted by the program, as shown in Figure 3.1.

Figure 3.1
Find the produce with zero totals.

	A	B	C
1	Apples	45	
2	Oranges	12	
3	Grapefruit	86	
4	Lemons	0	LOW

NOTE

Refer to the section "Object Variables" in Chapter 4 for more information on the Set statement.

Offsetting isn't only for single cells—you can use it with ranges. You can shift the focus of a range over in the same way you can shift the active cell. The following line refers to B2:D4 (see Figure 3.2):

```
Range("A1:C3").Offset(1,1)
```

Figure 3.2
Offsetting a range:
Range("A1:C3").
Offset(1,1).
Select.

	A	B	C	D
1				
2				
3				
4				
5				

Use the Resize Property to Change the Size of a Range

The `Resize` property enables you to change the size of a range based on the location of the active cell. You can create a new range as needed. The syntax for the `Resize` property is

```
Range.Resize(RowSize, ColumnSize)
```

To create a range B3:D13, use the following:

```
Range("B3").Resize(RowSize:=11, ColumnSize:=3)
```

Or here's a simpler way to create this range:

```
Range("B3").Resize(11, 3)
```

But what if you need to resize by only a row or a column—not both? You do not have to enter both the row and the column parameters.

If you need to expand by two columns, use one of the following:

```
Range("B3").Resize(ColumnSize:=2)
```

or

```
Range("B3").Resize(,2)
```

Both lines mean the same thing. The choice is yours. Resizing just the rows is similar. You can use either of the following:

```
Range("B3").Resize(RowSize:=2)
```

or

```
Range("B3").Resize(2)
```

Once again, the choice is yours. It is a matter of readability of the code.

From the list of produce, find the zero totals and color the cells of the total and corresponding produce (see Figure 3.3):

```
Set Rng = Range("B1:B16").Find(What:="0", LookAt:=xlWhole, LookIn:=xlValues)
Rng.Offset(, -1).Resize(, 2).Interior.ColorIndex = 15
```

Notice that the `Offset` property was used first to move the active cell over. When you are resizing, the upper-left-corner cell must remain the same.

Figure 3.3

Resizing a range to extend the selection.

	A	B	
1	Apples	45	
2	Oranges	12	
3	Grapefruit	0	
4	Lemons	0	

Resizing isn't only for single cells—you can use it to resize an existing range. For example, if you have a named range but need it and the column next to it, use this:

```
Range("Produce").Resize(,2)
```

Remember, the number you resize by is the total number of rows/columns you want to include.

Use the Columns and Rows Properties to Specify a Range

The `Columns` and `Rows` properties refer to the columns and rows of a specified `Range` object, which can be a worksheet or a range of cells. They return a `Range` object referencing the rows or columns of the specified object.

You have seen the following line used, but what is it doing?

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

This line of code finds the last row in a sheet in which Column A has a value and places the row number of that `Range` object into `FinalRow`. This can be useful when you need to loop through a sheet row by row—you will know exactly how many rows you need to go through.

NOTE

Some properties of columns and rows require contiguous rows and columns to work properly. For example, if you were to use the following line of code, 9 would be the answer because only the first range would be evaluated:

```
Range("A1:B9, C10:D19").Rows.Count
```

However, if the ranges were grouped separately, the answer would be 19.

```
Range("A1:B9", "C10:D19").Rows.Count
```

Use the Union Method to Join Multiple Ranges

The `Union` method enables you to join two or more noncontiguous ranges. It creates a temporary object of the multiple ranges, which enables you to affect them together:

```
Application.Union(argument1, argument2, etc.)
```

The expression, `Application`, is not required. The following code joins two named ranges on the sheet, inserts the `=RAND()` formula, and bolds them:

```
Set UnionRange = Union(Range("Range1"), Range("Range2"))
With UnionRange
    .Formula = "=RAND()"
    .Font.Bold = True
End With
```

Use the Intersect Method to Create a New Range from Overlapping Ranges

The `Intersect` method returns the cells that overlap between two or more ranges:

```
Application.Intersect(argument1, argument2, etc.)
```

The expression, `Application`, is not required. The following code colors the overlapping cells of the two ranges:

```
Set IntersectRange = Intersect(Range("Range1"), Range("Range2"))
IntersectRange.Interior.ColorIndex = 6
```

Use the ISEMPTY Function to Check Whether a Cell Is Empty

The ISEMPTY function returns a Boolean value that indicates whether a single cell is empty: True if empty, False if not. The cell must truly be empty for the function to return True. Even if it has a space that you cannot see, Excel does not consider the cell to be empty:

```
IsEmpty(Cell)
```

You have several groups of data separated by a blank row. You want to make the separations a little more obvious. The following code goes down the data in Column A. When it finds an empty cell, it colors in the first four cells for that row (see Figure 3.4):

```
LastRow = Cells(Rows.Count, 1).End(xlUp).Row
For i = 1 To LastRow
    If IsEmpty(Cells(i, 1)) Then
        Cells(i, 1).Resize(1, 4).Interior.ColorIndex = 1
    End If
Next i
```

Figure 3.4
Colored rows separating data.

	A	B	C	D
1	Apples	Oranges	Grapefruit	Lemons
2	45	12	86	15
3	83%	19%	6%	58%
4				
5	Tomatoes	Cabbage	Lettuce	Green Peppers
6	58	24	31	0
7	72%	5%	87%	25%
8				
9	Potatoes	Yams	Onions	Garlic
10	10	61	26	29
11	33%	54%	26%	84%

Use the CurrentRegion Property to Select a Data Range

CurrentRegion returns a Range object representing a set of contiguous data. As long as the data is surrounded by one empty row and one empty column, you can select the table with CurrentRegion:

```
RangeObject.CurrentRegion
```

The following line selects A1:D3 because this is the contiguous range of cells around cell A1 (see Figure 3.5):

```
Range("A1").CurrentRegion.Select
```

This is useful if you have a table whose size is in constant flux.

Figure 3.5
Use CurrentRegion to select a range of contiguous data around the active cell.

	A	B	C	D
1	Apples	Oranges	Grapefruit	Lemons
2	79	69	16	92
3	9%	44%	19%	16%
4				

CASE STUDY: USING THE SPECIALCELLS METHOD TO SELECT SPECIFIC CELLS

Even Excel power users might not have encountered the Go To Special dialog box. If you press the F5 key in an Excel worksheet, you get the normal Go To dialog box (see Figure 3.6). In the lower-left corner of this dialog is a button labeled Special. Click that button to get to the superpowerful Go To Special dialog box (see Figure 3.7).

Figure 3.6

Although the Go To dialog doesn't seem useful, click the Special button in the lower-left corner.

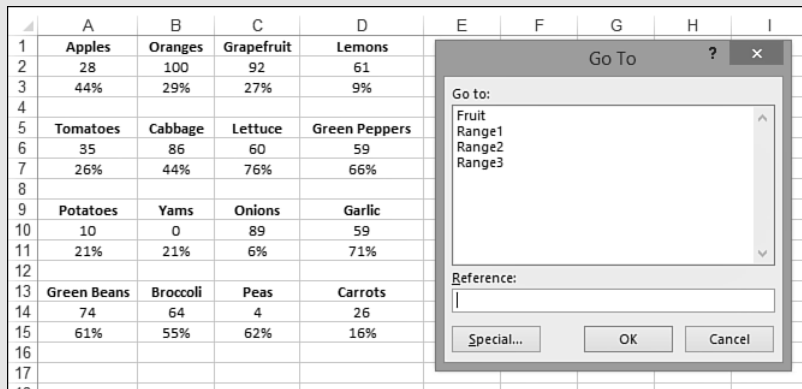
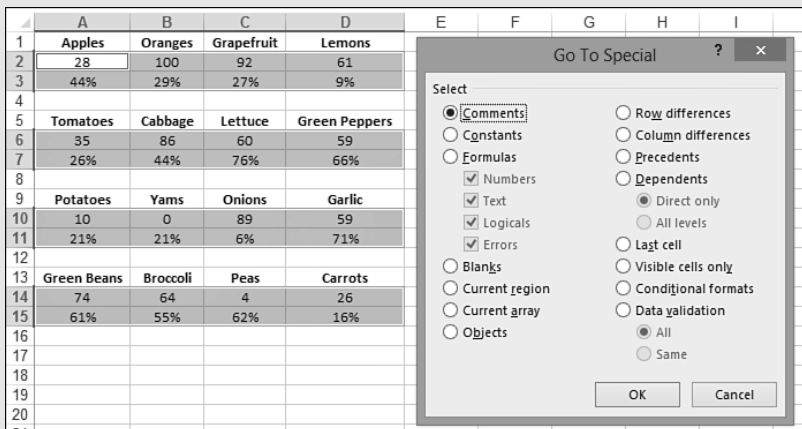


Figure 3.7

The Go To Special dialog has many incredibly useful selection tools, such as selecting only the formulas on a sheet.



In the Excel interface, the Go To Special dialog enables you to select only cells with formulas, only blank cells, or only the visible cells. Selecting only visible cells is excellent for grabbing the visible results of AutoFiltered data.

To simulate the Go To Special dialog in VBA, use the `SpecialCells` method. This enables you to act on cells that meet a certain criteria:

```
RangeObject.SpecialCells(Type, Value)
```

This method has two parameters: `Type` and `Value`. `Type` is one of the `xlCellType` constants:

```
xlCellTypeAllFormatConditions
xlCellTypeAllValidation
xlCellTypeBlanks
xlCellTypeComments
xlCellTypeConstants
xlCellTypeFormulas
xlCellTypeLastCell
xlCellTypeSameFormatConditions
xlCellTypeSameValidation
xlCellTypeVisible
```

`Value` is optional and can be one of the following:

```
xlErrors
xlLogical
xlNumbers
xlTextValues
```

The following code returns all the ranges that have conditional formatting set up. It produces an error if there are no conditional formats and adds a border around each contiguous section it finds:

```
Set rngCond = ActiveSheet.Cells.SpecialCells(xlCellTypeAllFormatConditions)
If Not rngCond Is Nothing Then
    rngCond.BorderAround xlContinuous
End If
```

Have you ever had someone send you a worksheet without all the labels filled in? Some people consider that the data shown in Figure 3.8 looks neat. They enter the `Region` field only once for each region. This might look aesthetically pleasing, but it is impossible to sort.

Figure 3.8

The blank cells in the `Region` column make it difficult to sort data tables such as this.

	A	B	C
1	Region	Product	Sales
2	North	ABC	766,469
3		DEF	776,996
4		XYZ	832,414
5	East	ABC	703,255
6		DEF	891,799
7		XYZ	897,949

Using the `SpecialCells` method to select all the blanks in this range is one way to fill in all the blank region cells quickly with the region found above them:

```
Sub FillIn()
    On Error Resume Next 'Need this because if there aren't any blank
    'cells, the code will error
    Range("A1").CurrentRegion.SpecialCells(xlCellTypeBlanks).FormulaR1C1 _
    = "=R[-1]C"
    Range("A1").CurrentRegion.Value = Range("A1").CurrentRegion.Value
End Sub
```


Referencing Tables

Tables are a special type of range that offers the convenience of referencing named ranges, but they are not created in the same manner. For more information on how to create a named table, see Chapter 6, “Create and Manipulate Names in VBA.”

The table itself is referenced using the standard method of referring to a ranged name. To refer to the data in Table1 in Sheet1, do this:

```
Worksheets(1).Range("Table1")
```

This references the data part of the table but does not include the header or total row. To include the header and total row, do this:

```
Worksheets(1).Range("Table1[#A1:1]")
```

What I really like about this feature is the ease of referencing specific columns of a table. You don't have to know how many columns to move in from a starting position or the letter/number of the column, and you don't have to use a FIND function. Instead, you can use the header name of the column. For example, do this to reference the Qty column of the table:

```
Worksheets(1).Range("Table1[Qty]")
```

Next Steps

Chapter 4, “Looping and Flow Control,” describes a fundamental component of any programming language: loops. If you have taken a programming class, you will be familiar with basic loop structures. VBA supports all the usual loops. In the next chapter, you'll also learn about a special loop, For Each...Next, which is unique to object-oriented programming such as VBA.

This page intentionally left blank

Index

Numerics

**32-bit and 64-bit compatible API
declarations, 511-512**

A

A1 references

case study, 102-103

versus R1C1 references, 99-100

**above-average records, returning with
formula-based conditions, 221**

**absolute references, using with R1C1
references, 104-105**

Access

ADO

records, adding to database, 473-
474

records, deleting, 478-479

records, retrieving, 475-476

records, summarizing, 479-480

records, updating, 476-478

tables, checking for existence of,
480-481

fields, adding to database, 482-483

shared Access database, creating, 471

accessing

Developer tab, 9

VBA help topics, 37-38

ActiveX controls

attaching macros to, 553-554

creating right-click menu, 299-300

Add method, 395-396

AddAboveAverage method, 392**adding**

- code to new workbooks, 302-303
- comments to names, 114
- controls to userforms, 181
- macro button
 - to Quick Access toolbar, 17
 - to Ribbon, 16
- trusted location to hard drive, 12

add-ins

- characteristics of, 555-556
- closing, 560
- hidden workbooks as alternative to, 561-562
- installing, 558-559
- removing, 560

Add-Ins group (Developer tab), 10**AddTop10 method, 393****AddUniqueValues method, 393-395****adjusting macro default settings, 11-14****ADO (ActiveX data objects)**

- versus DAO, 470
- fields, checking for in Access database, 481-482
- records
 - adding to Access database, 473-474
 - deleting in Access database, 478-479
 - retrieving from Access database, 475-476
 - summarizing in Access database, 479-480
 - updating in Access database, 476-478
- tables, checking for in Access database, 480-481
- tools of, 472-473

Advanced Filter command, extracting unique list of values with, 206-207**advanced filters**

- building, 205-206
- reports, creating from, 226-229
- xlFilterCopy parameter, 222-225

allowing macros outside of trusted locations, 13**Anderson, Ken, 437****APIs**

- declaring, 509-511
 - 32-bit and 64-bit compatible declarations, 511-512
- examples of, 512-517
- Windows API, 509

application-level events, 123, 140-148

- trapping, 160-161

applying data visualizations to pivot tables, 270**apps**

- defining with XML, 571-572
- Hello World
 - creating, 563-567
 - interactivity, adding, 568-570
- Napa Office 365 Development Tools, 582
- Office apps, JavaScript incorporation in, 581-582

Areas collection, returning noncontiguous ranges, 76**array formulas**

- entering, 108-109
- names, using with, 117-118

arrays

- declaring, 149-150
- dynamic, 155-156
- filling, 151-152
- for JavaScript, 574-575

- multidimensional, declaring, 150
- passing, 156
- reinitializing, 155
- retrieving data from, 152-153
- speeding up code with, 153-155

assigning

- macros to text boxes, form controls, or shapes, 18-19
- names
 - to formulas, 114-115
 - to numbers, 116
 - to strings, 115-116
 - to tables, 117

assignment operators (JavaScript), 578-579

attaching macros

- to ActiveX controls, 553-554
- to shapes, 552

attributes for Ribbon controls, 538-540

AutoFilter

- dynamic date range, selecting, 202-203
- field drop-downs, turning off, 229-230
- filtering
 - by color, 201
 - by icon, 201
- multiple items, selecting, 200
- replacing loops with, 197-203
- Search box, 200-201
- visible cells, selecting, 203

automation

- constant values, 456-457
- CreateObject function, 454
- early binding, referencing Word object, 451-453
- late binding, referencing Word object, 453-454

- New keyword, referencing Word application, 454

AutoShow feature (VBA), 255-257

AutoSort option (pivot tables), 246

AutoSum button and recording macros, 30-31

B

barriers to learning VBA, 7-9

BASIC, 33-34

- comparing to VBA, 8-15

bins, creating for frequency charts, 365-368

blank cells, removing in pivot table values area, 246

BookOpen() function, 309

breakpoints, 49

- setting, 53-54

building

- advanced filters, 205-206
- cell progress indicator, 294-295
- Data Model pivot table, 262-267
- list of unique combinations of two fields, 211
- web queries with VBA, 424-427

built-in charts, specifying, 333-334

buttons

- help buttons, 170-171
- HTML, 570-571
- macro button
 - adding to Quick Access toolbar, 17
 - adding to Ribbon, 16
- option buttons, adding to userforms, 186-187
- SpinButton control, 188-190
- toggle buttons, 491

C

calculated data fields (pivot tables), 267

calculated items (pivot tables), 268

calling userforms, 177-178

canceling scheduled macros, 429

capturing data periodically, 427-428

case of text, changing, 297-298

case studies

A1 versus R1C1 references, 102-103

combo charts, creating, 361-363

controls, adding to userforms, 181

custom Excel 2003 toolbar,
converting to Excel 2013, 547

custom functions, 306-307

fixing recorded code, 61-62

help buttons, 170-171

looping through all files in a
directory, 91-92

macros, recording, 22-23

military time, entering in cells, 136

multicolumn list boxes, 505

named ranges, using with
VLOOKUP() function, 120-121

page setup problems, error handling,
524

password cracking, 530

specific cells, selecting with
SpecialCells method, 74-76

standard add-in security, 559

storing macros and forms with hidden
code workbooks, 562

cell progress indicator, building, 294-295

cells

empty cells, verifying, 73

highlighting, 283-286

military time, entering, 136

selecting with SpecialCells method,
74-76

visible cells, selecting, 203

Cells property, selecting ranges, 68-69

changing

case of text, 297-298

size of ranges, 71

characteristics of standard add-ins, 555-556

chart sheet events, 123, 137-140

embedded chart events, trapping,
161-163

for workbook events, 129-132

charts

built-in charts, specifying, 333-334

combo charts, creating, 359-363

creating

in Excel 2003 through Excel 2013,
340-341

in Excel 2007 through Excel 2013,
340-341

in Excel 2013, 338-340

exporting as a graphic, 372-373

formatting

in Excel 2013, 343-350

with Format method, 355-359

with SetElement method, 350-355

frequency charts, creating bins for,
365-368

location of, specifying, 336-337

OHLC charts, creating, 364-365

pivot charts, creating, 373-375

pivot tables, 231

AutoSort option, 246

blank cells, removing in values
area, 246

calculated data fields, 267

- calculated items, 268
 - compatibility in different Excel versions, 231-232
 - conceptual filters, 250-253
 - daily dates, grouping, 241-243
 - Data Model pivot table, building, 262-267
 - fields, adding to data area, 234-237
 - multiple value fields, 240-241
 - percentages, displaying, 243-245
 - pivot cache, defining, 232-233
 - reports, replicating for every product, 246-249
 - size of, determining, 238-240
 - slicers, 252-259
 - timeline slicer, 259-262
 - placing comments in, 282-283
 - referencing, 332-333
 - referring to, 337
 - size of, specifying, 336-337
 - sparklines
 - creating, 399-401
 - creating in a dashboard, 414-418
 - formatting, 405-413
 - scaling, 401-404
 - stacked area charts, creating, 368-372
 - title of, specifying, 342-343
- check boxes, 485-486**
- class modules**
- application-level events, trapping, 160-161
 - collections, creating, 168-170
 - custom objects
 - creating, 163
 - Property Get procedures, 165-166
 - Property Let procedures, 165-166
 - referencing, 163-164
 - embedded chart events, trapping, 161-163
 - Excel state class module, 290-292
 - inserting, 159
- cleaning up recorded code, 56-60**
- clients, training on error handling, 526-527**
- closing add-ins, 560**
- code**
- adding to new workbooks, 302-303
 - breakpoints, 49
 - examining in recorded macros, 23-25
 - protecting, 529
 - recorded
 - cleaning up, 56-60
 - fixing, 61-62
 - speeding up with arrays, 153-155
 - stepping through, 46-48
- Code group (Developer tab), 9**
- collections, 34-35**
- and controls, 492-494
 - creating
 - in class modules, 168-170
 - in standard modules, 167-168
- ColName() function, 327**
- color of text, filtering, 201**
- color scales, adding to a range, 384-385**
- columns**
- copying, 223
 - referring to with R1C1 references, 105
 - remembering column numbers associated with column letters, 107-108
- Columns property, referring to ranges, 72**
- combining workbooks, 276-277**
- combo charts, creating, 359-363**
- ComboBox control, 182-185**

CommandButton control, 180-182**commands, extracting unique list of values with Advanced Filter, 206-207****comments**

- adding to names, 114
- charts, placing in, 282-283
- listing, 279-281
- resizing, 281-282

comparing

- A1 versus R1C1 references, 99-100
 - case study, 102-103
- DAO and ADO, 470

compatibility

- 32-bit and 64-bit compatible API declarations, 511-512
- Excel versions
 - error handling, 530-531
 - pivot tables, 231-232
- Excel8CompatibilityMode property, 588
- file types supporting macros, 10-11
- Version property, 587-588

complex criteria, working with, 214-215**conceptual filters, 250-253****conditional formatting**

- Add method, 395-396
- AddAboveAverage method, 392
- AddTop10 method, 393
- AddUniqueValues method, 393-395
- NumberFormat property, 398
- xlExpression version, 396-397

conditions

- formula-based, 216-221
- for If...Then...Else constructs, 90-95
- list of values, replacing with, 214-221

configuring pivot tables, 233-234**constants**

- retrieving value of, 457
- SetElement method, 352

ContainsText() function, 324-325**content management system, using Excel as, 434-437****controlling**

- form fields in Word, 465-467
- illegal windows closing on userforms, 191-193

controls**ActiveX**

- attaching macros to, 553-554
- right-click menu, creating, 299-300
- adding at runtime, 496-502
- adding to Ribbon, 536-540
- adding to tabs, 535-536
- check boxes, 485-486
- and collections, 492-494
- combo boxes, 182-185
- command buttons, 180-182
- graphics, 187-188
- labels, 180-182
- list boxes, 182-185
- MultiPage, 189-190
- programming, 180
- RefEdit, 489-490
- renaming, 180
- scrollbar, 491-493
- spin buttons, 188-190
- tab strips, 487-489
- text boxes, 180-182
- toggle buttons, 491

Controls group (Developer tab), 10

converting

- custom Excel 2003 toolbar to Excel 2013, 547
- files to an add-in, 558
- workbooks to an add-in, 556-558

copying

- columns, 223
 - subsets of, 224-225
- data to separate worksheets, 277-278
- formulas, 101-102

CreateObject function, 454**creating**

- bins for frequency charts, 365-368
- charts
 - combo charts, 359-363
 - in Excel 2003 through Excel 2013, 340-341
 - in Excel 2007 through Excel 2013, 340-341
 - in Excel 2013, 338-340
 - OHLC charts, 364-365
 - pivot charts, 373-375
 - stacked area charts, 368-372
- collections
 - in class modules, 168-170
 - in standard modules, 167-168
- custom objects, 163
- global names, 112
- local names, 113
- reports from advanced filters, 226-229
- sparklines, 399-401
- user-defined functions, 305-307
- userforms, 176-177
- web pages with VBA, 434
- web queries, 420-423

criteria ranges, joining

- with logical AND, 214
- with logical OR, 213

CSS (Cascading Style Sheets), 571**CSV files, importing, 273-274****CurrentRegion property, selecting ranges, 73-76****custom Excel 2003 toolbar, converting to Excel 2013, 547****custom functions**

- BookOpen(), 309
- case study, 306-307
- ColName(), 327
- ContainsText(), 324-325
- DateTime(), 312-313
- FirstNonZeroLength(), 318
- GetAddress(), 326-327
- IsEmailValid(), 313-315
- LastSaved(), 312
- MSubstitute(), 318-319
- MyFullName(), 308
- MyName(), 308
- NumFilesInCurDir(), 310-311
- NumUniqueValues(), 316
- RetrieveNumbers(), 320
- ReturnMaxs(), 326
- ReverseContents(), 325
- SheetExists(), 309-310
- SortConcat(), 321-323
- StaticRAND(), 327-328
- StringElement(), 321
- SumColor(), 315
- UniqueValues(), 316-318
- Weekday(), 320-321
- WinUserName(), 311-312

custom icons, applying to Ribbon, 545-546**custom objects**

- creating, 163
- Property Get procedures, 165-166
- Property Let procedures, 165-166
- referencing, 163-164

custom properties, creating with UDTs, 172-174

custom sort order, specifying, 293

Custom UI Editor, 543

customized data, transposing, 286-288

customizing the Ribbon

accessing the file structure, 537-542

Custom UI Editor, 543

D

daily dates, grouping in pivot tables, 241-243

DAO (data access objects) versus ADO, 470

dashboards

creating, 413-418

sparklines, creating in, 414-418

data bars

adding to a range, 380-384

using two colors in a range, 390-392

data visualizations

applying to pivot tables, 270

color scales, adding to a range, 384-385

data bars

adding to a range, 380-384

using two colors in a range, 390-392

icon sets

adding to a range, 385-388

creating for a subset of a range, 388-390

methods for, 378-379

properties for, 378-379

DateTime() function, 312-313

declaring

APIs, 509-511

32-bit and 64-bit compatible declarations, 511-512

arrays, 149-150

multidimensional, 150

UDTs, 172

default settings, adjusting for macros, 11-14

defining

apps with XML, 571-572

pivot cache for pivot tables, 232-233

deleting

blank cells in pivot table values area, 246

records from Access database, 478-479

deselecting noncontiguous cells, 288-290

Developer tab

accessing, 9

Add-Ins group, 10

Code group, 9

Controls group, 10

Modify group, 10

dialog boxes

Go To Special, replacing loops with, 204

Record Macro dialog, filling out, 14-15

directory files

listing, 271-273

looping through, 91-92

Disable All Macros with Notification setting, 13-14

displaying

drill-down recordsets on Pivot Table, 292-293

percentages in pivot tables, 243-245

Do loops, 85-89

Until clause, 87-88

While clause, 87-88

While...Wend loops, 88-89

drill-down recordsets, displaying on pivot tables, 292-293

dynamic arrays, 155-156

dynamic date range, selecting with AutoFilter, 202-203

E

early binding, referencing Word object, 451-453

embedded chart events, trapping, 161-163

empty cells, verifying, 73

enabling events, 125

encountering errors on purpose, 526

error handling, 519-522

client training, 526-527

encountering errors on purpose, 526

On Error GoTo syntax, 522-523

Excel version compatibility, 530-531

ignoring errors, 524

page setup problems, case study, 524

passwords, 529-530

protecting code, 529

Runtime Error 9: Subscript Out of Range, 527-528

Runtime Error 1004: Method Range of Object Global Failed, 528-529

warnings, suppressing, 526

error messages for custom Ribbon, troubleshooting, 548-551

events

application-level events, 140-148

trapping, 160-161

chart sheet events, 137-140

embedded chart events, trapping, 161-163

enabling, 125

levels of, 123-124

parameters, 124-125

for userforms, 178

workbook events, 125-132

chart sheet events, 129-132

sheet events, 129-132

worksheet events, 132-136

examining code in recorded macros, 23-25

examples of APIs, 512-517

Excel 2003

charts, creating, 340-341

custom toolbar, converting to Excel 2013, 547

Excel 2007, creating charts, 340-341

Excel 2010, creating charts, 340-341

Excel 2013

charts

creating, 338-340

formatting, 343-350

Data Model pivot table, building, 262-267

Excel state class module, 290-292

Excel8CompatibilityMode property, 588

Excel, using as content management system, 434-437

existence of names, checking for, 119

existing instance of Word, referencing with GetObject function, 455-456

exiting For...Next loops after condition is met, 83-84

exporting

charts as a graphic, 372-373

data to Word, 278-279

extracting unique list of values

with Advanced Filter, 206-207

with VBA code, 207-210

F

field drop-downs, turning off in AutoFilter, 229-230

fields

form fields, controlling in Word, 465-467

pivot table

adding to pivot table data area, 234-237

manual filtering, 249-250

suppressing subtotals in multiple row fields, 269

file types supporting macros, 10-11

filenames, retrieving from user, 193-194

files

converting to an add-in, 558

listing, 271-273

text files, parsing, 274-275

filling arrays, 151-152

filling out

forms, 496-502

Record Macro dialog, 14-15

Filter in Place, running, 221-222

filtering

Advanced Filter command, extracting unique list of values with, 206-207

advanced filters

building, 205-206

reports, creating from, 226-229

AutoFilter, turning off field drop-downs, 229-230

by color, 201

data to separate worksheets, 277-278

with Filter in Place, 221-222

by icon, 201

multiple items, selecting, 200

pivot tables

conceptual filters, 250-253

fields, 249-250

slicers, 252-259

xlFilterCopy parameter, 222-225

FirstNonZeroLength() function, 318

fixing recorded code, 61-62

flow control

If statements, nesting, 95-97

If...Then...Else construct, conditions, 90-95

loops

Do loops, 85-89

For...Next loops, 79-84

Select Case construct, 94-95

for loops (JavaScript), 575

For statement, using variables in, 82

form controls, assigning macros, 18-19

form fields, controlling in Word, 465-467

Format method, formatting charts with, 355-359

formatting. See also conditional formatting

charts

in Excel 2013, 343-350

win/loss charts, 412-413

with Format method, 355-359

with SetElement method, 350-355

sparklines, 405-413

forms

helping users fill out, 496-502

transparent forms, 505-506

formula-based conditions, 216-221

formulas

A1, replacing with R1C1 formulas, 106-107

copying, 101-102

names, assigning, 114-115

R1C1, 99-100

- array formulas, entering, 108-109
- rows and columns, referring to, 105
- switching to, 100-101
- using with absolute references, 104-105
- using with mixed references, 105
- using with relative references, 104

For...Next loops, 79-84

- exiting after condition is met, 83-84
- nesting, 84
- For statement, using variables in, 82
- variations of, 82-83

frequency charts, creating bins for, 365-368**functions**

- CreateObject, 454
- custom functions
 - BookOpen(), 309
 - case study, 306-307
 - ColName(), 327
 - ContainsText(), 324-325
 - DateTime(), 312-313
 - FirstNonZeroLength(), 318
 - GetAddress(), 326-327
 - IsEmailValid(), 313-315
 - LastSaved(), 312
 - MSubstitute(), 318-319
 - MyFullName(), 308
 - MyName(), 308
 - NumFilesInCurDir(), 310-311
 - NumUniqueValues(), 316
 - RetrieveNumbers(), 320
 - ReturnMaxs(), 326
 - ReverseContents(), 325
 - SheetExists(), 309-310
 - SortConcat(), 321-323

StaticRAND(), 327-328

StringElement(), 321

SumColor(), 315

UniqueValues(), 316-318

Weekday(), 320-321

WinUserName(), 311-312

GetObject, 455-456

InputBox(), 175-176

ISEMPTY(), 73

for JavaScript, 572-573

MsgBox(), 176

user-defined

- creating, 305-307

- sharing, 307-308

G**GetAddress() function**, 326-327**GetObject function**, 455-456**global names**, 111-112

- creating, 112

Go To Special dialog box, replacing loops with, 204**Graphic controls, adding to userforms**, 187-188**graphics**

- exporting charts as, 372-373

- SmartArt, 586

grouping daily dates in pivot tables, 241-243**H****hard drive, trusted locations**

- adding, 12

- allowing macros outside of, 13

Hello World app

- creating, 563-567

- interactivity, adding, 568-570

help buttons, case study, 170-171

help files (VBA), 37-38

recorded macros, examining, 39-46

hidden workbooks as alternative to add-ins, 561-562

hiding

names, 119

userforms, 177-178

highlighting cells, 283-286

hovering as means of querying, 52

HTML

buttons, 570-571

CSS, 571

tags, 570

hyperlinks

running macros from, 554

using in userforms, 495-496

I

icon sets

adding to a range, 385-388

creating for a subset of a range, 388-390

icons

custom icons, applying to Ribbon, 545-546

filtering, 201

Microsoft Office icons, applying to Ribbon, 544-545

If statements, nesting, 95-97

If...Then...Else construct, conditions, 90-95

ignoring errors, 524

illegal window closing, controlling on userforms, 191-193

Immediate window (VB Editor), 50-52

importing

CSV files, 273-274

text files, 439-445

input boxes, 175-176

protected password box, creating, 295-297

InputBox() function, 175-176

inserting class modules, 159

interactivity, adding to Hello World app, 568-570

interrupting macros, 125

Intersect method, creating new ranges from overlapping ranges, 72

IsEmailValid() function, 313-315

ISEMPTY() function, 73

Item property, 68

J

Jet Database Engine, 470

Jiang, Wei, 293

joining

criteria ranges

with logical AND, 214

with logical OR, 213

multiple ranges, 72

JS (JavaScript)

arrays, 574-575

functions, 572-573

if statements, 576

for loops, 575

math functions, 578-580

in Office apps, 581-582

operators, 578-579

select...case statement, 576-577

strings, 574

variables, 573-574

K

Kaji, Masaru, 273, 286

Kapor, Mitch, 29

keyboard shortcut, running macros with, 551

Klann, Daniel, 295

L

Label control, 180-182

LastSaved() function, 312

late binding, referencing Word object, 453-454

layout of pivot tables, changing from Design tab, 268

learning VBA, barriers to learning, 7-9

levels of events, 123-124

list of values, replacing with condition, 214-221

ListBox control, 182-185

listing comments, 279-281

listing files in a directory, 271-273

local names, 111-112

 comments, adding, 114

 creating, 113

location of charts, specifying, 336-337

logical AND, joining multiple criteria ranges, 214

logical operators (JavaScript), 578-579

logical OR, joining multiple criteria ranges, 213

loops

 Do loops, 85-89

 Until clause, 87-88

 While clause, 87-88

 While...Wend loops, 88-89

 For...Next loops, 79-84

 exiting after condition is met, 83-84

 nesting, 84

 For statement, using variables in, 82

 variations of, 82-83

 looping through all files in a directory, case study, 90-92

 for loops (JavaScript), 575

 replacing

 with AutoFilter, 197-203

 with Go To Special dialog box, 204

lower bound of arrays, declaring, 150

M

macro button

 adding to Quick Access toolbar, 17

 adding to Ribbon, 16

macros

 assigning

 to form control, 18-19

 to shape, 18-19

 to text box, 18-19

 attaching to shapes, 552

 default settings, adjusting, 11-14

 Disable All Macros with Notification setting, 13-14

 file types supporting, 10-11

 interrupting, 125

 recorded

 code, examining, 23-25

 examining, 39-46

 recording

 case study, 22-23

 code, examining, 23-25

 obstacles to, 21-31

 relative references, 26-29

 running

 with command button, 552

 from hyperlinks, 554

 with keyboard shortcut, 551

- scheduled macros, canceling, 429
 - scheduling, 429-432
- mathematical operators (JavaScript), 578-579**
- message boxes, 176**
- methods**
 - Add, 395-396
 - AddAboveAverage, 392
 - AddTop10, 393
 - AddUniqueValues, 393-395
 - for data visualizations, 378-379
 - Format, formatting charts with, 355-359
 - Intersect, creating new ranges from overlapping ranges, 72
 - OnTime, 427-432
 - parameters, 35-36
 - SetElement, formatting charts with, 350-355
 - SpecialCells
 - selecting cells with, 74-76
 - selecting data with, 298-299
 - Union, joining multiple ranges, 72
- Microsoft Office icons, applying to Ribbon, 544-545**
- Miles, Tommy, 275, 276, 279**
- military time, entering in cells, 136**
- mixed references, using with R1C1 references, 105**
- Moala, Ivan F., 283, 297**
- modeless userforms, 495**
- Modify group (Developer tab), 10**
- MsgBox() function, 176**
- MSubstitute() function, 318-319**
- multicolumn list boxes, case study, 505**
- multidimensional arrays, declaring, 150**
- MultiPage control, 189-190**
- multiple ranges, joining, 72**

- multiple value fields (pivot tables), 240-241**
- MyFullName() function, 308**
- MyName() function, 308**

N

- Name Manager, 111-112**
- named ranges, 66**
 - VLOOKUP() function, using with, 120-121
- names**
 - arrays, using with, 117-118
 - assigning
 - to formulas, 114-115
 - to numbers, 116
 - to strings, 115-116
 - to tables, 117
 - checking for existence of, 119
 - comments, adding, 114
 - deleting, 113
 - global, 111-112
 - creating, 112
 - hiding, 119
 - local, 111-112
 - creating, 113
 - reserved names, 118-119
- Napa Office 365 Development Tools, 582**
- nesting**
 - For...Next loops, 84
 - If statements, 95-97
- New keyword, referencing Word application, 454**
- noncontiguous cells, selecting/deselecting, 288-290**
- noncontiguous ranges, returning with Areas collection, 76**
- NumberFormat property, 398**
- numbers, assigning names to, 116**

NumFilesInCurDir() function, 310-311

NumUniqueValues() function, 316

O

Object Browser, 55-56

object variables, 89-90

object-oriented languages, 34

objects

collections, 34-35

custom objects

creating, 163

Property Get procedures, 165-166

Property Let procedures, 165-166

referencing, 163-164

DAO versus ADO, 470

in Microsoft Word, 458-465

programming, changes in Excel 2013,
583-586

properties, 36-37

RANGE, 65

referring to, 65

watching, 54-55

Word object

referencing with early binding,
451-453

referencing with late binding, 453-
454

WORKSHEET, 65

obstacles to recording macros, 21-31

**Office apps, JavaScript incorporation in,
581-582**

Offset property, referring to ranges, 69-70

**OHLC (Open-High-Low-Close) charts,
creating, 364-365**

Oliver, Nathan P., 271, 301

**On Error GoTo syntax, error handling, 522-
523**

OnTime method, 427-432

operators (JavaScript), 578-579

**option buttons, adding to userforms, 186-
187**

**overlapping ranges, creating new ranges
from, 72**

Ozgur, Suat Mehmet, 274

P

page setup problems, case study, 524

parameters, 35-36

event parameters, 124-125

xlFilterCopy, 222-225

parsing text files, 274-275

passing arrays, 156

passwords

cracking, 530

error handling, 529-530

protected password box, creating,
295-297

Peltier, Jon, 372

**percentages, displaying in pivot tables,
243-245**

Pieterse, Jan Karel, 512

pivot cache, defining, 232-233

pivot charts, creating, 373-375

pivot tables, 231

AutoSort option, 246

blank cells, removing in values area,
246

calculated data fields, 267

calculated items, 268

compatibility in different Excel
versions, 231-232

conceptual filters, 250-253

configuring, 233-234

daily dates, grouping, 241-243

Data Model pivot table, building,
262-267

- data visualizations, applying, 270
 - drill-down recordsets, displaying, 292-293
 - fields
 - adding to data area, 234-237
 - manual filtering, 249-250
 - multiple value fields, 240-241
 - layout, changing from Design tab, 268
 - percentages, displaying, 243-245
 - pivot cache, defining, 232-233
 - report layout, 269
 - reports, replicating for every product, 246-249
 - size of, determining, 238-240
 - slicers, 252-259
 - subtotals, suppressing for multiple row fields, 269
 - timeline slicer, 259-262
- Pope, Andy, 372**
- procedural languages, BASIC, 33-34**
- procedures**
- Property Get procedures, 165-166
 - Property Let procedures, 165-166
- programming**
- controls, 180
 - objects, changes in Excel 2013, 583-586
- programming languages**
- BASIC, 33-34
 - comparing to VBA, 8-15
 - object-oriented, 34
- Project Explorer window (VB Editor), 20-21**
- properties, 36-37**
- Cells, selecting ranges, 68-69
 - Columns, referring to ranges, 72
 - CurrentRegion, selecting ranges, 73-76

- custom properties, creating with UDTs, 172-174
- for data visualizations, 378-379
- Excel8CompatibilityMode, 588
- Item, 68
- NumberFormat, 398
- Offset, referring to ranges, 69-70
- Resize, 71
- Rows, referring to ranges, 72
- ShowDetail, 268
- Version, 587-588

Properties window (VB Editor), 21**Property Get procedures, 165-166****Property Let procedures, 165-166****protected password box, creating, 295-297****protecting code, 529****publishing data to a web page, 432-438**

Q

querying in VB Editor

- by hovering, 52
- with Watches window, 53

Quick Access toolbar, adding macro button, 17**Quick Analysis tool, 585**

R

R1C1 references, 99-100

- array formulas, entering, 108-109
- columns and rows, referring to, 105
- copying, 101-102
- replacing A1 formulas with, 106-107
- switching to, 100-101
- using with absolute references, 104-105
- using with mixed references, 105
- using with relative references, 104

RANGE object, 65

referring to, 65

ranges, 65

changing size of, 71

color scales, adding to, 384-385

creating from overlapping ranges, 72

data bars, adding to, 380-384

dynamic date ranges, selecting with
AutoFilter, 202-203

icon sets, adding to, 385-388

multiple, joining, 72

named ranges, 66

noncontiguous, returning with Areas
collection, 76

referring to

in other sheets, 67

shortcuts, 66

relative to other ranges, specifying,
67-68

selecting, 68-69

with CurrentRegion property,
73-76

specifying, 66

with Columns and Rows
properties, 72

subsets of, creating icon sets for, 388-
390

tables, referencing, 77

two-color data bars, 390-392

reading text files

to memory, 274-275

one row at a time, 445-449

Record Macro dialog, filling out, 14-15**recorded macros**

cleaning up, 56-60

examining, 39-46

fixing, 61-62

recording macros

case study, 22-23

code, examining, 23-25

obstacles to, 21-31

relative references, 26-29

RefEdit control, 489-490**referencing**

charts, 332-333

custom objects, 163-164

tables, 77

referring

to charts, 337

to ranges, 65

with Columns and Rows
properties, 72

with Offset property, 69-70

in other sheets, 67

relative to other ranges, 67-68

shortcuts, 66

reinitializing arrays, 155**relative references**

recording macros with, 26-29

using with R1C1 references, 104

RELS file, 537-543**removing**

blank cells in pivot table values area,
246

names, 113

standard add-ins, 560

renaming controls, 180**replacing**

A1 formulas with single R1C1
formula, 106-107

list of values with a condition, 214-
221

loops

with AutoFilter, 197-203

with Go To Special dialog box,
204

- replicating pivot table reports for every product, 246-249**
- report layout (pivot tables), 269**
- reports, creating from advanced filters, 226-229**
- reserved names, 118-119**
- Resize property, 71**
- resizing**
 - comments, 281-282
 - ranges, 71
- RetrieveNumbers() function, 320**
- retrieving**
 - constant values, 457
 - data from arrays, 152-153
 - data from the Web, 419-427
 - filenames from userforms, 193-194
 - records from Access database, 475-476
 - stock ticker averages, 301
 - unique combinations of two or more fields, 211
- returning above-average records with formula-based conditions, 221**
- returning noncontiguous ranges with Areas collection, 76**
- ReturnMaxs() function, 326**
- ReverseContents() function, 325**
- Ribbon**
 - controls, adding, 536-540
 - custom icons, applying, 545-546
 - customizing
 - accessing the file structure, 537-542
 - Custom UI Editor, 543
 - macro button, adding, 16
 - Microsoft Office icons, applying, 544-545
 - tabs, adding, 534-535

- right-click menu, creating for ActiveX object, 299-300**
- rows, referring to with R1C1 references, 105**
- Rows property, referring to ranges, 72**
- Ruiz, Juan Pablo, 290**
- running macros**
 - with command button, 552
 - from hyperlinks, 554
 - with keyboard shortcut, 551
- runtime, adding controls during, 496-502**
- Runtime Error 9: Subscript Out of Range, error handling, 527-528**
- Runtime Error 1004: Method Range of Object Global Failed, error handling, 528-529**

S

- saving new files as .xlsm file type, 11**
- scaling sparklines, 401-404**
- scheduling**
 - macros, 429-430
 - verbal reminders, 430-431
 - web query updates, 428-429
- scrollbar control, 491-493**
- SDI (single document interface), 533, 583-584**
- Search box (AutoFilter), 200-201**
- security**
 - Disable All Macros with Notification setting, 13-14
 - standard add-ins, 559
 - trusted locations
 - adding, 12
 - allowing macros outside of, 13
- Select Case construct, 94-95**

selecting

cells

- noncontiguous cells, 288-290
- with SpecialCells method, 74-76, 298-299
- visible cells, 203

dynamic date range with AutoFilter, 202-203

multiple items with AutoFilter, 200

ranges, 68-69

with CurrentRegion property, 73-76

separating worksheets into workbooks, 275-276

SetElement method

constants, 352

formatting charts with, 350-355

setting breakpoints, 53-54

settings, VB Editor, 20

shapes

attaching macros to, 552

macros, assigning, 18-19

shared Access database, creating, 471

sharing user-defined functions, 307-308

sheet events for workbook events, 129-132

SheetExists() function, 309-310

shortcuts, referring to ranges, 66

ShowDetail property, 268

size of charts, specifying, 336-337

size of pivot tables, determining, 238-240

sliders, 252-259

timeline slicer, 259-262

SmartArt, 586

SortConcat() function, 321-323

sorter() function, 323-324

sparklines

creating, 399-401, 414-418

formatting, 405-413

scaling, 401-404

SpecialCells method

selecting cells with, 74-76

selecting data with, 298-299

specifying

built-in charts, 333-334

custom sort order, 293

ranges, 66

with Columns and Rows properties, 72

in other sheets, 67

relative to other ranges, 67-68

speeding up code with arrays, 153-155

SpinButton control, 188-190

SQL servers, 483-484

stacked area charts, creating, 368-372

standard add-ins

characteristics of, 555-556

closing, 560

hidden workbooks as alternative to, 561-562

removing, 560

security, 559

standard modules, creating collections, 167-168

StaticRAND() function, 327-328

stepping through code, 46-48

stock ticker averages, retrieving, 301

storing macros and forms with hidden code workbooks, 562

StringElement() function, 321

strings

for JavaScript, 574

names, assigning, 115-116

subsets

of columns, copying, 224-225

of ranges, creating icon sets for, 388-390

SumColor() function, 315

summarizing records in Access database, 479-480

suppressing

 pivot table subtotals for multiple row fields, 269

 warnings, 526

switching to R1C1 references, 100-101

syntax, specifying ranges, 66

T

tab strips, 487-489

tables

 checking for existence of in Access database, 480-481

 names, assigning, 117

 referencing, 77

tabs

 adding to Ribbon, 534-535

 controls, adding, 535-536

tags (HTML), 570

text, changing case of, 297-298

text boxes, assigning macros to, 18-19

text files

 importing, 439-445

 parsing, 274-275

 reading one row at a time, 445-449

 writing, 449

TextBox control, 180-182

timeline slicer, 259-262

tips for cleaning up recorded code, 56-60

tips for using macro recorder, 31

title of charts, specifying, 342-343

toggle buttons, 491

toolbars

 custom Excel 2003 toolbar, converting to Excel 2013, 547

 UserForm, 485

tools of ADO, 472-473

training clients on error handling, 526-527

transparent forms, 506-505

transposing customized data, 286-288

trapping

 application-level events, 160-161

 embedded chart events, 161-163

troubleshooting custom Ribbon error messages, 548-551

trusted locations

 adding to hard drive, 12

 allowing macros outside of, 13

turning off field drop-downs in AutoFilter, 229-230

U

UDTs (user-defined types)

 custom properties, creating, 172-174

 declaring, 172

Union method, joining multiple ranges, 72

unique list of values, extracting

 with Advanced Filter command, 206-207

 with VBA code, 207-210

UniqueValues() function, 316-318

Until clause (Do loops), 87-88

updates, scheduling, 428-429

updating web queries, 423-424

upper bound of arrays, declaring, 150

Urtis, Tom, 281, 288, 292-294, 299

user-defined functions

 creating, 305-307

 sharing, 307-308

UserForm toolbar, 485

userforms

 calling, 177-178

controls

- adding at runtime, 496-502
- check boxes, 485-486
- combo boxes, 182-185
- command buttons, 180-182
- labels, 180-182
- list boxes, 182-185
- MultiPage control, 189-190
- programming, 180
- RefEdit, 489-490
- scrollbar, 491-493
- spin buttons, 188-190
- tab strips, 487-489
- text boxes, 180-182
- toggle buttons, 491
- creating, 176-177
- events, 178
- field entry, verifying, 191
- filenames, retrieving, 193-194
- graphics, adding, 187-188
- helping users fill out, 496-502
- hiding, 177-178
- hyperlinks, 495-496
- illegal window closing, controlling, 191-193
- input boxes, 175-176
- message boxes, 176
- modeless, 495
- option buttons, adding, 186-187

V

variables

- for JavaScript, 573-574
- object variables, 89-90
- in For statements, 82
- Variant-type, 151
- Variant-type variables, 151**

variations of For...Next loops, 82-83**VB Editor**

- code
 - breakpoints, 49
 - stepping through, 46-48
- controls, programming, 180
- Immediate window, 50-52
- Object Browser, 55-56
- Project Explorer window, 20-21
- Properties window, 21
- querying
 - by hovering, 52
 - with Watches window, 53
- settings, 20
- userforms, creating, 176-177

VBA (Visual Basic for Applications), 7

- AutoShow feature, 255-257
- comparing to BASIC, 8-15
- error handling, 519-522
 - client training, 526-527
 - On Error GoTo syntax, 522-523
 - ignoring errors, 524
- help files, 37-38
- learning, barriers to, 7-9
- pivot tables, creating, 232-240
- programming language components, 37
- web pages, creating with, 434
- web queries, building, 424-427

VBA Extensibility, adding code to new workbooks, 302-303**verbal reminders, scheduling, 430-431****verifying**

- empty cells, 73
- userform field entry, 191

Version property, 587-588**visible cells, selecting, 203**

VLOOKUP() function, using named ranges for, 120-121

W

Wallentin, Dennis, 277

warnings, suppressing, 526

Watches window (VB Editor)

- as means of querying, 53
- setting breakpoints, 53-54

WCL_FTP utility, 437-438

web pages

- creating with VBA, 434
- hyperlinks, using in userforms, 495-496
- publishing data to, 432-438

web queries

- capturing data periodically, 427-428
- creating, 420-423
- updates, scheduling, 428-429
- updating, 423-424

Weekday() function, 320-321

While clause (Do loops), 87-88

While...Wend loops, 88-89

Windows API, 509

win/loss charts, formatting, 412-413

WinUserName() function, 311-312

Word

- exporting data to, 278-279
- form fields, controlling, 465-467
- objects, 458-465

Word object, referencing

- with early binding, 451-453
- with late binding, 453-454

workbook events, 123, 125-132

- chart events, 129-132
- sheet events, 129-132

workbooks

- code, adding to, 302-303
- combining, 276-277
- converting to an add-in, 556-558
- creating from worksheets, 275-276
- Disable All Macros with Notification setting, 13-14
- Name Manager, 111-112

working with complex criteria, 214-215

worksheet events, 123, 132-136

WORKSHEET object, 65

worksheets, separating into workbooks, 275-276

writing text files, 449

X-Y-Z

XcelFiles, 283

xlExpression version (conditional formatting), 396-397

xlFilterCopy parameter, 222-225

.xslm file type, saving new files as, 11

XML, defining apps, 571-572

zipped files, accessing Ribbon file structure with, 537-542